ACOUSTIC AND SPEECH SIGNAL CLASSIFICATION: FROM FEATURES TO CLASSIFIERS

JIANG YUECHI

PhD

The Hong Kong Polytechnic University

2021

The Hong Kong Polytechnic University

Department of Electronic and Information Engineering

Acoustic and Speech Signal Classification: From Features to Classifiers

Jiang Yuechi

A thesis submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

July 2020

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____ (Signed)

Jiang Yuechi (Name of student)

Abstract

Nowadays, with the fast development of multimedia technologies, acoustic and speech signals are playing more and more important roles. Acoustic and speech signals can deliver beneficial information for applications such as security check, audio authentication, environment analysis, and context-aware navigation. Most of the time, the study of acoustic and speech signals is conducted by performing classification. For example, verifying the similarity of two acoustic signals can be treated as checking whether they belong to the same class or not; detecting the occurrence of an acoustic event can be treated as finding out a segment that belongs to a specific class. The work discussed in this thesis focuses on the classification of acoustic and speech signals, which is a fundamental problem covering a wide range of applications.

Three key components that form a classification system are 1) feature representations, 2) classifiers, and 3) feature transformation techniques. These components are all important to the success of a classification system and deserve a comprehensive investigation.

Good feature representations are crucial for the regular operation of a classification system. In general, a good feature representation should carry enough information for well describing the acoustic sample. This often implicitly requires the dimensionality of the feature representation to be high. In this thesis, we consider two high-dimensional feature representations, viz. the Gaussian supervector (GSV) and the identity vector (i-vector). GSV is fast in computation, but its dimensionality is unchangeable. I-vector has a changeable dimensionality, but its computation can be time-consuming owing to the requirement of estimating additional model parameters. To balance the computational efficiency and the dimensional flexibility, we propose feature representations based on the mixture of factor analyzers (MFA), such as the MFA latent vector (MFALV). MFALV is comparable to GSV and i-vector in effectiveness, and has a similar flexibility in dimensionality but a higher computational efficiency as compared to i-vector.

By analyzing the similarity between different feature representations, we propose the generic supervector, which generalizes GSV and MFALV. I-vector can then be obtained by post-processing the

generic supervector. It is noticed that the generic supervector can explain the structure of the classic convolutional neural network and the residual network.

The support vector machine (SVM) and the probabilistic linear discriminant analysis (PLDA) model are two prevalent classifiers for classifying high-dimensional feature representations, such as GSV and i-vector. Although PLDA may outperform SVM for speaker verification, it is inefficient in handling many training data, especially when the dimensionality of the feature representation is high. To address the inefficiency issue of PLDA, we propose a scalable formulation that enables it to do classification efficiently irrespective of the quantity of the training data.

The sparse representation (SR) and the SR-based classifier (SRC) are also good at classifying high-dimensional feature representations. Still, the computation of SR is slow because of the L1-norm constraint involved in the objective function. The collaborative representation (CR), which replaces the L1-norm constraint by the L2-norm constraint, is computationally more efficient than SR. To boost the discrimination ability of CR, we propose the discriminative CR (DCR), which incorporates the class information and thus better suits the classification tasks.

Two probabilistic models are also investigated, viz. the Gaussian mixture model (GMM) and the restricted Boltzmann machine (RBM). Although both can be used for probability estimation and classification, their different model assumptions determine their different applicability. GMM is suitable for processing low-dimensional decorrelated feature representations, whereas RBM is suitable for processing high-dimensional correlated feature representations. Both yet require a large number of training data. Another important use of RBM is to work as the basic building block for constructing a deep belief net (DBN). By adding a softmax layer at the end of DBN, a deep neural network (DNN) is formed. This DBN-DNN is a discriminative classification model. Our experiments then validate the importance of a high dimensionality for DBN-DNN to take effect.

If the original feature representation does not work well, suitable feature transformation techniques may help. Two feature transformation techniques are popular in the area of speech processing, viz. the nuisance attribute projection (NAP), and the linear discriminant analysis (LDA). As a generalization,

their kernel versions, viz. the kernel NAP (KNAP) and the kernel discriminant analysis (KDA), introduce an implicit feature mapping before performing the projection, which may be beneficial in some circumstances. The detailed derivations for the kernel-based formulations are given in this thesis, and comparative experiments are conducted to investigate the effectiveness of different feature transformation techniques.

To comprehensively investigate the performance of different feature representations, classifiers, and feature transformation techniques, we perform experiments on four different datasets, including two speech datasets for doing speaker identification tasks and two acoustic datasets for doing acoustic scene classification tasks. The experimental results and discussions reveal the characteristics of different feature representations, feature transformation techniques, and classifiers. In general, no one type of feature representation or classifier always surpasses the others for all conditions, which implies the importance of choosing a suitable combination. We hope these analyses may help devise new features representations, classifiers, and feature transformations.

Acknowledgements

I would like to express my appreciation to my chief supervisor, Dr. Frank H. F. Leung, for his guidance and support, and most importantly, his confidence on me. Without his guidance and confidence, I will never be able to embark on a research career.

I would also like to thank my parents for their patience and understanding, and the financial support. Special thanks are devoted to my girlfriend, Carmen, who is always standing with me and keeping encouraging me at all time.

The work described in this thesis was substantially supported by a grant from The Hong Kong Polytechnic University (Project Account Code: RUG7).

Statement of Originality

The following contributions reported in this thesis are claimed to be original.

1. *The analysis related to the Gaussian supervector (Chapter 3.1).* Gaussian supervector (GSV) originates from the comparison of two Gaussian mixture models using the Kullback-Leibler (KL) divergence. It can be shown that the KL divergence is upper bounded under certain conditions, where a detailed proof for the upper bound is given in Chapter 3.1. Besides, a simple way to determine the relevance factor used to compute GSV is proposed. Extensions of GSV are also proposed.
2. *The analysis related to i-vector and its relationship with GSV (Chapter 3.2).* The log-likelihood for estimating the model parameters of i-vector is derived in Chapter 3.2, which helps understand the characteristics of i-vector. The relationship between i-vector and GSV is analyzed, which shows that i-vector can be treated as an affine transformation of GSV.
3. *The feature representations based on mixture of factor analyzers (Chapter 3.3).* New feature representations are proposed based on the mixture of factor analyzers (MFA), such as the MFA latent vector (MFALV). MFALV is also compared to i-vector in terms of formulation and computational complexity. The theoretical analysis shows that MFALV can be more efficient in computation than i-vector, while maintaining a similar flexibility in dimensionality.
4. *The comprehensive theoretical and experimental comparisons of different feature representations (Chapters 3.4 & 4).* Theoretical comparisons are made for the characteristics and the computational complexity of different feature representations, including GSV, i-vector and MFA-based feature representations. Experiments on two speaker identification tasks are performed to evaluate the effectiveness and the efficiency of different feature representations.
5. *The generic supervector (Chapter 3.5).* The generic supervector is proposed as the generalization of GSV and MFA-based feature representations. It can also be used to explain the structure of a classic convolutional neural network and the robustness of the residual network.

6. *The formulation for restricted Boltzmann machine to be used for probability estimation in the case of real-valued input feature vector (Chapter 5.2.4).* This formulation makes restricted Boltzmann machine (RBM) able to be used as a probabilistic model for pattern recognition tasks.
7. *The theoretical and experimental comparisons between Gaussian mixture model and RBM (Chapters 5.3 & 6.1).* The analysis of Gaussian mixture model (GMM) and RBM reveals their different characteristics and applicable scenarios. GMM is good at handling low-dimensional decorrelated feature vectors, while RBM is good at handling high-dimensional correlated feature vectors. In some sense, GMM and RBM capture complementary information from the feature vector. The experimental results on acoustic scene classification also validates the importance of a high dimensionality for RBM to take effect.
8. *The scalable version of the probabilistic linear discriminant analysis model (Chapter 5.5.3).* The scalable probabilistic linear discriminant analysis (PLDA) enables a PLDA model to efficiently use many training data to perform classification.
9. *The discriminative collaborative representation (Chapter 5.6.3).* The discriminative collaborative representation (DCR) is proposed as an alternative to the collaborative representation (CR) and the sparse representation (SR). DCR is more discriminative than CR for pattern recognition, and is computationally more efficient than SR. This is demonstrated in an acoustic scene classification task and a speaker identification task.
10. *The comparisons of different classifiers in two acoustic scene classification tasks and two speaker identification tasks (Chapter 6).* The comparisons in terms of effectiveness and efficiency demonstrate the characteristics of different classifiers. In general, no one classifier always surpasses the others in all cases, and different classifiers have their own advantages and disadvantages.
11. *The derivation and the analysis of the linear discriminant analysis and the kernel discriminant analysis (Chapter 7.1).* Two versions of the linear discriminant analysis (LDA) and the kernel discriminant analysis (KDA) are derived in detail. KDA, as the kernel version of LDA, provides an implicit feature mapping using the kernel function. The existence of the kernel version is also discussed.

12. *The extension of the nuisance attribute projection to a general projection technique (Chapter 7.2).* The nuisance attribute projection (NAP) is extended to be a general projection technique like LDA. The kernel version of NAP (KNAP) is derived in detail, and the existence of the kernel version is discussed. The relationship between NAP and LDA is also analyzed, and the equivalence between NAP and LDA under certain conditions is proved.
13. *The experimental comparisons of LDA, KDA, NAP and KNAP as a projection technique (Chapter 8).* The experimental results in a speaker identification task implies that the kernel-based projection techniques do not always outperform the linear projection techniques. In addition, the effectiveness of the projection techniques also depends on the characteristics of the feature vector.

Author's Publications

Journal Papers

[1] Y. Jiang and F. H. F. Leung, "Source microphone recognition aided by a kernel-based projection method", *IEEE Trans. on Information Forensics and Security*, vol. 14, no. 11, pp. 2875-2886, 2019.

[2] Y. Jiang and F. H. F. Leung, "Vector-based feature representations for speech signals: from supervector to latent vector", *IEEE Trans. on Multimedia*, early access.

[3] Y. Jiang and F. H. F. Leung, "Investigating and improving the utility of probabilistic linear discriminant analysis for acoustic signal classification", *Digital Signal Processing*, major revision.

Conference Papers

[1] Y. Jiang and F. H. F. Leung, "Gaussian mixture model and Gaussian supervector for image classification," in *Proc. IEEE Int. Conf. on Digital Signal Processing (DSP)*, 2018, pp. 1-5.

[2] Y. Jiang and F. H. F. Leung, "Fisher discriminant analysis with new between-class scatter matrix for audio signal classification," in *Proc. IEEE Int. Conf. on Digital Signal Processing (DSP)*, 2018, pp. 1-5.

[3] Y. Jiang and F. H. F. Leung, "A class-dependent background model for speech signal feature extraction," in *Proc. IEEE Int. Conf. on Digital Signal Processing (DSP)*, 2018, pp. 1-5.

[4] Y. Jiang and F. H. F. Leung, "Comparison of supervector and majority voting in acoustic scene identification," in *Proc. IEEE Int. Conf. on Digital Signal Processing (DSP)*, 2018, pp. 1-5.

[5] Y. Jiang and F. H. F. Leung, "Discriminative collaborative representation and its application to audio signal classification," in *Proc. IEEE Int. Conf. on Pattern Recognition (ICPR)*, 2018, pp. 31-36.

[6] Y. Jiang and F. H. F. Leung, "Generalized Fisher discriminant analysis as a dimensionality reduction technique," in *Proc. IEEE Int. Conf. on Pattern Recognition (ICPR)*, 2018, pp. 994-999.

[7] Y. Jiang and F. H. F. Leung, "The scalable version of probabilistic linear discriminant analysis and its potential as a classifier for audio signal classification," in *Proc. IEEE Int. Joint Conf. on Neural Networks (IJCNN)*, 2018, pp. 1-7.

[8] Y. Jiang and F. H. F. Leung, "Using double regularization to improve the effectiveness and robustness of Fisher discriminant analysis as a projection technique," in *Proc. IEEE Int. Conf. on Neural Networks (IJCNN)*, 2018, pp. 1-7.

[9] Y. Jiang and F. H. F. Leung, "Using regularized Fisher discriminant analysis to improve the performance of Gaussian supervector in session and device identification," in *Proc. IEEE Int. Conf. on Neural Networks (IJCNN)*, 2017, pp. 705-712.

[10] Y. Jiang and F. H. F. Leung, "Mobile phone identification from speech recordings using weighted support vector machine," in *Proc. Annual Conference of the IEEE Industrial Electronics Society (IECON)*, 2016, pp. 963-968.

Table of Contents

CERTIFICATE OF ORIGINALITY ii

Abstract iii

Acknowledgements vi

Statement of Originality vii

Author's Publications x

Table of Contents xii

List of Figures xviii

List of Tables xx

List of Abbreviations xxi

Chapter 1 Introduction 1

1.1 Problem Definition 1

1.2 A Generic Classification Framework 2

1.3 Limitations of Existing Techniques and Proposed Methods 6

1.4 Outline of the Thesis 8

1.5 Datasets Description 9

Chapter 2 Literature Review 12

2.1 Feature Representations 12

2.2 Classifiers 14

2.3 Sample-level Feature Transformation Techniques 16

Chapter 3 Feature Representations: Theoretical Analysis 18

3.1 Gaussian Supervector 18

3.1.1 Basic Formulation 18

3.1.2 Rationale 20

3.1.3 Extensions 22

3.1.4 Relevance Factor 22

3.2 I-vector 23

3.2.1 Basic Formulation 23

3.2.2 Rationale 26

3.2.3 Relationship with GSV 27

3.3 Feature Representations based on Mixture of Factor Analyzers 29

3.3.1 Basic Formulation 29

3.3.2 Comparison with I-vector 33

3.4 Comparison of Different Vector-based Representations 35

3.4.1 Characteristics 35

3.4.2 Computational Complexity 37

3.4.2.1 Space Complexity 37

3.4.2.2 Time Complexity 38

3.5 Generic Supervector 39

3.5.1 A General Formulation 39

3.5.2 Relationship with Neural Networks 42

3.5.2.1 Interpreting Convolutional Neural Network 42

3.5.2.2 Interpreting Residual Network 43

Chapter 4 Feature Representations: Experimental Comparison 45

4.1 Experimental Comparison between GSV and Its Extensions 45

4.2 Experimental Comparison of Different Representations 46

4.2.1 Effectiveness and Efficiency of Different Feature Representations 46

4.2.2 Dimensionality Flexibility of I-vector and MFALV 49

Chapter 5 Classifiers: Theoretical Analysis 52

5.1 Gaussian Mixture Model 52

5.1.1 Expectation-Maximization Algorithm for Mixture Models 52

5.1.2 Fundamentals of GMM 53

5.1.3 Mixture Splitting Strategies 54

5.1.4 GMM for Classification 57

5.2 Restricted Boltzmann Machine 57

5.2.1 Fundamentals of RBM 58

5.2.2 Gaussian RBM 61

5.2.3 Deep Belief Net 62

5.2.4 RBM for Classification 63

5.3 Comparison between GMM and RBM 66

5.4 Support Vector Machine 67

5.4.1 Binary SVM 67

5.4.2 Multi-class SVM 70

5.4.3 Weighted SVM 71

5.5 Probabilistic Linear Discriminant Analysis 72

5.5.1 Fundamentals of PLDA 72

5.5.2 Original Formulation 74

5.5.2.1 Parameter Estimation 74

5.5.2.2 Class Label Prediction 76

5.5.3 Scalable Formulation 78

5.5.3.1 Parameter Estimation 78

5.5.3.2 Class Label Prediction 80

5.5.4 Computational Complexity 81

5.5.4.1 Parameter Estimation 81

5.5.4.2 Class Label Prediction 82

5.6 Dictionary-based Representations and Classifiers 83

5.6.1 Sparse Representation 83

5.6.2 Collaborative Representation 84

5.6.3 Discriminative Collaborative Representation 85

5.6.4 Minimum Residual-based Classifier 87

5.7 Comparison of SVM, PLDA and MRC 87

Chapter 6 Classifiers: Experimental Comparison 89

6.1 GMM vs. RBM 89

6.1.1 RBM as A Probabilistic Model 90

6.1.2 Discriminative Model based on RBM 92

6.2 PLDA vs. SVM 93

6.2.1 Scalable PLDA vs. Original PLDA 93

6.2.2 PLDA vs. SVM Using MFCC as the Feature 94

6.2.3 PLDA vs. SVM Using GSV and I-vector as the Feature 96

6.2.4 Linear SVM vs. Kernel SVM Using GSV and I-vector as the Feature 97

6.3 SR vs. DCR 99

6.4 Experimental Comparison of SVM, PLDA and MRC 100

Chapter 7 Feature Projection Techniques: Theoretical Analysis 103

7.1 Fisher Discriminant Analysis 103

7.1.1 Linear Discriminant Analysis 103

7.1.1.1 LDA Version 1 104

7.1.1.2 LDA Version 2 105

7.1.1.3 Brief Comparison between the Two Versions of LDA 105

7.1.2 Kernel Discriminant Analysis 106

7.1.2.1 KDA Version 1 108

7.1.2.2 KDA Version 2 110

7.1.2.3 Brief Comparison between the Two Versions of KDA 112

7.2 Nuisance Attribute Projection 112

7.2.1 Basic Formulation 113

7.2.2 Kernel Extension 115

7.2.3 Relationship with LDA 117

7.3 Brief Comparison between LDA and NAP 120

Chapter 8 Feature Projection Techniques: Experimental Comparison 121

Chapter 9 Conclusion 126

9.1 Major Findings 126

9.2 Future Directions 129

References 131

List of Figures

Figure 1.1 A generic classification framework for acoustic and speech signals.

Figure 3.1 The computation of GSV.

Figure 3.2 The computation of FALV (i-vector).

Figure 3.3 The computation of MFALV.

Figure 3.4 The computation of GSV, i-vector and MFALV.

Figure 3.5 A generic supervector framework.

Figure 3.6 An illustration of the convolution process in a convolutional layer.

Figure 4.1 GSV vs. its extensions for speaker identification employing SVM as the classifier.

Figure 4.2 Speaker identification results on Kingline081 employing PLDA as the classifier.

Figure 4.3 Speaker identification results on Ahumada employing PLDA as the classifier.

Figure 4.4 Time consumption for the construction process of different feature representations.

Figure 4.5 The effectiveness of i-vector and MFALV with different dimensionalities.

Figure 4.6 The efficiency of i-vector and MFALV with different dimensionalities.

Figure 5.1 An illustration of the mixture splitting strategies.

Figure 5.2 A depiction of RBM.

Figure 5.3 A depiction of DBN with $L + 1$ layers.

Figure 5.4 A depiction of the joint RBM.

Figure 5.5 A depiction of SVM with 2-dimensional data.

Figure 6.1 GMM vs. GRBM for acoustic scene classification.

Figure 6.2 DBN-softmax vs. DBN-DNN for acoustic scene classification.

Figure 6.3 Time consumption of original and scalable PLDA.

Figure 6.4 PLDA vs. SVM using MFCC as the feature representation.

Figure 6.5 PLDA vs. SVM using GSV and i-vector as the feature representation.

Figure 6.6 Linear SVM vs. kernel SVM using GSV as the feature representation.

Figure 6.7 Linear SVM vs. kernel SVM using i-vector as the feature representation.

Figure 6.8 SR vs. DCR for acoustic and speech signal classification.

Figure 6.9 Classification accuracy of DCR+MRC, SVM and PLDA on different datasets.

Figure 6.10 Time consumption of DCR+MRC, SVM and PLDA.

Figure 8.1 Linear SVM vs. kernel SVM using GSV as the raw feature representation.

Figure 8.2 NAP vs. kernel NAP using GSV as the raw feature representation.

Figure 8.3 LDA vs. KDA using GSV as the raw feature representation.

Figure 8.4 Linear SVM vs. kernel SVM using i-vector as the raw feature representation.

Figure 8.5 LDA vs. KDA using i-vector as the raw feature representation.

List of Tables

Table 1.1 Acoustic and speech datasets description.

Table 3.1 Characteristics of different vector-based representations.

Table 3.2 Computational complexity of different vector-based representations.

Table 6.1 Time consumption for SR and DCR (in seconds).

List of Abbreviations

Some commonly used abbreviations are given below.

CNN	convolutional neural network
CR	collaborative representation
CRC	CR-based classifier
DBN	deep belief net
DCR	discriminative collaborative representation
DNN	deep neural network
EM	expectation-maximization algorithm
FA	factor analysis
FALV	factor analysis latent vector
FASV	factor analysis supervector
FSV	fully concatenated supervector
GMM	Gaussian mixture model
GRBM	Gaussian restricted Boltzmann machine
GSV	Gaussian supervector
KDA	kernel discriminant analysis
KL	Kullback-Leibler
KNAP	kernel-based nuisance attribute projection
LDA	linear discriminant analysis
MAP	maximum a posteriori

MFA	mixture of factor analyzers
MFALV	MFA latent vector
MFASV	MFA supervector
MFCC	Mel-frequency cepstral coefficient
MLE	maximum likelihood estimation
MRC	minimum residual-based classifier
NAP	nuisance attribute projection
PLDA	probabilistic linear discriminant analysis
RBM	restricted Boltzmann machine
SR	sparse representation
SRC	SR-based classifier
SVM	support vector machine
UBM	universal background model
VSV	variance supervector
WSV	weight supervector

Chapter 1 Introduction

This research focuses on the classification of acoustic and speech signals. For simplicity, we assume that each acoustic or speech sample is assigned with only one class label. This means one sample can only belong to one class.

1.1 Problem Definition

An acoustic or speech sample may convey different kinds of information that meet different classification objectives. For example, a speech sample will naturally carry the information about the speaker, such as the timbre or speaking style, which can be extracted out and used for verifying the identity of the speaker [1]. The speech sample may also carry the information of the recording device, as the recorded speech can be treated as the convolution of the raw speech and the frequency response of the device [2]. The device information can then be used to validate the authentication of a claimed recording device for forensic purposes [3]. If an audio sample is recorded near the power grid, because of electromagnetic induction, it will embed the electric network frequency (ENF) signal [3], which can then be used for timestamp verification [4]. An acoustic or speech sample may also carry the information of the surrounding environment, such as the geometric information of the recording location or the acoustic information of the environmental sounds, which may help describe the geometry of the environment [5] or recognize the acoustic scenes useful for context-aware navigation [6].

In order to recognize the information an acoustic or speech sample carries; for example, which speaker contributes the speech, which device is used to record the speech, or in which environment the acoustic sample is recorded, we may need to do classification that predicts the class the sample belongs to. Usually, the classification process comprises three fundamental steps, which are: 1) feature extraction, 2) feature transformation, and 3) feature classification.

Feature extraction aims at generating feature vectors that can well describe the sample. These feature vectors are supposed to embed enough information for doing some specific tasks, such as the information of the speaker, the device, or the environment. A suitable feature vector has a significant influence on the performance of the designed classifier.

Feature transformation aims at mapping the original feature vector into another feature space, such that the mapped feature vector can better suit some specific purposes. For example, the mapped feature vector may be more discriminative, contain less interference information, has a reduced dimensionality, or has a more distributed characteristic. Feature transformation techniques are expected to improve the quality of the raw feature vector, but its effectiveness may be highly affected by the quality of the original feature vector.

Feature classification aims at constructing a classification model that can predict the feature vectors into their true classes. With different characteristics on the probability distribution or the dimensionality of the feature vectors, there can be different classification models. A mismatch of the characteristics of the feature vectors with the assumptions of the classification models may lead to failure of the classification system.

1.2 A Generic Classification Framework

Figure 1.1 depicts a generic classification framework for acoustic and speech signal classification tasks. There are two modules, one for feature extraction and the other for prediction. Here, the feature transformation steps are included as a part of the feature extraction module, as their outputs are still feature vectors. Nevertheless, some feature transformation techniques can be quite useful and deserve to be comprehensively investigated.

It is common that the length of an acoustic sample is not fixed; for example, one sample may last for 1s while another sample may last for 10s. However, it is usually necessary for the feature vectors to have the same dimensionality before they are used for doing classification. Therefore, we may first divide a variable-length sample into a sequence of equal-length frames and then extract one fixed-length feature vector from each frame, which is called “frame-level feature vector”. Each frame-level feature vector

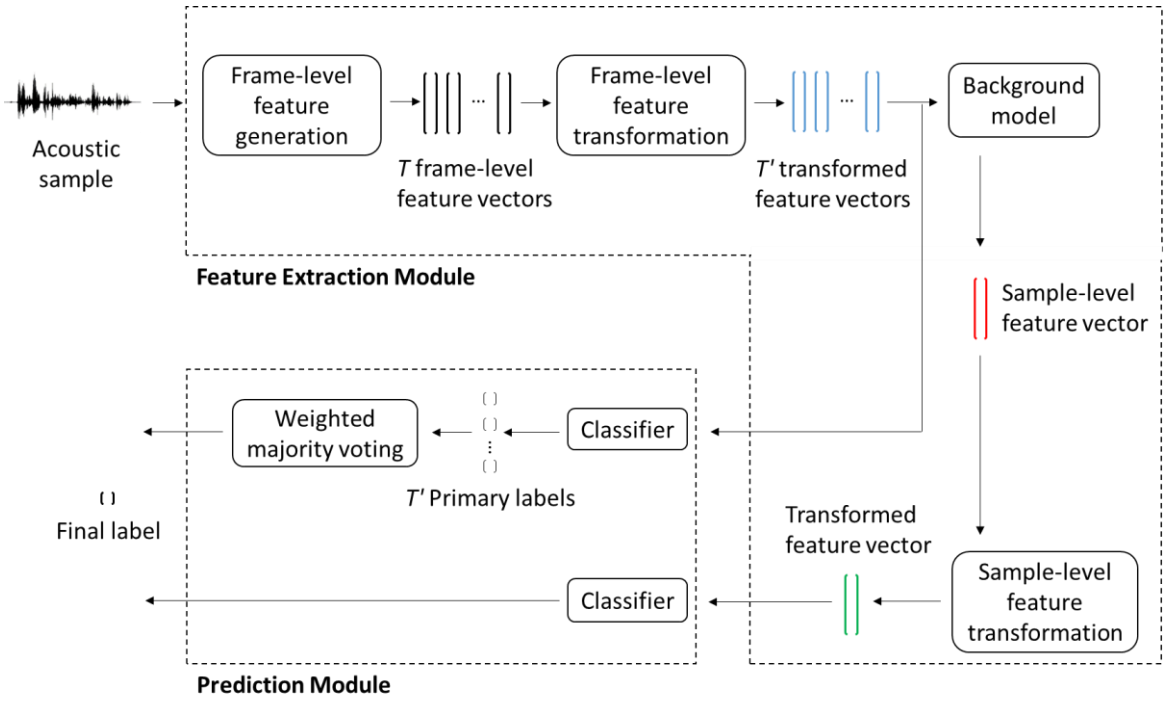


Figure 1.1 A generic classification framework for acoustic and speech signals.

can be transformed into another vector that benefits some specific purposes. For example, several consecutive frame-level feature vectors can be concatenated to form a longer feature vector, which captures some local characteristics.

Having obtained a sequence of frame-level feature vectors or transformed frame-level feature vectors, they can then be used to form a single vector, which is called "sample-level feature vector". This sample-level feature vector usually has a high dimensionality so that it captures enough information about all the frame-level feature vectors. Example sample-level feature vectors include the Gaussian supervector (GSV) [7] and the identity vector (i-vector) [8]. They are obtained based on a universal background model (UBM), whose details will be explained later. The sample-level feature vector may be further processed by some feature transformation techniques to improve its quality for some specific purposes, such as the Fisher linear discriminant analysis (LDA) or the nuisance attribute projection (NAP) [1]. The sample-level feature vector can then be used to train and test a classification model.

Referring to Figure 1.1, the frame-level feature vectors can also be directly used for doing classification. This means that each sample will be represented by multiple feature vectors. In this case, each frame-level feature vector will be treated as an independent feature vector and fed to the classification model

for training and testing, which leads to multiple predicted labels for each sample. Thus, suitable majority voting strategies are needed to combine these labels. Besides, it is crucial to choose a suitable classification model when using the majority voting strategies. It is because in this case, each sample is represented by multiple feature vectors, causing the number of feature vectors to be far more than the case where each sample is represented by only one feature vector. The concept of majority voting also generalizes well in the case where there is only one feature vector for each sample, as the weight of the feature vector will simply be one. In the following, we present four majority voting strategies.

Given a classification task, suppose there are totally K classes, and a classification model has been well trained, whose parameter is denoted by the parameter set θ. For an acoustic or speech sample s, suppose we have already obtained a sequence of frame-level feature vectors or transformed vectors, denoted as $s = \{\boldsymbol{x}_1, \boldsymbol{x}_2 \dots \boldsymbol{x}_T\}$. For each feature vector $\boldsymbol{x}_t$, the classification model may yield a probability $p(\boldsymbol{x}_t|k,\theta)$, which represents the likelihood if $\boldsymbol{x}_t$ belongs to class k. In order to make predictions, the classification model must be able to produce a posterior probability $p(k|\boldsymbol{x}_t,\theta)$, which represents the possibility of predicting $\boldsymbol{x}_t$ into class k among all the K classes. The relationship between the likelihood and the posterior probability is given by (1.1), where $p(k)$ is the prior probability of the k-th class and is assumed to be $1/K$, as all the classes are equally weighted.

$$p(k|\boldsymbol{x}_t,\theta) = \frac{p(\boldsymbol{x}_t|k,\theta)p(k)}{\sum_{j=1}^{K} p(\boldsymbol{x}_t|j,\theta)p(j)} = \frac{\frac{1}{K}p(\boldsymbol{x}_t|k,\theta)}{\frac{1}{K}\sum_{j=1}^{K} p(\boldsymbol{x}_t|j,\theta)} = \frac{p(\boldsymbol{x}_t|k,\theta)}{\sum_{j=1}^{K} p(\boldsymbol{x}_t|j,\theta)} \tag{1.1}$$

Having the posterior probabilities for each class, the predicted class label for sample s can be determined by the cooperation of all the T feature vectors using the following majority voting (MV) strategies as given by (1.2) ~ (1.5), where $Label(s)$ is the predicted label for the sample s, $\delta(.,.)$ is an indicator function given by (1.6), and $\ell(\boldsymbol{x}_t)$ is the predicted label for $\boldsymbol{x}_t$ as given by (1.7). As the majority voting schemes given by (1.3) ~ (1.5) require the existence of the likelihood, only probabilistic models or some specific classification models that can produce a probability can apply them.

$$Label(s) = \underset{k}{\operatorname{argmax}} \sum_{t=1}^{T} \delta\big(k, \ell(\boldsymbol{x}_t)\big) \tag{1.2}$$

$$Label(s) = \underset{k}{\operatorname{argmax}} \sum_{t=1}^{T} p(k \mid \boldsymbol{x}_t, \theta) \tag{1.3}$$

$$Label(s) = \underset{k}{\operatorname{argmax}} \sum_{t=1}^{T} p(k \mid \boldsymbol{x}_t, \theta) \cdot \delta\big(k, \ell(\boldsymbol{x}_t)\big) \tag{1.4}$$

$$Label(s) = \underset{k}{\operatorname{argmax}} \sum_{t=1}^{T} \ln p(\boldsymbol{x}_t|k, \theta) = \underset{k}{\operatorname{argmax}} \sum_{t=1}^{T} \big(\ln p(k|\boldsymbol{x}_t, \theta) + \ln \sum_{j=1}^{K} p(\boldsymbol{x}_t|j, \theta)\big)$$

$$= \underset{k}{\operatorname{argmax}} \sum_{t=1}^{T} \ln p(k|\boldsymbol{x}_t, \theta) \tag{1.5}$$

where

$$\delta(p, q) = \begin{cases} 1, & if\ p = q \\ 0, & if\ p \neq q \end{cases} \tag{1.6}$$

$$\ell(\boldsymbol{x}_t) = \underset{k}{\operatorname{argmax}}\, p(k|\boldsymbol{x}_t, \theta) \tag{1.7}$$

The first MV scheme given by (1.2) is a hard-assignment operation, meaning that each feature vector can only belong to one class. The second MV scheme given by (1.3) is a soft-assignment operation, meaning that each feature vector can belong to all the classes but with different possibilities. The third MV scheme given by (1.4) is a compromise between hard-assignment and soft-assignment, meaning that although each feature vector can only belong to one class, the possibility of belonging to that class is also considered, indicating its relevance. The fourth MV scheme given by (1.5) assumes that the feature vectors $\{\boldsymbol{x}_1, \boldsymbol{x}_2 \ldots \boldsymbol{x}_T\}$ are independently distributed, and thus their joint probability with respect to a class will be the product of their individual likelihoods. This joint probability can then be used for the class label prediction as given by (1.8), which leads to the MV scheme given by (1.5).

$$Label(s) = \underset{k}{\operatorname{argmax}} \prod_{t=1}^{T} p(\boldsymbol{x}_t|k, \theta) = \underset{k}{\operatorname{argmax}} \sum_{t=1}^{T} \ln p(\boldsymbol{x}_t|k, \theta) \tag{1.8}$$

The fourth MV scheme also conforms to the maximum likelihood estimation (MLE) [9] used in the expectation-maximization (EM) algorithm, which assumes that all the training vectors are independently distributed. Therefore, the model parameters are estimated to maximize the product of individual likelihoods, or equivalently, the sum of the log-likelihoods. Besides, this scheme is also in

agreement with the cross-entropy loss [10] used for deep neural networks (DNN), which is also a sum of logarithm values.

1.3 Limitations of Existing Techniques and Proposed Methods

In this thesis, we focus on three aspects of a classification system, i.e. 1) the sample-level feature representation, 2) the classifier, and 3) the feature projection technique. Feature projection techniques are specific feature transformation techniques, where the transformation is defined by a projection matrix.

Two widely used sample-level feature representations are the GSV and the i-vector [1]. As popular feature representations, GSV and i-vector have their own advantages and disadvantages. GSV is simple and fast in computation as it is based on the maximum a posteriori (MAP) adaptation of the UBM. However, its dimensionality depends on the UBM, which is unchangeable. As i-vector is based on a factor analysis (FA) model, there are additional parameters to be estimated besides the UBM. Therefore, the computation of i-vector is inefficient and can be time-consuming. Nonetheless, the dimensionality of i-vector depends on the factor-loading matrix, which is then changeable. To balance the computational efficiency and the flexibility in dimensionality, we propose feature representations based on a mixture of factor analyzers (MFA), such as the MFA latent vector (MFALV). MFALV is comparable to GSV and i-vector in terms of effectiveness. On comparing with i-vector, MFALV preserves similar flexibility in dimensionality while offering a higher computational efficiency.

Furthermore, we propose the generic supervector, which generalizes GSV and MFALV. The i-vector can also be obtained by post-processing the generic supervector. Interestingly, we find that the generic supervector can be used to interpret the feature representation learned by a convolutional neural network (CNN) and help intuitively understand the robustness of the residual network (ResNet) [107]. The formulation of the generic supervector may help design new feature representations.

The Gaussian mixture model (GMM) [9] and the restricted Boltzmann machine (RBM) [48] are popular generative models that can be used for probability estimation and pattern recognition. Both can be applied to acoustic signal classification tasks; however, there lacks a comparison of the characteristics

of GMM and RBM. To improve the understanding of GMM and RBM, we perform theoretical analyses and experimental comparisons on the characteristics and applicable scenarios of GMM and RBM. We also derive the formulation for RBM to estimate the probability of real-valued input feature vectors. In this way, RBM can be used as a probability estimator in the same way as GMM.

The probabilistic linear discriminant analysis (PLDA) model is the state-of-the-art backend for speaker verification [55]. However, its original formulation is not efficient when there are many training data, especially when the training data have a high dimensionality. This prevents PLDA from being a general-purpose classifier like support vector machine (SVM). While some scalable formulations have been proposed to make the model parameter estimation process efficient [78][79], we propose the scalable formulation that makes PLDA able to efficiently predict the class label.

The sparse representation-based classifier (SRC) has demonstrated improved performance over SVM for speaker identification [67]. However, due to the L1-norm constraint included in the formation of the sparse representation (SR), the computation of SRC is inefficient. By replacing the L1-norm constraint by the L2-norm constraint, the collaborative representation (CR) is proposed in [69], which significantly improves the computational efficiency. To further improve the performance of CR, we propose the discriminative CR (DCR), which improves the effectiveness of CR while keeping a similar computational efficiency. In addition, we also propose the minimum residual-based classifier (MRC), which uses SR, CR and DCR to make class label predictions. MRC can be treated as the generalization of SRC and CR-based classifier (CRC).

Two popular feature projection techniques are the nuisance attribute projection (NAP) and the Fisher linear discriminant analysis (LDA) [1]. NAP is usually applied to GSV while LDA is usually applied to i-vector. These two projection techniques are seldom compared in a fair treatment (viz. with the same feature representation and the same backend). In addition, their kernel extensions are not given enough attention. There are many variants of LDA. In this thesis, we focus on two fundamental formulations and make detailed derivations of their kernel-based versions, and then make a comparison between the two formulations. The original concept of NAP does not suit pattern recognition tasks well as its target is to remove the unwanted information from the original feature vector instead of increasing the

discrimination ability of the original feature vector. Nonetheless, by modifying the meaning of the parameters involved in the objective function of NAP, it becomes a general-purpose projection technique like LDA. We then derive its kernel version, which is the generalization of NAP. We also prove that NAP is equivalent to one formulation of LDA under certain conditions. This builds the connection between NAP and LDA.

Our major contributions are summarized as follows:

- The detailed analysis on the formulation and rationale of GSV and i-vector, the proposal of MFA-based feature representations that balance the computational efficiency and dimensional flexibility, and the theoretical and experimental comparisons of different feature representations
- The proposal of the generic supervector, which generalizes GSV and MFALV, and can be used to interpret the feature representation learned by a neural network
- The derivation of the formulation for RBM to estimate probabilities for real-valued input feature vectors, and the theoretical and experimental comparisons between GMM and RBM
- The proposal of the scalable formulation of PLDA that enables it to efficiently make class label predictions
- The proposal of DCR which improves the discriminativeness of CR, and the proposal of MRC which generalizes SRC and CRC
- The detailed analysis on LDA, NAP and their kernel extensions, and the proof of the equivalence of NAP and LDA under certain conditions

1.4 Outline of the Thesis

This thesis is organized into three parts, where each part is relatively independent and self-contained.

Part I includes Chapters 3 and 4, covering the analyses and discussions on several vector-based feature representations, including GSV, i-vector, and the proposed MFA-based feature representations. Specifically, Chapter 3 discusses the formulation and the rationale of GSV and i-vector. Some extensions of GSV are also covered. The MFA-based feature representations are then proposed and compared to GSV and i-vector. Finally, the generic supervector is proposed, which serves as the

generalization of GSV and MFALV. Chapter 4 covers comparative experiments on the effectiveness and efficiency of different feature representations. Some work related to Part I has been included in journal paper [2] and conference papers [1], [4] of Author's Publications.

Part II includes Chapters 5 and 6, covering the analyses and discussions on several classifiers, including GMM, RBM, SVM, PLDA, and MRC. Specifically, Chapter 5 discusses the formulation of GMM and RBM, and presents theoretical comparisons between GMM and RBM. The original formulation and the scalable formulation for PLDA are also analyzed. Binary SVM, multi-class SVM and weighted SVM (WSVM) are briefly introduced. The formulations of SR, CR and DCR are given, followed by the description of MRC. Chapter 6 covers comparative experiments on the effectiveness and efficiency of different classifiers. Some work related to Part II has been included in conference papers [3], [5], [7], [10] of Author's Publications.

Part III includes Chapters 7 and 8, covering the analyses and discussions on several feature projection techniques including LDA, NAP, and their kernel extensions. Specifically, Chapter 7 discusses two formulations of LDA and then derives their kernel-based formulations. NAP is extended to be a general-purpose projection technique, and the corresponding kernel-based formulation is derived. We then prove the equivalence of NAP and LDA under certain conditions. Chapter 8 covers comparative experiments on the performance of LDA, NAP, and their kernel extensions as feature transformation techniques. Some work related to Part III has been included in journal paper [1] and conference papers [2], [6], [8], [9] of Author's Publications.

Chapter 9 concludes the thesis, summarizing our major findings.

1.5 Datasets Description

In this thesis, two acoustic datasets and two speech datasets will be used to evaluate the performance of different feature representations, classifiers, and projection techniques. The acoustic datasets are used to do acoustic scene classification, while the speech datasets are used to do speaker identification. Details about the datasets are given in Table 1.1 and described as follows.

Table 1.1 Acoustic and speech datasets description.

Dataset	Task	Number of classes	Number of samples		Length of each sample	Average length
			Training	Testing		
Kingline081	Speaker identification	20	3997	1998	2s ~ 10s	4s
Ahumada		25	1199	1200	2s ~ 2min	13s
DCASE2013	Acoustic scene classification	10	100	100	30s	30s
TUT2016		15	1121	390	30s	30s

Kingline081 [80] is an American English speech corpus comprising continuous speeches with normal speeds. In our experiments, only a small portion is used, which consists of 20 native speakers' speeches. The speeches are recorded in 3 different sessions, where speeches in the first two sessions are used for training, and those in the third session are used for testing. For each session, there are about 100 samples contributed by each speaker. This leads to a training set of 3997 speech samples and a testing set of 1998 speech samples. The length of the sample varies from 2s to 10s, and the average length is about 4s.

Ahumada [81] is a Spanish speech corpus comprising speeches recorded using different devices with varying speeds. The contents include specific texts as well as spontaneous speeches. In our experiments, only a small portion is used, which comprises 25 speakers' telephone conversational speeches. The speeches are recorded in 4 different sessions, where speeches in two sessions are used for training, and those in the other two sessions are used for testing. For each session, there are about 24 samples contributed by each speaker. This leads to a training set of 1199 speech samples and a testing set of 1200 speech samples. The length of the sample varies from 2s to 2min. Most samples have a length of about 3s, and the average length is about 13s.

DCASE2013 [82] contains data from 10 different acoustic scenes. The public dataset consists of 100 acoustic samples, with each acoustic scene having 10 samples. The private dataset consists of 100 acoustic samples, with each acoustic scene having 10 samples. Each acoustic sample lasts for 30s. The public dataset is used for training, and the private dataset is used for testing.

TUT2016 [83] contains data from 15 different acoustic scenes. The development set consists of 1170 acoustic samples, with each acoustic scene having 78 samples. After deleting erroneous samples, there are 1121 acoustic samples in the development set. The evaluation set consists of 390 acoustic samples, with each acoustic scene having 26 samples. Each acoustic sample lasts for 30s. The development set is used for training and the evaluation set is used for testing.

Chapter 2 Literature Review

2.1 Feature Representations

The most widely used frame-level feature representation is the Mel-frequency cepstral coefficients (MFCC) vector [11]. An MFCC vector is obtained in four steps. First, the Fourier coefficients of a frame are calculated using the discrete Fourier transformation (DFT). Second, the DFT coefficients are filtered using the Mel-scale filterbanks, which simulates the perception of human ear. Third, the Mel coefficients take the logarithm, where convolutive noises are transformed into additive noises, yielding the logmel vector. Finally, the discrete cosine transform (DCT) is applied, which decorrelates the elements in the logmel vector and compresses the energy into a smaller number of coefficients, producing the MFCC vector [12]. The logmel vector itself also works well with DNN [13].

In a speech sample, it is believed that the signal intensity at a time instant can be predicted by the intensities at previous time instants, which leads to the linear predictive coding. Under this assumption and following the similar way of obtaining MFCC, we may have the linear predictive cepstral coefficients (LPCC) [11]. However, this assumption may not generalize well to other types of non-speech acoustic signals. For some specific tasks, MFCC may not be the best choice even if the signal is a speech. For example, in [14], it is shown that the energy band gap feature outperforms MFCC for speech recording device identification. In [15], it is shown that the constant-Q cepstral coefficients (CQCC), which are obtained by replacing DFT with the constant-Q transform (CQT) in the derivation, outperform MFCC for speech replay detection.

It is also viable to apply a DNN to extract frame-level feature representations. For example, in [16], feature representations are extracted by a convolutional deep belief net (CDBN), which is trained in an unsupervised manner. CDBN-based feature representation has been shown to outperform MFCC in speaker recognition when the training data are limited. A sequence of consecutive frame-level feature

representations can be concatenated and then processed by a DNN, yielding the bottleneck feature (BNF) [17][18], which embeds some temporal information existing in the speech signal. However, the temporal characteristics are not evident in the acoustic signals for acoustic scene classification [19]. The frame-level feature representation can also be transformed into a probability vector, whose elements are the posterior probabilities of a Gaussian mixture model (GMM), yielding the posteriorgram [20]. In our experiments, we shall only employ MFCC as the frame-level feature representation because of its simplicity and popularity among a wide range of applications, including speaker recognition [11], acquisition device recognition [2], acoustic environment identification [5], and acoustic scene classification [6].

The frame-level feature representations can be aggregated using majority voting strategies [21]. However, it is more convenient to describe one sample using one feature representation. This sample-level feature representation can be a probabilistic model, such as the GMM, which can be used for speaker identification [22] and verification [23]. To compare the distance or the similarity of two GMMs, we may adopt the Kullback-Leibler (KL) divergence. However, KL divergence is asymmetric, which means that the KL divergence between GMM a and GMM b are different from the KL divergence between GMM b and GMM a. This makes it not really a distance metric. In addition, the calculation of KL divergence is generally difficult, as it requires integrating over the whole feature space. Sometimes, it is even intractable if the probabilistic model is complicated. This difficulty raises the interest in vector-based feature representations [7], which are easier to compare and visualize.

Two widely used vector-based feature representations are GSV and i-vector [1]. GSV is applicable to speaker verification [7] and identification [24], acquisition device recognition [25][26], acoustic scene classification [27], and speech signal clustering [17]. The applications of i-vector include speaker verification [8][18], voice search [28], acoustic scene classification [29], speech replay detection [30], and acoustic signal clustering [31]. GSV is obtained by adapting the parameters of a GMM-based UBM using the maximum a posteriori (MAP) adaptation. The adaptation process is simple and fast, but the dimensionality of GSV depends on the number of mixture components in the UBM, which is unchangeable. I-vector is obtained by constructing a factor analysis (FA) model based on the parameters

of a GMM-based UBM. The computation can be time-consuming because of the additional parameters in the FA model. Nonetheless, the dimensionality of i-vector depends on the size of the factor-loading matrix, which is changeable. There are also studies focusing on reducing the computational burden of i-vector by simplifying its formulation [32][33]. The e-vector [34] is a variant of i-vector, which adopts a different estimation procedure for the factor-loading matrix. Fisher vector (FV), widely used for image classification tasks [35][36], is similar to GSV in formulation and applicable to speaker identification [24]. Different from the MAP adaption adopted by GSV, FV is based on the gradients of a GMM [37].

The UBM can also be a deep belief net (DBN) trained in an unsupervised manner [38], or a DNN used for speech recognition trained in a supervised way [39]. The latter works well while the former does not, which shows the importance of supervised training. A DNN can be applied to the raw frame-level feature representations to produce the DNN-based frame-level feature vectors, which are the activation vectors of a hidden layer. The activation vector can be the output of the intermediate hidden layer [18] or the last hidden layer [40]. These DNN-based frame-level feature vectors can be used to form an i-vector [18] or directly averaged to form a sample-level feature representation called the d-vector [40]. However, d-vector is not comparable to i-vector. By taking account of both the average and the standard deviation, the performance of x-vector is comparable to that of i-vector, or even better after data augmentation [41][42]. The hidden layer of a convolutional neural network (CNN) can also be used to construct sample-level feature representations that work well [43][44]. Still, it requires the input feature vector to have a high dimensionality and the training data to be abundant.

2.2 Classifiers

Most classifiers require constructing a classification model based on a set of training data. The classification model can then be used to predict the class labels of the testing data. In general, classification models can be divided into generative models and discriminative models [11]. Generative models aim at modeling the probability distributions of the training data, and making predictions based on the probabilities with respect to different classes. An example generative model is the GMM. Discriminative models aim at classifying the training data into different classes by finding separating hyperplanes or forming decision functions without modeling the underlying probability distribution.

Example discriminative models include the support vector machine (SVM) and DNN. There also exist model-free classifiers that do not build a model but directly use the training data to make predictions, such as the k nearest neighbor (KNN) classifier and the sparse representation-based classifier (SRC).

Two generative models are widely used in acoustic signal processing. One is GMM, and the other is the restricted Boltzmann machine (RBM). GMM is a famous generative model that covers a wide range of application areas, such as speaker recognition [1], speech acquisition device identification [45], speech replay detection [30], room identification [46], environmental sound classification [47], and acoustic scene classification [6]. It is famous for its usage as the UBM to provide posterior probabilities for the computation of GSV, i-vector, or other feature representations [11][24]. RBM, essentially a two-layer neural network, is also a generative model that can be used for probability estimation [48]. In addition, it can also be used for doing feature transformation [49]. DBN, which is formed by stacking multiple RBMs [50], can be used to initialize a stacked autoencoder (SAE) for dimensionality reduction [51], or used to initialize a DNN for speech recognition [52]. As a brief comparison, GMM is good at handling low-dimensional decorrelated feature vectors, whereas DNN is good at handling high-dimensional correlated feature vectors. The combination of GMM and DNN, such as using DNN to produce the BNF and then using GMM to produce the posterior probabilities [18], may make the best use of their characteristics.

SVM has a wide range of applications, including classification, regression, and novelty detection [53]. It has been widely used for high-dimensional feature classification tasks, such as speaker recognition [11], speech acquisition device identification [25][26], acoustic environment identification [5], acoustic scene classification [29], and environmental sound classification [54]. SVM has been the state-of-the-art backend for speaker verification in the earlier days [8], while currently it is the probabilistic linear discriminant analysis (PLDA) model [55]. Besides speaker verification, PLDA has also been applied to speaker clustering [56] and voice search [28]. CNN is also a good classification model, which can outperform SVM and PLDA for speaker identification and verification [57][58]. However, it requires a large number of training data. Some sequence classification models are also applicable to acoustic signal classification, such as the hidden Markov model (HMM) [54] and the recurrent neural network

(RNN) [19]. However, as the acoustic sample is related to only one class label instead of a sequence of labels, the sequential information may not be critical. In addition, acoustic signals for acoustic scene classification do not possess strong temporal characteristics [19].

The sparse representation (SR), which is essentially a feature transformation technique, has been widely used in different digital signal processing tasks [59], including face recognition [60], handwritten digit recognition [61], music genre classification [62], speech recognition [63], speaker verification [64] and recognition [65], and speech recording device verification [66]. The SRC, which is a model-free classifier and specially designed for classifying SR [60], has demonstrated improved performance in speaker identification over SVM [67] and the cosine distance-based classifier [68]. However, as an L1-norm constraint is included in the objective function of SR, the computation is very slow. In order to improve the computational efficiency, the L1-norm constraint can be replaced by the L2-norm constraint, yielding the collaborative representation (CR) [69]. The CR-based classifier (CRC) gives a similar performance as that of SRC in face recognition but maintains a much lower computational complexity.

2.3 Sample-level Feature Transformation Techniques

Feature transformation aims at mapping a feature representation from its original feature space to another feature space. The mapping can be done, e.g., by a projection matrix that maps the feature representation to a projected space, or a probability function that maps the feature representation to a probabilistic space. Although feature transformations can be applied to frame-level feature representations, a frame-level feature representation does not carry much information and thus is not very exploitable.

Two projection techniques have been popular in speaker recognition studies [1], viz. the nuisance attribute projection (NAP) and the Fisher linear discriminant analysis (LDA). LDA is widely used as a dimensionality reduction technique for different applications such as speaker verification [1] and face recognition [70]. Its kernel extension (KDA) [71][72], which includes an implicit feature mapping before performing projection, can be more powerful [73]. NAP is widely used for removing unwanted

information from the original feature vector, which improves the quality of the feature representation for speaker verification [74] and face recognition [75]. The kernel extension of NAP (KNAP) is also useful [76][77]. There are also some other projection techniques, such as restricted Boltzmann machine (RBM) [49], factor analysis (FA), and probabilistic principal component analysis (PPCA) [33].

Chapter 3 Feature Representations: Theoretical Analysis

3.1 Gaussian Supervector

3.1.1 Basic Formulation

Gaussian supervector (GSV) is obtained by modifying the model parameters of a universal background model (UBM), which is usually a GMM. Suppose a GMM-based UBM has already been constructed, whose parameter set is denoted as $\theta = \{\pi_m, \boldsymbol{\mu}_m, \boldsymbol{\sigma}_m | m = 1,2 \dots M\}$, where M is the total number of Gaussian mixture components in the UBM; π_m, $\boldsymbol{\mu}_m$ and $\boldsymbol{\sigma}_m$ are the corresponding weight, mean vector, and standard deviation vector of the m-th mixture component respectively (assuming the covariance of each Gaussian component is diagonal).

Given an acoustic or speech sample denoted as $s = \{\boldsymbol{x}_1, \boldsymbol{x}_2 \dots \boldsymbol{x}_T\}$, where T is the number of frame-level feature vectors obtained. For each vector $\boldsymbol{x}_t$, a posterior probability $p(m|\boldsymbol{x}_t, \theta)$ for each mixture component as given by (3.1) needs to be calculated, where $p_g(\boldsymbol{x}_t|\boldsymbol{\mu}_m, \boldsymbol{\sigma}_m)$ is the Gaussian probability density function with parameters $\{\boldsymbol{\mu}_m, \boldsymbol{\sigma}_m\}$.

$$p(m|\boldsymbol{x}_t, \theta) = \frac{\pi_m p_g(\boldsymbol{x}_t|\boldsymbol{\mu}_m, \boldsymbol{\sigma}_m)}{\sum_{j=1}^{M} \pi_j p_g(\boldsymbol{x}_t|\boldsymbol{\mu}_j, \boldsymbol{\sigma}_j)} \tag{3.1}$$

Having obtained the posterior probability, the sufficient statistics are needed to be calculated based on the maximum likelihood estimation (MLE) as given by (3.2) ~ (3.4), where $E_m[\boldsymbol{x}^0]$, $E_m[\boldsymbol{x}]$ and $E_m[\boldsymbol{x}^2]$ represent the zero-order, first-order, and second-order statistics for the m-th mixture component respectively. These statistics are posterior expectations. The operator $\circ$ in (3.4) denotes the element-wise multiplication (also known as the Hadamard product).

$$E_m[\boldsymbol{x}^0] = \frac{1}{T}\sum_{t=1}^{T} p(m|\boldsymbol{x}_t, \theta) \tag{3.2}$$

$$E_m[\boldsymbol{x}] = \frac{\sum_{t=1}^{T} p(m|\boldsymbol{x}_t,\theta)\boldsymbol{x}_t}{\sum_{t=1}^{T} p(m|\boldsymbol{x}_t,\theta)} \tag{3.3}$$

$$E_m[\boldsymbol{x}^2] = \frac{\sum_{t=1}^{T} p(m|\boldsymbol{x}_t,\theta)\boldsymbol{x}_t \circ \boldsymbol{x}_t}{\sum_{t=1}^{T} p(m|\boldsymbol{x}_t,\theta)} \tag{3.4}$$

The posterior expectations are then combined with the original model parameters of the UBM to form the adapted parameters as given by (3.5) ~ (3.7), where $\tilde{\pi}_m$, $\tilde{\boldsymbol{\mu}}_m$ and $\tilde{\boldsymbol{\sigma}}_m$ are the adapted weight, adapted mean vector and adapted standard deviation vector respectively. This adaptation process is based on the maximum a posteriori (MAP) adaptation. The operation $(.)^{\circ(.)}$ is the element-wise power operation (also known as the Hadamard power). η is a scale factor automatically determined during the computation to ensure that the adapted weight $\tilde{\pi}_m$ adds up to one. α_m is the adaptation coefficient given by (3.8), where r is the relevance factor [23].

$$\tilde{\pi}_m = \eta(\alpha_m E_m[\boldsymbol{x}^0] + (1-\alpha_m)\pi_m) \tag{3.5}$$

$$\tilde{\boldsymbol{\mu}}_m = \alpha_m E_m[\boldsymbol{x}] + (1-\alpha_m)\boldsymbol{\mu}_m \tag{3.6}$$

$$\tilde{\boldsymbol{\sigma}}_m = (\alpha_m E_m[\boldsymbol{x}^2] + (1-\alpha_m)(\boldsymbol{\sigma}_m^{\circ 2} + \boldsymbol{\mu}_m^{\circ 2}) - \tilde{\boldsymbol{\mu}}_m^{\circ 2})^{\circ \frac{1}{2}} \tag{3.7}$$

where

$$\alpha_m = \frac{E_m[\boldsymbol{x}^0]}{E_m[\boldsymbol{x}^0] + r/T} \tag{3.8}$$

Having the adapted parameters, the GSV is then obtained by column-wisely concatenating the adapted mean vectors $\tilde{\boldsymbol{\mu}}_m$ for $m = 1,2 \ldots M$ as given by (3.9), where the superscript T denotes the transpose operation [7]. If the dimensionality of a frame-level feature vector $\boldsymbol{x}_t$ is $D \times 1$, the dimensionality of GSV is $MD \times 1$. An illustration of the computation of GSV is shown in Figure 3.1.

$$\boldsymbol{X}_{GSV} = [\tilde{\boldsymbol{\mu}}_1^T \quad \tilde{\boldsymbol{\mu}}_2^T \quad \cdots \quad \tilde{\boldsymbol{\mu}}_M^T]^T \tag{3.9}$$

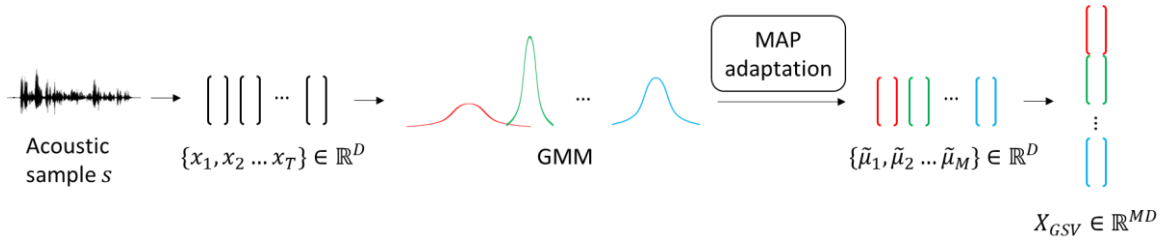


Figure 3.1 The computation of GSV.

3.1.2 Rationale

The idea of GSV arises from an approximation of the KL divergence between two GMM distributions. Before the emergence of vector-based representations, an acoustic or speech sample is represented by a probabilistic model, such as an adapted GMM [23]. Given a sample a and a sample b, suppose their corresponding adapted GMMs are represented by $\tilde{\theta}^{(a)} = \{\tilde{\pi}_m^{(a)}, \tilde{\boldsymbol{\mu}}_m^{(a)}, \tilde{\boldsymbol{\sigma}}_m^{(a)} | m = 1,2 \dots M\}$ and $\tilde{\theta}^{(b)} = \{\tilde{\pi}_m^{(b)}, \tilde{\boldsymbol{\mu}}_m^{(b)}, \tilde{\boldsymbol{\sigma}}_m^{(b)} | m = 1,2 \dots M\}$ respectively. The distance between $\tilde{\theta}^{(a)}$ and $\tilde{\theta}^{(b)}$ can be approximated by the KL divergence given by (3.10), where $p_m^{(a)}(\boldsymbol{x})$ and $p_m^{(b)}(\boldsymbol{x})$ are the Gaussian probability density functions with parameters $\{\tilde{\boldsymbol{\mu}}_m^{(a)}, \tilde{\boldsymbol{\sigma}}_m^{(a)}\}$ and $\{\tilde{\boldsymbol{\mu}}_m^{(b)}, \tilde{\boldsymbol{\sigma}}_m^{(b)}\}$ respectively [7].

$$D_{KL}\left(\tilde{\theta}^{(a)} || \tilde{\theta}^{(b)}\right) = \int_{x=-\infty}^{+\infty} \left(\sum_{m=1}^{M} \tilde{\pi}_m^{(a)} p_m^{(a)}(\boldsymbol{x})\right) \ln\left(\frac{\sum_{m=1}^{M} \tilde{\pi}_m^{(a)} p_m^{(a)}(\boldsymbol{x})}{\sum_{m=1}^{M} \tilde{\pi}_m^{(b)} p_m^{(b)}(\boldsymbol{x})}\right) d\boldsymbol{x} \tag{3.10}$$

It is proved in [84] that the KL divergence between two GMMs is upper bounded by the weighted sum of the KL divergence between their individual Gaussian components as given by (3.11).

$$D_{KL}\left(\tilde{\theta}^{(a)} || \tilde{\theta}^{(b)}\right) \leq D_{KL}\left(\tilde{\pi}_m^{(a)} || \tilde{\pi}_m^{(b)}\right) + \sum_{m=1}^{M} \tilde{\pi}_m^{(a)} D_{KL}\left(p_m^{(a)}(\boldsymbol{x}) \middle\| p_m^{(b)}(\boldsymbol{x})\right) \tag{3.11}$$

If we assume that $\tilde{\pi}_m^{(a)} = \tilde{\pi}_m^{(b)} = \pi_m$ and $\tilde{\boldsymbol{\sigma}}_m^{(a)} = \tilde{\boldsymbol{\sigma}}_m^{(b)} = \boldsymbol{\sigma}_m$, namely only the mean parameter is adapted, then the upper bound of $D_{KL}\left(\tilde{\theta}^{(a)} || \tilde{\theta}^{(b)}\right)$ can be expanded as in (3.12), where $\boldsymbol{\Sigma}_m$ is a diagonal matrix whose elements on the principal diagonal are the elements of $\boldsymbol{\sigma}_m^{\circ 2}$.

$$D_{KL}\left(\tilde{\pi}_m^{(a)} || \tilde{\pi}_m^{(b)}\right) + \sum_{m=1}^{M} \tilde{\pi}_m^{(a)} D_{KL}\left(p_m^{(a)}(\boldsymbol{x}) \middle\| p_m^{(b)}(\boldsymbol{x})\right)$$

$$= D_{KL}(\pi_m || \pi_m) + \sum_{m=1}^{M} \pi_m D_{KL}\left(p_m^{(a)}(\boldsymbol{x}) \middle\| p_m^{(b)}(\boldsymbol{x})\right)$$

$$= 0 + \sum_{m=1}^{M} \pi_m \int_{x=-\infty}^{+\infty} p_m^{(a)}(\boldsymbol{x}) \ln \frac{p_m^{(a)}(\boldsymbol{x})}{p_m^{(b)}(\boldsymbol{x})} d\boldsymbol{x}$$

$$= \sum_{m=1}^{M} \pi_m \int p_m^{(a)}(\boldsymbol{x}) \frac{1}{2} \left(\left(\boldsymbol{x} - \widetilde{\boldsymbol{\mu}}_m^{(b)} \right)^T \boldsymbol{\Sigma}_m^{-1} \left(\boldsymbol{x} - \widetilde{\boldsymbol{\mu}}_m^{(b)} \right) - \left(\boldsymbol{x} - \widetilde{\boldsymbol{\mu}}_m^{(a)} \right)^T \boldsymbol{\Sigma}_m^{-1} \left(\boldsymbol{x} - \widetilde{\boldsymbol{\mu}}_m^{(a)} \right) \right) d\boldsymbol{x}$$

$$= \frac{1}{2} \sum_{m=1}^{M} \pi_m \int p_m^{(a)}(\boldsymbol{x}) \left(\left(\widetilde{\boldsymbol{\mu}}_m^{(b)} \right)^T \boldsymbol{\Sigma}_m^{-1} \widetilde{\boldsymbol{\mu}}_m^{(b)} - \left(\widetilde{\boldsymbol{\mu}}_m^{(a)} \right)^T \boldsymbol{\Sigma}_m^{-1} \widetilde{\boldsymbol{\mu}}_m^{(a)} + 2 \left(\widetilde{\boldsymbol{\mu}}_m^{(a)} - \widetilde{\boldsymbol{\mu}}_m^{(b)} \right)^T \boldsymbol{\Sigma}_m^{-1} x \right) d\boldsymbol{x}$$

$$= \frac{1}{2} \sum_{m=1}^{M} \pi_m \int p_m^{(a)}(\boldsymbol{x}) \left(\left(\widetilde{\boldsymbol{\mu}}_m^{(b)} \right)^T \boldsymbol{\Sigma}_m^{-1} \widetilde{\boldsymbol{\mu}}_m^{(b)} - \left(\widetilde{\boldsymbol{\mu}}_m^{(a)} \right)^T \boldsymbol{\Sigma}_m^{-1} \widetilde{\boldsymbol{\mu}}_m^{(a)} \right) d\boldsymbol{x}$$

$$+ \frac{1}{2} \sum_{m=1}^{M} \pi_m \int p_m^{(a)}(\boldsymbol{x}) \left(2 \left(\widetilde{\boldsymbol{\mu}}_m^{(a)} - \widetilde{\boldsymbol{\mu}}_m^{(b)} \right)^T \boldsymbol{\Sigma}_m^{-1} x \right) d\boldsymbol{x}$$

$$= \frac{1}{2} \sum_{m=1}^{M} \pi_m \left(\left(\widetilde{\boldsymbol{\mu}}_m^{(b)} \right)^T \boldsymbol{\Sigma}_m^{-1} \widetilde{\boldsymbol{\mu}}_m^{(b)} - \left(\widetilde{\boldsymbol{\mu}}_m^{(a)} \right)^T \boldsymbol{\Sigma}_m^{-1} \widetilde{\boldsymbol{\mu}}_m^{(a)} \right) \int p_m^{(a)}(\boldsymbol{x}) d\boldsymbol{x}$$

$$+ \frac{1}{2} \sum_{m=1}^{M} \pi_m \left(2 \left(\widetilde{\boldsymbol{\mu}}_m^{(a)} - \widetilde{\boldsymbol{\mu}}_m^{(b)} \right)^T \boldsymbol{\Sigma}_m^{-1} \right) \int p_m^{(a)}(\boldsymbol{x}) \boldsymbol{x} d\boldsymbol{x}$$

$$= \frac{1}{2} \sum_{m=1}^{M} \pi_m \left(\left(\widetilde{\boldsymbol{\mu}}_m^{(b)} \right)^T \boldsymbol{\Sigma}_m^{-1} \widetilde{\boldsymbol{\mu}}_m^{(b)} - \left(\widetilde{\boldsymbol{\mu}}_m^{(a)} \right)^T \boldsymbol{\Sigma}_m^{-1} \widetilde{\boldsymbol{\mu}}_m^{(a)} \right) + \frac{1}{2} \sum_{m=1}^{M} \pi_m \left(2 \left(\widetilde{\boldsymbol{\mu}}_m^{(a)} - \widetilde{\boldsymbol{\mu}}_m^{(b)} \right)^T \boldsymbol{\Sigma}_m^{-1} \right) \widetilde{\boldsymbol{\mu}}_m^{(a)}$$

$$= \frac{1}{2} \sum_{m=1}^{M} \pi_m \left(\left(\widetilde{\boldsymbol{\mu}}_m^{(b)} \right)^T \boldsymbol{\Sigma}_m^{-1} \widetilde{\boldsymbol{\mu}}_m^{(b)} + \left(\widetilde{\boldsymbol{\mu}}_m^{(a)} \right)^T \boldsymbol{\Sigma}_m^{-1} \widetilde{\boldsymbol{\mu}}_m^{(a)} - 2 \left(\widetilde{\boldsymbol{\mu}}_m^{(b)} \right)^T \boldsymbol{\Sigma}_m^{-1} \widetilde{\boldsymbol{\mu}}_m^{(a)} \right)$$

$$= \frac{1}{2} \sum_{m=1}^{M} \pi_m \left(\widetilde{\boldsymbol{\mu}}_m^{(a)} - \widetilde{\boldsymbol{\mu}}_m^{(b)} \right)^T \boldsymbol{\Sigma}_m^{-1} \left(\widetilde{\boldsymbol{\mu}}_m^{(a)} - \widetilde{\boldsymbol{\mu}}_m^{(b)} \right)$$

$$= \sum_{m=1}^{M} \left(\sqrt{\frac{\pi_m}{2}} \boldsymbol{\Sigma}_m^{-\frac{1}{2}} \widetilde{\boldsymbol{\mu}}_m^{(a)} - \sqrt{\frac{\pi_m}{2}} \boldsymbol{\Sigma}_m^{-\frac{1}{2}} \widetilde{\boldsymbol{\mu}}_m^{(b)} \right)^T \left(\sqrt{\frac{\pi_m}{2}} \boldsymbol{\Sigma}_m^{-\frac{1}{2}} \widetilde{\boldsymbol{\mu}}_m^{(a)} - \sqrt{\frac{\pi_m}{2}} \boldsymbol{\Sigma}_m^{-\frac{1}{2}} \widetilde{\boldsymbol{\mu}}_m^{(b)} \right) \tag{3.12}$$

The result in (3.12) can be seen as the Euclidean distance between two supervectors, where each supervector has the expression as given by (3.13), which is a normalized version of GSV. We name (3.13) as the normalized GSV (nGSV).

$$\boldsymbol{X}_{nGSV} = \left[\left(\sqrt{\frac{\pi_1}{2}} \boldsymbol{\Sigma}_1^{-\frac{1}{2}} \widetilde{\boldsymbol{\mu}}_1 \right)^T \quad \left(\sqrt{\frac{\pi_2}{2}} \boldsymbol{\Sigma}_2^{-\frac{1}{2}} \widetilde{\boldsymbol{\mu}}_2 \right)^T \quad \cdots \quad \left(\sqrt{\frac{\pi_M}{2}} \boldsymbol{\Sigma}_M^{-\frac{1}{2}} \widetilde{\boldsymbol{\mu}}_M \right)^T \right]^T \tag{3.13}$$

Hence, by assuming $\tilde{\pi}_m^{(a)} = \tilde{\pi}_m^{(b)} = \pi_m$ and $\widetilde{\boldsymbol{\sigma}}_m^{(a)} = \widetilde{\boldsymbol{\sigma}}_m^{(b)} = \boldsymbol{\sigma}_m$, the KL divergence between $\tilde{\theta}^{(a)}$ and $\tilde{\theta}^{(b)}$ are upper bounded by the Euclidean distance between $\boldsymbol{X}_{nGSV}^{(a)}$ and $\boldsymbol{X}_{nGSV}^{(b)}$, as given by (3.14). That

is to say, the similarity between two acoustic samples can be approximated by the similarity between their corresponding nGSVs, which lays the foundation for GSV.

$$D_{KL}\left(\tilde{\theta}^{(a)}||\tilde{\theta}^{(b)}\right) \leq \left\|\boldsymbol{X}_{nGSV}^{(a)} - \boldsymbol{X}_{nGSV}^{(b)}\right\|^2 \tag{3.14}$$

3.1.3 Extensions

GSV is obtained by concatenating the adapted mean vectors of all the mixture components. It is then natural to think whether concatenating other adapted parameters may provide some benefits, including the adapted weights and the adapted standard deviation vectors. This leads to the weight supervector (WSV) as given by (3.15), which is the concatenation of the adapted weight parameters; and the variance supervector (VSV) as given by (3.16), which is the concatenation of the adapted standard deviation parameters. By concatenating WSV, GSV, and VSV, a supervector with an even higher dimensionality can be obtained as given by (3.17), which is named as fully concatenated supervector (FSV). The dimensionality of WSV, GSV, VSV, and FSV are $M \times 1$, $MD \times 1$, $MD \times 1$ and $(M + 2MD) \times 1$ respectively.

$$\boldsymbol{X}_{WSV} = [\tilde{\pi}_1 \quad \tilde{\pi}_2 \quad \cdots \quad \tilde{\pi}_M]^T \tag{3.15}$$

$$\boldsymbol{X}_{VSV} = [\tilde{\boldsymbol{\sigma}}_1^T \quad \tilde{\boldsymbol{\sigma}}_2^T \quad \cdots \quad \tilde{\boldsymbol{\sigma}}_M^T]^T \tag{3.16}$$

$$\boldsymbol{X}_{FSV} = [\boldsymbol{X}_{WSV}^T \quad \boldsymbol{X}_{GSV}^T \quad \boldsymbol{X}_{VSV}^T]^T \tag{3.17}$$

3.1.4 Relevance Factor

The relevance factor r involved in the calculation of the supervector is a hyperparameter that needs to be determined. Given a speech sample $s = \{\boldsymbol{x}_1, \boldsymbol{x}_2 \ldots \boldsymbol{x}_T\}$ and a GMM-based UBM with M mixture components, we propose to determine the value of r using (3.18), where β is a scaling factor. According to (3.2) and (3.18), we have $\sum_{m=1}^{M} E_m[\boldsymbol{x}^0] = 1$ and $\sum_{m=1}^{M} r/T = \beta$. So, when $\beta = 1$, we have $\sum_{m=1}^{M} r/T = \sum_{m=1}^{M} E_m[\boldsymbol{x}^0]$. According to (3.5) ~ (3.7), the adapted parameters depend on both the statistics of the sample s and the statistics of the UBM. If the sample s fits well to some mixture component i but fits poorly to some mixture component j, such that $E_i[\boldsymbol{x}^0] > r/T$ and $E_j[\boldsymbol{x}^0] < r/T$,

we have $\alpha_i > 0.5$ and $\alpha_j < 0.5$. This indicates the adapted parameters of the i-th mixture component will depend more on the statistics of the specific sample s instead of the UBM, whereas the adapted parameters of the j-th mixture component will depend more on the statistics of the UBM. When $\beta = 1$, the statistics of the sample s and the statistics of the UBM are equally weighted. The larger the zero-order statistics, the more the adapted parameters will be tuned towards the specific sample s, and vice versa. β controls the general dependence of the adapted parameters on the sample or the UBM. The smaller the β, the more the adapted parameters will depend on the sample; the larger the β, the more the adapted parameters will depend on the UBM.

$$r = \beta \frac{T}{M} \tag{3.18}$$

Determining the value of r using β is more convenient than directly determining the value of r because $\beta = 1$ is the critical point, and the value of β can be chosen using $\beta = 1$ as a reference. In contrast, there is no guidance on choosing the value of r.

3.2 I-vector

3.2.1 Basic Formulation

The identity vector (i-vector) is obtained by estimating a factor analysis (FA) model based on a GMM-based UBM [8]. Given a GMM-based UBM denoted as $\theta = \{\pi_m, \boldsymbol{\mu}_m, \boldsymbol{\sigma}_m | m = 1,2 \dots M\}$, it is assumed that a sample $s = \{\boldsymbol{x}_1, \boldsymbol{x}_2 \dots \boldsymbol{x}_T\}$ is represented by an unobservable supervector $\boldsymbol{X}_s$, which is generated by a latent vector $\boldsymbol{z}_s$ as expressed by (3.19), where $\boldsymbol{\mu}_{UBM}$ is the concatenation of the mean vectors of the UBM as given by (3.20), $\boldsymbol{V}$ is the factor-loading matrix, and $\boldsymbol{\varepsilon}_s$ is a noise vector with zero mean and diagonal covariance $\boldsymbol{\Psi}$. The subscript s means that the supervector and the latent vector depend on a specific sample s.

$$\boldsymbol{X}_s = \boldsymbol{\mu}_{UBM} + \boldsymbol{V}\boldsymbol{z}_s + \boldsymbol{\varepsilon}_s \tag{3.19}$$

where

$$\boldsymbol{\mu}_{UBM} = [\boldsymbol{\mu}_1^T \quad \boldsymbol{\mu}_2^T \quad \cdots \quad \boldsymbol{\mu}_M^T]^T \tag{3.20}$$

If the dimensionality of $\boldsymbol{x}_t$ is $D \times 1$, then the dimensionality of $\boldsymbol{X}_s$ is $MD \times 1$, but the dimensionality of $\boldsymbol{z}_s$ can be smaller than $MD \times 1$, depending on the size of $\boldsymbol{V}$. To estimate the model parameters $\{\boldsymbol{V}, \boldsymbol{\Psi}\}$, we first need to compute the zero-order, first-order and second-order centralized Baum-Welch statistics for each mixture component in the UBM as given by (3.21) ~ (3.23) respectively, where $p(m|\boldsymbol{x}_t, \theta)$ is the GMM posterior probability given by (3.1) [8].

$$\hat{E}_m[\boldsymbol{x}^0] = \sum_{t=1}^{T} p(m|\boldsymbol{x}_t, \theta) \tag{3.21}$$

$$\hat{E}_m[\boldsymbol{x}] = \sum_{t=1}^{T} p(m|\boldsymbol{x}_t, \theta)(\boldsymbol{x}_t - \boldsymbol{\mu}_m) \tag{3.22}$$

$$\hat{E}_m[\boldsymbol{x}\boldsymbol{x}^T] = \sum_{t=1}^{T} p(m|\boldsymbol{x}_t, \theta)(\boldsymbol{x}_t - \boldsymbol{\mu}_m)(\boldsymbol{x}_t - \boldsymbol{\mu}_m)^T \tag{3.23}$$

The Baum-Welch statistics are frame-level statistics, which are then used to compute the sample-level zero-order, first-order, and second-order statistics as given by (3.24) ~ (3.26). The subscript s means that these statistics depend on a specific sample s, and $\boldsymbol{I}$ is the identity matrix whose size is $D \times D$.

$$E_s[\boldsymbol{X}^0] = \begin{bmatrix} \hat{E}_1[\boldsymbol{x}^0]\boldsymbol{I} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \hat{E}_2[\boldsymbol{x}^0]\boldsymbol{I} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \hat{E}_M[\boldsymbol{x}^0]\boldsymbol{I} \end{bmatrix} \tag{3.24}$$

$$E_s[\boldsymbol{X}] = [\hat{E}_1[\boldsymbol{x}]^T \quad \hat{E}_2[\boldsymbol{x}]^T \quad \cdots \quad \hat{E}_M[\boldsymbol{x}]^T]^T \tag{3.25}$$

$$E_s[\boldsymbol{X}\boldsymbol{X}^T] = \begin{bmatrix} \hat{E}_1[\boldsymbol{x}\boldsymbol{x}^T]\boldsymbol{I} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \hat{E}_2[\boldsymbol{x}\boldsymbol{x}^T]\boldsymbol{I} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \hat{E}_M[\boldsymbol{x}\boldsymbol{x}^T]\boldsymbol{I} \end{bmatrix} \tag{3.26}$$

Having the sample-level statistics, $\{\boldsymbol{V}, \boldsymbol{\Psi}\}$ are then estimated using the EM algorithm [86]. In the E-step, the current estimated $\{\boldsymbol{V}, \boldsymbol{\Psi}\}$ are fixed while the posterior mean and the posterior covariance of $\boldsymbol{z}_s$ are computed using (3.27) and (3.28) respectively.

$$E[\boldsymbol{z}_s] = (\boldsymbol{I} + \boldsymbol{V}^T\boldsymbol{\Psi}^{-1}E_s[\boldsymbol{X}^0]\boldsymbol{V})^{-1}\boldsymbol{V}^T\boldsymbol{\Psi}^{-1}E_s[\boldsymbol{X}] \tag{3.27}$$

$$E[\boldsymbol{z}_s\boldsymbol{z}_s^T] = (\boldsymbol{I} + \boldsymbol{V}^T\boldsymbol{\Psi}^{-1}E_s[\boldsymbol{X}^0]\boldsymbol{V})^{-1} + E[\boldsymbol{z}_s]E[\boldsymbol{z}_s]^T \tag{3.28}$$

In the M-step, the posterior mean and the posterior covariance of $\boldsymbol{z}_s$ are fixed while $\{\boldsymbol{V}, \boldsymbol{\Psi}\}$ are re-estimated by solving (3.29) and (3.30). S denotes the number of training samples, and diag{.} is an operation that retains all the elements on the principal diagonal but sets all other elements to zero.

$$\sum_{s=1}^{S} E_s[\boldsymbol{X}^0]\boldsymbol{V}E[\boldsymbol{z}_s\boldsymbol{z}_s^T] = \sum_{s=1}^{S} E_s[\boldsymbol{X}]E[\boldsymbol{z}_s]^T \tag{3.29}$$

$$\boldsymbol{\Psi} = \frac{1}{\sum_{s=1}^{S} E_s[\boldsymbol{X}^0]} \text{diag}\left\{\sum_{s=1}^{S}(E_s[\boldsymbol{X}\boldsymbol{X}^T] - \boldsymbol{V}E[\boldsymbol{z}_s]E_s[\boldsymbol{X}]^T)\right\} \tag{3.30}$$

The EM algorithm for estimating the parameters of i-vector is summarized in Algorithm 3.1, where the superscript i indicates the number of EM iterations having been performed, and P is the total number of EM iterations. The initial values of $\boldsymbol{V}$ and $\boldsymbol{\Psi}$ are matrices with ones on the principal diagonal but zeros elsewhere.

Algorithm 3.1 EM algorithm for i-vector

1:	Initialization:	$\theta^{(0)} = \left\{\boldsymbol{V}^{(0)}, \boldsymbol{\Psi}^{(0)}\right\}$
2:	**For** s = 1 to S	
3:	Compute $E_s[\boldsymbol{X}^0]$, $E_s[\boldsymbol{X}]$ and $E_s[\boldsymbol{X}\boldsymbol{X}^T]$ using (3.24) ~ (3.26)	
4:	**End**	
5:	**For** i = 1 to P	
6:	Set $\{\boldsymbol{V}, \boldsymbol{\Psi}\} = \left\{\boldsymbol{V}^{(i-1)}, \boldsymbol{\Psi}^{(i-1)}\right\}$	
7:	**For** s = 1 to S	
8:	E-step:	$E[\boldsymbol{z}_s] = (\boldsymbol{I} + \boldsymbol{V}^T\boldsymbol{\Psi}^{-1}E_s[\boldsymbol{X}^0]\boldsymbol{V})^{-1}\boldsymbol{V}^T\boldsymbol{\Psi}^{-1}E_s[\boldsymbol{X}]$
9:		$E[\boldsymbol{z}_s\boldsymbol{z}_s^T] = (\boldsymbol{I} + \boldsymbol{V}^T\boldsymbol{\Psi}^{-1}E_s[\boldsymbol{X}^0]\boldsymbol{V})^{-1} + E[\boldsymbol{z}_s]E[\boldsymbol{z}_s]^T$
10:	**End**	
11:	M-step:	Solve $\sum_{s=1}^{S} E_s[\boldsymbol{X}^0]\boldsymbol{V}^{(i)}E[\boldsymbol{z}_s\boldsymbol{z}_s^T] = \sum_{s=1}^{S} E_s[\boldsymbol{X}]E[\boldsymbol{z}_s]^T$ for $\boldsymbol{V}^{(i)}$
12:		$\boldsymbol{\Psi}^{(i)} = \frac{1}{\sum_{s=1}^{S} E_s[\boldsymbol{X}^0]} \text{diag}\left\{\sum_{s=1}^{S}\left(E_s[\boldsymbol{X}\boldsymbol{X}^T] - \boldsymbol{V}^{(i)}E[\boldsymbol{z}_s]E_s[\boldsymbol{X}]^T\right)\right\}$
13:	Update θ:	$\theta^{(i)} = \left\{\boldsymbol{V}^{(i)}, \boldsymbol{\Psi}^{(i)}\right\}$
14:	**End**	

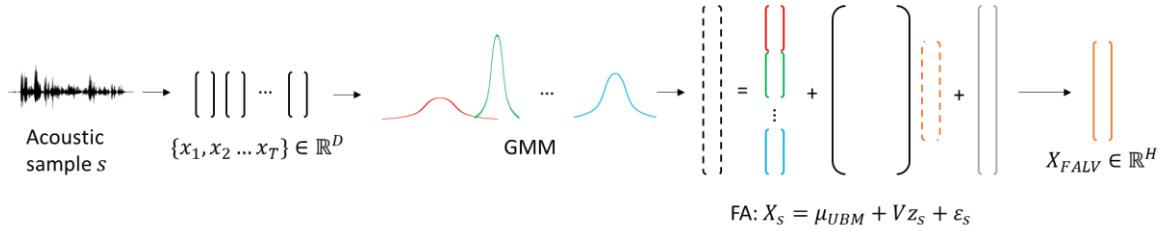


Figure 3.2 The computation of FALV (i-vector).

Having obtained $\{\boldsymbol{V}, \boldsymbol{\Psi}\}$, the i-vector corresponding to the sample s is obtained as the posterior mean of the latent vector $\boldsymbol{z}_s$ as given by (3.31) [8]. We call i-vector the FA latent vector (FALV), because it is based on an FA model. A corresponding supervector is obtained as the posterior mean of $\boldsymbol{X}_s$ as given by (3.32), and we name it as the FA supervector (FASV). An illustration of the computation of FALV (i-vector) is shown in Figure 3.2, where H is the dimensionality of $\boldsymbol{z}_s$.

$$\boldsymbol{X}_{FALV} = E[\boldsymbol{z}_s] \tag{3.31}$$

$$\boldsymbol{X}_{FASV} = E[\boldsymbol{X}_s] = \boldsymbol{\mu}_{UBM} + \boldsymbol{V}E[\boldsymbol{z}_s] = \boldsymbol{\mu}_{UBM} + \boldsymbol{V}\boldsymbol{X}_{FALV} \tag{3.32}$$

3.2.2 Rationale

Consider a set of S training samples denoted as $\{a_1, a_2 \ldots a_S\}$, where the s-th sample is represented by a sequence of T_s frame-level feature vectors, denoted as $a_s = \left\{\boldsymbol{x}_1^{(s)}, \boldsymbol{x}_2^{(s)} \ldots \boldsymbol{x}_{T_s}^{(s)}\right\}$. According to [8] and [86], the log-likelihood of the sample s, denoted as $\mathcal{L}(s)$, should be expressed as

$$\mathcal{L}(s) = \sum_{m=1}^{M}\sum_{t=1}^{T_s} p\left(m\middle|\boldsymbol{x}_t^{(s)}, \theta_{GMM}\right) \ln \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Psi}_m|^{1/2}}$$

$$-\frac{1}{2}\sum_{m=1}^{M}\sum_{t=1}^{T_s} p\left(m\middle|\boldsymbol{x}_t^{(s)}, \theta_{GMM}\right)\left(\boldsymbol{x}_t^{(s)} - (\boldsymbol{\mu}_m + \boldsymbol{V}_m\boldsymbol{z}_s)\right)^T \boldsymbol{\Psi}_m^{-1}\left(\boldsymbol{x}_t^{(s)} - (\boldsymbol{\mu}_m + \boldsymbol{V}_m\boldsymbol{z}_s)\right) \tag{3.33}$$

where $p(m|\boldsymbol{x}_t^{(s)}, \theta_{GMM})$ is the GMM-based posterior probability given by (3.1), $\boldsymbol{\mu}_m$ is the mean of the m-th Gaussian component in the GMM, $\boldsymbol{V}_m$ and $\boldsymbol{\Psi}_m$ are the m-th sub-matrix of $\boldsymbol{V}$ and the m-th sub-matrix of $\boldsymbol{\Psi}$ respectively as given by (3.34).

$$\boldsymbol{V} = \begin{bmatrix} \boldsymbol{V}_1 \\ \boldsymbol{V}_2 \\ \vdots \\ \boldsymbol{V}_M \end{bmatrix}, \ \Psi = \begin{bmatrix} \boldsymbol{\Psi}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Psi}_2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{\Psi}_M \end{bmatrix} \tag{3.34}$$

Equation (3.33) can be reformulated to

$$\mathcal{L}(s) = \sum_{m=1}^{M} \sum_{t=1}^{T_s} p\left(m \middle| \boldsymbol{x}_t^{(s)}, \theta_{GMM}\right) \ln \frac{1}{(2\pi)^{D/2} |\boldsymbol{\Psi}_m|^{1/2}} e^{-\frac{1}{2}\left(\boldsymbol{x}_t^{(s)} - (\boldsymbol{\mu}_m + \boldsymbol{V}_m \boldsymbol{z}_s)\right)^T \boldsymbol{\Psi}_m^{-1} \left(\boldsymbol{x}_t^{(s)} - (\boldsymbol{\mu}_m + \boldsymbol{V}_m \boldsymbol{z}_s)\right)}$$

$$= \sum_{m=1}^{M} \sum_{t=1}^{T_s} \ln \left(\frac{1}{(2\pi)^{D/2} |\boldsymbol{\Psi}_m|^{1/2}} e^{-\frac{1}{2}\left(\boldsymbol{x}_t^{(s)} - (\boldsymbol{\mu}_m + \boldsymbol{V}_m \boldsymbol{z}_s)\right)^T \boldsymbol{\Psi}_m^{-1} \left(\boldsymbol{x}_t^{(s)} - (\boldsymbol{\mu}_m + \boldsymbol{V}_m \boldsymbol{z}_s)\right)} \right)^{p\left(m \middle| \boldsymbol{x}_t^{(s)}, \theta_{GMM}\right)} \tag{3.35}$$

Using (3.35), the model parameters $\{\boldsymbol{V}, \boldsymbol{\Psi}\}$ of i-vector can then be treated as obtained by maximizing the log-likelihood $\mathcal{L}$ given by (3.36), where $p_g(\boldsymbol{x}_t^{(s)} | \boldsymbol{\mu}_m + \boldsymbol{V}_m \boldsymbol{z}_s, \boldsymbol{\Psi}_m)$ is the Gaussian probability density function with mean $(\boldsymbol{\mu}_m + \boldsymbol{V}_m \boldsymbol{z}_s)$ and covariance $\boldsymbol{\Psi}_m$.

$$\mathcal{L} = \sum_{s=1}^{S} \mathcal{L}(s) = \sum_{s=1}^{S} \sum_{m=1}^{M} \sum_{t=1}^{T_s} \ln p_g \left(\boldsymbol{x}_t^{(s)} \middle| \boldsymbol{\mu}_m + \boldsymbol{V}_m \boldsymbol{z}_s, \boldsymbol{\Psi}_m \right)^{p\left(m \middle| \boldsymbol{x}_t^{(s)}, \theta_{GMM}\right)} \tag{3.36}$$

In this way, i-vector can be parameterized by $\theta_{i\text{-}vector} = \{\pi_m, \boldsymbol{\mu}_m, \boldsymbol{\sigma}_m, \boldsymbol{V}_m, \boldsymbol{\Psi}_m | m = 1,2 \dots M\}$, where $\{\pi_m, \boldsymbol{\mu}_m, \boldsymbol{\sigma}_m\}$ are the parameters of the GMM-based UBM, and $\{\boldsymbol{V}_m, \boldsymbol{\Psi}_m\}$ are the parameters of the FA model.

3.2.3 Relationship with GSV

Although (3.19) is an FA model, the parameters cannot be estimated using the standard FA formulation because the supervector $\boldsymbol{X}_s$ is unobservable. In a standard FA model, an observable supervector $\boldsymbol{Y}$ is related to a latent vector $\boldsymbol{y}$ by the expression given by (3.37), where $\boldsymbol{\mu}$ is the mean, $\boldsymbol{W}$ is the factor-loading matrix, and $\boldsymbol{\varepsilon}$ is the noise term following a Gaussian distribution with zero mean and diagonal covariance $\boldsymbol{\Sigma}$. Since the supervector $\boldsymbol{Y}$ is observable, the latent vector $\boldsymbol{y}$ can be estimated from $\boldsymbol{Y}$.

$$\boldsymbol{Y} = \boldsymbol{\mu} + \boldsymbol{W}\boldsymbol{y} + \boldsymbol{\varepsilon} \tag{3.37}$$

The model parameters $\{\boldsymbol{\mu}, \boldsymbol{W}, \boldsymbol{\Sigma}\}$ can be estimated using the EM algorithm [87]. In the E-step, the posterior mean and the posterior covariance are calculated, as given by (3.38) and (3.39) respectively. In the M-step, the model parameters are re-estimated using the posterior expectations in the E-step.

$$E[\boldsymbol{y}] = (\boldsymbol{I} + \boldsymbol{W}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{W})^{-1}\boldsymbol{W}^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{Y} - \boldsymbol{\mu}) \tag{3.38}$$

$$E[\boldsymbol{y}\boldsymbol{y}^T] = (\boldsymbol{I} + \boldsymbol{W}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{W})^{-1} + E[\boldsymbol{y}]E[\boldsymbol{y}]^T \tag{3.39}$$

Comparing (3.38) and (3.39) with (3.27) and (3.28), we may substitute $\boldsymbol{y}$, $\boldsymbol{W}$, $\boldsymbol{\Sigma}$, $\boldsymbol{Y}$ and $\boldsymbol{\mu}$ such that

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{z}_s \\ \boldsymbol{W} &= \boldsymbol{V} \\ \boldsymbol{\Sigma} &= E_s[\boldsymbol{X}^0]^{-1}\boldsymbol{\Psi} \\ \boldsymbol{Y} - \boldsymbol{\mu} &= E_s[\boldsymbol{X}^0]^{-1}E_s[\boldsymbol{X}] \end{aligned} \tag{3.40}$$

The E-step for a standard FA then becomes (3.41) and (3.42), which is the E-step for estimating the parameters for i-vector.

$$E[\boldsymbol{z}_s] = (\boldsymbol{I} + \boldsymbol{V}^T(E_s[\boldsymbol{X}^0]^{-1}\boldsymbol{\Psi})^{-1}\boldsymbol{V})^{-1}\boldsymbol{V}^T(E_s[\boldsymbol{X}^0]^{-1}\boldsymbol{\Psi})^{-1}(E_s[\boldsymbol{X}^0]^{-1}E_s[\boldsymbol{X}]) \tag{3.41}$$

$$E[\boldsymbol{z}_s\boldsymbol{z}_s^T] = (\boldsymbol{I} + \boldsymbol{V}^T(E_s[\boldsymbol{X}^0]^{-1}\boldsymbol{\Psi})^{-1}\boldsymbol{V})^{-1} + E[\boldsymbol{z}_s]E[\boldsymbol{z}_s]^T \tag{3.42}$$

Through the reformulation, the latent vector $\boldsymbol{z}_s$ then corresponds to a supervector $E_s[\boldsymbol{X}^0]^{-1}E_s[\boldsymbol{X}]$. This supervector is actually the mean-shifted GSV with $r = 0$ as given by (3.43), which indicates that i-vector can be treated as an affine transformation of GSV.

$$E_s[\boldsymbol{X}^0]^{-1}E_s[\boldsymbol{X}] = \begin{bmatrix} \hat{E}_1[\boldsymbol{x}^0]^{-1}\hat{E}_1[\boldsymbol{x}] \\ \hat{E}_2[\boldsymbol{x}^0]^{-1}\hat{E}_2[\boldsymbol{x}] \\ \vdots \\ \hat{E}_M[\boldsymbol{x}^0]^{-1}\hat{E}_M[\boldsymbol{x}] \end{bmatrix} = \begin{bmatrix} \frac{\sum_{t=1}^{T} p(1|\boldsymbol{x}_t,\theta)(\boldsymbol{x}_t - \boldsymbol{\mu}_1)}{\sum_{t=1}^{T} p(1|\boldsymbol{x}_t,\theta)} \\ \frac{\sum_{t=1}^{T} p(2|\boldsymbol{x}_t,\theta)(\boldsymbol{x}_t - \boldsymbol{\mu}_2)}{\sum_{t=1}^{T} p(2|\boldsymbol{x}_t,\theta)} \\ \vdots \\ \frac{\sum_{t=1}^{T} p(M|\boldsymbol{x}_t,\theta)(\boldsymbol{x}_t - \boldsymbol{\mu}_M)}{\sum_{t=1}^{T} p(M|\boldsymbol{x}_t,\theta)} \end{bmatrix} = \begin{bmatrix} E_1[\boldsymbol{x}] - \boldsymbol{\mu}_1 \\ E_2[\boldsymbol{x}] - \boldsymbol{\mu}_2 \\ \vdots \\ E_M[\boldsymbol{x}] - \boldsymbol{\mu}_M \end{bmatrix} = \boldsymbol{X}_{GSV} - \boldsymbol{\mu}_{UBM} \tag{3.43}$$

3.3 Feature Representations based on Mixture of Factor Analyzers

3.3.1 Basic Formulation

Albeit i-vector assumes an FA model, the parameter estimation formulation is different from that of a standard FA model. Besides, the model parameters are estimated based on a GMM, making it resemble a mixture of factor analyzers (MFA). Since i-vector is neither the result of the standard FA nor MFA, we consider whether we could form a supervector and a latent vector directly based on an MFA. This yields the MFA supervector (MFASV) and the MFA latent vector (MFALV). In the following, we start by explaining MFA.

An MFA consisting of M factor analyzers can be represented by $\theta_{MFA} = \{\hat{\pi}_m, \hat{\boldsymbol{\mu}}_m, \hat{\boldsymbol{V}}_m, \hat{\boldsymbol{\Psi}} | m = 1,2 \dots M\}$, where $\hat{\pi}_m$, $\hat{\boldsymbol{\mu}}_m$ and $\hat{\boldsymbol{V}}_m$ are the weight, mean, and factor-loading matrix of the m-th factor analyzer respectively. $\hat{\boldsymbol{\Psi}}$ is the covariance parameter of the noise [88]. Different from i-vector which assumes that an unobservable supervector $\boldsymbol{X}_s$ follows an FA model given by (3.19), on using MFA, we assume that each observable frame-level feature vector $\boldsymbol{x}_t$ follows an MFA. With this assumption, $\boldsymbol{x}_t$ can be expressed as (3.44), where $\boldsymbol{z}_t$ is the latent vector following a Gaussian distribution with zero mean and identity covariance, and $\boldsymbol{\varepsilon}_t$ is the noise term following a Gaussian distribution with zero mean and diagonal covariance $\hat{\boldsymbol{\Psi}}$. It should be noted that, for a sample $s = \{\boldsymbol{x}_1, \boldsymbol{x}_2 \dots \boldsymbol{x}_T\}$, in the model assumption of MFA, there will be T latent vectors corresponding to s, denoted as $\{\boldsymbol{z}_1, \boldsymbol{z}_2 \dots \boldsymbol{z}_T\}$. This is different from the model assumption of i-vector, where only one latent vector is corresponding to s, which is $\boldsymbol{z}_s$.

$$\boldsymbol{x}_t = \sum_{m=1}^{M} \hat{\pi}_m \left(\hat{\boldsymbol{\mu}}_m + \hat{\boldsymbol{V}}_m \boldsymbol{z}_t\right) + \boldsymbol{\varepsilon}_t \tag{3.44}$$

The probability produced by the m-th factor analyzer and that produced by the MFA are then the Gaussian probabilities as given by (3.45) and (3.46) respectively, where $p_g\left(\boldsymbol{x}_t | \hat{\boldsymbol{\mu}}_m, \hat{\boldsymbol{V}}_m \hat{\boldsymbol{V}}_m^T + \hat{\boldsymbol{\Psi}}\right)$ is the Gaussian probability density function with mean $\hat{\boldsymbol{\mu}}_m$ and covariance $\left(\hat{\boldsymbol{V}}_m \hat{\boldsymbol{V}}_m^T + \hat{\boldsymbol{\Psi}}\right)$.

$$p(\boldsymbol{x}_t|m,\theta_{MFA}) = p_g\left(\boldsymbol{x}_t\middle|\widehat{\boldsymbol{\mu}}_m, \widehat{\boldsymbol{V}}_m\widehat{\boldsymbol{V}}_m^T + \widehat{\boldsymbol{\Psi}}\right) \tag{3.45}$$

$$p(\boldsymbol{x}_t|\theta_{MFA}) = \sum_{m=1}^{M} \hat{\pi}_m p_g\left(\boldsymbol{x}_t\middle|\widehat{\boldsymbol{\mu}}_m, \widehat{\boldsymbol{V}}_m\widehat{\boldsymbol{V}}_m^T + \widehat{\boldsymbol{\Psi}}\right) \tag{3.46}$$

The posterior probability with respect to the m-th factor analyzer is then given by (3.47).

$$p(m|\boldsymbol{x}_t,\theta_{MFA}) = \frac{\hat{\pi}_m p_g\left(\boldsymbol{x}_t\middle|\widehat{\boldsymbol{\mu}}_m, \widehat{\boldsymbol{V}}_m\widehat{\boldsymbol{V}}_m^T + \widehat{\boldsymbol{\Psi}}\right)}{\sum_{j=1}^{M} \hat{\pi}_j p_g\left(\boldsymbol{x}_t\middle|\widehat{\boldsymbol{\mu}}_j, \widehat{\boldsymbol{V}}_j\widehat{\boldsymbol{V}}_j^T + \widehat{\boldsymbol{\Psi}}\right)} \tag{3.47}$$

Given a set of S training acoustic samples $\{a_1, a_2 \ldots a_S\}$ where the s-th sample consists of T_s frame-level feature vectors. Suppose there are totally N frame-level feature vectors obtained from these training samples, denoted as $\{\boldsymbol{x}_1, \boldsymbol{x}_2 \ldots \boldsymbol{x}_N\}$, where $N = \sum_{s=1}^{S} T_s$. The corresponding latent vectors are denoted as $\{\boldsymbol{z}_1, \boldsymbol{z}_2 \ldots \boldsymbol{z}_N\}$. The model parameters of an MFA can be estimated using the EM algorithm [88]. In the E-step, the posterior mean and the posterior covariance of the latent vector are computed using (3.48) and (3.49) respectively.

$$E_m[\boldsymbol{z}_n] = \left(\boldsymbol{I} + \widehat{\boldsymbol{V}}_m^T\widehat{\boldsymbol{\Psi}}^{-1}\widehat{\boldsymbol{V}}_m\right)^{-1}\widehat{\boldsymbol{V}}_m^T\widehat{\boldsymbol{\Psi}}^{-1}(\boldsymbol{x}_n - \widehat{\boldsymbol{\mu}}_m) \tag{3.48}$$

$$E_m[\boldsymbol{z}_n\boldsymbol{z}_n^T] = \left(\boldsymbol{I} + \widehat{\boldsymbol{V}}_m^T\widehat{\boldsymbol{\Psi}}^{-1}\widehat{\boldsymbol{V}}_m\right)^{-1} + E_m[\boldsymbol{z}_n]E_m[\boldsymbol{z}_n]^T \tag{3.49}$$

In the M-step, the model parameters $\{\hat{\pi}_m, \widehat{\boldsymbol{\mu}}_m, \widehat{\boldsymbol{V}}_m, \widehat{\boldsymbol{\Psi}}\}$ are re-estimated using the updated posterior mean and the updated posterior covariance. In order to re-estimate $\widehat{\boldsymbol{V}}_m$ and $\widehat{\boldsymbol{\Psi}}$ together, we may form an augmented factor-loading matrix $\widetilde{\boldsymbol{V}}_m$ as given by (3.50). The posterior mean $E_m[\boldsymbol{z}_n]$ and the posterior covariance $E_m[\boldsymbol{z}_n\boldsymbol{z}_n^T]$ also need to be augmented accordingly [88].

$$\widetilde{\boldsymbol{V}}_m = [\widehat{\boldsymbol{V}}_m \quad \widehat{\boldsymbol{\mu}}_m],\ E_m[\tilde{\boldsymbol{z}}_n] = \begin{bmatrix} E_m[\boldsymbol{z}_n] \\ 1 \end{bmatrix},\ E_m[\tilde{\boldsymbol{z}}_n\tilde{\boldsymbol{z}}_n^T] = \begin{bmatrix} E_m[\boldsymbol{z}_n\boldsymbol{z}_n^T] & E_m[\boldsymbol{z}_n] \\ E_m[\boldsymbol{z}_n]^T & 1 \end{bmatrix} \tag{3.50}$$

Using the augmented notations, the M-step is then given by (3.51) ~ (3.53).

$$\hat{\pi}_m = \frac{1}{N}\sum_{n=1}^{N} p(m|\boldsymbol{x}_n,\theta_{MFA}) \tag{3.51}$$

$$\widetilde{\boldsymbol{V}}_m = \left(\sum_{n=1}^{N} p(m|\boldsymbol{x}_n,\theta_{MFA})\boldsymbol{x}_n E_m[\tilde{\boldsymbol{z}}_n]^T\right)\left(\sum_{n=1}^{N} p(m|\boldsymbol{x}_n,\theta_{MFA})E_m[\tilde{\boldsymbol{z}}_n\tilde{\boldsymbol{z}}_n^T]\right)^{-1} \tag{3.52}$$

$$\widehat{\boldsymbol{\Psi}} = \frac{1}{N}\mathrm{diag}\left\{\sum_{n=1}^{N}\sum_{m=1}^{M} p(m|\boldsymbol{x}_n,\theta_{MFA})\left(\boldsymbol{x}_n - \widetilde{\boldsymbol{V}}_m E_m[\tilde{\boldsymbol{z}}_n]\right)\boldsymbol{x}_n^T\right\} \tag{3.53}$$

The EM algorithm for MFA is summarized in Algorithm 3.2, where the superscript i indicates the number of EM iterations having been performed, and P is the total number of EM iterations. The initial value of $\hat{\boldsymbol{\mu}}_m$ is the value of $\boldsymbol{\mu}_m$ of the GMM-based UBM; the initial value of $\hat{\pi}_m$ is $1/M$, and the initial values of $\hat{\boldsymbol{V}}_m$ and $\hat{\boldsymbol{\Psi}}$ are matrices with ones on the principal diagonal but zeros elsewhere.

Having $\{\hat{\pi}_m, \hat{\boldsymbol{\mu}}_m, \hat{\boldsymbol{V}}_m, \hat{\boldsymbol{\Psi}}\}$, for a sample $s = \{\boldsymbol{x}_1, \boldsymbol{x}_2 \dots \boldsymbol{x}_T\}$, the m-th sub-vector of its corresponding MFA latent vector (MFALV) is obtained as the weighted average (i.e., $\bar{E}_m[\boldsymbol{z}]$) of the posterior means of the latent vectors as given by (3.54), where $E_m[\boldsymbol{z}_t]$ is given by (3.48), namely $E_m[\boldsymbol{z}_t] = \left(\boldsymbol{I} + \hat{\boldsymbol{V}}_m^T \hat{\boldsymbol{\Psi}}^{-1} \hat{\boldsymbol{V}}_m\right)^{-1} \hat{\boldsymbol{V}}_m^T \hat{\boldsymbol{\Psi}}^{-1} (\boldsymbol{x}_t - \hat{\boldsymbol{\mu}}_m)$. The MFALV is then the concatenation of its sub-vectors as given by (3.55). An illustration of the computation of MFALV is shown in Figure 3.3, where the dimensionality of $\boldsymbol{z}_t$ is $\frac{H}{M} \times 1$ (assuming H is an integer multiple of M) and thus the dimensionality of MFALV is $H \times 1$.

$$\boldsymbol{X}_{MFALV,m} = \bar{E}_m[\boldsymbol{z}] = \frac{\sum_{t=1}^{T} p(m|\boldsymbol{x}_t, \theta_{MFA}) E_m[\boldsymbol{z}_t]}{\sum_{t=1}^{T} p(m|\boldsymbol{x}_t, \theta_{MFA})} \tag{3.54}$$

$$\boldsymbol{X}_{MFALV} = [\boldsymbol{X}_{MFALV,1}^T \quad \boldsymbol{X}_{MFALV,2}^T \quad \cdots \quad \boldsymbol{X}_{MFALV,M}^T]^T \tag{3.55}$$

The m-th sub-vector of the MFA supervector (MFASV) is obtained in a similar way to FASV as given by (3.56). The MFASV is then the concatenation of its sub-vectors as given by (3.57).

$$\boldsymbol{X}_{MFASV,m} = \hat{\boldsymbol{\mu}}_m + \hat{\boldsymbol{V}}_m \boldsymbol{X}_{MFALV,m} \tag{3.56}$$

$$\boldsymbol{X}_{MFASV} = [\boldsymbol{X}_{MFASV,1}^T \quad \boldsymbol{X}_{MFASV,2}^T \quad \cdots \quad \boldsymbol{X}_{MFASV,M}^T]^T \tag{3.57}$$

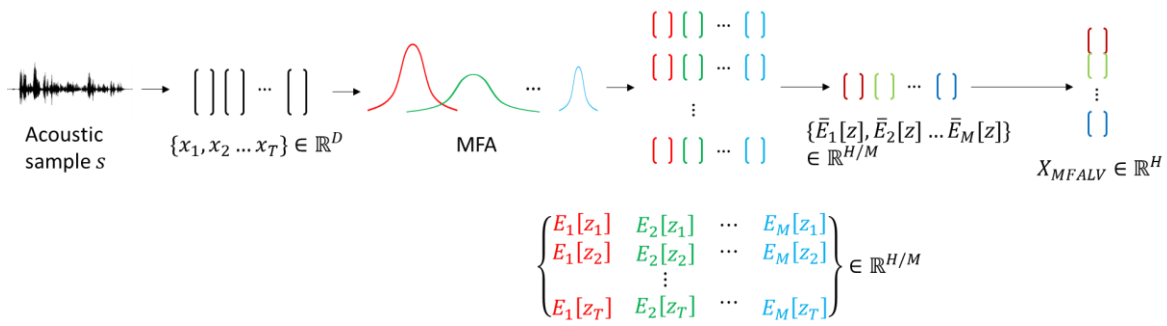


Figure 3.3 The computation of MFALV.

Algorithm 3.2 EM algorithm for MFA

1: Initialization: $\theta^{(0)} = \left\{\hat{\pi}_m^{(0)}, \hat{\boldsymbol{\mu}}_m^{(0)}, \hat{\boldsymbol{V}}_m^{(0)}, \hat{\boldsymbol{\Psi}}^{(0)} \middle| m = 1,2 \dots M\right\}$

2: **For** i = 1 to P

3: Set $\{\pi_m, \boldsymbol{\mu}_m, \boldsymbol{V}_m, \boldsymbol{\Psi} | m = 1,2 \dots M\} = \left\{\hat{\pi}_m^{(i-1)}, \hat{\boldsymbol{\mu}}_m^{(i-1)}, \hat{\boldsymbol{V}}_m^{(i-1)}, \hat{\boldsymbol{\Psi}}^{(i-1)} | m = 1,2 \dots M\right\}$

4: **For** m = 1 to M

5: **For** n = 1 to N

6: $p_m(\boldsymbol{x}_n) = \dfrac{\pi_m p_g(\boldsymbol{x}_n | \boldsymbol{\mu}_m, \boldsymbol{V}_m \boldsymbol{V}_m^T + \boldsymbol{\Psi})}{\sum_{j=1}^{M} \pi_j p_g\left(\boldsymbol{x}_n \middle| \boldsymbol{\mu}_j, \boldsymbol{V}_j \boldsymbol{V}_j^T + \boldsymbol{\Psi}\right)}$

7: E-step: $E_m^{(i)}[\boldsymbol{z}_n] = (\boldsymbol{I} + \boldsymbol{V}_m^T \boldsymbol{\Psi}^{-1} \boldsymbol{V}_m)^{-1} \boldsymbol{V}_m^T \boldsymbol{\Psi}^{-1} (\boldsymbol{x}_n - \boldsymbol{\mu}_m)$

8: $E_m^{(i)}[\boldsymbol{z}_n \boldsymbol{z}_n^T] = (\boldsymbol{I} + \boldsymbol{V}_m^T \boldsymbol{\Psi}^{-1} \boldsymbol{V}_m)^{-1} + E_m^{(i)}[\boldsymbol{z}_n] E_m^{(i)}[\boldsymbol{z}_n]^T$

9: Form $E_m^{(i)}[\tilde{\boldsymbol{z}}_n]$ and $E_m^{(i)}[\tilde{\boldsymbol{z}}_n \tilde{\boldsymbol{z}}_n^T]$ from $E_m^{(i)}[\boldsymbol{z}_n]$ and $E_m^{(i)}[\boldsymbol{z}_n \boldsymbol{z}_n^T]$ using (3.50)

10: **End**

11: $\hat{\pi}_m^{(i)} = \dfrac{1}{N} \sum_{n=1}^{N} p_m(\boldsymbol{x}_n)$

12: M-step: $\tilde{\boldsymbol{V}}_m^{(i)} = \left(\sum_{n=1}^{N} p_m(\boldsymbol{x}_n) \boldsymbol{x}_n E_m^{(i)}[\tilde{\boldsymbol{z}}_n]^T\right) \left(\sum_{n=1}^{N} p_m(\boldsymbol{x}_n) E_m^{(i)}[\tilde{\boldsymbol{z}}_n \tilde{\boldsymbol{z}}_n^T]\right)^{-1}$

13: $\begin{bmatrix} \hat{\boldsymbol{V}}_m^{(i)} & \hat{\boldsymbol{\mu}}_m^{(i)} \end{bmatrix} = \tilde{\boldsymbol{V}}_m^{(i)}$

14: **End**

15: Estimate noise covariance: $\hat{\boldsymbol{\Psi}}^{(i)} = \dfrac{1}{N} \operatorname{diag}\left\{\sum_{n=1}^{N} \sum_{m=1}^{M} p_m(\boldsymbol{x}_n) \left(\boldsymbol{x}_n - \tilde{\boldsymbol{V}}_m^{(i)} E_m^{(i)}[\tilde{\boldsymbol{z}}_n]\right) \boldsymbol{x}_n^T\right\}$

16: Update θ: $\theta^{(i)} = \left\{\hat{\pi}_m^{(i)}, \hat{\boldsymbol{\mu}}_m^{(i)}, \hat{\boldsymbol{V}}_m^{(i)}, \hat{\boldsymbol{\Psi}}^{(i)} \middle| m = 1,2 \dots M\right\}$

17: **End**

3.3.2 Comparison with I-vector

Suppose we have S training samples denoted as $\{a_1, a_2 \dots a_S\}$, and the s-th sample is represented as a sequence of T_s frame-level feature vectors, i.e., $a_s = \left\{\boldsymbol{x}_1^{(s)}, \boldsymbol{x}_2^{(s)} \dots \boldsymbol{x}_{T_s}^{(s)}\right\}$. Suppose a GMM-based UBM has been constructed, which is parameterized by $\theta_{GMM} = \{\pi_m, \boldsymbol{\mu}_m, \boldsymbol{\sigma}_m | m = 1,2 \dots M\}$. There are several differences between i-vector and MFALV.

First, the model parameters are different. The model parameters used to compute an i-vector are $\theta_{i\text{-}vector} = \{\pi_m, \boldsymbol{\mu}_m, \boldsymbol{\sigma}_m, \boldsymbol{V}_m, \boldsymbol{\Psi}_m | m = 1,2 \dots M\}$, which is a fusion of the parameters of the GMM-based UBM and the parameters of an FA model. On the other hand, the model parameters used to compute an MFALV are $\theta_{MFA} = \{\hat{\pi}_m, \hat{\boldsymbol{\mu}}_m, \hat{\boldsymbol{V}}_m, \hat{\boldsymbol{\Psi}} | m = 1,2 \dots M\}$, which only depend on the parameters of the MFA instead of the GMM, even though these parameters are initialized to some parameters of the GMM. From this perspective, i-vector is based on a GMM-based UBM, whereas MFALV is based on an MFA-based UBM.

Second, the objective function is different. Finding the parameters $\{\boldsymbol{V}_m, \boldsymbol{\Psi}_m\}$ for i-vector can be regarded as the optimization problem given by (3.58), where $\boldsymbol{z}_s$ is the latent vector corresponding to $\left\{\boldsymbol{x}_1^{(s)}, \boldsymbol{x}_2^{(s)} \dots \boldsymbol{x}_{T_s}^{(s)}\right\}$. In contrast, finding the parameters $\{\hat{\boldsymbol{V}}_m, \hat{\boldsymbol{\Psi}}\}$ for MFALV can be regarded as the optimization problem given by (3.59), where $\left\{\boldsymbol{z}_1^{(s)}, \boldsymbol{z}_2^{(s)} \dots \boldsymbol{z}_{T_s}^{(s)}\right\}$ are the latent vectors corresponding to $\left\{\boldsymbol{x}_1^{(s)}, \boldsymbol{x}_2^{(s)} \dots \boldsymbol{x}_{T_s}^{(s)}\right\}$. For i-vector, the mean and the exponent involved in the Gaussian probability density function are dependent on the GMM-based UBM. For MFALV, the mean and the exponent involved in the Gaussian probability density function are dependent on the MFA-based UBM. This implies that MFALV needs to estimate more parameters than i-vector, which renders MFALV more flexible. However, the increased parameter space may cause instability issues.

$$\max_{\boldsymbol{V}_m, \boldsymbol{\Psi}_m} \mathcal{L} = \sum_{s=1}^{S} \sum_{m=1}^{M} \sum_{t=1}^{T_s} \ln p_g \left(\boldsymbol{x}_t^{(s)} \middle| \boldsymbol{\mu}_m + \boldsymbol{V}_m \boldsymbol{z}_s, \boldsymbol{\Psi}_m\right)^{p\left(m \middle| \boldsymbol{x}_t^{(s)}, \theta_{GMM}\right)} \tag{3.58}$$

$$\max_{\hat{\boldsymbol{V}}_m, \hat{\boldsymbol{\Psi}}} \mathcal{L} = \sum_{s=1}^{S} \sum_{m=1}^{M} \sum_{t=1}^{T_s} \ln p_g \left(\boldsymbol{x}_t^{(s)} \middle| \hat{\boldsymbol{\mu}}_m + \hat{\boldsymbol{V}}_m \boldsymbol{z}_t^{(s)}, \hat{\boldsymbol{\Psi}}\right)^{\hat{\pi}_m} \tag{3.59}$$

Third, the model assumption is different. According to the model assumption of i-vector, all the frame-level feature vectors in a sample share the same latent vector. On the other hand, according to the model assumption of MFALV, each frame-level feature vector will have its own latent vector. This latent vector sharing mechanism may render i-vector more robust, as it captures the common characteristics across different frame-level feature vectors. Nevertheless, MFALV treats each frame-level feature vector as an independent individual, which may provide a better depiction if there are large variations across different frame-level feature vectors.

Fourth, the time complexity of the EM algorithm used for model parameter estimation is different. Suppose the dimensionality of i-vector and MFALV is $H \times 1$, then the E-step for i-vector requires computing the inverse of $(\boldsymbol{I} + \boldsymbol{V}^T \boldsymbol{\Psi}^{-1} E_S[\boldsymbol{X}^0] \boldsymbol{V})$, whose size is $H \times H$. For MFALV, each sub-vector will have a dimensionality of $\frac{H}{M} \times 1$ (assuming H is an integer multiple of M), and thus the size of $\left(\boldsymbol{I} + \widehat{\boldsymbol{V}}_m^T \widehat{\boldsymbol{\Psi}}^{-1} \widehat{\boldsymbol{V}}_m\right)$ involved in the E-step will be $\frac{H}{M} \times \frac{H}{M}$. The E-step of i-vector is operated on a sample, whereas the E-step of MFALV is operated on all the frame-level feature vectors. Suppose the time complexity of inverting an $H \times H$ matrix is $O(H^3)$ ($O(.)$ is the big-O notation upper bounding the run time of an algorithm), according to Algorithm 3.1 and Algorithm 3.2, the time complexity of the E-step for i-vector is approximately $O(PSH^3)$, and that for MFALV is approximately $O\left(PMN\left(\frac{H}{M}\right)^3\right)$. If each sample consists of T frame-level feature vectors, meaning that $N = ST$, then the time complexity of the E-step for MFALV becomes $O\left(PMST\left(\frac{H}{M}\right)^3\right) = \frac{T}{M^2} \cdot O(PSH^3)$. This implies that the model parameter estimation process of MFALV can be more efficient than that of i-vector, especially when T is small or M is large. A small value of T indicates that the acoustic sample is short. Nevertheless, as the dimensionality of i-vector can be any integer value, but the dimensionality of MFALV can only be an integer multiple of M, i-vector can be more flexible in terms of dimensionality.

3.4 Comparison of Different Vector-based Representations

3.4.1 Characteristics

GSV is obtained by the MAP adaptation of a GMM-based UBM. The necessary parameters used to construct a GSV are simply those of the GMM, denoted as $\theta_{GSV} = \{\pi_m, \boldsymbol{\mu}_m, \boldsymbol{\sigma}_m | m = 1,2 \dots M\}$.

I-vector is obtained by estimating an FA model centered on a GMM-based UBM. The necessary parameters used to construct an i-vector are a fusion of the parameters of the GMM and the parameters of an FA model, denoted as $\theta_{i\text{-}vector} = \{\pi_m, \boldsymbol{\mu}_m, \boldsymbol{\sigma}_m, \boldsymbol{V}_m, \boldsymbol{\Psi}_m | m = 1,2 \dots M\}$. The parameters of i-vector can also be treated as the parameters of an MFA-based UBM, with the restrictions that, 1) given a feature vector $\boldsymbol{x}_t$, the posterior probability of the m-th factor analyzer in the MFA equals the posterior probability of the m-th Gaussian component in the GMM, namely $p(m|\boldsymbol{x}_t, \theta_{GMM}) = p(m|\boldsymbol{x}_t, \theta_{MFA})$, 2) the mean parameter of the m-th factor analyzer equals the mean parameter of the m-th Gaussian component, and 3) in the MFA, the noise covariance is different for different factor analyzers.

MFALV is obtained based on an MFA-based UBM. The necessary parameters used to construct an MFALV are those of the MFA, denoted as $\theta_{MFALV} = \{\hat{\pi}_m, \hat{\boldsymbol{\mu}}_m, \hat{\boldsymbol{V}}_m, \hat{\boldsymbol{\Psi}} | m = 1,2 \dots M\}$, where the mean parameters are initialized from the GMM-based UBM. From this point of view, MFALV is obtained by estimating an MFA model initially centered on a GMM-based UBM.

As GSV only requires the parameters of a GMM, its computation is simple and fast. As i-vector and MFALV require further estimating an FA or MFA from a GMM, their computation can be time-consuming. Nonetheless, the dimensionality of GSV is fixed to be $MD \times 1$, where D is the dimensionality of a frame-level feature vector, and M is the number of mixture components in the UBM. In contrast, the dimensionality of i-vector and MFALV is changeable, which depends on the size of the factor-loading matrix in the FA and MFA respectively. In general, different feature representations have their own advantages and disadvantages.

Although they are different, GSV, i-vector, and MFALV share a common characteristic, which is the adoption of a UBM. The UBM serves two primary purposes. First, it provides some prior information

as it is usually trained using a large number of unlabeled data. This prior information includes the information from different samples. Second, it provides feature alignment. For example, suppose an acoustic sample is a sequence of different events, but where the events occur is unknown. The mixture components in the UBM can then be regarded as representing the statistics of different events. For example, suppose the 1st mixture component represents event A, the 2nd mixture component represents event B, and so forth. In a GSV, the elements $1{\sim}D$ may then represent the possibility that event A occurs in this sample, the elements $(D+1){\sim}2D$ may then represent the possibility that event B occurs in this sample, and so forth. Therefore, the same element in different GSVs is representing the same attribute. A similar analysis also applies to i-vector and MFALV.

A summary of the characteristics of different vector-based representations is given in Table 3.1. As the model parameters for i-vector can be treated as a special MFA with some restrictions, we categorize i-vector to be based on an MFA-based UBM.

Table 3.1 Characteristics of different vector-based representations.

UBM	Name	Description	Dimensionality
GMM	WSV	Concatenation of adapted weights	M
	GSV	Concatenation of adapted means	MD
	VSV	Concatenation of adapted standard deviations	MD
	FSV	Concatenation of WSV, GSV and VSV	$M+2MD$
	nGSV	Normalization of GSV by weight and covariance	MD
MFA	FALV (i-vector)	Posterior mean of latent vector	$1, 2, \ldots \infty$
	FASV	Affine transformation of FALV	MD
	MFALV	Expectation of posterior means of latent vectors	$M, 2M, \ldots \infty$
	MFASV	Affine transformation of MFALV	MD

3.4.2 Computational Complexity

This section analyzes the space complexity of storing the model parameters used to compute a GSV, an i-vector and an MFALV, as well as the time complexity of computing a GSV, an i-vector or an MFALV when the model parameters are ready.

3.4.2.1 Space Complexity

Suppose a frame-level feature vector has a dimensionality of $D \times 1$, and the UBM has M mixture components. Since the computation of GSV depends only on the parameters of the GMM-based UBM, the space complexity of GSV is the memory space required by the GMM parameterized by $\theta_{GMM} = \{\pi_m, \boldsymbol{\mu}_m, \boldsymbol{\sigma}_m | m = 1,2 \dots M\}$. As the size of π_m, $\boldsymbol{\mu}_m$ and $\boldsymbol{\sigma}_m$ is 1×1, $D \times 1$ and $D \times 1$ respectively, the space complexity of GSV is given by (3.60), where γ is the memory space used to store a scalar value.

$$sp_{GSV} = \gamma(M + MD + MD) = \gamma M(1 + 2D) \tag{3.60}$$

Suppose the dimensionality of i-vector and MFALV is $H \times 1$, where H is an integer multiple of M such that $H = \lambda M$ with λ being a positive integer. This assumption is based on the fact that the dimensionality of MFALV can only be an integer multiple of M. Since the computation of i-vector depends on both the parameters of the GMM-based UBM and the parameters of an FA model, the space complexity of i-vector is the memory space required by the GMM parameterized by $\theta_{GMM} = \{\pi_m, \boldsymbol{\mu}_m, \boldsymbol{\sigma}_m | m = 1,2 \dots M\}$ and the FA model parameterized by $\theta_{FA} = \{\boldsymbol{V}_m, \boldsymbol{\Psi}_m | m = 1,2 \dots M\}$. As the size of $\boldsymbol{V}_m$ and $\boldsymbol{\Psi}_m$ is $D \times H$ and $D \times D$ respectively, the space complexity of i-vector is given by (3.61). (Notice that $\boldsymbol{\Psi}_m$ is a diagonal matrix such that only the diagonal elements need to be stored).

$$sp_{i\text{-}vector} = \gamma\big((M + MD + MD) + (MDH + MD)\big) = \gamma M(1 + (3 + \lambda M)D) \tag{3.61}$$

Since the computation of MFALV depends only on the parameters of the MFA-based UBM, the space complexity of MFALV is the memory space required by the MFA parameterized by $\theta_{MFA} = \{\hat{\pi}_m, \widehat{\boldsymbol{\mu}}_m, \widehat{\boldsymbol{V}}_m, \widehat{\boldsymbol{\Psi}} | m = 1,2 \dots M\}$. As the size of $\hat{\pi}_m$, $\widehat{\boldsymbol{\mu}}_m$, $\widehat{\boldsymbol{V}}_m$ and $\widehat{\boldsymbol{\Psi}}$ is 1×1, $D \times 1$, $D \times \frac{H}{M}$ and $D \times D$

respectively, the space complexity of MFALV is given by (3.62). (Notice that $\widehat{\boldsymbol{\Psi}}$ is a diagonal matrix such that only the diagonal elements need to be stored).

$$sp_{MFALV} = \gamma\left(M + MD + MD \times \frac{H}{M} + D\right) = \gamma M\left(1 + (1+\lambda)D + \frac{D}{M}\right) \tag{3.62}$$

As can be seen from (3.60) ~ (3.62), GSV has the lowest space complexity, i-vector has the highest space complexity, and MFALV has the space complexity in between. This implies that i-vector needs more memory space than MFALV to store the model parameters used for computation.

3.4.2.2 Time Complexity

Given an acoustic sample s denoted as $s = \{\boldsymbol{x}_1, \boldsymbol{x}_2 \ldots \boldsymbol{x}_T\}$, the computation of GSV is based on $E_m[\boldsymbol{x}]$ given by (3.3). Thus, most of the computation time will be consumed by computing the posterior probability $p(m|\boldsymbol{x}_t, \theta)$. Since there are T frame-level feature vectors obtained from the acoustic sample, the time complexity of GSV is approximately given by (3.63), where $O(.)$ is the big-O notation upper bounding the running time of an algorithm.

$$t_{GSV} = O(MDT) \tag{3.63}$$

The computation of i-vector is based on $E[\boldsymbol{z}_s]$ given by (3.27). The matrices involved in computation, i.e., $\boldsymbol{V}$, $\boldsymbol{\Psi}$, $E_s[\boldsymbol{X}^0]$ and $E_s[\boldsymbol{X}]$, have sizes of $MD \times H$, $MD \times MD$, $MD \times MD$ and $MD \times 1$ respectively. Suppose the inversion of a matrix with a size of $A \times A$ has a time complexity of $O(A^3)$, the multiplication of matrices with sizes $A \times B$ and $B \times C$ has a time complexity of $O(ABC)$, and the inverse of $\boldsymbol{\Psi}$ has been pre-computed. The time complexity of i-vector can then be approximated by (3.64), with the assumption that $\lambda = O(D)$, i.e., λ has at most the same order of magnitude as D. (Notice that only the term with the highest order of magnitude is preserved).

$$\begin{aligned} t_{i\text{-}vector} &= O(H \times MD \times MD \times MD \times H) + O(H^3) + O(H \times H \times MD \times MD \times 1) \\ &= O(H^2M^3D^3) + O(H^3) + O(H^2M^2D^2) = O(\lambda^2M^5D^3) + O(\lambda^3M^3) + O(\lambda^2M^4D^2) \\ &= O(\lambda^2M^5D^3) = O(M^5D^5) \end{aligned} \tag{3.64}$$

The computation of MFALV is based on $E_m[\boldsymbol{z}_t]$ used in (3.54), namely $E_m[\boldsymbol{z}_t] = \left(\boldsymbol{I} + \widehat{\boldsymbol{V}}_m^T \widehat{\boldsymbol{\Psi}}^{-1} \widehat{\boldsymbol{V}}_m\right)^{-1} \widehat{\boldsymbol{V}}_m^T \widehat{\boldsymbol{\Psi}}^{-1} (\boldsymbol{x}_t - \widehat{\boldsymbol{\mu}}_m)$. The matrices involved in computation, i.e., $\widehat{\boldsymbol{V}}_m$ and $\widehat{\boldsymbol{\Psi}}$, have sizes of $D \times \frac{H}{M}$ and $D \times D$ respectively. Suppose the inverse of $\widehat{\boldsymbol{\Psi}}$ has been pre-computed, the time complexity of MFALV can then be approximated by (3.65), with the assumption that $\lambda = O(D)$.

$$t_{MFALV} = MT \times \left(O\left(\frac{H}{M} \times D \times D \times \frac{H}{M}\right) + O\left(\frac{H^3}{M^3}\right) + O\left(\frac{H}{M} \times \frac{H}{M} \times D \times D \times 1\right)\right)$$

$$= MT \times \left(O\left(\frac{H^2 D^2}{M^2}\right) + O\left(\frac{H^3}{M^3}\right) + O\left(\frac{H^2 D^2}{M^2}\right)\right) = MT \times (O(\lambda^2 D^2) + O(\lambda^3) + O(\lambda^2 D^2))$$

$$= MT \times O(\lambda^2 D^2) = O(MD^4 T) \tag{3.65}$$

As can be seen from (3.63) ~ (3.65), if T is relatively small such that $T \ll M^4 D^4$, computing GSV will be much more efficient than computing i-vector and MFALV. If $T \ll M^4 D$, computing MFALV will be much more efficient than computing i-vector.

A summary of the computational complexity of GSV, i-vector and MFALV is given in Table 3.2.

Table 3.2 Computational complexity of different vector-based representations.

Name	Parameters	Space complexity	Time complexity
GSV	GMM	$\gamma M(1 + 2D)$	$O(MDT)$
FALV (i-vector)	GMM, FA	$\gamma M(1 + (3 + \lambda M)D)$	$O(M^5 D^5)$
MFALV	MFA	$\gamma M(1 + (1 + \lambda)D + D/M)$	$O(MD^4 T)$

3.5 Generic Supervector

3.5.1 A General Formulation

According to the analyses and discussions in previous sections, we may illustrate the computation of GSV, i-vector and MFALV using Figure 3.4. As can be seen from Figure 3.4, all of them are based on a background model, which is a mixture model, such as a GMM or an MFA. The background model

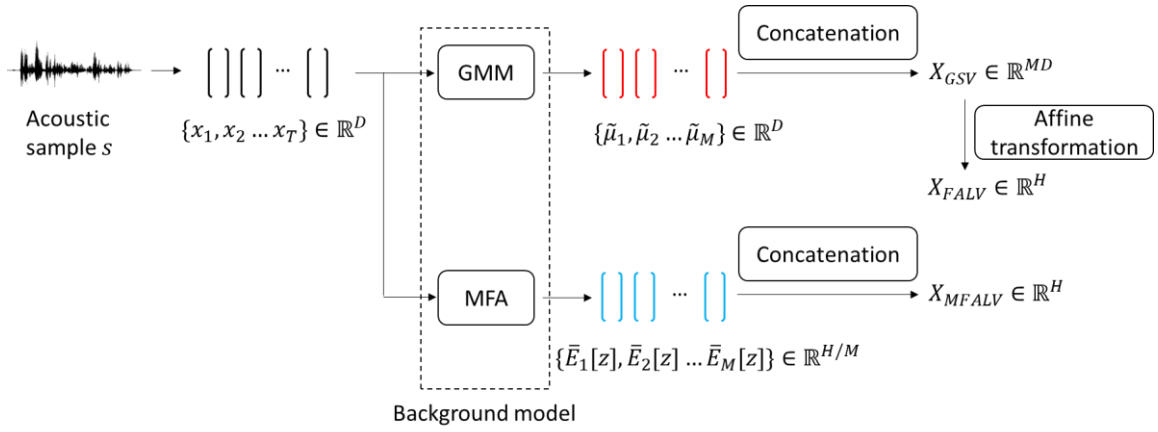


Figure 3.4 The computation of GSV, i-vector and MFALV.

provides a feature mapping that maps a sequence of input vectors (e.g., $\boldsymbol{x}_1, \boldsymbol{x}_2 \dots \boldsymbol{x}_T$) to a sequence of output vectors (e.g., $\tilde{\boldsymbol{\mu}}_1, \tilde{\boldsymbol{\mu}}_2 \dots \tilde{\boldsymbol{\mu}}_M$). The output vectors are then concatenated to form a single feature vector, whose dimensionality can be higher (e.g., the dimensionality of GSV) or lower (e.g., the dimensionality of MFALV) than the input feature dimensionality D. We call this single feature vector the generic supervector. The number of output vectors, i.e., M, depends only on the number of mixture components in the background model, instead of the number of input vectors.

Based on these observations, we propose a generic formulation for the generic supervector $\boldsymbol{X}_{SV}(s)$ as given by (3.66), where the m-th sub-vector $\boldsymbol{X}_{SV,m}(s)$ is given by (3.67).

$$\boldsymbol{X}_{SV}(s) = \left[\left(\boldsymbol{X}_{SV,1}(s)\right)^T \quad \left(\boldsymbol{X}_{SV,2}(s)\right)^T \quad \cdots \quad \left(\boldsymbol{X}_{SV,M}(s)\right)^T\right]^T \tag{3.66}$$

where

$$\boldsymbol{X}_{SV,m}(s) = a_m f_m(s|\Theta) + b_m f_m(S|\Theta) \tag{3.67}$$

In (3.67), s represents an acoustic sample, denoted as $s = \{\boldsymbol{x}_1, \boldsymbol{x}_2 \dots \boldsymbol{x}_T\}$; S represents a set of acoustic samples; Θ represents the model parameters of the background model, which is a collection of sub-models, denoted as $\Theta = \{\Theta_1, \Theta_2 \dots \Theta_M\}$; f_m is a feature mapping that maps a sequence of vectors to a single vector based on Θ; a_m and b_m are weighting coefficients. $f_m(s|\Theta)$ and $f_m(S|\Theta)$ are called the mapped vector and the calibration vector respectively. The mapped vector is based on the statistics of the background model and the sample s. The calibration vector is based on the statistics of the

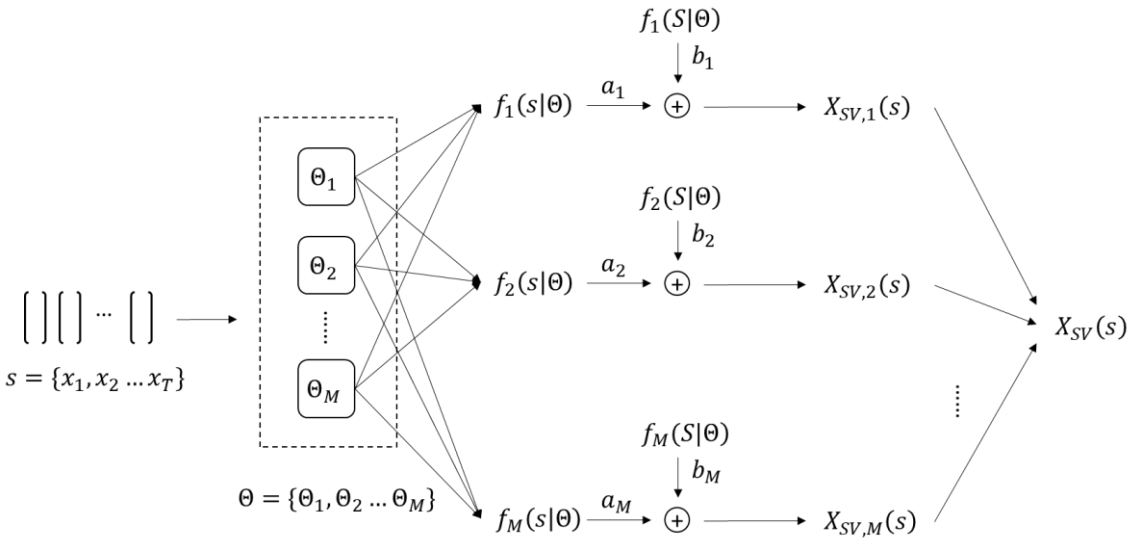


Figure 3.5 A generic supervector framework.

background model and a set of samples, which may provide some calibration. An illustration of the generic supervector is shown in Figure 3.5.

The formulation of GSV fits well to this generic formulation by comparing (3.67) and (3.6). The generic supervector then becomes GSV with the corresponding parameters given by (3.68). I-vector can then be obtained by post-processing the generic supervector.

$$\begin{gathered}\Theta_m = \{\pi_m, \boldsymbol{\mu}_m, \boldsymbol{\sigma}_m\} \\ f_m(s|\Theta) = \frac{1}{\sum_{t=1}^{T} p(m|\boldsymbol{x}_t, \Theta)} \sum_{t=1}^{T} p(m|\boldsymbol{x}_t, \Theta)\boldsymbol{x}_t \\ f_m(S|\Theta) = \boldsymbol{\mu}_m \\ a_m = \alpha_m \\ b_m = (1 - \alpha_m)\end{gathered} \tag{3.68}$$

The formulation of MFALV also fits well to the generic formulation by comparing (3.67) and (3.54). The generic supervector then becomes MFALV with the corresponding parameters given by (3.69).

$$\begin{gathered}\Theta_m = \{\hat{\pi}_m, \hat{\boldsymbol{\mu}}_m, \hat{\boldsymbol{V}}_m, \hat{\boldsymbol{\Psi}}\} \\ f_m(s|\Theta) = \frac{1}{\sum_{t=1}^{T} p(m|\boldsymbol{x}_t, \Theta)} \sum_{t=1}^{T} p(m|\boldsymbol{x}_t, \Theta)\left(\boldsymbol{I} + \hat{\boldsymbol{V}}_m^T \hat{\boldsymbol{\Psi}}^{-1} \hat{\boldsymbol{V}}_m\right)^{-1} \hat{\boldsymbol{V}}_m^T \hat{\boldsymbol{\Psi}}^{-1} \boldsymbol{x}_t \\ f_m(S|\Theta) = \left(\boldsymbol{I} + \hat{\boldsymbol{V}}_m^T \hat{\boldsymbol{\Psi}}^{-1} \hat{\boldsymbol{V}}_m\right)^{-1} \hat{\boldsymbol{V}}_m^T \hat{\boldsymbol{\Psi}}^{-1} \hat{\boldsymbol{\mu}}_m \\ a_m = 1 \\ b_m = -1\end{gathered} \tag{3.69}$$

GSV and MFALV are homogeneous supervectors, where the sub-models in the background model are of the same type. It is also feasible to construct heterogeneous supervectors where the sub-models are of different types.

3.5.2 Relationship with Neural Networks

It is interesting that the feature mapping f_m can be regarded as the activation function of the m-th neuron in a neural network. Therefore, the feature representations learned by the hidden layers of a neural network, such as a convolutional neural network (CNN), can be interpreted using the generic supervector.

3.5.2.1 Interpreting Convolutional Neural Network

In the convolutional layer of a classic CNN, the input vector is convolved with a weight vector, which is a rectangular window if the input is an image. The sliding rectangular window will move from one position to another on the image. At each position, a scalar value is obtained, which is the inner product of the window and the corresponding image patch. The scalar values obtained at different positions form a filtered image, which is the result of convolving the input image with the sliding window. The sliding window is also called a filter. In a convolutional layer, each filter yields a filtered image, and multiple filters yield multiple filtered images.

On using CNN to process an image sample s, the image is actually treated as a sequence of image patches, denoted as $s = \{\boldsymbol{x}_1, \boldsymbol{x}_2 \dots \boldsymbol{x}_T\}$, where $\boldsymbol{x}_t$ is the t-th vectorized image patch. The convolution process can then be expressed as (3.70), where $\boldsymbol{w}_m$ is the m-th vectorized filter, and $*$ denotes the convolution operator.

$$\boldsymbol{w}_m * s = \begin{bmatrix} \boldsymbol{w}_m^T \boldsymbol{x}_1 \\ \boldsymbol{w}_m^T \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{w}_m^T \boldsymbol{x}_T \end{bmatrix} = \begin{bmatrix} \boldsymbol{w}_m^T & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{w}_m^T & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{w}_m^T \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix} \tag{3.70}$$

If there exist M filters, denoted as $W = \{\boldsymbol{w}_1, \boldsymbol{w}_2 \dots \boldsymbol{w}_M\}$, the vectorized output of a convolutional layer can then be expressed as (3.71). An illustration of the convolution process is shown in Figure 3.6.

$$W * s = [(\boldsymbol{w}_1 * s)^T \quad (\boldsymbol{w}_2 * s)^T \quad \cdots \quad (\boldsymbol{w}_M * s)^T]^T \tag{3.71}$$

The feature representation produced by a convolutional layer, viz. $W * s$, can then be interpreted using the formulation of the generic supervector, with the corresponding parameters given by (3.72).

$$\begin{aligned} \Theta_m &= \{\boldsymbol{w}_m\} \\ f_m(s|\Theta) &= \boldsymbol{w}_m * s \\ f_m(S|\Theta) &= \mathbf{0} \\ a_m &= 1 \\ b_m &= 0 \end{aligned} \tag{3.72}$$

The feature representation produced by a deep CNN can be interpreted as a multi-level supervector, which is a deep supervector.

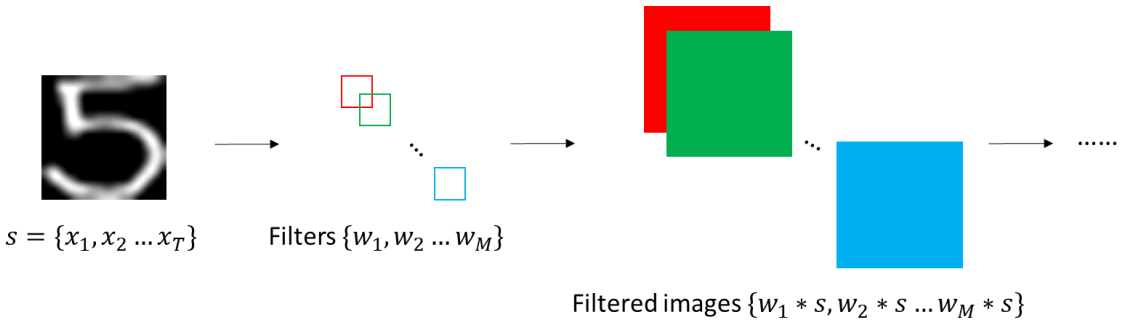


Figure 3.6 An illustration of the convolution process in a convolutional layer.

3.5.2.2 Interpreting Residual Network

As shown in [107], when CNN has a deep structure, further increasing the depth may degrade the performance. This phenomenon can be intuitively explained using (3.72). Since the calibration vector $f_m(S|\Theta)$ does not occur in the formulation, the output of a convolutional layer highly depends on the output of the previous layer, which may lead to instability issues. Fortunately, the degradation of performance can be alleviated by adopting a residual network (ResNet) structure [107], where the output of a convolutional layer becomes (3.73). In (3.73), $\boldsymbol{w}_0$ is a feature transformation used to ensure $\boldsymbol{w}_m * s$ and $\boldsymbol{w}_0 * s$ have the same dimensionality. If $\boldsymbol{w}_m * s$ already has the same dimensionality as the input image sample s, $\boldsymbol{w}_0 * s$ is then the same as the input image s, meaning that $\boldsymbol{w}_0$ performs an identity feature mapping [107].

$$\boldsymbol{w}_m * s + \boldsymbol{w}_0 * s = \begin{bmatrix} \boldsymbol{w}_m^T \boldsymbol{x}_1 \\ \boldsymbol{w}_m^T \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{w}_m^T \boldsymbol{x}_T \end{bmatrix} + \begin{bmatrix} \boldsymbol{w}_0^T \boldsymbol{x}_1 \\ \boldsymbol{w}_0^T \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{w}_0^T \boldsymbol{x}_T \end{bmatrix} = \left(\begin{bmatrix} \boldsymbol{w}_m^T & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{w}_m^T & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{w}_m^T \end{bmatrix} + \begin{bmatrix} \boldsymbol{w}_0^T & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{w}_0^T & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{w}_0^T \end{bmatrix} \right) \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix} \tag{3.73}$$

The ResNet structure can be interpreted using the formulation of the generic supervector, with the corresponding parameters given by (3.74).

$$\begin{gathered} \Theta_m = \{\boldsymbol{w}_m, \boldsymbol{w}_0\} \\ f_m(s|\Theta) = \boldsymbol{w}_m * s \\ f_m(S|\Theta) = \boldsymbol{w}_0 * s \\ a_m = 1 \\ b_m = 1 \end{gathered} \tag{3.74}$$

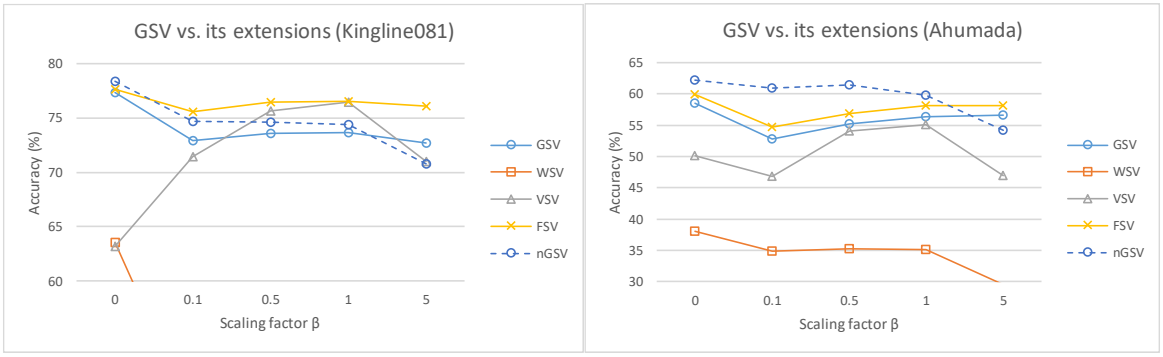
As can be seen from (3.74), there exist a calibration vector in the formulation of the ResNet structure. The existence of this calibration vector may help explain the robustness of ResNet intuitively.

Chapter 4 Feature Representations: Experimental Comparison

4.1 Experimental Comparison between GSV and Its Extensions

This section briefly compares the performance of GSV, nGSV, WSV, VSV, and FSV in two speaker identification tasks. The two speech corpora are Kingline081 and Ahumada. The frame-level feature vector is the 20-dimensional MFCC vector extracted using the Hamming window with 40ms frame length and 20ms frame shift. The training data are also used to construct the UBM with 128 mixture components. The dimensionality of GSV, nGSV, WSV, VSV and FSV is 2560×1, 2560×1, 128×1, 2560×1 and 5248×1 respectively.

The experimental results in the two speaker identification tasks are shown in Figure 4.1, where Figure 4.1 (a) shows the results on the Kingline081 dataset, and Figure 4.1 (b) shows the results on the Ahumada dataset. Some accuracy values are too low to be shown in the figure. The classifier is a linear SVM implemented using LIBSVM [85] with default parameters. As shown in the figure, GSV outperforms WSV and VSV in general, implying that the adapted mean parameter is the most important



(a) (b)

Figure 4.1 GSV vs. its extensions for speaker identification employing SVM as the classifier. (a) Results on Kingline081. (b) Results on Ahumada.

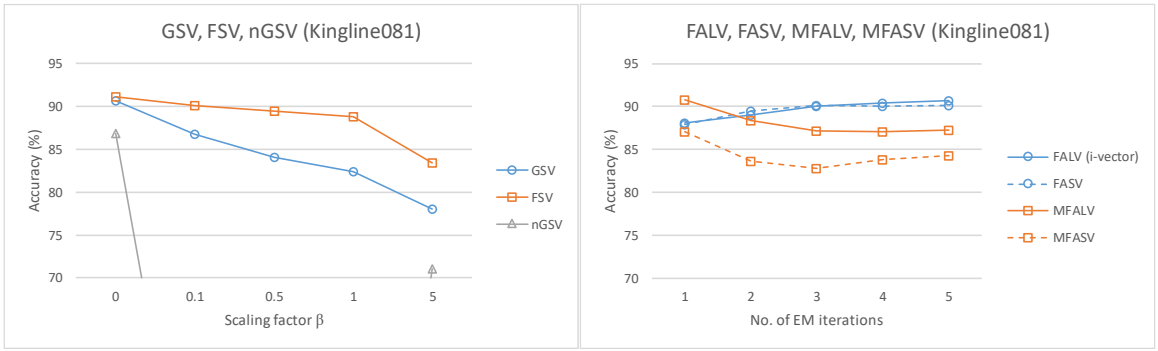
among all parameters. WSV gives the worst performance, probably owing to its low dimensionality, which makes it unable to carry enough information. Nevertheless, the weight parameter and the standard deviation parameter are useful supplements to GSV, as demonstrated by the improved performance offered by FSV. The normalization operation may also be useful, as nGSV outperforms GSV and FSV for some values of β. The scaling factor also plays an important role. The smaller the value of β, the more the supervector will depend on the statistics of the specific sample instead of the statistics of the UBM. This will make the supervectors less similar to each other because the statistics of different samples can be quite different, while the statistics of the UBM are shared among all supervectors.

4.2 Experimental Comparison of Different Representations

This section compares different vector-based representations in two speaker identification tasks, in terms of their effectiveness and efficiency. The two speech corpora are Kingline081 and Ahumada. The frame-level feature vector is the 20-dimensional MFCC vector extracted using the Hamming window with 40ms frame length and 20ms frame shift. The GMM-based UBM is obtained using the mixture splitting technique [89]. The parameters of the FA model used to construct FALV (i-vector) and FASV are estimated using Algorithm 3.1. The parameters of the MFA used to construct MFALV and MFASV are estimated using Algorithm 3.2. For each speech corpus, the training data are also used to train the GMM-based UBM, the FA model, and the MFA. PLDA is employed as the classifier, which is implemented using the scalable formulation [90]. The latent vector in the PLDA model has the same dimensionality as the feature representation, and the model parameters are estimated using 2 EM iterations.

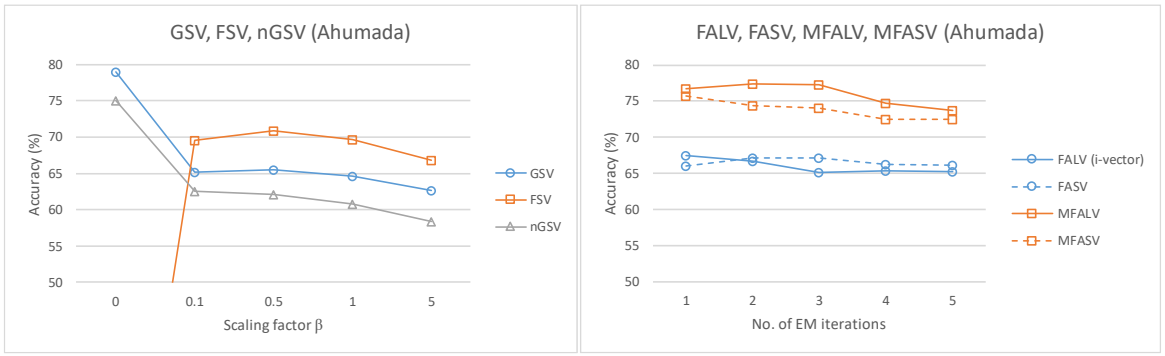
4.2.1 Effectiveness and Efficiency of Different Feature Representations

This sub-section compares the effectiveness and efficiency of different feature representations under the condition of the same dimensionality (except for FSV). The UBM has 128 mixture components, and all the feature representations have a dimensionality of 2560×1, except for FSV, whose dimensionality is 5248×1. GSV and its extensions are calculated with different values of the scaling



(a) (b)

Figure 4.2 Speaker identification results on Kingline081 employing PLDA as the classifier. (a) Results achieved by GSV and its extensions. (b) Results achieved by i-vector, MFALV and their extensions.



(a) (b)

Figure 4.3 Speaker identification results on Ahumada employing PLDA as the classifier. (a) Results achieved by GSV and its extensions. (b) Results achieved by i-vector, MFALV and their extensions.

factor. The model parameters of i-vector, MFALV and their extensions are estimated using different numbers of EM iterations.

The identification accuracy results on Kingline081 and Ahumada are shown in Figure 4.2 and Figure 4.3 respectively. Figure 4.2 (a) and Figure 4.3 (a) show the identification accuracy achieved by GSV and its extensions, while Figure 4.2 (b) and Figure 4.3 (b) show the identification accuracy achieved by i-vector, MFALV and their extensions. Some accuracy values are too low to be displayed.

As shown in Figure 4.2 (a) and Figure 4.3 (a), GSV and its extensions are sensitive to the scaling factor. As explained before, the scaling factor controls the similarity between different GSVs. The smaller the scaling factor, the less similar different GSVs will be. As PLDA itself is an FA model, which assumes

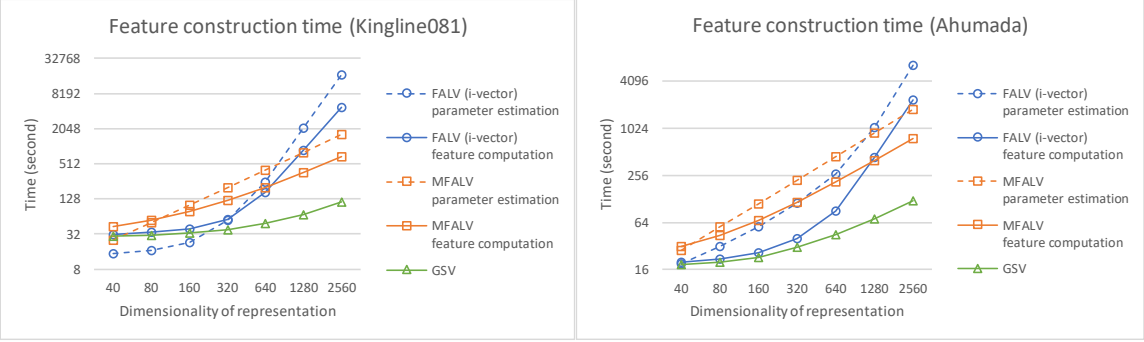
that the feature representations should follow a Gaussian distribution, the scaling factor seems also controlling how well the PLDA model assumption has been fulfilled.

As shown in Figure 4.2 (b) and Figure 4.3 (b), i-vector, MFALV and their extensions are sensitive to the number of EM iterations used to estimate the model parameters, but more EM iterations do not mean better performance. Certainly, the more the EM iterations, the more accurate the model parameters will be estimated towards the model assumption. However, if the model assumption is ill-posed, a more precise estimation will incur a greater deviation.

The highest accuracy achieved by GSV is comparable to that achieved by i-vector (Figure 4.2) or even better (Figure 4.3). As explained before, i-vector is an affine transformation of GSV. This transformation may incur information loss. The highest accuracy achieved by MFALV is comparable to that achieved by i-vector (Figure 4.2) or even better (Figure 4.3), which demonstrates the effectiveness of the proposed representation.

It is also noticed that the latent vectors (i.e., FALV and MFALV) tend to outperform their corresponding supervectors (i.e., FASV and MFASV). One possible reason is that, FASV and MFASV are the affine transformation of FALV and MFALV respectively, so the latent vectors are the original feature representation while the supervectors are the transformed feature representation. This transformation may bring distortion. From another perspective, the latent vector and the supervector can be regarded as the essence and the appearance of a sample respectively. So, the latent vector serves as the cause of the supervector, while the supervector serves as the effect of the latent vector. From this angle, the latent vector is expected to be purer.

The time used to construct GSV, i-vector, and MFALV using Kingline081 and Ahumada is shown in Figure 4.4, where Figure 4.4 (a) shows the construction time using the Kingline081 dataset, and Figure 4.4 (b) shows the construction time using the Ahumada dataset. The time consumption is estimated by running the MATLAB implementations on an iMac desktop computer with 32G memory. For GSV, time is consumed for feature computation only, as there is no need to estimate additional model parameters. For i-vector and MFALV, time is consumed for both feature computation and model



(a) (b)

Figure 4.4 Time consumption for the construction process of different feature representations. (a) Time consumption on Kingline081. (b) Time consumption on Ahumada.

parameter estimation. GSV is constructed with $\beta = 0.1$, and the model parameters for i-vector and MFALV are estimated with 1 EM iteration, just for illustration. UBMs with the number of mixture components varying from 2 to 128 are used to construct the feature representations, so the dimensionality of the feature representations varies from 40×1 to 2560×1.

As shown in Figure 4.4, GSV has the highest computational efficiency among all, as it does not need extra parameter estimation procedures. When the dimensionality of the feature representation is high, the construction process of MFALV can be more efficient than that of i-vector, as the former crumbles the large factor-loading matrix into small matrices such that the matrix inversion computation is faster. Albeit GSV is the fastest in computation, its dimensionality is strictly restricted by the number of mixture components in the UBM. That is to say, the dimensionality of GSV is fixed for a given UBM. In contrast, the dimensionality of i-vector and MFALV is changeable, depending on the size of the factor-loading matrix, even if the UBM is fixed.

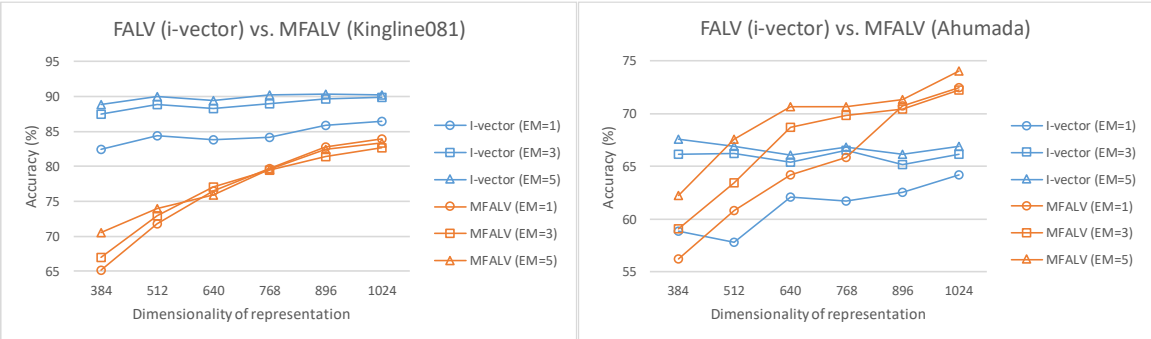
4.2.2 Dimensionality Flexibility of I-vector and MFALV

This sub-section compares the effectiveness and efficiency of i-vector and MFALV when their dimensionality varies. The UBM has 128 mixture components, but the size of the factor-loading matrix varies, resulting in the dimensionality of i-vector and MFALV varying from 384×1 to 1024×1. The number of EM iterations used to estimate the model parameters of i-vector and MFALV varies.

The identification accuracy results on Kingline081 and Ahumada are shown in Figure 4.5, where Figure 4.5 (a) shows the identification accuracy using the Kingline081 dataset, and Figure 4.5 (b) shows the identification accuracy using the Ahumada dataset. It can be seen from the figure that the effectiveness of i-vector is quite robust to the change of the dimensionality. Even if the dimensionality of i-vector is as low as 384×1, the performance can still be good. In contrast, the effectiveness of MFALV is highly dependent on the dimensionality. The lower the dimensionality, the lower the effectiveness.

The different characteristics of i-vector and MFALV are caused by the differences in their model assumptions. In brief, i-vector captures the common characteristics of all the frame-level feature vectors in a sample, whereas MFALV captures the characteristics of individual frame-level feature vectors and then takes the weighted average. Suppose the dimensionality of i-vector and MFALV is $H \times 1$. From the perspective of i-vector, the common characteristics are carried by an $H \times 1$ vector. From the perspective of MFALV, the individual characteristics are carried by an $\frac{H}{M} \times 1$ vector (M is the number of mixture components in the UBM). If H is small and M is large, the $\frac{H}{M} \times 1$ vector may not be able to capture enough information.

These observations imply that the dimensionality of i-vector can be chosen to be very low without suffering from performance degradation. In contrast, the dimensionality of MFALV should be chosen to be relatively high so as to prevent performance degradation.

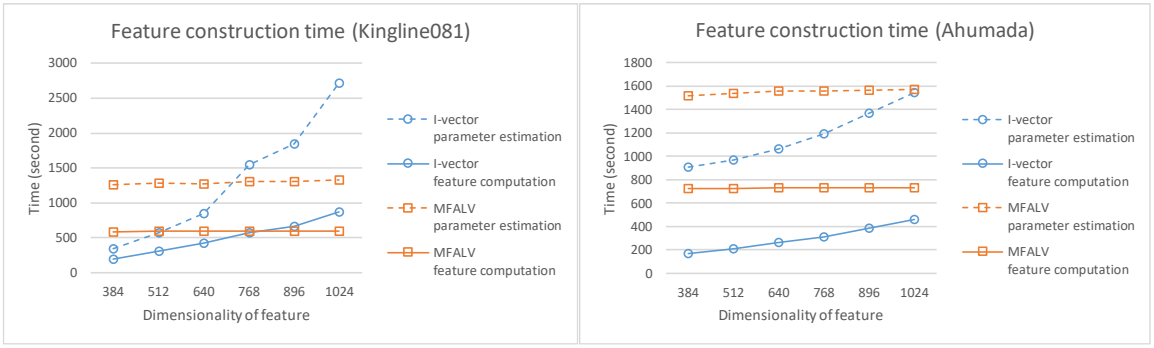


(a) (b)

Figure 4.5 The effectiveness of i-vector and MFALV with different dimensionalities. (a) Results on Kingline081. (b) Results on Ahumada.

The time used to construct i-vector and MFALV using Kingline081 and Ahumada is shown in Figure 4.6, where Figure 4.6 (a) shows the construction time using the Kingline081 dataset, and Figure 4.6 (b) shows the construction time using the Ahumada dataset. The time consumption is estimated by running the MATLAB implementations on an iMac desktop computer with 32G memory. The model parameters for i-vector and MFALV are estimated with 1 EM iteration. The UBM has 128 mixture components.

It can be seen from the figure that the computational efficiency of MFALV is quite robust to the change of the dimensionality. Even if the dimensionality of MFALV is as high as 1024×1, the time consumption can still be low. In contrast, the computational efficiency of i-vector is highly dependent on the dimensionality. The higher the dimensionality, the lower the computational efficiency. These observations imply that the dimensionality of MFALV can be chosen to be very high without suffering from computational inefficiency, while the dimensionality of i-vector should be chosen to be relatively low in order to lighten the computational burden.



(a) (b)

Figure 4.6 The efficiency of i-vector and MFALV with different dimensionalities. (a) Time consumption on Kingline081. (b) Time consumption on Ahumada.

Chapter 5 Classifiers: Theoretical Analysis

5.1 Gaussian Mixture Model

This section starts by describing a general mixture model. It then introduces the basic formulas of the Gaussian mixture model (GMM), which is a popular probabilistic model applicable to speech signal processing tasks, either used for doing classification [22] or feature extraction [7]. GMM is able to smoothly approximate any probability density function [9]. The expectation-maximization (EM) algorithm used to estimate the model parameters of GMM is also introduced. The EM algorithm is an iterative algorithm that requires some initialized parameters. Rather than a random initialization of the parameters, the mixture splitting strategy can be adopted.

5.1.1 Expectation-Maximization Algorithm for Mixture Models

A mixture model is a mixture of many single models. Different single models have different parameters and describe different probability distributions, so combining these single models to form a mixture model may better describe the distribution of a set of data. Suppose a mixture model consists of M mixture components, denoted as $\theta = \{\theta_1, \theta_2, \dots \theta_M\}$. Given a vector $\boldsymbol{x}$, the probability that the mixture model assigns to it is the weighted sum of the probabilities assigned by the mixture components as given by (5.1), where the weighting coefficient π_m adds up to 1 to ensure $p(\boldsymbol{x}|\theta)$ is a probability distribution.

$$p(\boldsymbol{x}|\theta) = \sum_{m=1}^{M} \pi_m p(\boldsymbol{x}|\theta_m) \tag{5.1}$$

Given a set of training vectors $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots \boldsymbol{x}_N\}$, the parameters of the mixture model can be estimated by maximizing the log-likelihood given by (5.2).

$$\mathcal{L} = \sum_{n=1}^{N} \ln\ p(\boldsymbol{x}_n|\theta) = \sum_{n=1}^{N} \ln \sum_{m=1}^{M} \pi_m p(\boldsymbol{x}_n|\theta_m) \tag{5.2}$$

However, as $\mathcal{L}$ involves the logarithm of the summation, it may be challenging to optimize (5.2) directly. Suppose a training vector $\boldsymbol{x}_n$ is generated by the probability distribution described by one of the M mixture components. For $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_N\}$, there can be a set of corresponding latent variables $\{z_1, z_2, \ldots z_N\}$, where $z_n \in \{1,2 \ldots M\}$ tells which mixture component generates $\boldsymbol{x}_n$. With the help of this latent variable, we may then apply the expectation-maximization (EM) algorithm to optimize the parameters of the mixture model [91].

The EM algorithm includes an E-step and an M-step. In the E-step, the posterior probability of the latent variable is computed as given by (5.3), which is the probability that the latent variable z_n equals m given the observation $\boldsymbol{x}_n$. As z_n is a latent variable, we will never know its exact value, but only the probabilities of being different values.

$$p(z_n = m|\boldsymbol{x}_n, \theta) = \frac{p(\boldsymbol{x}_n, z_n=m|\theta)}{p(\boldsymbol{x}_n|\theta)} = \frac{p(\boldsymbol{x}_n|z_n=m,\theta)p(z_n=m|\theta)}{p(\boldsymbol{x}_n|\theta)} = \frac{\pi_m p(\boldsymbol{x}_n|\theta_m)}{\sum_{j=1}^{M} \pi_j p(\boldsymbol{x}_n|\theta_j)} \tag{5.3}$$

In the M-step, with the posterior probability given by (5.3) fixed, the model parameters are re-estimated using the posterior probability in the E-step, by maximizing $Q(\theta', \theta)$ defined in (5.4) with respect to θ'. θ' is the new estimation of θ.

$$Q(\theta', \theta) = \sum_{n=1}^{N} \sum_{m=1}^{M} p(z_n = m|\boldsymbol{x}_n, \theta) \ln p(\boldsymbol{x}_n, z_n = m|\theta') \tag{5.4}$$

The EM algorithm will run for several iterations until convergence or the total number of iterations exceeds some threshold. In each iteration, the E-step computes the posterior probability based on the model parameters from its previous iteration, and the M-step re-estimates the parameters based on the posterior probability obtained in the E-step.

5.1.2 Fundamentals of GMM

Gaussian mixture model (GMM), as a famous type of mixture model, is a mixture of many single Gaussian models combined in a weighted manner. A GMM with M Gaussian components can be parameterized by $\theta = \{\pi_m, \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m | m = 1,2 \ldots M\}$, where π_m, $\boldsymbol{\mu}_m$ and $\boldsymbol{\Sigma}_m$ represents the weight, the mean, and the covariance of the m-th Gaussian component.

Given a set of training vectors $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_N\}$ and a set of initialized parameters $\theta = \{\pi_m, \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m | m = 1,2 \ldots M\}$, based on the aforementioned EM algorithm, the E-step computes the posterior probability of $\boldsymbol{x}_n$ being generated by the m-th Gaussian component as given by (5.5), where $p_g(\boldsymbol{x}_n | \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)$ is the Gaussian probability density function with mean $\boldsymbol{\mu}_m$ and covariance $\boldsymbol{\Sigma}_m$.

$$p(m|\boldsymbol{x}_n, \theta) = \frac{\pi_m p_g(\boldsymbol{x}_n|\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)}{\sum_{j=1}^{M} \pi_j p_g(\boldsymbol{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \tag{5.5}$$

The M-step then, making use of the posterior probability in (5.5), computes a new set of model parameters $\theta' = \{\pi'_m, \boldsymbol{\mu}'_m, \boldsymbol{\Sigma}'_m | m = 1,2 \ldots M\}$ by maximizing $Q(\theta', \theta)$ with respect to θ', yielding (5.6) [91].

$$\frac{\partial Q}{\partial \theta'} = \mathbf{0} \quad \Rightarrow \quad \begin{aligned} \pi'_m &= \frac{1}{N} \sum_{n=1}^{N} p(m|\boldsymbol{x}_n, \theta) \\ \boldsymbol{\mu}'_m &= \frac{\sum_{n=1}^{N} p(m|\boldsymbol{x}_n, \theta) \boldsymbol{x}_n}{\sum_{n=1}^{N} p(m|\boldsymbol{x}_n, \theta)} \\ \boldsymbol{\Sigma}'_m &= \frac{\sum_{n=1}^{N} p(m|\boldsymbol{x}_n, \theta)(\boldsymbol{x}_n - \boldsymbol{\mu}'_m)(\boldsymbol{x}_n - \boldsymbol{\mu}'_m)^T}{\sum_{n=1}^{N} p(m|\boldsymbol{x}_n, \theta)} \end{aligned} \tag{5.6}$$

The parameters of the Gaussian components in a GMM can be initialized randomly or using the K-means algorithm [91]. After initialization, the parameters can be re-estimated for several EM iterations until convergence. The EM algorithm is summarized in Algorithm 5.1, where the superscript i indicates the number of EM iterations having been performed, and I is the total number of EM iterations.

5.1.3 Mixture Splitting Strategies

Most of the time, it is safe to assume that the covariance matrix $\boldsymbol{\Sigma}_m$ for each Gaussian component in a GMM is diagonal [9]. The parameters of a GMM can then be expressed as $\theta_M = \{\pi_m, \boldsymbol{\mu}_m, \boldsymbol{\sigma}_m | m = 1,2 \ldots M\}$, where the i-th element of $\boldsymbol{\sigma}_m$ is the square root of the ii-th element of $\boldsymbol{\Sigma}_m$. With this assumption, we may then adopt a mixture splitting strategy to initialize the model parameters in a deterministic way instead of randomly [89]. This strategy starts from the model parameters of a single Gaussian model (i.e., 1-component GMM), and increases the number of mixture components step by step to M. Two mixture splitting strategies are given in Algorithm 5.2 and Algorithm 5.3. The subscript of θ_i and the superscript of $\{\pi_m^{(i)}, \boldsymbol{\mu}_m^{(i)}, \boldsymbol{\Sigma}_m^{(i)}\}$ indicate that the number of Gaussian components in the GMM θ_i is i. The parameters of θ_1 are simply those of a single Gaussian model. The operator ∪

Algorithm 5.1 EM algorithm for GMM

1:	Initialization:		$\theta^{(0)} = \left\{\pi_m^{(0)}, \boldsymbol{\mu}_m^{(0)}, \boldsymbol{\Sigma}_m^{(0)} \middle| m = 1,2 \dots M\right\}$
2:	**For** i = 1 to I		
3:	**For** m = 1 to M		
4:		E-step:	$p\left(m\middle|\boldsymbol{x}_n, \theta^{(i-1)}\right) = \frac{\pi_m^{(i-1)} p_g\left(\boldsymbol{x}_n\middle|\boldsymbol{\mu}_m^{(i-1)}, \boldsymbol{\Sigma}_m^{(i-1)}\right)}{\sum_{j=1}^{M} \pi_j^{(i-1)} p_g\left(\boldsymbol{x}_n\middle|\boldsymbol{\mu}_j^{(i-1)}, \boldsymbol{\Sigma}_j^{(i-1)}\right)}$
5:			$\pi_m^{(i)} = \frac{1}{N}\sum_{n=1}^{N} p\left(m\middle|\boldsymbol{x}_n, \theta^{(i-1)}\right)$
6:		M-step:	$\boldsymbol{\mu}_m^{(i)} = \frac{\sum_{n=1}^{N} p\left(m\middle|\boldsymbol{x}_n, \theta^{(i-1)}\right)\boldsymbol{x}_n}{\sum_{n=1}^{N} p(m\middle|\boldsymbol{x}_n, \theta^{(i-1)})}$
7:			$\boldsymbol{\Sigma}_m^{(i)} = \frac{\sum_{n=1}^{N} p\left(m\middle|\boldsymbol{x}_n, \theta^{(i-1)}\right)\left(\boldsymbol{x}_n - \boldsymbol{\mu}_m^{(i)}\right)\left(\boldsymbol{x}_n - \boldsymbol{\mu}_m^{(i)}\right)^T}{\sum_{n=1}^{N} p(m\middle|\boldsymbol{x}_n, \theta^{(i-1)})}$
8:	**End**		
9:	Update θ:		$\theta^{(i)} = \left\{\pi_m^{(i)}, \boldsymbol{\mu}_m^{(i)}, \boldsymbol{\Sigma}_m^{(i)} \middle| m = 1,2 \dots M\right\}$
10:	**End**		

Algorithm 5.2 GMM mixture splitting strategy I

1:	Initialization:	$\theta_1 = \left\{\pi_1^{(1)}, \boldsymbol{\mu}_1^{(1)}, \boldsymbol{\sigma}_1^{(1)}\right\} = \left\{1, \boldsymbol{\mu}_1^{(1)}, \boldsymbol{\sigma}_1^{(1)}\right\}$
2:	**For** i = 2 to M	
3:	Find largest weight:	$j = \underset{m}{\operatorname{argmax}}\left\{\pi_m^{(i-1)} \middle| m = 1,2 \dots i-1\right\}$
4:	Split mixture component:	$\theta_i = \left\{\pi_m^{(i-1)}, \boldsymbol{\mu}_m^{(i-1)}, \boldsymbol{\sigma}_m^{(i-1)} \middle| m = 1 \dots i-1, m \neq j\right\}$ $\cup \left\{0.5\pi_j^{(i-1)}, \boldsymbol{\mu}_j^{(i-1)} - 0.2\boldsymbol{\sigma}_j^{(i-1)}, \boldsymbol{\sigma}_j^{(i-1)}\right\}$ $\cup \left\{0.5\pi_j^{(i-1)}, \boldsymbol{\mu}_j^{(i-1)} + 0.2\boldsymbol{\sigma}_j^{(i-1)}, \boldsymbol{\sigma}_j^{(i-1)}\right\}$
5:	Re-estimate θ_i using Algorithm 5.1	
6:	**End**	

realizes the union of two sets. In Algorithm 5.3, the operation $\lfloor . \rfloor$ outputs the largest integer that is not

Algorithm 5.3 GMM mixture splitting strategy II (faster version)

1: Initialization: $\theta_1 = \left\{\pi_1^{(1)}, \boldsymbol{\mu}_1^{(1)}, \boldsymbol{\sigma}_1^{(1)}\right\} = \left\{1, \boldsymbol{\mu}_1^{(1)}, \boldsymbol{\sigma}_1^{(1)}\right\}$

2: $L = \lfloor \log_2 M \rfloor$

3: **For** i = 1 to L

4: Split mixture component:
$$\theta_{2^i} = \left\{0.5\pi_m^{(2^{i-1})}, \boldsymbol{\mu}_m^{(2^{i-1})} - 0.2\boldsymbol{\sigma}_m^{(2^{i-1})}, \boldsymbol{\sigma}_m^{(2^{i-1})} \middle| m = 1 \dots 2^{i-1}\right\}$$
$$\cup \left\{0.5\pi_m^{(2^{i-1})}, \boldsymbol{\mu}_m^{(2^{i-1})} + 0.2\boldsymbol{\sigma}_m^{(2^{i-1})}, \boldsymbol{\sigma}_m^{(2^{i-1})} \middle| m = 1 \dots 2^{i-1}\right\}$$

5: Re-estimate θ_{2^i} using Algorithm 5.1

6: **End**

7: **For** $i = 2^L + 1$ to M

8: Find largest weight: $j = \underset{m}{\operatorname{argmax}} \left\{\pi_m^{(i-1)} | m = 1,2 \dots i-1\right\}$

9: Split mixture component:
$$\theta_i = \left\{\pi_m^{(i-1)}, \boldsymbol{\mu}_m^{(i-1)}, \boldsymbol{\sigma}_m^{(i-1)} \middle| m = 1 \dots i-1, m \neq j\right\}$$
$$\cup \left\{0.5\pi_j^{(i-1)}, \boldsymbol{\mu}_j^{(i-1)} - 0.2\boldsymbol{\sigma}_j^{(i-1)}, \boldsymbol{\sigma}_j^{(i-1)}\right\}$$
$$\cup \left\{0.5\pi_j^{(i-1)}, \boldsymbol{\mu}_j^{(i-1)} + 0.2\boldsymbol{\sigma}_j^{(i-1)}, \boldsymbol{\sigma}_j^{(i-1)}\right\}$$

10: Re-estimate θ_i using Algorithm 5.1

11: **End**

larger than its input real value.

The first mixture splitting strategy increases the number of Gaussian components one by one, so M Gaussian components will require splitting for $M - 1$ times. The second mixture splitting strategy first doubles the number of Gaussian components and then increases the number of Gaussian components one by one, so M Gaussian components will require splitting for $\lfloor \log_2 M \rfloor + M - 2^{\lfloor \log_2 M \rfloor}$ times. It is obvious that $\lfloor \log_2 M \rfloor + M - 2^{\lfloor \log_2 M \rfloor} \leq M - 1$, since $\lfloor \log_2 M \rfloor \leq 2^{\lfloor \log_2 M \rfloor} - 1$, implying that the second splitting strategy is more efficient. An illustration of the two mixture splitting strategies is shown in Figure 5.1.

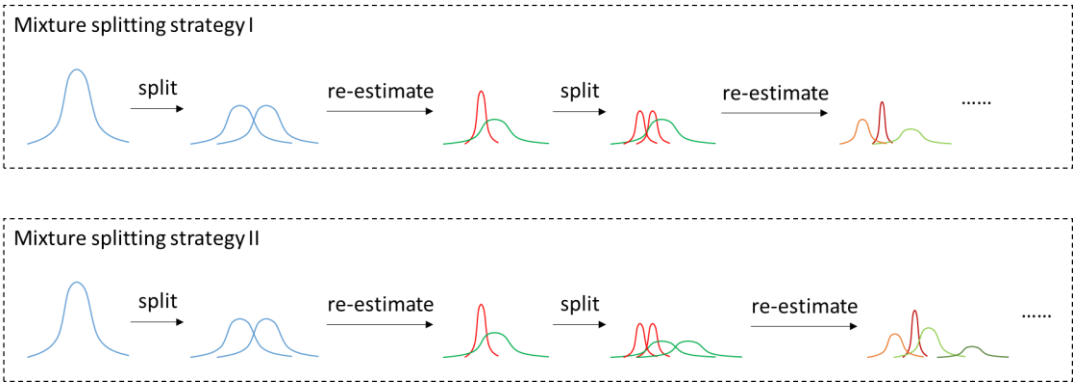


Figure 5.1 An illustration of the mixture splitting strategies.

In the experiments described in this thesis, any GMM will be constructed using Algorithm 5.3, as it is faster. Besides, we also assume that the number of Gaussian components in the GMM will be a power of 2, which simplifies the construction process and saves the construction time.

5.1.4 GMM for Classification

In order to employ GMM for doing classification, we may construct a class-specific GMM for each class using only the training data from that class. This results in a total of K class-specific GMMs for K different classes, denoted as $\{\theta_1, \theta_2, \ldots \theta_K\}$. The label $\ell(\boldsymbol{y})$ of a testing vector $\boldsymbol{y}$ can then be predicted by finding the class-specific GMM giving the largest posterior probability as given by (5.7). In the experiments described in this thesis, we will first utilize all the training data to train a class-independent GMM, and then utilize the parameters of the class-independent GMM as the initialized model parameters to train the class-specific GMMs. The class-independent GMM provides a better initialization, as the number of training data in a single class will be much smaller than the number of all the training data.

$$\ell(\boldsymbol{y}) = \underset{k}{\operatorname{argmax}}\, p(k|\boldsymbol{y}) = \underset{k}{\operatorname{argmax}} \frac{p(\boldsymbol{y}|\theta_k)}{\sum_{j=1}^{K} p(\boldsymbol{y}|\theta_j)} = \underset{k}{\operatorname{argmax}}\, p(\boldsymbol{y}|\theta_k) \tag{5.7}$$

5.2 Restricted Boltzmann Machine

This section introduces the basic concept and basic formulas for restricted Boltzmann machine (RBM), and describes how it can be used for pattern recognition tasks. RBM is a probabilistic model and thus can be used for probability estimation. This enables it to be directly used for classification tasks [48].

Stacking multiple RBMs produces the deep belief net (DBN), which can be used to initialize a deep neural network (DNN). The DNN can then be fine-tuned for different purposes, such as dimensionality reduction [51] and pattern recognition [50].

5.2.1 Fundamentals of RBM

Restricted Boltzmann machine (RBM) is a two-layer neural network. It consists of a visible layer where each neuron is called a visible unit, and a hidden layer where each neuron is called a hidden unit. The visible layer is usually used to receive the input, while the hidden layer is usually used to produce the output. Both the visible and hidden units follow a Bernoulli distribution and take the value of 0 or 1.

RBM is an energy-based model, which assumes that a form of energy exists inside the network as given by (5.8), where $\boldsymbol{v}$ is the visible vector and v_i is the i-th visible unit, $\boldsymbol{h}$ is the hidden vector and h_j is the j-th hidden unit, $\boldsymbol{b}$ is the bias term for the visible layer, $\boldsymbol{c}$ is the bias term for the hidden layer, and $\boldsymbol{W}$ is the weight matrix connecting the visible layer and the hidden layer. $E(\boldsymbol{v}, \boldsymbol{h})$ is the energy existing in the RBM [48][92]. A depiction of RBM is shown in Figure 5.2.

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\sum_{i \in visible} b_i v_i - \sum_{j \in hidden} c_j h_j - \sum_i \sum_j h_j W_{ji} v_i = -\boldsymbol{b}^T \boldsymbol{v} - \boldsymbol{c}^T \boldsymbol{h} - \boldsymbol{h}^T \boldsymbol{W} \boldsymbol{v} \quad (5.8)$$

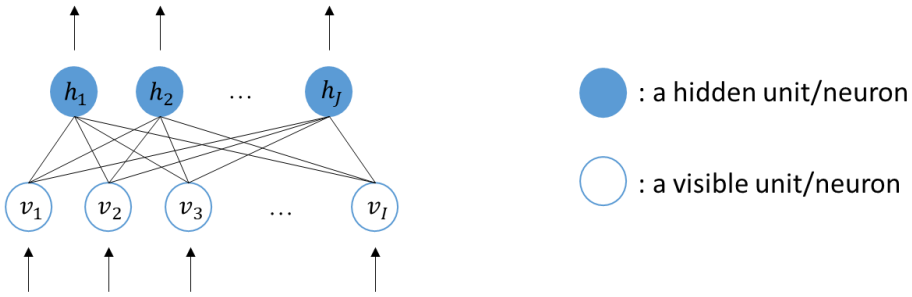


Figure 5.2 A depiction of RBM.

In an RBM, a probability is associated with $(\boldsymbol{v}, \boldsymbol{h})$ as given by (5.9), where Z is the "partition function" taking the sum over all possible values of $\boldsymbol{v}$ and $\boldsymbol{h}$ as given by (5.10), serving as the normalization term to make $p(\boldsymbol{v}, \boldsymbol{h})$ a probability distribution [48].

$$p(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{Z} e^{-E(\boldsymbol{v}, \boldsymbol{h})} \quad (5.9)$$

where

$$Z = \sum_{\boldsymbol{v}} \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})} \tag{5.10}$$

By marginalizing out $\boldsymbol{h}$, we can calculate the probability that the RBM assigns to $\boldsymbol{v}$ as given by (5.11), where $F(\boldsymbol{v})$ is defined as the free energy.

$$p(\boldsymbol{v}) = \sum_{\boldsymbol{h}} p(\boldsymbol{v},\boldsymbol{h}) = \frac{1}{Z}\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})} = \frac{1}{Z} e^{-F(\boldsymbol{v})} \tag{5.11}$$

Given a visible vector $\boldsymbol{v}$, the parameters $\theta = \{\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{c}\}$ of an RBM are adjusted by raising the probability $p(\boldsymbol{v})$ or lowering the free energy $F(\boldsymbol{v})$. This free energy determines the probability the RBM assigns to a visible vector. The lower the free energy, the higher the probability. This assumption resembles the behaviors of the molecules in a thermodynamic system, where the higher the energy, the more unstable the molecules will be. Therefore, we prefer lower energy in the system.

The gradient of $\ln p(\boldsymbol{v})$ with respect to θ can be computed as given by (5.12) [92].

$$\frac{\partial \ln p(\boldsymbol{v})}{\partial \theta} = -\sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}) \frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial \theta} + \sum_{\boldsymbol{v}'} \sum_{\boldsymbol{h}} p(\boldsymbol{v}',\boldsymbol{h}) \frac{\partial E(\boldsymbol{v}',\boldsymbol{h})}{\partial \theta} \tag{5.12}$$

Based on (5.12), we can then have the gradients with respect to $\boldsymbol{W}$, $\boldsymbol{b}$ and $\boldsymbol{c}$ as given by (5.13) [48][92], where $\langle . \rangle_{p(\boldsymbol{h}|\boldsymbol{v})}$ is the expectation over the distribution $p(\boldsymbol{h}|\boldsymbol{v})$.

$$\begin{aligned}
\frac{\partial \ln p(\boldsymbol{v})}{\partial \boldsymbol{W}} &= \sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}) \boldsymbol{h}\boldsymbol{v}^T - \sum_{\boldsymbol{v}'} \sum_{\boldsymbol{h}} p(\boldsymbol{v}',\boldsymbol{h}) \boldsymbol{h}\boldsymbol{v}'^T = \langle \boldsymbol{h}\boldsymbol{v}^T \rangle_{p(\boldsymbol{h}|\boldsymbol{v})} - \langle \boldsymbol{h}\boldsymbol{v}^T \rangle_{p(\boldsymbol{v},\boldsymbol{h})} \\
\frac{\partial \ln p(\boldsymbol{v})}{\partial \boldsymbol{b}} &= \sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}) \boldsymbol{v} - \sum_{\boldsymbol{v}'} \sum_{\boldsymbol{h}} p(\boldsymbol{v}',\boldsymbol{h}) \boldsymbol{v}' = \langle \boldsymbol{v} \rangle_{p(\boldsymbol{h}|\boldsymbol{v})} - \langle \boldsymbol{v} \rangle_{p(\boldsymbol{v},\boldsymbol{h})} \\
\frac{\partial \ln p(\boldsymbol{v})}{\partial \boldsymbol{c}} &= \sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}) \boldsymbol{h} - \sum_{\boldsymbol{v}'} \sum_{\boldsymbol{h}} p(\boldsymbol{v}',\boldsymbol{h}) \boldsymbol{h} = \langle \boldsymbol{h} \rangle_{p(\boldsymbol{h}|\boldsymbol{v})} - \langle \boldsymbol{h} \rangle_{p(\boldsymbol{v},\boldsymbol{h})}
\end{aligned} \tag{5.13}$$

Computing the expectation over the distribution $p(\boldsymbol{v},\boldsymbol{h})$ is difficult, but we can approximate it using Gibbs sampling [92]. Given a training vector $\boldsymbol{x}$, Gibbs sampling is given by (5.14), where $\boldsymbol{v}^{(p)}$ and $\boldsymbol{h}^{(p)}$ are the sampled visible vector and hidden vector respectively after performing p steps of Gibbs sampling, and $\boldsymbol{h}^{(p)} \sim p(\boldsymbol{h}|\boldsymbol{v}^{(p)})$ means sampling a vector $\boldsymbol{h}^{(p)}$ from the distribution with the probability $p(\boldsymbol{h}|\boldsymbol{v}^{(p)})$.

$$\begin{aligned}
\boldsymbol{v}^{(0)} &= \boldsymbol{x} \\
\boldsymbol{h}^{(0)} &\sim p(\boldsymbol{h}|\boldsymbol{v}^{(0)}) \\
\boldsymbol{v}^{(1)} &\sim p(\boldsymbol{v}|\boldsymbol{h}^{(0)}) \\
\boldsymbol{h}^{(1)} &\sim p(\boldsymbol{h}|\boldsymbol{v}^{(1)}) \\
&\vdots \\
\boldsymbol{v}^{(p+1)} &\sim p(\boldsymbol{v}|\boldsymbol{h}^{(p)}) \\
\boldsymbol{h}^{(p+1)} &\sim p(\boldsymbol{h}|\boldsymbol{v}^{(p+1)})
\end{aligned} \tag{5.14}$$

Given a training vector, the gradients of its log probability are then approximated using p steps of Gibbs sampling as given by (5.15), which is known as the contrastive divergence (CD) algorithm [48].

$$\begin{aligned}
\frac{\partial \ln p(\boldsymbol{v})}{\partial \boldsymbol{W}} &\approx \boldsymbol{h}^{(0)}\left(\boldsymbol{v}^{(0)}\right)^T - \boldsymbol{h}^{(p)}\left(\boldsymbol{v}^{(p)}\right)^T \\
\frac{\partial \ln p(\boldsymbol{v})}{\partial \boldsymbol{b}} &\approx \boldsymbol{v}^{(0)} - \boldsymbol{v}^{(p)} \\
\frac{\partial \ln p(\boldsymbol{v})}{\partial \boldsymbol{c}} &\approx \boldsymbol{h}^{(0)} - \boldsymbol{h}^{(p)}
\end{aligned} \tag{5.15}$$

Usually, 1 step of Gibbs sampling is good enough, which is used in our experiments. The complete algorithm for estimating the parameters of an RBM can be found in [92]. To prevent the randomness caused by sampling, we use the expected value of $\boldsymbol{h}$ instead of sampling a value from the distribution with the probability $p(\boldsymbol{h}|\boldsymbol{v}^{(p)})$. As $\boldsymbol{h}$ is a binary vector, its expected value is then equal to $p(\boldsymbol{h} = \boldsymbol{1}|\boldsymbol{v}^{(p)})$. This results in the gradients of the log probability as given by (5.16), where $\boldsymbol{h}^{(0)}$, $\boldsymbol{v}^{(1)}$ and $\boldsymbol{h}^{(1)}$ are vectors whose i-th elements are given by (5.17).

$$\begin{aligned}
\frac{\partial \ln p(\boldsymbol{x})}{\partial \boldsymbol{W}} &\approx \boldsymbol{h}^{(0)}\left(\boldsymbol{v}^{(0)}\right)^T - \boldsymbol{h}^{(1)}\left(\boldsymbol{v}^{(1)}\right)^T \\
\frac{\partial \ln p(\boldsymbol{x})}{\partial \boldsymbol{b}} &\approx \boldsymbol{v}^{(0)} - \boldsymbol{v}^{(1)} \\
\frac{\partial \ln p(\boldsymbol{x})}{\partial \boldsymbol{c}} &\approx \boldsymbol{h}^{(0)} - \boldsymbol{h}^{(1)}
\end{aligned} \tag{5.16}$$

where

$$\begin{aligned}
\boldsymbol{v}^{(0)} &= \boldsymbol{x} \\
h_i^{(0)} &= E_{\boldsymbol{h}|\boldsymbol{v}^{(0)}}[h_i] = p(h_i = 1|\boldsymbol{v}^{(0)}) \\
v_i^{(1)} &= E_{\boldsymbol{v}|\boldsymbol{h}^{(0)}}[v_i] = p(v_i = 1|\boldsymbol{h}^{(0)}) \\
h_i^{(1)} &= E_{\boldsymbol{h}|\boldsymbol{v}^{(1)}}[h_i] = p(h_i = 1|\boldsymbol{v}^{(1)})
\end{aligned} \tag{5.17}$$

The posterior probability $p(h_i = 1|\boldsymbol{v})$ used in (5.17) can be calculated as given by (5.18), where $\boldsymbol{W}_{i,:}$ is the i-th row of $\boldsymbol{W}$, and $\mathrm{sigm}(.)$ is the sigmoid function.

$$p(h_i = 1|\boldsymbol{v}) = \frac{p(h_i=1,\boldsymbol{v})}{p(h_i=1,\boldsymbol{v})+p(h_i=0,\boldsymbol{v})} = \frac{\exp\{-E(\boldsymbol{v},h_i=1)\}}{\exp\{-E(\boldsymbol{v},h_i=1)\}+\exp\{-E(\boldsymbol{v},h_i=0)\}} = \frac{\exp\{\boldsymbol{b}^T\boldsymbol{v}+c_i+\boldsymbol{W}_{i,:}\boldsymbol{v}\}}{\exp\{\boldsymbol{b}^T\boldsymbol{v}+c_i+\boldsymbol{W}_{i,:}\boldsymbol{v}\}+\exp\{\boldsymbol{b}^T\boldsymbol{v}\}}$$

$$= \frac{\exp\{c_i+\boldsymbol{W}_{i,:}\boldsymbol{v}\}}{\exp\{c_i+\boldsymbol{W}_{i,:}\boldsymbol{v}\}+1} = \text{sigm}(c_i + \boldsymbol{W}_{i,:}\boldsymbol{v}) \tag{5.18}$$

The posterior probability $p(v_i = 1|\boldsymbol{h})$ is calculated in the same way, as given by (5.19), where $\boldsymbol{W}_{:,i}$ is the i-th column of $\boldsymbol{W}$.

$$p(v_i = 1|\boldsymbol{h}) = \text{sigm}(b_i + \boldsymbol{h}^T\boldsymbol{W}_{:,i}) \tag{5.19}$$

Having the gradients, $\theta = \{\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{c}\}$ can then be estimated using gradient ascent to raise the log-likelihood $\ln p(\boldsymbol{v})$.

5.2.2 Gaussian RBM

Bernoulli RBM can handle binary input vectors only; however, acoustic or speech feature vectors usually have real values. In this case, we may assume that the visible unit follows a Gaussian distribution with zero mean and identity covariance, while the hidden unit still follows a Bernoulli distribution [48]. This RBM is called Gaussian RBM (GRBM), and the energy function has the form as given by (5.20) [52].

$$E(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{2}(\boldsymbol{v} - \boldsymbol{b})^T(\boldsymbol{v} - \boldsymbol{b}) - \boldsymbol{c}^T\boldsymbol{h} - \boldsymbol{h}^T\boldsymbol{W}\boldsymbol{v} \tag{5.20}$$

In GRBM, the posterior probability of the hidden unit is the same as (5.18), while the posterior probability of the visible unit is different from (5.19), which is then given by (5.21). In (5.21), Δ contains the terms independent of $\boldsymbol{v}$ which are useless in the integration, and D is the dimensionality of $\boldsymbol{v}$.

$$p(\boldsymbol{v}|\boldsymbol{h}) = \frac{p(\boldsymbol{v},\boldsymbol{h})}{p(\boldsymbol{h})} = \frac{p(\boldsymbol{v},\boldsymbol{h})}{\int p(\boldsymbol{v},\boldsymbol{h})d\boldsymbol{v}} = \frac{\exp\{-E(\boldsymbol{v},\boldsymbol{h})\}}{\int \exp\{-E(\boldsymbol{v},\boldsymbol{h})\}d\boldsymbol{v}} = \frac{\exp\left\{-\frac{1}{2}(\boldsymbol{v}-\boldsymbol{b})^T(\boldsymbol{v}-\boldsymbol{b})+\boldsymbol{c}^T\boldsymbol{h}+\boldsymbol{h}^T\boldsymbol{W}\boldsymbol{v}\right\}}{\int \exp\left\{-\frac{1}{2}(\boldsymbol{v}-\boldsymbol{b})^T(\boldsymbol{v}-\boldsymbol{b})+\boldsymbol{c}^T\boldsymbol{h}+\boldsymbol{h}^T\boldsymbol{W}\boldsymbol{v}\right\}d\boldsymbol{v}}$$

$$= \frac{\exp\left\{-\frac{1}{2}\left(\boldsymbol{v}-(\boldsymbol{b}+\boldsymbol{W}^T\boldsymbol{h})\right)^T\left(\boldsymbol{v}-(\boldsymbol{b}+\boldsymbol{W}^T\boldsymbol{h})\right)+\Delta\right\}}{\int \exp\left\{-\frac{1}{2}\left(\boldsymbol{v}-(\boldsymbol{b}+\boldsymbol{W}^T\boldsymbol{h})\right)^T\left(\boldsymbol{v}-(\boldsymbol{b}+\boldsymbol{W}^T\boldsymbol{h})\right)+\Delta\right\}d\boldsymbol{v}} = \frac{\exp\left\{-\frac{1}{2}\left(\boldsymbol{v}-(\boldsymbol{b}+\boldsymbol{W}^T\boldsymbol{h})\right)^T\left(\boldsymbol{v}-(\boldsymbol{b}+\boldsymbol{W}^T\boldsymbol{h})\right)\right\}}{\int \exp\left\{-\frac{1}{2}\left(\boldsymbol{v}-(\boldsymbol{b}+\boldsymbol{W}^T\boldsymbol{h})\right)^T\left(\boldsymbol{v}-(\boldsymbol{b}+\boldsymbol{W}^T\boldsymbol{h})\right)\right\}d\boldsymbol{v}}$$

$$= \frac{\exp\left\{-\frac{1}{2}\left(\boldsymbol{v}-\left(\boldsymbol{b}+\boldsymbol{W}^T\boldsymbol{h}\right)\right)^T\left(\boldsymbol{v}-\left(\boldsymbol{b}+\boldsymbol{W}^T\boldsymbol{h}\right)\right)\right\}}{2\pi^{D/2}} = p_g(\boldsymbol{v}|\boldsymbol{b}+\boldsymbol{W}^T\boldsymbol{h}, \boldsymbol{I}) \tag{5.21}$$

As can be seen from (5.21), $p(\boldsymbol{v}|\boldsymbol{h})$ is actually a Gaussian distribution with mean $(\boldsymbol{b}+\boldsymbol{W}^T\boldsymbol{h})$ and identity covariance. The gradients for GRBM have the same expression as (5.16), but with the visible and hidden vectors given by (5.22).

$$\begin{aligned} \boldsymbol{v}^{(0)} &= \boldsymbol{x} \\ h_i^{(0)} &= E_{\boldsymbol{h}|\boldsymbol{v}^{(0)}}[h_i] = p(h_i = 1|\boldsymbol{v}^{(0)}) \\ \boldsymbol{v}^{(1)} &= E_{\boldsymbol{v}|\boldsymbol{h}^{(0)}}[\boldsymbol{v}] = \boldsymbol{b}+\boldsymbol{W}^T\boldsymbol{h}^{(0)} \\ h_i^{(1)} &= E_{\boldsymbol{h}|\boldsymbol{v}^{(1)}}[h_i] = p(h_i = 1|\boldsymbol{v}^{(1)}) \end{aligned} \tag{5.22}$$

5.2.3 Deep Belief Net

Deep Belief Net (DBN) consists of one input layer and many hidden layers, where each pair of adjacent layers forms an RBM [50]. Suppose a DBN consists of $L+1$ layers, then the 1st layer and the 2nd layer form the 1st RBM, the 2nd layer and the 3rd layer form the 2nd RBM, and so forth. Totally there are L RBMs. The parameters of these RBMs are adjusted sequentially. That is to say, after adjusting the parameters of the l-th RBM, its parameters are fixed. Then, we start adjusting the parameters of the $(l+1)$-th RBM, whose visible units are the hidden units of the l-th RBM. If the input has real-valued data, the 1st RBM should be a GRBM, while the others are Bernoulli RBMs. A depiction of DBN is shown in Figure 5.3.

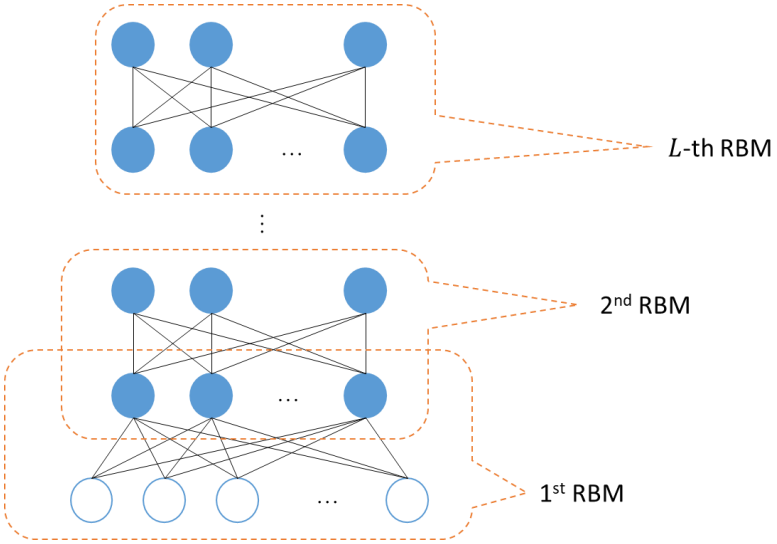


Figure 5.3 A depiction of DBN with $L+1$ layers.

After training the DBN by training a stack of RBMs one by one, a softmax layer can then be added to the last layer of DBN to form a DNN classifier. This DBN-DNN can then be fine-tuned using the cross-entropy loss function [10]. The number of neurons in the softmax layer is equal to the number of classes, meaning that the output of each neuron can represent a posterior probability of a class.

Suppose a vector $\boldsymbol{x}$ is the input to a softmax layer, whose elements are the weighted sums of the neurons' activations in the previous hidden layer. Suppose the output vector is denoted as $\boldsymbol{y}$, whose dimensionality is $K \times 1$, where K is the number of classes. Then, the softmax layer can be expressed as given by (5.23), where x_k and y_k denote the k-th element of $\boldsymbol{x}$ and $\boldsymbol{y}$ respectively.

$$y_k = \frac{\exp(x_k)}{\sum_{j=1}^{K} \exp(x_j)} \tag{5.23}$$

Given a set of training vectors used to train the DNN, denoted as $\{\boldsymbol{x}_1, \boldsymbol{x}_2 \dots \boldsymbol{x}_N\}$, where $\boldsymbol{x}_n$ is the n-th training vector. Suppose the corresponding output vectors of the softmax layer is $\{\boldsymbol{y}_1, \boldsymbol{y}_2 \dots \boldsymbol{y}_N\}$, and the corresponding labels are $\{\boldsymbol{l}_1, \boldsymbol{l}_2 \dots \boldsymbol{l}_N\}$, which are one-hot vectors with a dimensionality of $K \times 1$. That is to say, if $\boldsymbol{x}_n$ belongs to class k, only the k-th element in the label vector $\boldsymbol{l}_n$ will be 1 and the remaining will be 0. Then, the cross-entropy loss is expressed as given by (5.24), where $\ln(.)$ is an element-wise operation. As $\boldsymbol{l}_n$ is a one-hot vector, only one element in $\boldsymbol{y}_n$ will be involved in the loss function.

$$loss = -\frac{1}{N}\sum_{n=1}^{N} \boldsymbol{l}_n^T \ln(\boldsymbol{y}_n) \tag{5.24}$$

Given a testing vector, each neuron in the softmax layer of DBN-DNN produces a posterior probability as given by (5.23). The class label of the testing vector can then be predicted as the class having the largest posterior probability.

5.2.4 RBM for Classification

Apart from stacking multiple RBMs to form a discriminative model, such as the DBN-DNN, we may also use RBM as a probability estimator. Given a vector $\boldsymbol{x}$, an RBM assigns a probability $p(\boldsymbol{x})$ to it based on the free energy $F(\boldsymbol{x})$ as given by (5.25).

$$p(\boldsymbol{x}) = \frac{1}{Z}\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{x},\boldsymbol{h})} = \frac{1}{Z} e^{-F(\boldsymbol{x})} \tag{5.25}$$

For a Bernoulli RBM with the energy function given by (5.8), using the factorization trick in [92], the free energy can be expressed as (5.26), where h_j is the j-th hidden unit.

$$F(\boldsymbol{x}) = -\ln \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{x},\boldsymbol{h})} = -\boldsymbol{b}^T\boldsymbol{x} - \sum_j \ln \sum_{h_j} \exp\{h_j(c_j + \boldsymbol{W}_{j,:}\boldsymbol{x})\} \tag{5.26}$$

Noting that h_j can only have a value of 0 or 1, the free energy is further simplified to

$$F(\boldsymbol{x}) = -\boldsymbol{b}^T\boldsymbol{x} - \sum_j \ln(\exp\{(c_j + \boldsymbol{W}_{j,:}\boldsymbol{x})\} + 1) \tag{5.27}$$

Similarly, for a GRBM with the energy function given by (5.20), the free energy then becomes

$$F(\boldsymbol{x}) = \frac{1}{2}(\boldsymbol{x} - \boldsymbol{b})^T(\boldsymbol{x} - \boldsymbol{b}) - \sum_j \ln(\exp\{(c_j + \boldsymbol{W}_{j,:}\boldsymbol{x})\} + 1) \tag{5.28}$$

Although the free energy can be computed, the partition function Z is still intractable and so is the probability $p(\boldsymbol{x})$. Therefore, we may not be able to train a class-specific RBM for each class and compute the relative probability, as different RBMs will have different partition functions.

To avoid the calculation of the partition function, we may train a joint model [48]. Suppose we have a set of training vectors $\{\boldsymbol{x}_1, \boldsymbol{x}_2 \ldots \boldsymbol{x}_N\}$, where $\boldsymbol{x}_n$ is the n-th training vector having a dimensionality of $D \times 1$, and a set of corresponding label vectors $\{\boldsymbol{l}_1, \boldsymbol{l}_2 \ldots \boldsymbol{l}_N\}$, where $\boldsymbol{l}_n$ is a one-hot vector having a dimensionality of $K \times 1$. The joint model is described as follows.

In the training stage, a pair of training vector and training label $\{\boldsymbol{x}_n, \boldsymbol{l}_n\}$ are concatenated to form an augmented training vector $\tilde{\boldsymbol{x}}_n$ whose dimensionality is $(D + K) \times 1$. The augmented training vectors $\{\tilde{\boldsymbol{x}}_1, \tilde{\boldsymbol{x}}_2 \ldots \tilde{\boldsymbol{x}}_N\}$ are then used to adjust the parameters of the joint RBM. A depiction of the joint RBM is shown in Figure 5.4.

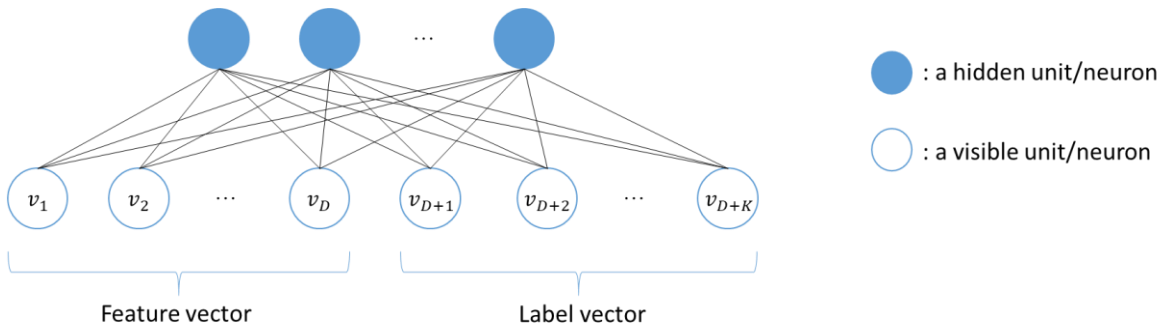


Figure 5.4 A depiction of the joint RBM.

If the training vectors have binary values, the energy function has the form given by (5.29), where $\tilde{\boldsymbol{x}}$ is the augmented vector, $\boldsymbol{x}$ is the original vector, $\boldsymbol{l}$ is its corresponding label vector, $\boldsymbol{W}^{(x)}$ is the weight matrix connecting $\boldsymbol{x}$ and $\boldsymbol{h}$ with $\boldsymbol{b}^{(x)}$ being the bias, $\boldsymbol{W}^{(l)}$ is the weight matrix connecting $\boldsymbol{l}$ and $\boldsymbol{h}$ with $\boldsymbol{b}^{(l)}$ being the bias, $\boldsymbol{c}$ is the bias for $\boldsymbol{h}$, $\widetilde{\boldsymbol{W}}$ is the concatenation of $\boldsymbol{W}^{(x)}$ and $\boldsymbol{W}^{(l)}$, and $\tilde{\boldsymbol{b}}$ is the concatenation of $\boldsymbol{b}^{(x)}$ and $\boldsymbol{b}^{(l)}$. The training procedure is then the same as training a Bernoulli RBM with parameters $\{\widetilde{\boldsymbol{W}}, \tilde{\boldsymbol{b}}, \boldsymbol{c}\}$.

$$E(\tilde{\boldsymbol{x}}, \boldsymbol{h}) = -\left(\boldsymbol{b}^{(x)}\right)^T \boldsymbol{x} - \boldsymbol{h}^T \boldsymbol{W}^{(x)} \boldsymbol{x} - \left(\boldsymbol{b}^{(l)}\right)^T \boldsymbol{l} - \boldsymbol{h}^T \boldsymbol{W}^{(l)} \boldsymbol{l} - \boldsymbol{c}^T \boldsymbol{h} = -\tilde{\boldsymbol{b}}^T \tilde{\boldsymbol{x}} - \boldsymbol{c}^T \boldsymbol{h} - \boldsymbol{h}^T \widetilde{\boldsymbol{W}} \tilde{\boldsymbol{x}} \quad (5.29)$$

where

$$\tilde{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{l} \end{bmatrix}, \ \widetilde{\boldsymbol{W}} = [\boldsymbol{W}^{(x)} \quad \boldsymbol{W}^{(l)}], \ \tilde{\boldsymbol{b}} = \begin{bmatrix} \boldsymbol{b}^{(x)} \\ \boldsymbol{b}^{(l)} \end{bmatrix} \quad (5.30)$$

If the training vectors have real values, the energy function should have the form as given by (5.31), which is the fusion of a Gaussian RBM and a Bernoulli RBM.

$$E(\tilde{\boldsymbol{x}}, \boldsymbol{h}) = \frac{1}{2}\left(\boldsymbol{x} - \boldsymbol{b}^{(x)}\right)^T \left(\boldsymbol{x} - \boldsymbol{b}^{(x)}\right) - \boldsymbol{h}^T \boldsymbol{W}^{(x)} \boldsymbol{x} - \left(\boldsymbol{b}^{(l)}\right)^T \boldsymbol{l} - \boldsymbol{h}^T \boldsymbol{W}^{(l)} \boldsymbol{l} - \boldsymbol{c}^T \boldsymbol{h}$$

$$= \frac{1}{2}\left(\boldsymbol{x} - \boldsymbol{b}^{(x)}\right)^T \left(\boldsymbol{x} - \boldsymbol{b}^{(x)}\right) - \left(\boldsymbol{b}^{(l)}\right)^T \boldsymbol{l} - \boldsymbol{c}^T \boldsymbol{h} - \boldsymbol{h}^T \widetilde{\boldsymbol{W}} \tilde{\boldsymbol{x}} \quad (5.31)$$

The gradients for training this heterogeneous RBM have the same expression as (5.16), but with the visible and hidden vectors given by

$$\begin{gathered} \boldsymbol{v}^{(0)} = \tilde{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{x}^{(0)} \\ \boldsymbol{l}^{(0)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{l} \end{bmatrix} \\ h_i^{(0)} = E_{\boldsymbol{h}|\boldsymbol{v}^{(0)}}[h_i] = p\left(h_i = 1 \middle| \boldsymbol{v}^{(0)}\right) = \text{sigm}\left(c_i + \widetilde{\boldsymbol{W}}_{i,:} \boldsymbol{v}^{(0)}\right) \\ \boldsymbol{v}^{(1)} = \begin{bmatrix} \boldsymbol{x}^{(1)} \\ \boldsymbol{l}^{(1)} \end{bmatrix} = \begin{bmatrix} E_{\boldsymbol{x}|\boldsymbol{h}^{(0)}}[\boldsymbol{x}] \\ E_{\boldsymbol{l}|\boldsymbol{h}^{(0)}}[\boldsymbol{l}] \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}^{(x)} + \left(\boldsymbol{W}^{(x)}\right)^T \boldsymbol{h}^{(0)} \\ p(\boldsymbol{l} = \boldsymbol{1} | \boldsymbol{h}^{(0)}) \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}^{(x)} + \left(\boldsymbol{W}^{(x)}\right)^T \boldsymbol{h}^{(0)} \\ p(l_1 = 1 | \boldsymbol{h}^{(0)}) \\ \vdots \\ p(l_K = 1 | \boldsymbol{h}^{(0)}) \end{bmatrix} \\ h_i^{(1)} = E_{\boldsymbol{h}|\boldsymbol{v}^{(1)}}[h_i] = p\left(h_i = 1 \middle| \boldsymbol{v}^{(1)}\right) = \text{sigm}\left(c_i + \widetilde{\boldsymbol{W}}_{i,:} \boldsymbol{v}^{(1)}\right) \end{gathered} \quad (5.32)$$

where

$$p(l_k = 1 | \boldsymbol{h}) = \text{sigm}\left(b_k^{(l)} + \boldsymbol{h}^T \boldsymbol{W}_{:,k}^{(l)}\right) \quad (5.33)$$

In the testing stage, a testing vector $\boldsymbol{y}$ is concatenated with trial label vectors $\{\boldsymbol{\ell}_1, \boldsymbol{\ell}_2 \dots \boldsymbol{\ell}_K\}$, where $\boldsymbol{\ell}_k$ has a dimensionality of $K \times 1$, and the k-th element is 1 while other elements are 0. Denote the concatenated testing vectors as $\{\tilde{\boldsymbol{y}}_1, \tilde{\boldsymbol{y}}_2 \dots \tilde{\boldsymbol{y}}_K\}$, whose dimensionality is $(D+K) \times 1$. Using the factorization trick in [92], the free energy of $\tilde{\boldsymbol{y}}_k$ is then given by (5.34), where $\Delta(\boldsymbol{y})$ is a term only related to $\boldsymbol{y}$ but not $\boldsymbol{\ell}_k$ and $\boldsymbol{h}$.

$$F(\tilde{\boldsymbol{y}}_k) = -\ln \sum_{\boldsymbol{h}} e^{-E(\tilde{\boldsymbol{y}}_k, \boldsymbol{h})} = \Delta(\boldsymbol{y}) - \left(\boldsymbol{b}^{(l)}\right)^T \boldsymbol{\ell}_k - \sum_j \ln\left(\exp\left\{\left(c_j + \widetilde{\boldsymbol{W}}_{j,:} \tilde{\boldsymbol{y}}_k\right)\right\} + 1\right) \quad (5.34)$$

We can then predict the label of $\boldsymbol{y}$ by finding the largest posterior probability or the lowest free energy among $\{\tilde{\boldsymbol{y}}_1, \tilde{\boldsymbol{y}}_2 \dots \tilde{\boldsymbol{y}}_K\}$ as given by (5.35).

$$\ell(\boldsymbol{y}) = \underset{k}{\operatorname{argmax}}\, p(k|\boldsymbol{y}) = \underset{k}{\operatorname{argmax}} \frac{p(\tilde{\boldsymbol{y}}_k)}{\sum_{j=1}^{K} p(\tilde{\boldsymbol{y}}_j)} = \underset{k}{\operatorname{argmax}} \frac{\frac{1}{Z} e^{-F(\tilde{\boldsymbol{y}}_k)}}{\sum_{j=1}^{K} \frac{1}{Z} e^{-F(\tilde{\boldsymbol{y}}_j)}} = \underset{k}{\operatorname{argmax}}\, e^{-F(\tilde{\boldsymbol{y}}_k)}$$

$$= \underset{k}{\operatorname{argmin}}\, F(\tilde{\boldsymbol{y}}_k) = \underset{k}{\operatorname{argmin}} \left\{-\left(\boldsymbol{b}^{(l)}\right)^T \boldsymbol{\ell}_k - \sum_j \ln\left(\exp\left\{\left(c_j + \widetilde{\boldsymbol{W}}_{j,:} \tilde{\boldsymbol{y}}_k\right)\right\} + 1\right)\right\} \quad (5.35)$$

5.3 Comparison between GMM and RBM

To effectively employ GMM as a classifier, the feature vectors are supposed to be independent of each other and follow the Gaussian distribution. To estimate a Gaussian distribution from a set of training vectors, the dimensionality of the vectors should be considerably smaller than the number of vectors; otherwise, the covariance will not be accurately estimated. Most of the time, it is assumed that the covariance of each Gaussian component in a GMM is diagonal, which implicitly assumes that the elements inside a feature vector should be independent of each other, i.e., the elements should be decorrelated. In addition, as can be seen from the EM algorithm given in Algorithm 5.1, the model parameters of a GMM capture the inter-feature relationship (i.e., the relationship between the elements in the same dimension of different feature vectors), instead of the intra-feature relationship (i.e., the relationship between the elements in different dimensions of a feature vector). Therefore, GMM can capture the global characteristics across different feature vectors.

To effectively employ RBM as a classifier, the feature vectors are expected to have a high dimensionality such that enough information is stored in a feature vector. As can be seen from the

energy function of RBM given by (5.8), the model assumes that the variables in the visible layer are related by a weight matrix, meaning that the elements inside a feature vector should be correlated. This is contrary to the implicit assumption of GMM that the elements inside a feature vector should be decorrelated. From this perspective, RBM actually captures the intra-feature relationship instead of the inter-feature relationship. For acoustic or speech signals, the input to RBM can be a sequence of consecutive feature vectors. In this way, RBM may capture the local characteristics of the signal. By increasing the length of the sequence, it may capture more characteristics, but they are still local characteristics as it cannot capture the characteristics across different samples. Similar analyses also apply to DBN-DNN, as DBN is trained by stacking multiple RBMs.

In summary, GMM can handle low-dimensional decorrelated features, whereas RBM can handle high-dimensional correlated features. GMM can capture global characteristics, whereas RBM can capture local characteristics. From this perspective, GMM and RBM seem complementary to each other, and the ways of combing them, such as using RBM to extract features so that the local characteristics are embedded in the features, and using GMM for classification purposes so that the global characteristics are taken into consideration, may produce some interesting results.

5.4 Support Vector Machine

This section starts by briefly introducing the basic formulation of support vector machine (SVM), which is a mature classification model having a wide range of applications [53]. Originally SVM is designed for doing binary classification, but it can be easily extended to handle multi-class cases by forming multiple binary SVMs. By introducing a weight parameter, SVM can be extended to weighted SVM (WSVM), where different training data can be unequally weighted to emphasize their importance or relevance to the model.

5.4.1 Binary SVM

A two-class SVM classifier is a maximum margin classifier [53]. Given a training set consisting of data belonging to two classes, denoted as the positive class and the negative class, SVM aims at finding a separating hyperplane that separates the positive data and the negative data into different sides of the

hyperplane. The hyperplane is obtained by maximizing the margin, which is defined as the smallest distance between the data and the hyperplane [53].

Given a set of training vectors $\{\boldsymbol{x}_1, \boldsymbol{x}_2 \dots \boldsymbol{x}_N\}$, and the corresponding training labels $\{l_1, l_2 \dots l_N\}$ where $l_n \in \{-1, +1\}$ indicates whether the training vector $\boldsymbol{x}_n$ belongs to the positive class or the negative class. Then the optimization problem of SVM can be formulated as given by (5.36), where $\{\boldsymbol{w}, b\}$ are the parameters of the hyperplane represented by $\boldsymbol{w}^T\boldsymbol{x}_n + b = 0$. A depiction of SVM is shown in Figure 5.5.

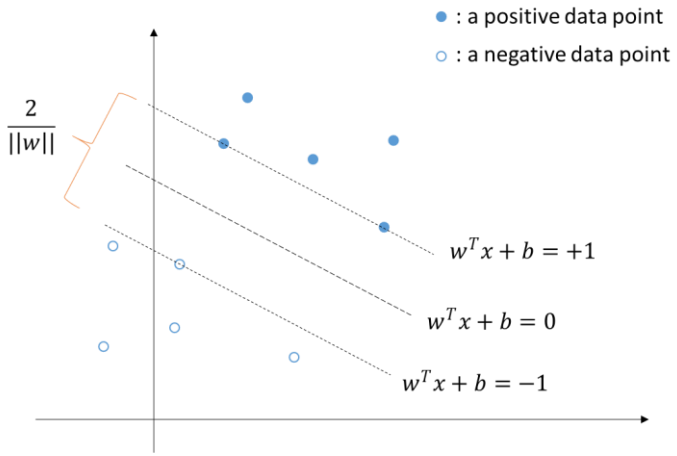


Figure 5.5 A depiction of SVM with 2-dimensional data.

$$\begin{aligned} &\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 \\ &s.t. \\ &\boldsymbol{w}^T\boldsymbol{x}_n + b \geq +1 \quad for\ l_n = +1 \\ &\boldsymbol{w}^T\boldsymbol{x}_n + b \leq -1 \quad for\ l_n = -1 \end{aligned} \tag{5.36}$$

With the fact that l_n can only take the value of $+1$ or -1, the optimization problem in (5.36) is simplified to

$$\begin{aligned} &\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 \\ &s.t. \\ &l_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) - 1 \geq 0 \quad for\ n = 1,2 \dots N \end{aligned} \tag{5.37}$$

After training an SVM, for a testing vector $\boldsymbol{y}$, the class label can be predicted according to the sign of $(\boldsymbol{w}^T\boldsymbol{y} + b)$. If $\boldsymbol{w}^T\boldsymbol{y} + b > 0$, $\boldsymbol{y}$ belongs to the positive class; if $\boldsymbol{w}^T\boldsymbol{y} + b < 0$, $\boldsymbol{y}$ belongs to the negative class.

The optimization problem given by (5.37) can also be reformulated using the Lagrangian function $L(\boldsymbol{w}, b, \lambda)$ as given by (5.38), where $\{\lambda_1, \lambda_2 \ldots \lambda_N\}$ are Lagrange multipliers and should satisfy the nonnegativity constraints, i.e., $\lambda_n \geq 0$ for $n = 1,2 \ldots N$.

$$L(\boldsymbol{w}, b, \lambda) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{n=1}^{N} \lambda_n (l_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) - 1) \tag{5.38}$$

The solution to the optimization problem in (5.37) can then be obtained by minimizing the Lagrangian function in (5.38) with respect to $\{\boldsymbol{w}, b\}$ subject to the nonnegativity constraints of λ_n. As the optimization problem defined by (5.38) and the corresponding constraints form a convex quadratic programming problem, its solution is equivalent to that obtained by maximizing the dual formulation $\tilde{L}(\lambda)$ with respect to λ_n as given by (5.39).

$$\begin{gathered} \max_{\lambda} \tilde{L}(\lambda) = \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m l_n l_m \boldsymbol{x}_n^T \boldsymbol{x}_m \\ s.t. \\ \lambda_n \geq 0 \quad for \ n = 1,2 \ldots N \\ \sum_{n=1}^{N} \lambda_n \, l_n = 0 \end{gathered} \tag{5.39}$$

In the case where the training data are inseparable by the hyperplane, the inequality constraints in (5.37) can be relaxed by adding a slack variable ξ_n for each $\boldsymbol{x}_n$. Then, the optimization problem becomes (5.40), where C is a pre-defined constant parameter. When $C \to \infty$, the optimization process will force $\xi_n \to 0$, (5.40) is then equivalent to (5.37).

$$\begin{gathered} \min_{\boldsymbol{w}, b, \xi} \frac{1}{2}\|\boldsymbol{w}\|^2 + C \sum_{n=1}^{N} \xi_n \\ s.t. \\ \xi_n \geq 0 \quad for \ n = 1,2 \ldots N \\ l_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) \geq 1 - \xi_n \quad for \ n = 1,2 \ldots N \end{gathered} \tag{5.40}$$

The optimization problem for the dual formulation then becomes

$$\begin{gathered} \max_{\lambda} \tilde{L}(\lambda) = \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m l_n l_m \boldsymbol{x}_n^T \boldsymbol{x}_m \\ s.t. \\ 0 \leq \lambda_n \leq C \quad for \ n = 1,2 \ldots N \\ \sum_{n=1}^{N} \lambda_n \, l_n = 0 \end{gathered} \tag{5.41}$$

Solving (5.41) yields the expression of $\boldsymbol{w}$ as given by (5.42).

$$\boldsymbol{w} = \sum_{n=1}^{N} \lambda_n l_n \boldsymbol{x}_n \tag{5.42}$$

Substituting $\boldsymbol{w}$ in the decision boundary $\boldsymbol{w}^T\boldsymbol{y} + b = 0$ gives

$$\sum_{n=1}^{N} \lambda_n l_n \boldsymbol{x}_n^T \boldsymbol{y} + b = 0 \tag{5.43}$$

The advantage of using the dual formulation is that the objective function given by (5.39) or (5.41) only involves the inner product of the training vectors, and the decision boundary $\boldsymbol{w}^T\boldsymbol{y} + b = 0$ only involves the inner product of the training vector and the testing vector. This makes it possible to apply the kernel trick.

5.4.2 Multi-class SVM

The original SVM is a binary classifier serving two-class classification tasks. It is viable to extend SVM to tackle multi-class classification tasks by constructing several binary SVMs. There are two common ways to construct a set of binary SVMs for multi-class purposes: one being the one-versus-one strategy, and the other being the one-versus-rest strategy [53][95].

Given a set of training vectors $\{\boldsymbol{x}_1, \boldsymbol{x}_2 \dots \boldsymbol{x}_N\}$ belonging to K different classes with the corresponding training labels $\{l_1, l_2 \dots l_N\}$, where $l_n \in \{1,2 \dots K\}$. On using the one-versus-one strategy, totally $K(K-1)/2$ binary SVMs are constructed, where the ij-th SVM aims at finding the hyperplane separating the training data of class i from those of class j by solving the optimization problem given by (5.44). The superscript ij indicates that $\{\boldsymbol{w}^{(ij)}, b^{(ij)}\}$ are the parameters of the hyperplane that separates class i from class j [95].

$$\begin{gathered}
\min_{\boldsymbol{w}^{(ij)}, b^{(ij)}, \xi^{(ij)}} \ \frac{1}{2}\left\|\boldsymbol{w}^{(ij)}\right\|^2 + C\sum_{n, l_n = i \text{ or } j} \xi_n^{(ij)} \\
s.t. \\
\xi_n^{(ij)} \geq 0 \\
\left(\boldsymbol{w}^{(ij)}\right)^T \boldsymbol{x}_n + b^{(ij)} \geq 1 - \xi_n^{(ij)} \quad for \ l_n = i \\
\left(\boldsymbol{w}^{(ij)}\right)^T \boldsymbol{x}_n + b^{(ij)} \leq -1 + \xi_n^{(ij)} \quad for \ l_n = j
\end{gathered} \tag{5.44}$$

On using the one-versus-rest strategy, totally K binary SVMs are constructed, where the k-th SVM aims at finding the hyperplane separating the training data of class k from those of the remaining $(K-1)$ classes. By this means, the k-th SVM solves the optimization problem as given by (5.45) [95].

$$\begin{gathered}\min_{\boldsymbol{w}^{(k)},b^{(k)},\xi^{(k)}} \ \frac{1}{2}\left\|\boldsymbol{w}^{(k)}\right\|^2 + C\sum_{n=1}^{N}\xi_n^{(k)} \\ s.t. \\ \xi_n^{(k)} \geq 0 \\ \left(\boldsymbol{w}^{(k)}\right)^T \boldsymbol{x}_n + b^{(k)} \geq 1 - \xi_n^{(k)} \quad for \ l_n = k \\ \left(\boldsymbol{w}^{(k)}\right)^T \boldsymbol{x}_n + b^{(k)} \leq -1 + \xi_n^{(k)} \quad for \ l_n \neq k\end{gathered} \tag{5.45}$$

On using the sequential minimal optimization (SMO) algorithm to find the solution of SVM, the computation time is approximately proportional to the number of training data [53]. Suppose the number of training data in each class is N/K, then the time complexity of the one-versus-one strategy is approximately given by (5.46), where γ is the proportional factor, $\frac{2N}{K}$ is the number of training data used to train each binary SVM, $\frac{K(K-1)}{2}$ is the total number of binary SVMs, and $O(.)$ is the big-O notation upper bounding the running time of an algorithm.

$$t_{one-one} = \gamma \times \frac{2N}{K} \times \frac{K(K-1)}{2} = \gamma N(K-1) = O(NK) \tag{5.46}$$

The time complexity of the one-versus-rest strategy can be approximated in a similar way as given by (5.47). Each binary SVM will be trained with all the training data, and there will be totally *K* binary SVMs to be trained.

$$t_{one-rest} = \gamma \times N \times K = O(NK) \tag{5.47}$$

Using the aforementioned approximation, the two strategies seem to have similar time complexities. Nevertheless, the comprehensive experimental results in [95] show that the former one always consumes less computation time while keeping the performance to be almost the same.

5.4.3 Weighted SVM

During the modeling process, it is viable to introduce a weighting coefficient for each training datum to indicate its importance or relevance to the model. The resulting model is called the weighted support vector machine (WSVM) [96]. For a binary WSVM, the optimization problem becomes (5.48), where W_n is the weighting coefficient for the n-th training datum.

$$\begin{gathered}\min_{\boldsymbol{w},b,\xi} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{n=1}^{N} W_n \xi_n \\ s.t. \\ \xi_n \geq 0 \quad for\ n = 1,2 \dots N \\ l_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) \geq 1 - \xi_n \quad for\ n = 1,2 \dots N\end{gathered} \tag{5.48}$$

It has been shown in [96] that WSVM performs significantly better than SVM when there exist outliers in the training set. However, in general, it may be challenging to determine the weighting coefficients.

5.5 Probabilistic Linear Discriminant Analysis

This section discusses the basic formulation of the probabilistic linear discriminant analysis (PLDA), including the formulation for model parameter estimation and the formulation for class label prediction. PLDA has been the state-of-the-art backend for speaker verification studies; however, the original formulas are inefficient in handling a large number of training data. To address this scalability issue, scalable formulas for model parameter estimation have been proposed in [78][79]. In this section, we propose the scalable formulas for class label prediction, which completes the scalable PLDA that can deal with a large number of training data.

5.5.1 Fundamentals of PLDA

Suppose we have a set of training data belonging to K different classes, and in each class, there are J training vectors. Totally there are KJ training vectors. This set of training data can be denoted as $\{\boldsymbol{x}_{11}, \boldsymbol{x}_{12} \dots \boldsymbol{x}_{1J}, \boldsymbol{x}_{21}, \boldsymbol{x}_{22} \dots \boldsymbol{x}_{2J} \dots \dots \boldsymbol{x}_{K1}, \boldsymbol{x}_{K2} \dots \boldsymbol{x}_{KJ}\}$, where $\boldsymbol{x}_{kj}$ represents the j-th training vector in the k-th class. In a PLDA model, we assume that a vector $\boldsymbol{x}_{kj}$ is generated by two latent vectors as given by (5.49), where $\boldsymbol{\mu}$ is the mean vector, $\boldsymbol{h}_k$ is the between-class latent vector, $\boldsymbol{w}_{kj}$ is the within-class latent vector, $\boldsymbol{\varepsilon}_{kj}$ is the noise term, $\boldsymbol{F}$ and $\boldsymbol{G}$ are factor-loading matrices corresponding to $\boldsymbol{h}_k$ and $\boldsymbol{w}_{kj}$ respectively [97]. The between-class latent vector $\boldsymbol{h}_k$ reflects the class information, while the within-class latent vector $\boldsymbol{w}_{kj}$ reflects the variation of the vector $\boldsymbol{x}_{kj}$ among all the vectors in class k. Therefore, the between-class latent vectors are supposed to be the same for all the training vectors in the same class, while the within-class latent vectors can be different even if the training vectors are in the same class.

$$\boldsymbol{x}_{kj} = \boldsymbol{\mu} + \boldsymbol{F}\boldsymbol{h}_k + \boldsymbol{G}\boldsymbol{w}_{kj} + \boldsymbol{\varepsilon}_{kj} \tag{5.49}$$

A PLDA model also follows the assumption of a standard FA model, namely, both $\boldsymbol{h}_k$ and $\boldsymbol{w}_{kj}$ should follow a Gaussian distribution with zero mean and identity covariance, and the noise vector $\boldsymbol{\varepsilon}_{kj}$ should follow a Gaussian distribution with zero mean and diagonal covariance $\boldsymbol{\Sigma}$, as given by (5.50).

$$\begin{aligned} \boldsymbol{h}_k &\sim p_g(\boldsymbol{h}_k|\boldsymbol{0}, \boldsymbol{I}) \\ \boldsymbol{w}_{kj} &\sim p_g(\boldsymbol{w}_{kj}|\boldsymbol{0}, \boldsymbol{I}) \\ \boldsymbol{\varepsilon}_{kj} &\sim p_g(\boldsymbol{\varepsilon}_{kj}|\boldsymbol{0}, \boldsymbol{\Sigma}) \end{aligned} \tag{5.50}$$

With the above model assumptions, $\boldsymbol{x}_{kj}$ should also follow a Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance ($\boldsymbol{F}\boldsymbol{F}^T + \boldsymbol{G}\boldsymbol{G}^T + \boldsymbol{\Sigma}$) as given by the probability density function in (5.51). So, a PLDA model can be parameterized by $\theta = \{\boldsymbol{\mu}, \boldsymbol{F}, \boldsymbol{G}, \boldsymbol{\Sigma}\}$.

$$p(\boldsymbol{x}_{kj}) = p_g(\boldsymbol{x}_{kj}|\boldsymbol{\mu}, \boldsymbol{F}\boldsymbol{F}^T + \boldsymbol{G}\boldsymbol{G}^T + \boldsymbol{\Sigma}) \tag{5.51}$$

As all the training vectors in class k should have the same between-class latent vector $\boldsymbol{h}_k$, in order to estimate the factor-loading matrix $\boldsymbol{F}$, all the training vectors $\{\boldsymbol{x}_{k1}, \boldsymbol{x}_{k2} \ldots \boldsymbol{x}_{kJ}\}$ need to be considered as a whole [97]. The expression in (5.49) then becomes

$$\begin{bmatrix} \boldsymbol{x}_{k1} \\ \boldsymbol{x}_{k2} \\ \vdots \\ \boldsymbol{x}_{kJ} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu} \\ \vdots \\ \boldsymbol{\mu} \end{bmatrix} + \begin{bmatrix} \boldsymbol{F} & \boldsymbol{G} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{F} & \boldsymbol{0} & \boldsymbol{G} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{F} & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{G} \end{bmatrix} \begin{bmatrix} \boldsymbol{h}_k \\ \boldsymbol{w}_{k1} \\ \boldsymbol{w}_{k2} \\ \vdots \\ \boldsymbol{w}_{kJ} \end{bmatrix} + \begin{bmatrix} \boldsymbol{\varepsilon}_{k1} \\ \boldsymbol{\varepsilon}_{k2} \\ \vdots \\ \boldsymbol{\varepsilon}_{kJ} \end{bmatrix} \tag{5.52}$$

Equation (5.52) can be simplified to (5.53) by using the simplified notations given by (5.54).

$$\boldsymbol{X}_k = \boldsymbol{U} + \boldsymbol{R}\boldsymbol{Y}_k + \boldsymbol{\xi}_k \tag{5.53}$$

where

$$\boldsymbol{X}_k = \begin{bmatrix} \boldsymbol{x}_{k1} \\ \boldsymbol{x}_{k2} \\ \vdots \\ \boldsymbol{x}_{kJ} \end{bmatrix},\ \boldsymbol{U} = \begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu} \\ \vdots \\ \boldsymbol{\mu} \end{bmatrix},\ \boldsymbol{R} = \begin{bmatrix} \boldsymbol{F} & \boldsymbol{G} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{F} & \boldsymbol{0} & \boldsymbol{G} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{F} & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{G} \end{bmatrix},\ \boldsymbol{Y}_k = \begin{bmatrix} \boldsymbol{h}_k \\ \boldsymbol{w}_{k1} \\ \boldsymbol{w}_{k2} \\ \vdots \\ \boldsymbol{w}_{kJ} \end{bmatrix},\ \boldsymbol{\xi}_k = \begin{bmatrix} \boldsymbol{\varepsilon}_{k1} \\ \boldsymbol{\varepsilon}_{k2} \\ \vdots \\ \boldsymbol{\varepsilon}_{kJ} \end{bmatrix} \tag{5.54}$$

As the noise vectors are independent of each other, $\boldsymbol{\xi}_k$ will follow a Gaussian distribution with zero mean and diagonal covariance $\boldsymbol{\Psi}$, where $\boldsymbol{\Psi}$ is given by (5.55).

$$\boldsymbol{\Psi} = \begin{bmatrix} \boldsymbol{\Sigma} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{\Sigma} \end{bmatrix} \tag{5.55}$$

As $\boldsymbol{Y}_k$ and $\boldsymbol{\xi}_k$ are assumed to be independent of each other, $\boldsymbol{X}_k$ should also follow a Gaussian distribution with mean $\boldsymbol{U}$ and covariance $(\boldsymbol{R}\boldsymbol{R}^T + \boldsymbol{\Psi})$ as given by (5.56), which is the joint distribution of $\{\boldsymbol{x}_{k1}, \boldsymbol{x}_{k2} \ldots \boldsymbol{x}_{kJ}\}$ [97][98].

$$p(\boldsymbol{X}_k) = p\left(\boldsymbol{x}_{k1}, \boldsymbol{x}_{k2} \ldots \boldsymbol{x}_{kJ}\right) = p_g(\boldsymbol{X}_k | \boldsymbol{U}, \boldsymbol{R}\boldsymbol{R}^T + \boldsymbol{\Psi}) \tag{5.56}$$

According to (5.56), a PLDA model can also be parameterized by $\theta = \{\boldsymbol{U}, \boldsymbol{R}, \boldsymbol{\Psi}\}$, which is the parameters of a standard FA model. As a matter of fact, PLDA has a close relationship with FA and Fisher linear discriminant analysis (LDA). If $\boldsymbol{h}_k$ in (5.49) disappears, it then becomes an FA model, which is an unsupervised model carrying no class information. From this perspective, PLDA is the generalization of FA. On the other hand, the between-class factor-loading matrix $\boldsymbol{F}$ plays a similar role to the between-class scatter matrix of LDA, and the within-class factor-loading matrix $\boldsymbol{G}$ plays a similar role to the within-class scatter matrix of LDA [97]. From this perspective, PLDA resembles LDA.

5.5.2 Original Formulation

5.5.2.1 Parameter Estimation

As the expression of a PLDA model can be formulated into the form of a standard FA, the model parameters $\theta = \{\boldsymbol{U}, \boldsymbol{R}, \boldsymbol{\Psi}\}$ can be estimated in the same way as that of FA by using the EM algorithm [97][87]. In the E-step, the model parameters are kept unchanged, and the posterior expectations are calculated using the current parameters. The posterior expected mean and the posterior expected covariance of $\boldsymbol{Y}_k$ are given by (5.57) and (5.58), respectively.

$$E[\boldsymbol{Y}_k] = (\boldsymbol{I} + \boldsymbol{R}^T\boldsymbol{\Psi}^{-1}\boldsymbol{R})^{-1}\boldsymbol{R}^T\boldsymbol{\Psi}^{-1}(\boldsymbol{X}_k - \boldsymbol{U}) \tag{5.57}$$

$$E\left[\boldsymbol{Y}_k\boldsymbol{Y}_k^T\right] = (\boldsymbol{I} + \boldsymbol{R}^T\boldsymbol{\Psi}^{-1}\boldsymbol{R})^{-1} + E[\boldsymbol{Y}_k]E[\boldsymbol{Y}_k]^T \tag{5.58}$$

In the M-step, the posterior expectations are kept unchanged, and the model parameters are re-estimated using the posterior expectations. To ease the computation, we may first reformulate (5.49) into (5.59), which consists of a concatenated matrix $\boldsymbol{V}$ and a concatenated latent vector $\boldsymbol{z}_{kj}$.

$$\boldsymbol{x}_{kj} = \boldsymbol{\mu} + \boldsymbol{V}\boldsymbol{z}_{kj} + \boldsymbol{\varepsilon}_{kj} \tag{5.59}$$

where

$$\boldsymbol{V} = [\boldsymbol{F} \quad \boldsymbol{G}],\ \boldsymbol{z}_{kj} = \begin{bmatrix}\boldsymbol{h}_k \\ \boldsymbol{w}_{kj}\end{bmatrix} \tag{5.60}$$

We then re-estimate the parameters $\{\boldsymbol{\mu}, \boldsymbol{V}, \boldsymbol{\Sigma}\}$ using (5.61) ~ (5.63), where diag{.} sets all the non-diagonal elements in a matrix to zero [97]. The posterior expectations $E[\boldsymbol{z}_{kj}]$ and $E[\boldsymbol{z}_{kj}\boldsymbol{z}_{kj}^T]$ can be obtained from the posterior expectations $E[\boldsymbol{Y}_k]$ and $E\left[\boldsymbol{Y}_k\boldsymbol{Y}_k^T\right]$ by comparing the expressions of $\boldsymbol{z}_{kj}$ and $\boldsymbol{Y}_k$ given by (5.60) and (5.54) respectively. We can see that $E[\boldsymbol{z}_{kj}]$ is just a sub-vector of $E[\boldsymbol{Y}_k]$, and $E[\boldsymbol{z}_{kj}\boldsymbol{z}_{kj}^T]$ is a sub-matrix of $E\left[\boldsymbol{Y}_k\boldsymbol{Y}_k^T\right]$. The parameters $\{\boldsymbol{\mu}, \boldsymbol{V}, \boldsymbol{\Sigma}\}$ can then be used to reconstruct $\{\boldsymbol{U}, \boldsymbol{R}, \boldsymbol{\Psi}\}$ for the E-step.

$$\boldsymbol{\mu} = \frac{1}{KJ}\sum_{k=1}^{K}\sum_{j=1}^{J}\boldsymbol{x}_{kj} \tag{5.61}$$

$$\boldsymbol{V} = \left(\sum_{k=1}^{K}\sum_{j=1}^{J}(\boldsymbol{x}_{kj} - \boldsymbol{\mu})E[\boldsymbol{z}_{kj}]^T\right)\left(\sum_{k=1}^{K}\sum_{j=1}^{J}E[\boldsymbol{z}_{kj}\boldsymbol{z}_{kj}^T]\right)^{-1} \tag{5.62}$$

$$\boldsymbol{\Sigma} = \frac{1}{KJ}\sum_{k=1}^{K}\sum_{j=1}^{J}\text{diag}\left\{(\boldsymbol{x}_{kj} - \boldsymbol{\mu})(\boldsymbol{x}_{kj} - \boldsymbol{\mu})^T - \boldsymbol{V}E[\boldsymbol{z}_{kj}](\boldsymbol{x}_{kj} - \boldsymbol{\mu})^T\right\} \tag{5.63}$$

The EM algorithm for the original formulation is summarized in Algorithm 5.4, where N is the total number of EM iterations, and the superscript n indicates the number of EM iterations having been performed. The mean parameter $\boldsymbol{\mu}$ is the mean of all the training vectors and needs to be computed for only once. The factor-loading matrices $\boldsymbol{F}$ and $\boldsymbol{G}$, and the noise covariance $\boldsymbol{\Sigma}$, are initialized to have all ones on the principal diagonal and all zeros on other positions.

Algorithm 5.4 EM algorithm for the original formulation of PLDA

1:	Initialization:	$\theta^{(0)} = \{\boldsymbol{\mu}, \boldsymbol{F}^{(0)}, \boldsymbol{G}^{(0)}, \boldsymbol{\Sigma}^{(0)}\}$
2:	Compute mean:	$\boldsymbol{\mu} = \frac{1}{KJ}\sum_{k=1}^{K}\sum_{j=1}^{J}\boldsymbol{x}_{kj}$
3:	**For** n = 1 to N	
4:		Form $\{\boldsymbol{U}, \boldsymbol{R}, \boldsymbol{\Psi}\}$ from $\{\boldsymbol{\mu}, \boldsymbol{F}^{(n-1)}, \boldsymbol{G}^{(n-1)}, \boldsymbol{\Sigma}^{(n-1)}\}$
5:	E-step:	$E^{(n)}[\boldsymbol{Y}_k] = (\boldsymbol{I} + \boldsymbol{R}^T\boldsymbol{\Psi}^{-1}\boldsymbol{R})^{-1}\boldsymbol{R}^T\boldsymbol{\Psi}^{-1}(\boldsymbol{X}_k - \boldsymbol{U})$
6:		$E^{(n)}[\boldsymbol{Y}_k\boldsymbol{Y}_k^T] = (\boldsymbol{I} + \boldsymbol{R}^T\boldsymbol{\Psi}^{-1}\boldsymbol{R})^{-1} + E^{(n)}[\boldsymbol{Y}_k]E^{(n)}[\boldsymbol{Y}_k]^T$
7:		Obtain $E^{(n)}[\boldsymbol{z}_{kj}]$ and $E^{(n)}[\boldsymbol{z}_{kj}\boldsymbol{z}_{kj}^T]$ from $E^{(n)}[\boldsymbol{Y}_k]$ and $E^{(n)}[\boldsymbol{Y}_k\boldsymbol{Y}_k^T]$
8:		$\boldsymbol{V}^{(n)} = \left(\sum_{k=1}^{K}\sum_{j=1}^{J}(\boldsymbol{x}_{kj} - \boldsymbol{\mu})E^{(n)}[\boldsymbol{z}_{kj}]^T\right)\left(\sum_{k=1}^{K}\sum_{j=1}^{J}E^{(n)}[\boldsymbol{z}_{kj}\boldsymbol{z}_{kj}^T]\right)^{-1}$
9:	M-step:	$\boldsymbol{\Sigma}^{(n)} = \frac{1}{KJ}\sum_{k=1}^{K}\sum_{j=1}^{J}\mathrm{diag}\left\{(\boldsymbol{x}_{kj} - \boldsymbol{\mu})(\boldsymbol{x}_{kj} - \boldsymbol{\mu})^T - \boldsymbol{V}^{(n)}E^{(n)}[\boldsymbol{z}_{kj}](\boldsymbol{x}_{kj} - \boldsymbol{\mu})^T\right\}$
10:		$[\boldsymbol{F}^{(n)} \quad \boldsymbol{G}^{(n)}] = \boldsymbol{V}^{(n)}$
11:	Update θ:	$\theta^{(n)} = \{\boldsymbol{\mu}, \boldsymbol{F}^{(n)}, \boldsymbol{G}^{(n)}, \boldsymbol{\Sigma}^{(n)}\}$
12:	**End**	

5.5.2.2 Class Label Prediction

Given a set of training vectors denoted as $\{\boldsymbol{x}_{11}, \boldsymbol{x}_{12} \ldots \boldsymbol{x}_{1J}, \boldsymbol{x}_{21}, \boldsymbol{x}_{22} \ldots \boldsymbol{x}_{2J} \ldots \ldots \boldsymbol{x}_{K1}, \boldsymbol{x}_{K2} \ldots \boldsymbol{x}_{KJ}\}$, and a testing vector $\boldsymbol{y}$. If $\boldsymbol{y}$ indeed belongs to class k, then $\boldsymbol{y}$ and $\{\boldsymbol{x}_{k1}, \boldsymbol{x}_{k2} \ldots \boldsymbol{x}_{kJ}\}$ should be jointly distributed, while $\boldsymbol{y}$ and the other training vectors should be independently distributed [98][99]. This yields the expression of the joint probability of $\boldsymbol{y}$ and all the training vectors as given by (5.64), where $\boldsymbol{X}_k$ represents all the training vectors in class k.

$$p(\boldsymbol{y}, \boldsymbol{X}_1, \ldots \boldsymbol{X}_{k-1}, \boldsymbol{X}_k, \boldsymbol{X}_{k+1}, \ldots \boldsymbol{X}_K) = p(\boldsymbol{y}, \boldsymbol{X}_k)\prod_{i=1, i\neq k}^{K} p(\boldsymbol{X}_i) = p(\boldsymbol{y}|\boldsymbol{X}_k)\prod_{i=1}^{K} p(\boldsymbol{X}_i) \quad (5.64)$$

The class label of $\boldsymbol{y}$ can then be determined by finding the class that provides the highest joint probability given by (5.64), which is equivalent to finding the conditional probability $p(\boldsymbol{y}|\boldsymbol{X}_k)$ concerning all the training vectors in class k. The prediction criterion is then given by (5.65), where $\ell(\boldsymbol{y})$ represents the predicted label and $\ell(\boldsymbol{y}) \in \{1,2 \dots K\}$.

$$\ell(\boldsymbol{y}) = \underset{k}{\operatorname{argmax}}\, p(\boldsymbol{y}|\boldsymbol{X}_k) \prod_{i=1}^{K} p(\boldsymbol{X}_i) = \underset{k}{\operatorname{argmax}}\, p(\boldsymbol{y}|\boldsymbol{X}_k) \tag{5.65}$$

The conditional probability $p(\boldsymbol{y}|\boldsymbol{X}_k)$ can be derived from the joint probability $p(\boldsymbol{y}, \boldsymbol{X}_k)$, since both $\boldsymbol{y}$ and $\boldsymbol{X}_k$ follow the Gaussian distribution. The joint probability of $\boldsymbol{y}$ and $\boldsymbol{X}_k$ can be obtained in a similar way to (5.56), which is a Gaussian distribution as given by (5.66), where $\widetilde{\boldsymbol{U}}$, $\widetilde{\boldsymbol{R}}$ and $\widetilde{\boldsymbol{\Psi}}$ are augmented matrices as given by (5.67).

$$p\left(\boldsymbol{y}, \boldsymbol{x}_{k1}, \boldsymbol{x}_{k2} \dots \boldsymbol{x}_{kJ}\right) = p(\boldsymbol{y}, \boldsymbol{X}_k) = p(\boldsymbol{X}_k, \boldsymbol{y}) = p_g\left(\begin{bmatrix} \boldsymbol{X}_k \\ \boldsymbol{y} \end{bmatrix} \middle| \widetilde{\boldsymbol{U}}, \widetilde{\boldsymbol{R}}\widetilde{\boldsymbol{R}}^T + \widetilde{\boldsymbol{\Psi}}\right) \tag{5.66}$$

where

$$\widetilde{\boldsymbol{U}} = \begin{bmatrix} \boldsymbol{U} \\ \boldsymbol{\mu} \end{bmatrix},\ \widetilde{\boldsymbol{R}} = \begin{bmatrix} \begin{bmatrix} & \boldsymbol{R} & \end{bmatrix} & \begin{matrix} \boldsymbol{0} \\ \vdots \\ \boldsymbol{0} \end{matrix} \\ \begin{matrix} \boldsymbol{F} & \boldsymbol{0} & \cdots & \boldsymbol{0} \end{matrix} & \boldsymbol{G} \end{bmatrix},\ \widetilde{\boldsymbol{\Psi}} = \begin{bmatrix} \boldsymbol{\Psi} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\Sigma} \end{bmatrix} \tag{5.67}$$

To ease the processing, we may partition the joint mean $\widetilde{\boldsymbol{U}}$ and the joint covariance ($\widetilde{\boldsymbol{R}}\widetilde{\boldsymbol{R}}^T + \widetilde{\boldsymbol{\Psi}}$) according to (5.68), where $\boldsymbol{\Psi}_{yy}$, $\boldsymbol{\Psi}_{yX}$, $\boldsymbol{\Psi}_{Xy}$ and $\boldsymbol{\Psi}_{XX}$ are matrix blocks given by (5.69).

$$\widetilde{\boldsymbol{U}} = \begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{U} \end{bmatrix},\ \widetilde{\boldsymbol{R}}\widetilde{\boldsymbol{R}}^T + \widetilde{\boldsymbol{\Psi}} = \begin{bmatrix} \boldsymbol{\Psi}_{yy} & \boldsymbol{\Psi}_{yX} \\ \boldsymbol{\Psi}_{Xy} & \boldsymbol{\Psi}_{XX} \end{bmatrix} \tag{5.68}$$

where

$$\begin{aligned} \boldsymbol{\Psi}_{yy} &= \boldsymbol{F}\boldsymbol{F}^T + \boldsymbol{G}\boldsymbol{G}^T + \boldsymbol{\Sigma} \\ \boldsymbol{\Psi}_{yX} &= \begin{bmatrix} \boldsymbol{F}\boldsymbol{F}^T & \boldsymbol{F}\boldsymbol{F}^T & \cdots & \boldsymbol{F}\boldsymbol{F}^T \end{bmatrix} \\ \boldsymbol{\Psi}_{Xy} &= \begin{bmatrix} \boldsymbol{F}\boldsymbol{F}^T & \boldsymbol{F}\boldsymbol{F}^T & \cdots & \boldsymbol{F}\boldsymbol{F}^T \end{bmatrix}^T \\ \boldsymbol{\Psi}_{XX} &= \begin{bmatrix} \boldsymbol{\Psi}_{yy} & \boldsymbol{F}\boldsymbol{F}^T & \boldsymbol{F}\boldsymbol{F}^T & \cdots \\ \boldsymbol{F}\boldsymbol{F}^T & \boldsymbol{\Psi}_{yy} & \boldsymbol{F}\boldsymbol{F}^T & \cdots \\ \boldsymbol{F}\boldsymbol{F}^T & \boldsymbol{F}\boldsymbol{F}^T & \boldsymbol{\Psi}_{yy} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \end{aligned} \tag{5.69}$$

Using the partitioned matrices given by (5.68), the conditional probability $p(\boldsymbol{y}|\boldsymbol{X}_k)$ can be expressed as (5.70), which is a Gaussian distribution with mean $\boldsymbol{\mu}_{y|k}$ and covariance $\boldsymbol{\Psi}_{y|k}$ given by (5.71) and (5.72) respectively [100].

$$p(\boldsymbol{y}|\boldsymbol{X}_k) = p_g\left(\boldsymbol{y}|\boldsymbol{\mu}_{y|k}, \boldsymbol{\Psi}_{y|k}\right) \tag{5.70}$$

where

$$\boldsymbol{\mu}_{y|k} = \boldsymbol{\mu} + \boldsymbol{\Psi}_{yX}\boldsymbol{\Psi}_{XX}^{-1}(\boldsymbol{X}_k - \boldsymbol{U}) \tag{5.71}$$

$$\boldsymbol{\Psi}_{y|k} = \boldsymbol{\Psi}_{yy} - \boldsymbol{\Psi}_{yX}\boldsymbol{\Psi}_{XX}^{-1}\boldsymbol{\Psi}_{Xy} \tag{5.72}$$

Having the conditional probability for each class, the class label of $\boldsymbol{y}$ can then be determined by finding the largest conditional probability as given by (5.73).

$$\ell(\boldsymbol{y}) = \underset{k}{\operatorname{argmax}}\, p(\boldsymbol{y}|\boldsymbol{X}_k) = \underset{k}{\operatorname{argmax}}\, p_g\left(\boldsymbol{y}|\boldsymbol{\mu}_{y|k}, \boldsymbol{\Psi}_{y|k}\right) \tag{5.73}$$

5.5.3 Scalable Formulation

5.5.3.1 Parameter Estimation

As can be seen from (5.57) and (5.58), the E-step requires computing the inverse of $(\boldsymbol{I} + \boldsymbol{R}^T\boldsymbol{\Psi}^{-1}\boldsymbol{R})$. Suppose the sizes of factor-loading matrices $\boldsymbol{F}$ and $\boldsymbol{G}$ are both $D \times H$, and the number of training vectors in each class is J, then the size of $(\boldsymbol{I} + \boldsymbol{R}^T\boldsymbol{\Psi}^{-1}\boldsymbol{R})$ will be $(J+1)H \times (J+1)H$. This means that the size of $(\boldsymbol{I} + \boldsymbol{R}^T\boldsymbol{\Psi}^{-1}\boldsymbol{R})$ is proportional to the number of training vectors. When the number of training vectors in each class is too large, the computation of the E-step is inefficient, or even infeasible if memory is not enough.

Facing this scalability issue, a scalable formulation for the E-step is proposed in [78], which partitions the large matrix $(\boldsymbol{I} + \boldsymbol{R}^T\boldsymbol{\Psi}^{-1}\boldsymbol{R})$ into matrix blocks. Then, the technique of block matrix inversion [100] is applied. The E-step then becomes (5.74) and (5.75), while the M-step is kept unchanged. The EM algorithm for this scalable formulation is summarized in Algorithm 5.5.

Algorithm 5.5 EM algorithm for the scalable formulation of PLDA

1:	Initialization:	$\theta^{(0)} = \{\boldsymbol{\mu}, \boldsymbol{F}^{(0)}, \boldsymbol{G}^{(0)}, \boldsymbol{\Sigma}^{(0)}\}$
2:	Compute mean:	$\boldsymbol{\mu} = \frac{1}{KJ}\sum_{k=1}^{K}\sum_{j=1}^{J}\boldsymbol{x}_{kj}$
3:	**For** n = 1 to N	
4:		Set $\{\boldsymbol{F}, \boldsymbol{G}, \boldsymbol{\Sigma}\} = \{\boldsymbol{F}^{(n-1)}, \boldsymbol{G}^{(n-1)}, \boldsymbol{\Sigma}^{(n-1)}\}$
5:		Compute $\{\boldsymbol{L}, \boldsymbol{\Lambda}, \boldsymbol{M}\}$ using $\{\boldsymbol{F}, \boldsymbol{G}, \boldsymbol{\Sigma}\}$
6:		$E^{(n)}[\boldsymbol{h}_k] = \boldsymbol{M}(\boldsymbol{F}^T - \boldsymbol{\Lambda}^T\boldsymbol{G}^T)\boldsymbol{\Sigma}^{-1}\sum_{j=1}^{J}(\boldsymbol{x}_{kj} - \boldsymbol{\mu})$
7:	E-step:	$E^{(n)}[\boldsymbol{w}_{kj}] = \boldsymbol{L}^{-1}\boldsymbol{G}^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_{kj} - \boldsymbol{\mu}) - \boldsymbol{\Lambda}E^{(n)}[\boldsymbol{h}_k]$
8:		$E^{(n)}[\boldsymbol{z}_{kj}] = \begin{bmatrix} E^{(n)}[\boldsymbol{h}_k] \\ E^{(n)}[\boldsymbol{w}_{kj}] \end{bmatrix}$
9:		$E^{(n)}[\boldsymbol{z}_{kj}\boldsymbol{z}_{kj}^T] = \begin{bmatrix} \boldsymbol{M} & -\boldsymbol{M}\boldsymbol{\Lambda}^T \\ -\boldsymbol{\Lambda}\boldsymbol{M} & \boldsymbol{L}^{-1} + \boldsymbol{\Lambda}\boldsymbol{M}\boldsymbol{\Lambda}^T \end{bmatrix} + E^{(n)}[\boldsymbol{z}_{kj}]E^{(n)}[\boldsymbol{z}_{kj}]^T$
10:		$\boldsymbol{V}^{(n)} = \left(\sum_{k=1}^{K}\sum_{j=1}^{J}(\boldsymbol{x}_{kj} - \boldsymbol{\mu})E^{(n)}[\boldsymbol{z}_{kj}]^T\right)\left(\sum_{k=1}^{K}\sum_{j=1}^{J}E^{(n)}[\boldsymbol{z}_{kj}\boldsymbol{z}_{kj}^T]\right)^{-1}$
11:	M-step:	$\boldsymbol{\Sigma}^{(n)} = \frac{1}{KJ}\sum_{k=1}^{K}\sum_{j=1}^{J}\mathrm{diag}\left\{(\boldsymbol{x}_{kj} - \boldsymbol{\mu})(\boldsymbol{x}_{kj} - \boldsymbol{\mu})^T - \boldsymbol{V}^{(n)}E^{(n)}[\boldsymbol{z}_{kj}](\boldsymbol{x}_{kj} - \boldsymbol{\mu})^T\right\}$
12:		$[\boldsymbol{F}^{(n)} \quad \boldsymbol{G}^{(n)}] = \boldsymbol{V}^{(n)}$
13:	Update θ:	$\theta^{(n)} = \{\boldsymbol{\mu}, \boldsymbol{F}^{(n)}, \boldsymbol{G}^{(n)}, \boldsymbol{\Sigma}^{(n)}\}$
14:	**End**	

$$E[\boldsymbol{z}_{kj}] = \begin{bmatrix} E[\boldsymbol{h}_k] \\ E[\boldsymbol{w}_{kj}] \end{bmatrix} = \begin{bmatrix} \boldsymbol{M}(\boldsymbol{F}^T - \boldsymbol{\Lambda}^T\boldsymbol{G}^T)\boldsymbol{\Sigma}^{-1}\sum_{j=1}^{J}(\boldsymbol{x}_{kj} - \boldsymbol{\mu}) \\ \boldsymbol{L}^{-1}\boldsymbol{G}^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_{kj} - \boldsymbol{\mu}) - \boldsymbol{\Lambda}E[\boldsymbol{h}_k] \end{bmatrix} \tag{5.74}$$

$$E[\boldsymbol{z}_{kj}\boldsymbol{z}_{kj}^T] = \begin{bmatrix} \boldsymbol{M} & -\boldsymbol{M}\boldsymbol{\Lambda}^T \\ -\boldsymbol{\Lambda}\boldsymbol{M} & \boldsymbol{L}^{-1} + \boldsymbol{\Lambda}\boldsymbol{M}\boldsymbol{\Lambda}^T \end{bmatrix} + E[\boldsymbol{z}_{kj}]E[\boldsymbol{z}_{kj}]^T \tag{5.75}$$

where

$$\begin{aligned} \boldsymbol{L} &= \boldsymbol{I} + \boldsymbol{G}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{G} \\ \boldsymbol{\Lambda} &= \boldsymbol{L}^{-1}\boldsymbol{G}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{F} \\ \boldsymbol{M} &= (\boldsymbol{I} + J\boldsymbol{F}^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{F} - \boldsymbol{G}\boldsymbol{\Lambda}))^{-1} \end{aligned} \tag{5.76}$$

5.5.3.2 Class Label Prediction

As can be seen from (5.71) and (5.72), the mean and the covariance of the conditional probability require computing the inverse of $\boldsymbol{\Psi}_{XX}$. Suppose the sizes of $\boldsymbol{F}$ and $\boldsymbol{G}$ are both $D \times H$, and the number of training vectors in each class is J, then the size of $\boldsymbol{\Psi}_{XX}$ will be $JD \times JD$. This means that the size of $\boldsymbol{\Psi}_{XX}$ is proportional to the number of training vectors, and the computation is inefficient if the number of training vectors in each class is too large.

Fortunately, according to (5.69), $\boldsymbol{\Psi}_{XX}$ possesses a symmetric structure, implying that its inverse should also possess a similar structure as given by (5.77), where $\boldsymbol{P}^{(J)}$ and $\boldsymbol{Q}^{(J)}$ are matrix blocks, and the superscript J indicates the number of training vectors in class k.

$$\boldsymbol{\Psi}_{XX}^{-1} = \begin{bmatrix} \boldsymbol{\Psi}_{yy} & \boldsymbol{F}\boldsymbol{F}^T & \boldsymbol{F}\boldsymbol{F}^T & \cdots \\ \boldsymbol{F}\boldsymbol{F}^T & \boldsymbol{\Psi}_{yy} & \boldsymbol{F}\boldsymbol{F}^T & \cdots \\ \boldsymbol{F}\boldsymbol{F}^T & \boldsymbol{F}\boldsymbol{F}^T & \boldsymbol{\Psi}_{yy} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}^{-1} = \begin{bmatrix} \boldsymbol{P}^{(J)} & \boldsymbol{Q}^{(J)} & \boldsymbol{Q}^{(J)} & \cdots \\ \boldsymbol{Q}^{(J)} & \boldsymbol{P}^{(J)} & \boldsymbol{Q}^{(J)} & \cdots \\ \boldsymbol{Q}^{(J)} & \boldsymbol{Q}^{(J)} & \boldsymbol{P}^{(J)} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{5.77}$$

It can be seen that $\boldsymbol{P}^{(J)}$ and $\boldsymbol{Q}^{(J)}$ should satisfy the equation given by (5.78).

$$\begin{bmatrix} \boldsymbol{\Psi}_{yy} & \boldsymbol{F}\boldsymbol{F}^T & \boldsymbol{F}\boldsymbol{F}^T & \cdots \\ \boldsymbol{F}\boldsymbol{F}^T & \boldsymbol{\Psi}_{yy} & \boldsymbol{F}\boldsymbol{F}^T & \cdots \\ \boldsymbol{F}\boldsymbol{F}^T & \boldsymbol{F}\boldsymbol{F}^T & \boldsymbol{\Psi}_{yy} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} \boldsymbol{P}^{(J)} & \boldsymbol{Q}^{(J)} & \boldsymbol{Q}^{(J)} & \cdots \\ \boldsymbol{Q}^{(J)} & \boldsymbol{P}^{(J)} & \boldsymbol{Q}^{(J)} & \cdots \\ \boldsymbol{Q}^{(J)} & \boldsymbol{Q}^{(J)} & \boldsymbol{P}^{(J)} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \cdots \\ \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} & \cdots \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{5.78}$$

By equating the results on both sides of (5.78), we obtain the linear equations

$$\begin{aligned} \boldsymbol{\Psi}_{yy}\boldsymbol{P}^{(J)} + (J-1)\boldsymbol{F}\boldsymbol{F}^T\boldsymbol{Q}^{(J)} &= \boldsymbol{I} \\ \boldsymbol{\Psi}_{yy}\boldsymbol{Q}^{(J)} + \boldsymbol{F}\boldsymbol{F}^T\boldsymbol{P}^{(J)} + (J-2)\boldsymbol{F}\boldsymbol{F}^T\boldsymbol{Q}^{(J)} &= \boldsymbol{0} \end{aligned} \tag{5.79}$$

By substituting $\boldsymbol{\Psi}_{yy}$ with $(\boldsymbol{F}\boldsymbol{F}^T + \boldsymbol{G}\boldsymbol{G}^T + \boldsymbol{\Sigma})$, (5.79) becomes

$$\begin{aligned} (\boldsymbol{F}\boldsymbol{F}^T + \boldsymbol{G}\boldsymbol{G}^T + \boldsymbol{\Sigma})\boldsymbol{P}^{(J)} + (J-1)\boldsymbol{F}\boldsymbol{F}^T\boldsymbol{Q}^{(J)} &= \boldsymbol{I} \\ (\boldsymbol{F}\boldsymbol{F}^T + \boldsymbol{G}\boldsymbol{G}^T + \boldsymbol{\Sigma})\boldsymbol{Q}^{(J)} + \boldsymbol{F}\boldsymbol{F}^T\boldsymbol{P}^{(J)} + (J-2)\boldsymbol{F}\boldsymbol{F}^T\boldsymbol{Q}^{(J)} &= \boldsymbol{0} \end{aligned} \tag{5.80}$$

Solving (5.80) yields

$$\begin{aligned}\boldsymbol{P}^{(J)} &= \{\boldsymbol{F}\boldsymbol{F}^T + \boldsymbol{G}\boldsymbol{G}^T + \boldsymbol{\Sigma} - (J-1)\boldsymbol{F}\boldsymbol{F}^T(\boldsymbol{G}\boldsymbol{G}^T + \boldsymbol{\Sigma} + (J-1)\boldsymbol{F}\boldsymbol{F}^T)^{-1}\boldsymbol{F}\boldsymbol{F}^T\}^{-1} \\ \boldsymbol{Q}^{(J)} &= -\{\boldsymbol{G}\boldsymbol{G}^T + \boldsymbol{\Sigma} + (J-1)\boldsymbol{F}\boldsymbol{F}^T\}^{-1}\boldsymbol{F}\boldsymbol{F}^T\boldsymbol{P}^{(J)}\end{aligned} \tag{5.81}$$

Having obtained the expressions of $\boldsymbol{P}^{(J)}$ and $\boldsymbol{Q}^{(J)}$, the inverse of $\boldsymbol{\Psi}_{XX}$ can then be expressed in terms of $\boldsymbol{P}^{(J)}$ and $\boldsymbol{Q}^{(J)}$. Consequently, the mean $\boldsymbol{\mu}_{y|k}$ and the covariance $\boldsymbol{\Psi}_{y|k}$ used in the conditional probability given by (5.70) can be expressed in terms of $\boldsymbol{P}^{(J)}$ and $\boldsymbol{Q}^{(J)}$ as given by (5.82) and (5.83) respectively. The conditional probability $p(\boldsymbol{y}|\boldsymbol{X}_k)$ can then be calculated efficiently.

$$\boldsymbol{\mu}_{y|k} = \boldsymbol{\mu} + \boldsymbol{\Psi}_{yX}\boldsymbol{\Psi}_{XX}^{-1}(\boldsymbol{X}_k - \boldsymbol{U}) = \boldsymbol{\mu} + \left(\boldsymbol{F}\boldsymbol{F}^T\boldsymbol{P}^{(J)} + (J-1)\boldsymbol{F}\boldsymbol{F}^T\boldsymbol{Q}^{(J)}\right)\sum_{j=1}^{J}(\boldsymbol{x}_{kj} - \boldsymbol{\mu}) \tag{5.82}$$

$$\boldsymbol{\Psi}_{y|k} = \boldsymbol{\Psi}_{yy} - \boldsymbol{\Psi}_{yX}\boldsymbol{\Psi}_{XX}^{-1}\boldsymbol{\Psi}_{Xy} = (\boldsymbol{F}\boldsymbol{F}^T + \boldsymbol{G}\boldsymbol{G}^T + \boldsymbol{\Sigma}) - J\left(\boldsymbol{F}\boldsymbol{F}^T\boldsymbol{P}^{(J)} + (J-1)\boldsymbol{F}\boldsymbol{F}^T\boldsymbol{Q}^{(J)}\right)\boldsymbol{F}\boldsymbol{F}^T \tag{5.83}$$

5.5.4 Computational Complexity

5.5.4.1 Parameter Estimation

On using the EM algorithm for parameter estimation, most of the computation time will be consumed by the E-step. We also assume that the time consumption of the E-step can be approximated by the time consumption of calculating the posterior expected mean of the latent vector, as the computation of the posterior expected mean requires more matrix operations than the computation of the posterior expected covariance. Suppose the inversion of a matrix with a size of $A \times A$ has a time complexity of $O(A^3)$, the multiplication of matrices with sizes $A \times B$ and $B \times C$ has a time complexity of $O(ABC)$. Suppose the sizes of factor-loading matrices $\boldsymbol{F}$ and $\boldsymbol{G}$ are both $D \times H$, and the number of training vectors in each class is J. Let $H = \gamma D$, where γ is a positive number.

In the original formulation, as given by (5.57), the matrices involved in computing $E[\boldsymbol{Y}_k]$, i.e., $(\boldsymbol{I} + \boldsymbol{R}^T\boldsymbol{\Psi}^{-1}\boldsymbol{R})$, $\boldsymbol{R}$, $\boldsymbol{\Psi}$ and $(\boldsymbol{X}_k - \boldsymbol{U})$, have sizes of $(J+1)H \times (J+1)H$, $JD \times (J+1)H$, $JD \times JD$ and $JD \times 1$ respectively. Suppose $(\boldsymbol{I} + \boldsymbol{R}^T\boldsymbol{\Psi}^{-1}\boldsymbol{R})$ and $\boldsymbol{\Psi}^{-1}$ have been pre-computed, the time

complexity of the model parameter estimation process can then be approximated by (5.84), where we neglect lower-order terms and assume $J > 1$ and $\gamma < D$.

$$t_{model} \approx K \times \left(O\left(\left((J+1)H \right)^3 \right) + O\big((J+1)H \times (J+1)H \times JD \times JD \times 1 \big) \right)$$

$$= K \times \big(O((J+1)^3 H^3) + O((J+1)^2 J^2 H^2 D^2) \big) = K \times \big(O((J+1)^3 \gamma^3 D^3) + O((J+1)^2 J^2 \gamma^2 D^4) \big)$$

$$= O(KJ^4 \gamma^2 D^4) \tag{5.84}$$

In the scalable formulation, as given by (5.74), the matrices involved in computing $E[\boldsymbol{z}_{kj}]$, i.e., $\boldsymbol{F}$, $\boldsymbol{G}$, $\boldsymbol{\Sigma}$, $\boldsymbol{L}$, $\boldsymbol{\Lambda}$, $\boldsymbol{M}$, $(\boldsymbol{x}_{kj} - \boldsymbol{\mu})$ and $E[\boldsymbol{h}_k]$, have sizes of $D \times H$, $D \times H$, $D \times D$, $H \times H$, $H \times H$, $H \times H$, $D \times 1$ and $H \times 1$ respectively. Suppose $\boldsymbol{\Sigma}^{-1}$ has been pre-computed, and the computational complexity of $E[\boldsymbol{z}_{kj}]$ can be approximated by that of $E[\boldsymbol{w}_{kj}]$ (because the computational complexity of $E[\boldsymbol{w}_{kj}]$ and $E[\boldsymbol{h}_k]$ is similar). The time complexity of the model parameter estimation process can then be approximated by (5.85), where we neglect lower-order terms and assume $\gamma < D$.

$$t'_{model} \approx KJ \times \big(O(H^3) + O(H \times H \times D \times D \times 1) + O(H \times H \times 1) \big)$$

$$= KJ \times \big(O(H^3) + O(H^2 D^2) + O(H^2) \big) = KJ \times \big(O(\gamma^3 D^3) + O(\gamma^2 D^4) + O(\gamma^2 D^2) \big)$$

$$= O(KJ\gamma^2 D^4) \tag{5.85}$$

By comparing (5.84) and (5.85), it can be seen that if J is very large, namely, the number of training data is large, using the scalable formulation will be much more efficient than using the original formulation to estimate the model parameters.

5.5.4.2 Class Label Prediction

When using the conditional probability to predict class labels, as given by (5.70), most of the computation time is spent in computing the conditional mean $\boldsymbol{\mu}_{y|k}$ and the conditional covariance $\boldsymbol{\Psi}_{y|k}$. Since the computational complexity of $\boldsymbol{\mu}_{y|k}$ and $\boldsymbol{\Psi}_{y|k}$ is similar, the time complexity of the class label prediction process can be approximated by the time complexity of computing $\boldsymbol{\Psi}_{y|k}$.

In the original formulation, as given by (5.72), the matrices involved in computation, i.e., $\boldsymbol{\Psi}_{yy}$, $\boldsymbol{\Psi}_{yX}$, $\boldsymbol{\Psi}_{XX}$ and $\boldsymbol{\Psi}_{Xy}$, have sizes of $D \times D$, $D \times JD$, $JD \times JD$ and $JD \times D$ respectively. Suppose $\boldsymbol{F}\boldsymbol{F}^T$, $\boldsymbol{G}\boldsymbol{G}^T$, $(\boldsymbol{F}\boldsymbol{F}^T + \boldsymbol{G}\boldsymbol{G}^T + \boldsymbol{\Sigma})$ and $\boldsymbol{\Psi}_{\mathrm{XX}}^{-1}$ have been pre-computed, the time complexity of the prediction process can then be approximated by (5.86).

$$t_{pred} \approx O(D \times JD \times JD \times D) = O(J^2 D^4) \tag{5.86}$$

In the scalable formulation, as given by (5.83), the matrices involved in computation, i.e., $\boldsymbol{F}$, $\boldsymbol{G}$, $\boldsymbol{\Sigma}$, $\boldsymbol{P}^{(J)}$ and $\boldsymbol{Q}^{(J)}$, have sizes of $D \times H$, $D \times H$, $D \times D$, $D \times D$ and $D \times D$ respectively. Suppose $\boldsymbol{F}\boldsymbol{F}^T$, $\boldsymbol{G}\boldsymbol{G}^T$ and $(\boldsymbol{F}\boldsymbol{F}^T + \boldsymbol{G}\boldsymbol{G}^T + \boldsymbol{\Sigma})$ have been pre-computed, the time complexity of the prediction process can then be approximated by (5.87).

$$t'_{pred} \approx O(D \times D \times D \times D) = O(D^4) \tag{5.87}$$

By comparing (5.86) and (5.87), it can be seen that the time complexity of the prediction process when using the scalable formulation is independent of the number of training data. This characteristic makes the scalable formulation able to handle a large number of data.

5.6 Dictionary-based Representations and Classifiers

This section introduces two dictionary-based representations, i.e., the sparse representation (SR) and the collaborative representation (CR). The two representations have similar objective functions and similar performance in face recognition, but CR is computationally more efficient than SR [69]. In order to further improve the discrimination ability of CR for doing classification tasks, we propose a discriminative CR (DCR) which carries the class information. The minimum residual-based classifier (MRC) is also described, which is used for classifying dictionary-based representations.

5.6.1 Sparse Representation

Suppose we have a dictionary denoted as $\boldsymbol{B}$, which is a matrix consisting of a set of basis vectors as given by (5.88), where $\boldsymbol{b}_n$ is the n-th basis vector. If the dimensionality of a basis vector is $D \times 1$, the size of the dictionary $\boldsymbol{B}$ will then be $D \times N$.

$$\boldsymbol{B} = [\boldsymbol{b}_1 \quad \boldsymbol{b}_2 \quad \cdots \quad \boldsymbol{b}_N] \tag{5.88}$$

From the perspective of the sparse representation (SR), a feature vector $\boldsymbol{x}$ is approximated as a linear combination of the basis vectors in the dictionary with some constraints. Specifically, given a feature vector $\boldsymbol{x}$, its corresponding SR, denoted as $\boldsymbol{X}_{SR}$, is given by (5.89), where $\boldsymbol{y}$ is the coefficient vector aiming to reconstruct $\boldsymbol{x}$ [60].

$$\boldsymbol{X}_{SR} = \underset{\boldsymbol{y}}{\operatorname{argmin}} \|\boldsymbol{y}\|_0 \quad s.t. \quad \boldsymbol{x} = \boldsymbol{B}\boldsymbol{y} \tag{5.89}$$

Since the optimization problem given by (5.89) involves the L0-norm constraint, which is difficult to solve, researchers approximate the solution by solving an alternative problem with the L1-norm constraint, as given by (5.90). It can be shown that, under the condition that the solution is sparse enough, the solution to (5.89) is equal to the solution to (5.90) [60].

$$\boldsymbol{X}_{SR} = \underset{\boldsymbol{y}}{\operatorname{argmin}} \|\boldsymbol{y}\|_1 \quad s.t. \quad \boldsymbol{x} = \boldsymbol{B}\boldsymbol{y} \tag{5.90}$$

When the feature vector or the basis vectors are noisy, it may not be suitable to exactly reconstruct the feature vector using the basis vectors. In this case, a relaxed version can be adopted to compute the SR as given by (5.91), where λ is a pre-defined regularization parameter. Equation (5.91) is referred to as the unconstrained basis pursuit denoising (BPDN) problem [101].

$$\boldsymbol{X}_{SR} = \underset{\boldsymbol{y}}{\operatorname{argmin}} \frac{1}{2} \|\boldsymbol{x} - \boldsymbol{B}\boldsymbol{y}\|_2^2 + \lambda \|\boldsymbol{y}\|_1 \tag{5.91}$$

5.6.2 Collaborative Representation

Since the L1-norm constraint is involved in the formulation of SR, no analytical solution for finding $\boldsymbol{X}_{SR}$ exists. Nevertheless, there are many algorithms that can solve it efficiently, such as the matching pursuit algorithm and the basis pursuit algorithm [61], and some fast algorithms such as the Homotopy method and the augmented Lagrangian method [101]. However, as shown in [109], the sparseness may not be the key to the success of SR, as increasing the sparseness does not yield performance improvement for image classification tasks. In addition, using the L2-norm constraint can even be more robust than using the L1-norm constraint, although the latter yields a sparser representation [110].

Therefore, in [69], the L1-norm constraint is replaced by the L2-norm constraint, yielding the collaborative representation (CR). Given a feature vector $\boldsymbol{x}$, its corresponding CR, denoted as $\boldsymbol{X}_{CR}$, is computed as

$$\boldsymbol{X}_{CR} = \underset{\boldsymbol{y}}{\operatorname{argmin}} \|\boldsymbol{x} - \boldsymbol{B}\boldsymbol{y}\|_2^2 + \lambda \|\boldsymbol{y}\|_2^2 \tag{5.92}$$

As shown in [69], for face recognition tasks, CR gives similar performance to SR, but is more efficient in computation as an analytical solution exists, which is given by

$$\boldsymbol{X}_{CR} = (\boldsymbol{B}^T\boldsymbol{B} + \lambda \boldsymbol{I})^{-1}\boldsymbol{B}^T\boldsymbol{x} \tag{5.93}$$

5.6.3 Discriminative Collaborative Representation

Essentially, SR and CR are unsupervised representations, as the dictionary usually carries no class information. However, when used for doing pattern recognition tasks, the dictionary is usually composed of the training data, which then carries the class information. The dictionary $\boldsymbol{B}$ can then be expressed as (5.94), where K denotes the total number of classes, and $\boldsymbol{B}^{(k)}$ denotes the k-th sub dictionary. Suppose the basis vectors in the dictionary are just the training vectors, if the dimensionality of a training vector is $D \times 1$ and there are N_k training vectors for class k, then the size of $\boldsymbol{B}^{(k)}$ is $D \times N_k$.

$$\boldsymbol{B} = [\boldsymbol{B}^{(1)} \quad \boldsymbol{B}^{(2)} \quad \cdots \quad \boldsymbol{B}^{(K)}] \tag{5.94}$$

Obviously, we have the relationships as given by (5.95), where $\boldsymbol{y}^{(k)}$ denotes the k-th sub coefficient vector that corresponds to the k-th sub dictionary, and N is the total number of training vectors. The dimensionality of $\boldsymbol{y}$ and $\boldsymbol{y}^{(k)}$ is $N \times 1$ and $N_k \times 1$ respectively.

$$\begin{aligned} N &= \textstyle\sum_{k=1}^{K} N_k \\ \boldsymbol{B}\boldsymbol{y} &= \textstyle\sum_{k=1}^{K} \boldsymbol{B}^{(k)}\boldsymbol{y}^{(k)} \end{aligned} \tag{5.95}$$

In the formulation of $\boldsymbol{X}_{CR}$, the class information is embedded through the L2-norm constraint. This sparsity constraint tries to minimize the norm of the coefficient vector $\boldsymbol{y}$, expecting that the coefficients corresponding to the target class will be more dominant than those corresponding to the non-target class.

In order to explicitly embed the class information into CR, as inspired by [102], we adopt the objective function $Q(\boldsymbol{y})$ as given by (5.96), where we introduce an extra regularization term into the original objective function of CR, with η being the corresponding regularization parameter. λ tends to regularize the coefficient vector to be sparse, whereas η tends to regularize the coefficient vector to be dense. These two regularization terms are actually confronting each other, but finding a balance between them may improve the quality of the representation.

$$Q(\boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{B}\boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{y}\|_2^2 + \eta\sum_{k=1}^{K}\left\|\boldsymbol{B}^{(k)}\boldsymbol{y}^{(k)} - \frac{1}{K}\sum_{i=1}^{K}\boldsymbol{B}^{(i)}\boldsymbol{y}^{(i)}\right\|_2^2 \tag{5.96}$$

The discriminative collaborative representation (DCR) is then given by

$$\boldsymbol{X}_{DCR} = \underset{\boldsymbol{y}}{\operatorname{argmin}}\, Q(\boldsymbol{y}) \tag{5.97}$$

Similar to CR, an analytical solution also exists for DCR, which can be obtained by setting the derivative of $Q(\boldsymbol{y})$ to zero. To compute the derivative of $Q(\boldsymbol{y})$ with respect to $\boldsymbol{y}$, we first reformulate the expression of $Q(\boldsymbol{y})$ as

$$\begin{aligned} Q(\boldsymbol{y}) &= (\boldsymbol{x} - \boldsymbol{B}\boldsymbol{y})^T(\boldsymbol{x} - \boldsymbol{B}\boldsymbol{y}) + \lambda\boldsymbol{y}^T\boldsymbol{y} + \eta\sum_{k=1}^{K}\left(\boldsymbol{B}^{(k)}\boldsymbol{y}^{(k)} - \frac{1}{K}\boldsymbol{B}\boldsymbol{y}\right)^T\left(\boldsymbol{B}^{(k)}\boldsymbol{y}^{(k)} - \frac{1}{K}\boldsymbol{B}\boldsymbol{y}\right) \\ &= \boldsymbol{x}^T\boldsymbol{x} + \boldsymbol{y}^T\boldsymbol{B}^T\boldsymbol{B}\boldsymbol{y} - 2\boldsymbol{x}^T\boldsymbol{B}\boldsymbol{y} + \lambda\boldsymbol{y}^T\boldsymbol{y} + \eta\sum_{k=1}^{K}\boldsymbol{y}^{(k)^T}\boldsymbol{B}^{(k)^T}\boldsymbol{B}^{(k)}\boldsymbol{y}^{(k)} + \eta\sum_{k=1}^{K}\frac{1}{K^2}\boldsymbol{y}^T\boldsymbol{B}^T\boldsymbol{B}\boldsymbol{y} \\ &\quad -2\eta\sum_{k=1}^{K}\frac{1}{K}\boldsymbol{y}^T\boldsymbol{B}^T\boldsymbol{B}^{(k)}\boldsymbol{y}^{(k)} \\ &= \boldsymbol{x}^T\boldsymbol{x} - 2\boldsymbol{x}^T\boldsymbol{B}\boldsymbol{y} + \lambda\boldsymbol{y}^T\boldsymbol{y} + \left(1 - \frac{\eta}{K}\right)\boldsymbol{y}^T\boldsymbol{B}^T\boldsymbol{B}\boldsymbol{y} + \eta\sum_{k=1}^{K}\boldsymbol{y}^{(k)^T}\boldsymbol{B}^{(k)^T}\boldsymbol{B}^{(k)}\boldsymbol{y}^{(k)} \end{aligned} \tag{5.98}$$

Define a diagonal block matrix $\boldsymbol{L}$ as

$$\boldsymbol{L} = \begin{bmatrix} \boldsymbol{B}^{(1)^T}\boldsymbol{B}^{(1)} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{B}^{(2)^T}\boldsymbol{B}^{(2)} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{B}^{(K)^T}\boldsymbol{B}^{(K)} \end{bmatrix} \tag{5.99}$$

Then we have

$$Q(\boldsymbol{y}) = \boldsymbol{x}^T\boldsymbol{x} - 2\boldsymbol{x}^T\boldsymbol{B}\boldsymbol{y} + \lambda\boldsymbol{y}^T\boldsymbol{y} + \left(1 - \frac{\eta}{K}\right)\boldsymbol{y}^T\boldsymbol{B}^T\boldsymbol{B}\boldsymbol{y} + \eta\boldsymbol{y}^T\boldsymbol{L}\boldsymbol{y} \tag{5.100}$$

The derivative of $Q(\boldsymbol{y})$ with respect to $\boldsymbol{y}$ is

$$\frac{\partial Q(\boldsymbol{y})}{\partial \boldsymbol{y}} = -2\boldsymbol{B}^T\boldsymbol{x} + 2\lambda\boldsymbol{y} + \left(2 - \frac{2\eta}{K}\right)\boldsymbol{B}^T\boldsymbol{B}\boldsymbol{y} + 2\eta\boldsymbol{L}\boldsymbol{y} \tag{5.101}$$

Setting the derivative to zero yields the optimal value of $\boldsymbol{y}$, which is $\boldsymbol{X}_{DCR}$ as given by (5.102). We can see that DCR is the generalization of CR, and the former becomes the latter when $\eta = 0$.

$$\boldsymbol{X}_{DCR} = (\lambda\boldsymbol{I} + \frac{K-\eta}{K}\boldsymbol{B}^T\boldsymbol{B} + \eta\boldsymbol{L})^{-1}\boldsymbol{B}^T\boldsymbol{x} \tag{5.102}$$

5.6.4 Minimum Residual-based Classifier

After obtaining the SR, CR or DCR corresponding to the feature vector $\boldsymbol{x}$, the next step is to perform classification. Both the SR-based classifier (SRC) [60] and the CR-based classifier (CRC) [69] can be treated as a minimum residual-based classifier (MRC), which carries out classification based on the reconstruction errors when using different sub dictionaries to approximate the original feature vector. The residual with respect to the k-th sub dictionary, denoted as r_k, is given by (5.103), where $\boldsymbol{X}_R$ represents SR, CR or DCR, and $\boldsymbol{X}_R^{(k)}$ is the sub coefficient vector in $\boldsymbol{X}_R$ that corresponds to the basis vectors of the k-th sub dictionary.

$$r_k = \left\|\boldsymbol{x} - \boldsymbol{B}^{(k)}\boldsymbol{X}_R^{(k)}\right\|_2^2 \tag{5.103}$$

The predicted label $\ell(\boldsymbol{x})$ is then the index of the class with the minimum residual as given by

$$\ell(\boldsymbol{x}) = \underset{k}{\operatorname{argmin}}\, r_k \tag{5.104}$$

5.7 Comparison of SVM, PLDA and MRC

SVM is a discriminative classification model, which draws hyperplanes in the feature space to separate feature vectors into different classes. This means that it requires the feature vectors to be spatially separable. If there are overlaps between feature vectors belonging to different classes, SVM will probably fail. The separability of the feature vectors also implicitly assumes a high dimensionality of the feature vectors, as high-dimensional feature vectors have a higher chance to be separable.

PLDA is a probabilistic model, which describes the distribution of the feature vectors using a latent variable model. It assumes that the feature vectors are generated by latent variables following a standard Gaussian distribution, and thus should be able to be decomposed into low-dimensional latent vectors. The latent variable model also requires the feature vectors to be highly structured, such that the decomposition is plausible. Nevertheless, feature vectors belonging to different classes can be overlapped, as PLDA does classification based on the probabilities of belonging to different classes, instead of the spatial locations.

MRC is a dictionary-based classifier, which assumes that the feature vector can be decomposed as a linear combination of the basis vectors in the dictionary. This assumption requires a set of high-quality basis vectors. It also implicitly assumes a high dimensionality of the feature vectors, so as to successfully perform the decomposition. Similar to PLDA, MRC does not require the feature vectors to be separable.

Chapter 6 Classifiers: Experimental Comparison

6.1 GMM vs. RBM

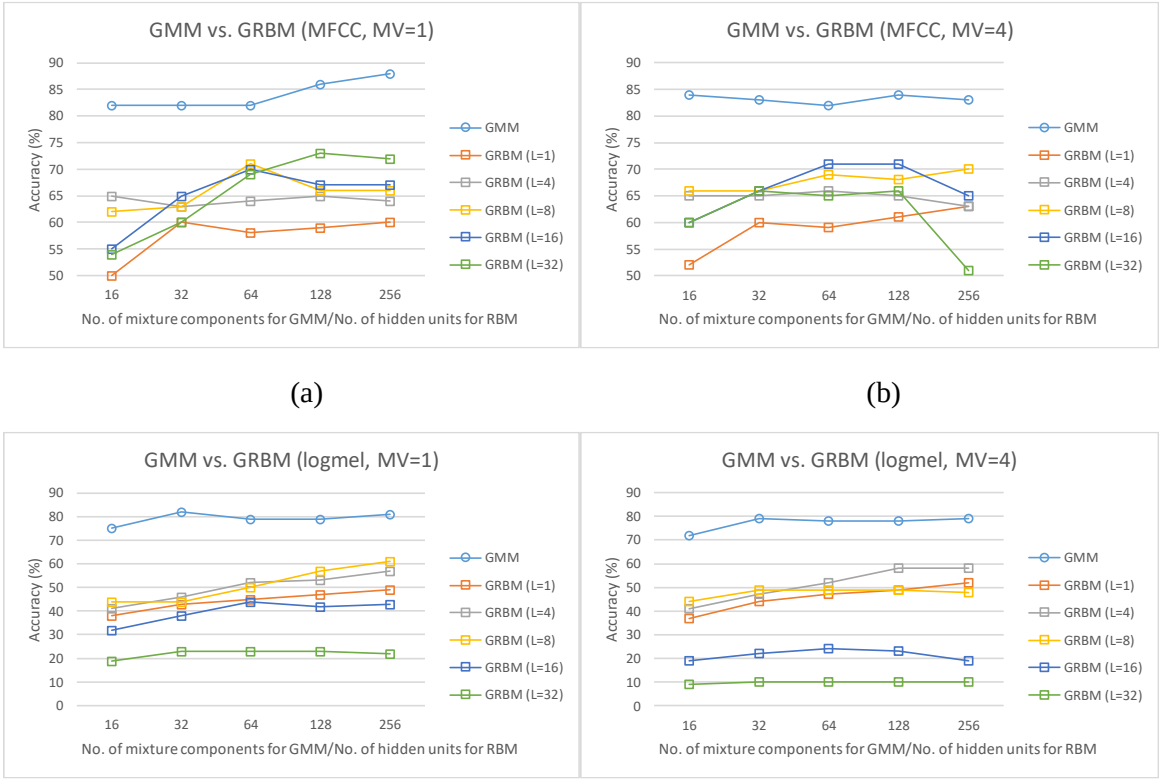
This section compares the performance of GMM and RBM in an acoustic scene classification task using the DCASE2013 dataset. Each acoustic sample is transformed into a sequence of 20-dimensional MFCC vectors or 40-dimensional logmel vectors (logmel is simply MFCC without DCT) using the Hamming window with 40ms frame length and 20ms frame shift. As each acoustic sample is corresponding to a sequence of feature vectors (i.e., MFCC vectors or logmel vectors), the classifier will produce a sequence of predicted labels correspondingly. We then adopt the first majority voting (MV) strategy as given by (1.2), which is the simplest strategy generally applicable, and the fourth MV strategy as given by (1.5), which applies to probabilistic models such as GMM and RBM. Regarding RBM, the dimensionality of the feature vector should be high, which is not satisfied by the low-dimensional MFCC vector or logmel vector. Therefore, we adopt the neighboring feature concatenation strategy as illustrated in (6.1), where $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3 \ldots\}$ is the sequence of MFCC vectors or logmel vectors obtained from an acoustic sample s, $\{\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3 \ldots\}$ is the sequence of concatenated MFCC vectors or concatenated logmel vectors that are used to represent the sample s for training and testing the classifier, L is the concatenation length, and H is the concatenation shift. To reduce the computational burden induced by the higher dimensionality and the similarity of consecutive concatenated vectors, we adopt $H = L/2$. If the dimensionality of an MFCC vector or a logmel vector is $D \times 1$, the dimensionality of a concatenated vector will be $DL \times 1$.

$$\{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3 \ldots\} \quad \Longrightarrow \quad \{\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3 \ldots\} = \left\{ \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_L \end{bmatrix}, \begin{bmatrix} \boldsymbol{x}_{1+H} \\ \boldsymbol{x}_{2+H} \\ \vdots \\ \boldsymbol{x}_{L+H} \end{bmatrix}, \begin{bmatrix} \boldsymbol{x}_{1+2H} \\ \boldsymbol{x}_{2+2H} \\ \vdots \\ \boldsymbol{x}_{L+2H} \end{bmatrix} \ldots \right\} \tag{6.1}$$

6.1.1 RBM as A Probabilistic Model

This sub-section evaluates the performance of the heterogenous RBM, which is trained based on maximizing the log-likelihood, similar to the training process of GMM. The ability of GMM and RBM as a probability estimator can then be compared by observing their classification results.

The experimental results are shown in Figure 6.1, where Figure 6.1 (a) and (b) illustrate the results of using MFCC as the frame-level feature vector, and Figure 6.1 (c) and (d) illustrate the results of using logmel as the frame-level feature vector. The horizontal axis indicates the number of Gaussian components if the classifier is GMM, or the number of hidden units if the classifier is RBM. As the feature vectors have real values, we use GRBM and normalize the feature vectors to have a zero mean and unit covariance using the training data. The GRBM is trained for 500 epochs with a learning rate of 0.01 and a momentum of 0.9, implemented using the DeepLearnToolbox [93]. The GMM is trained



(a) (b)

(c) (d)

Figure 6.1 GMM vs. GRBM for acoustic scene classification. (a) MFCC with the first MV strategy. (b) MFCC with the fourth MV strategy. (c) Logmel with the first MV strategy. (d) Logmel with the fourth MV strategy.

using the EM algorithm together with the mixture splitting strategy II, implemented using Voicebox [94].

As shown in Figure 6.1, GRBM gives a rather worse performance than GMM, implying that GRBM is not good at working as a probabilistic model for acoustic signals. There are three possible reasons that may explain the poor performance of using RBM as a probability estimator. First, the energy-based model nature of RBM may not well describe the characteristics of acoustic features. Second, in order to apply RBM for probability estimation, a concatenated feature vector (e.g., a concatenated MFCC vector or a concatenated logmel vector) and its corresponding label vector (which is a one-hot vector) are jointly trained as a heterogeneous RBM. Since the feature vector has real values but the label vector has binary values, the relationship between different data types may not be well learned by the heterogenous RBM. In addition, the relationship between the feature vector and the label vector may not be well modeled by the linear transformation as given by (5.31) (the energy function assumes that the feature vector and the label vector are linearly related by the weight matrix and the hidden units). Third, GMM estimates the probability in an unsupervised manner, while RBM estimates the probability in a supervised manner (because the feature vector and the label vector are concatenated as the input of RBM). It seems unsupervised probability estimation is better than supervised probability estimation.

The two MV strategies give very similar performances. Nevertheless, the first MV strategy seems slightly better, even though the fourth MV strategy is specially designed for probabilistic models. In addition, a higher dimensionality of the feature vector is vital to GRBM, as $L = 1$ gives the worst performance when MFCC is used as the frame-level feature vector (Figure 6.1 (a) and (b)) (The larger the L, the higher the dimensionality). On the other hand, the dimensionality of the feature vector should not be too high, as $L = 32$ gives the worst performance when logmel is used as the frame-level feature vector (Figure 6.1 (c) and (d)). GRBM assumes that the elements in a feature vector are correlated. By concatenating neighboring frame-level feature vectors (i.e., increasing L), GRBM learns the relationship between adjacent frame-level feature vectors, and thus may provide a better performance.

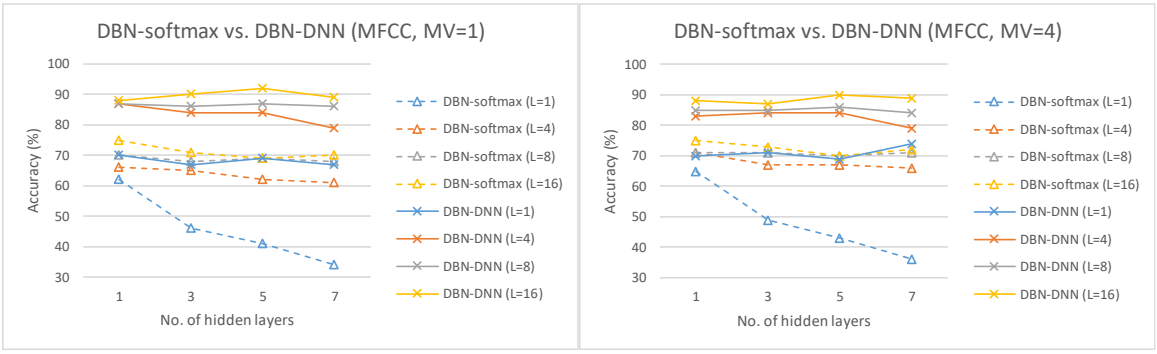
By comparing the top two plots with the bottom two plots in Figure 6.1, MFCC seems outperforming logmel, no matter employing GMM or GRBM as the classifier. Since the elements in an MFCC vector

are decorrelated due to the DCT operation, MFCC fits well to the model assumption of GMM. Since the elements in a logmel vector may be correlated, the logmel vector may not follow a multivariate Gaussian distribution. This violates not only the model assumption of GMM, but also the Gaussian assumption of the visible units in GRBM.

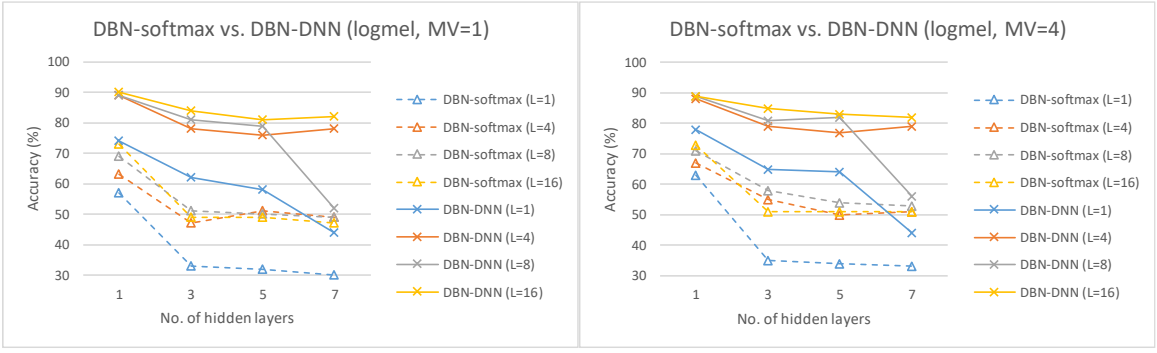
6.1.2 Discriminative Model based on RBM

Besides being directly used for probability estimation, RBM can also be used as the building block of a DBN, which can then be fine-tuned in a supervised manner.

This sub-section compares two discriminative models based on DBN. The first model, named as DBN-softmax, is built by first training a DBN and then training the last softmax layer with the parameters of the DBN fixed. Hence, the DBN is used to produce feature representations, while the softmax layer is used for doing classification. The second model, named as DBN-DNN, is built by further fine-tuning the parameters of the DBN-softmax using backpropagation. The DBN is trained as a series of RBMs



(a) (b)



(c) (d)

Figure 6.2 DBN-softmax vs. DBN-DNN for acoustic scene classification. (a) MFCC with the first MV strategy. (b) MFCC with the fourth MV strategy. (c) Logmel with the first MV strategy. (d) Logmel with the fourth MV strategy.

where each RBM is trained for 500 epochs with a learning rate of 0.01 and a momentum of 0.9. The softmax layer is trained for 100 epochs with a learning rate of 0.1 and a momentum of 0.9. The DBN-DNN is fine-tuned for 500 epochs with a learning rate of 0.1 and a momentum of 0.9. The number of neurons in each hidden layer of DBN-softmax and DBN-DNN is 256. The experimental results using MFCC and logmel as the frame-level feature vector are shown in Figure 6.2, where DBN-softmax and DBN-DNN are investigated with different numbers of hidden layers.

It can be seen that DBN-DNN significantly outperforms DBN-softmax. This indicates that DBN trained in an unsupervised manner does not produce an expressive feature representation. Increasing the number of hidden layers even further degrades the quality of the feature representation, but the degradation with increasing layers may be mitigated by increasing the dimensionality of the feature vector (i.e., increasing the value of L). This observation validates the importance of feature dimensionality to DBN-softmax and DBN-DNN, especially when they have a very deep structure. In other words, if the feature vector has a low dimensionality, it may not provide enough information to the DNN, and consequently, the deeper layers cannot learn useful representations.

6.2 PLDA vs. SVM

6.2.1 Scalable PLDA vs. Original PLDA

This sub-section validates the computational speedup achieved by the scalable formulation of PLDA in an acoustic scene classification task using the DCASE2013 dataset. The feature representation is GSV. UBMs with the number of mixture components varying from 2 to 64 are used to construct the GSV, so the dimensionality of the GSV varies from 40×1 to 1280×1. The scaling factor β used to compute the GSV is set to 0.1. The dimensionality of the latent vectors in the PLDA model is the same as that of the GSV, and the number of EM iterations for model parameter estimation is 1.

The computation time for model parameter estimation and class label prediction is measured by running the MATLAB codes on a Windows 10 laptop with 8G memory. The time consumption of the original and scalable formulations is illustrated in Figure 6.3. As can be seen from the figure, the computational efficiency of the scalable formulation is much higher than that of the original formulation, especially

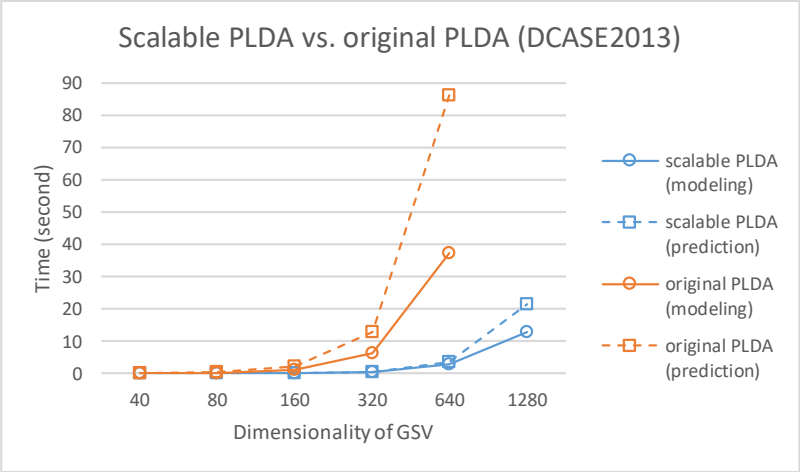


Figure 6.3 Time consumption of original and scalable PLDA.

when the dimensionality of the feature representation is high. Since the original formulation requires inverting large matrices whose sizes are proportional to the quantity of the training data and the dimensionality of the feature representation, either a large quantity or a high dimensionality will make the computation inefficient, or even infeasible if exceeding the memory space.

6.2.2 PLDA vs. SVM Using MFCC as the Feature

This sub-section compares the performance of PLDA and SVM in an acoustic scene classification task using the DCASE2013 dataset, with MFCC being the feature representation. Each acoustic sample is transformed into a sequence of 20-dimensional MFCC vectors, and thus the classifier will produce a sequence of predicted labels for each acoustic sample. Only the first MV strategy is adopted to handle multiple predicted labels, because PLDA or SVM does not produce a probability associated with a predicted label. We also adopt the neighboring feature concatenation strategy as given by (6.1), which may increase the amount of information a feature vector carries.

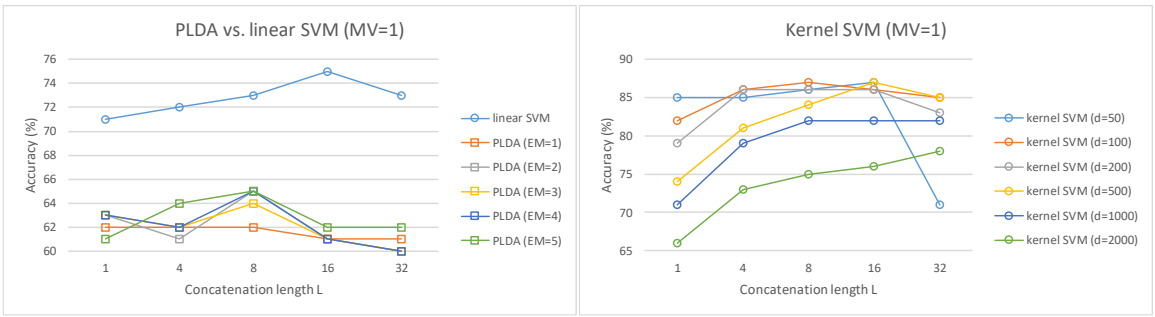
The dimensionality of the latent vectors in a PLDA model is the same as that of the feature vector, and the number of EM iterations for model parameter estimation is varied from 1 to 5. SVM is implemented using LIBSVM [85] with default parameters. In particular, we consider both the linear version and the kernel version. The kernel SVM uses a Gaussian kernel as given by (6.2), where $\ker(.,.)$ is a kernel function, $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ represent two feature vectors in the original feature space, $\phi(\boldsymbol{x}_i)$ and $\phi(\boldsymbol{x}_j)$ represent the two mapped feature vectors in the kernel space, and d is the kernel parameter. The

advantage of using a kernel is that the original feature vector can be mapped to a higher-dimensional space by exploiting the relationship between the elements in a feature vector (i.e., the dimensionality of $\boldsymbol{\phi}(\boldsymbol{x}_i)$ will be higher than that of $\boldsymbol{x}_i$). A high dimensionality may benefit SVM, as high-dimensional feature vectors will have a higher chance to be linearly separable. In particular, when using the Gaussian kernel, $\boldsymbol{\phi}(\boldsymbol{x}_i)$ will have infinite dimensions [106].

$$\ker(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{\phi}(\boldsymbol{x}_i)^T \boldsymbol{\phi}(\boldsymbol{x}_j) = \exp\left\{-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{d}\right\} \tag{6.2}$$

The experimental results are shown in Figure 6.4. As can be seen from Figure 6.4 (a), both PLDA and the linear SVM give a poor performance when MFCC vectors are directly used for classification. PLDA gives a rather worse performance, probably owing to the violation of the model assumption. The linear SVM gives a relatively better performance, probably owing to the simplicity of the model assumption. As can be seen from Figure 6.4 (b), the kernel SVM has significantly better performance than the linear SVM, as the original feature vector is mapped to a higher-dimensional space in the kernel SVM.

It is also noticed that increasing the dimensionality of the concatenated MFCC vector (i.e., increasing L) may improve the performance of both PLDA and SVM, but a larger value of L may not be better. The value of L seems to have a higher influence on the performance of the kernel SVM than that of the linear SVM. Since the kernel SVM exploits the intra-feature characteristics (i.e., the relationship between different elements in a feature vector), increasing L will include more neighboring feature vectors into a single concatenated feature vector, which then enables the kernel SVM to exploit the



(a) (b)

Figure 6.4 PLDA vs. SVM using MFCC as the feature representation. (a) Performance of PLDA and the linear SVM. (b) Performance of the kernel SVM.

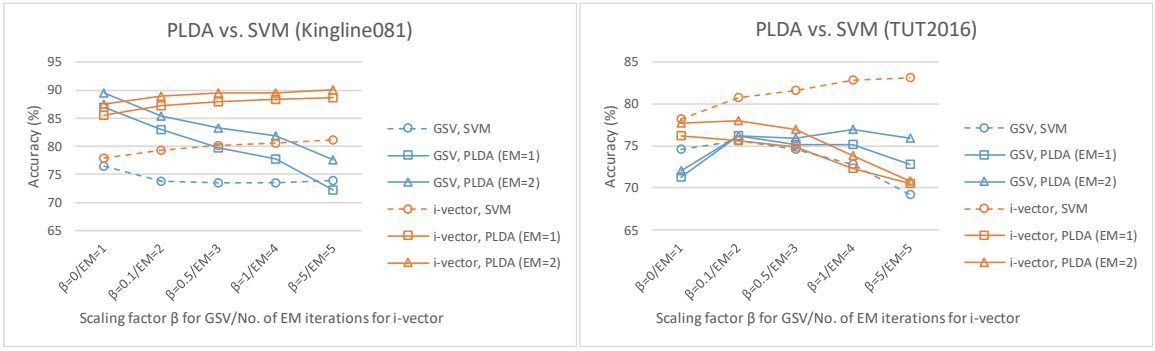
inter-feature characteristics too (because multiple feature vectors are combined into a single feature vector).

In addition, the effectiveness of increasing L also highly depends on the choice of the kernel parameter when the kernel SVM is used. The best choice of L is seemingly proportional to the choice of the kernel parameter d.

6.2.3 PLDA vs. SVM Using GSV and I-vector as the Feature

This sub-section compares the performance of PLDA and SVM in a speaker identification task using the Kingline081 dataset and an acoustic scene classification task using the TUT2016 dataset, with GSV and i-vector being the feature representation. The 1280-dimensional GSV is obtained based on a 64-mixture UBM, with the scaling factor β varying from 0 to 5. The 1280-dimensional i-vector is obtained based on a 64-mixture UBM, with the number of EM iterations varying from 1 to 5. The dimensionality of the latent vectors in the PLDA model is 1280×1, and the number of EM iterations for model parameter estimation varies from 1 to 2. The SVM is a linear SVM implemented using LIBSVM [85] with default parameters.

The experimental results are illustrated in Figure 6.5. As shown in Figure 6.5 (a), PLDA significantly outperforms the linear SVM for speaker identification no matter the feature representation is GSV or i-vector. However, as shown in Figure 6.5 (b), PLDA may not outperform the linear SVM for acoustic scene classification, and the linear SVM even significantly outperforms PLDA when the feature



(a) (b)

Figure 6.5 PLDA vs. SVM using GSV and i-vector as the feature representation. (a) Results on Kingline081. (b) Results on TUT2016.

representation is i-vector. The different behaviors of PLDA on different tasks are probably caused by the different feature distributions of different datasets.

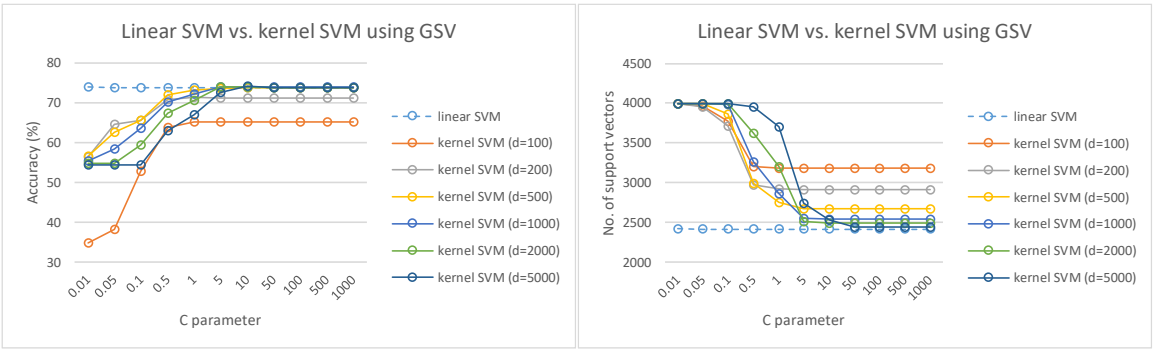
First, the MFCC vectors obtained from speech signals carry rich information and are probably following the Gaussian distribution, and therefore the GSV and the i-vector are probably of good quality, since they are constructed from a GMM-based UBM. The MFCC vectors obtained from non-speech acoustic signals may not carry as rich information as those obtained from human speeches, and thus the GSV and the i-vector may not carry enough information.

Second, the GSV and the i-vector obtained from non-speech acoustic signals may violate the assumption of the PLDA model. A PLDA model assumes that the feature representation is generated by two latent vectors. This implicitly assumes that the feature representation should be highly structured and carry rich information, such that it can be well described by a latent variable model.

6.2.4 Linear SVM vs. Kernel SVM Using GSV and I-vector as the Feature

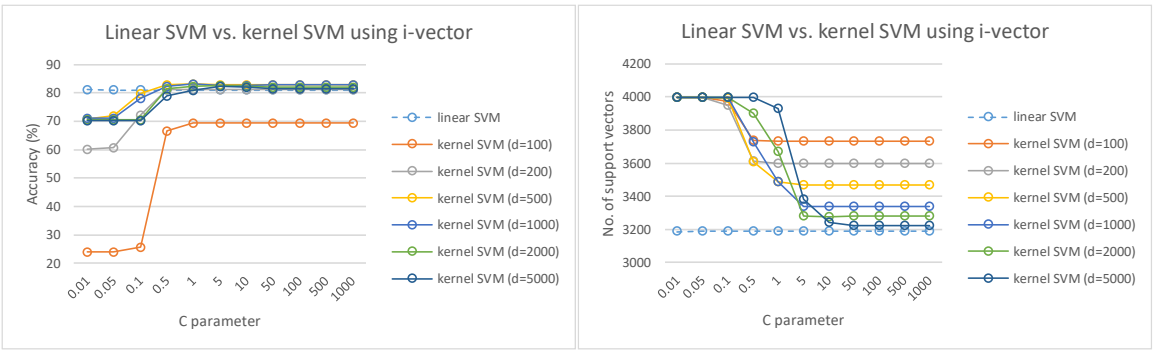
This sub-section compares the performance of the linear SVM and the kernel SVM in a speaker identification task using the Kingline081 dataset, with GSV and i-vector being the feature representation. The kernel SVM uses a Gaussian kernel as given by (6.2). The GSV has a dimensionality of 1280×1, computed with $\beta = 0.1$ based on a 64-mixture UBM. The i-vector has a dimensionality of 1280×1, computed with 5 EM iterations based on a 64-mixture UBM. For both the linear SVM and the kernel SVM, the parameter C varies from 0.01 to 1000. For the kernel SVM, the kernel parameter d varies from 100 to 5000.

The identification results using GSV and i-vector are shown in Figure 6.6 and Figure 6.7 respectively. The performance of the linear SVM and the kernel SVM is compared by varying the model parameters, and the number of support vectors under different model parameters are also recorded. As shown in Figure 6.6 and Figure 6.7, the highest identification accuracy achieved by the linear SVM and the kernel SVM is quite similar. Although the kernel SVM can map the original feature representations to a higher dimensional space where the mapped high-dimensional feature representations have a higher chance to



(a) (b)

Figure 6.6 Linear SVM vs. kernel SVM using GSV as the feature representation. (a) The identification accuracy under different parameters. (b) The number of support vectors under different parameters.



(a) (b)

Figure 6.7 Linear SVM vs. kernel SVM using i-vector as the feature representation. (a) The identification accuracy under different parameters. (b) The number of support vectors under different parameters.

be linearly separable, the mapping may be unnecessary in this case, as the original feature representation (i.e., the GSV or the i-vector) already has a high dimensionality.

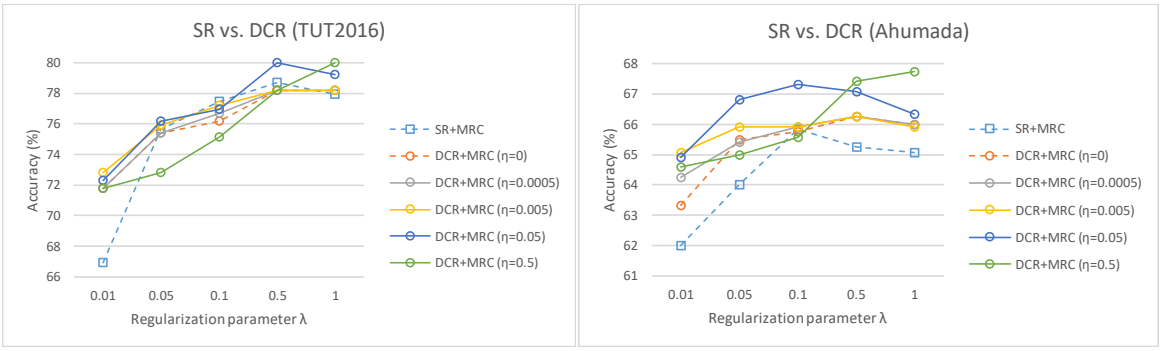
It can be seen that the performance of the linear SVM is insensitive to the value of the parameter C, whereas the performance of the kernel SVM is sensitive to the value of the parameter C. For the kernel SVM, the experimental results show that the larger the value of C, the better the performance. This observation can be explained as follows. On using the kernel SVM, the original feature representation is mapped to a higher dimensional space. In the case of Gaussian kernel, the space has infinite dimensions. In that extremely high dimensional space, the mapped feature representations can be easily separated by a hyperplane. Thus, the value of C should be large enough to ensure that most mapped training data are lying on the correct side of the separating hyperplane.

We also notice that the performance of the kernel SVM is strongly related to the number of support vectors. It seems that the fewer the support vectors, the better is the performance. This is probably because that the more the support vectors, the higher is the chance that SVM overfits the training data.

6.3 SR vs. DCR

This section briefly compares the performance of SR, CR, and DCR in an acoustic scene classification task using the TUT2016 dataset, and a speaker identification task using the Ahumada dataset. The feature representation is the 1280-dimensional i-vector based on a 64-mixture UBM. The model parameters used to compute the i-vector is estimated with 5 EM iterations. The SR is obtained using SparseLab [103], while the CR and the DCR are implemented using MATLAB. The classifier is MRC, and the training and testing feature representations are normalized to have a unit length.

The classification performance of SR, CR, and DCR is illustrated in Figure 6.8. It should be noted that CR is DCR with $\eta = 0$. The time consumption of computing the SR with $\lambda = 1$ and the time



(a) (b)

Figure 6.8 SR vs. DCR for acoustic and speech signal classification. (a) Results on TUT2016. (b) Results on Ahumada.

Table 6.1 Time consumption for SR and DCR (in seconds).

	TUT2016	**Ahumada**
SR+MRC ($\lambda = 1$)	908s	2868s
DCR+MRC ($\lambda = 1, \eta = 1$)	9s	16s

consumption of computing the DCR with $\lambda = 1$ and $\eta = 1$ are given in Table 6.1, which is obtained by running the MATLAB codes on a Windows 10 laptop with 8G memory.

As shown in Figure 6.8, SR slightly outperforms CR for acoustic scene classification (Figure 6.8 (a)), whereas CR slightly outperforms SR for speaker identification (Figure 6.8 (b)). With a suitably chosen value of η, DCR outperforms both SR and CR for both tasks. This validates the usefulness of the additional regularization term involved in the objective function of DCR. More importantly, the computation of DCR is much more efficient than that of SR, as shown in Table 6.1, since DCR has an analytic solution while SR does not.

6.4 Experimental Comparison of SVM, PLDA and MRC

This section compares the performance of the linear SVM, PLDA, and MRC in two acoustic scene classification tasks using the DCASE2013 and TUT2016 datasets, and two speaker identification tasks using the Ahumada and Kingline081 datasets. The feature representation is the 1280-dimensional i-vector based on a 64-mixture UBM. The model parameters used to compute the i-vector is estimated with 5 EM iterations. On using MRC, the feature representation is normalized to have a unit length, and the DRC is computed with different values of λ and η. SVM is implemented using LIBSVM [85] with $C = 1$, as varying the value of C does not make any significant difference. PLDA is implemented using the scalable formulation, and the latent vectors in the PLDA model have the same dimensionality as the feature representation. The model parameters are estimated by 1 and 2 EM iterations, as more EM iterations do not help.

The classification accuracies of using different classifiers are illustrated in Figure 6.9. The time consumption, including that for model parameter estimation and class label prediction, is illustrated in Figure 6.10, where DCR is computed with $\lambda = 1$ and $\eta = 1$, and the parameters of the PLDA model are estimated with 1 EM iteration. The time is estimated by running the MATLAB codes on a Windows 10 laptop with 8G memory.

As shown in Figure 6.9, SVM achieves the highest accuracy on TUT2016, PLDA achieves the highest accuracy on Kingline081, and DCR+MRC achieves the highest accuracy on DCASE2013 and

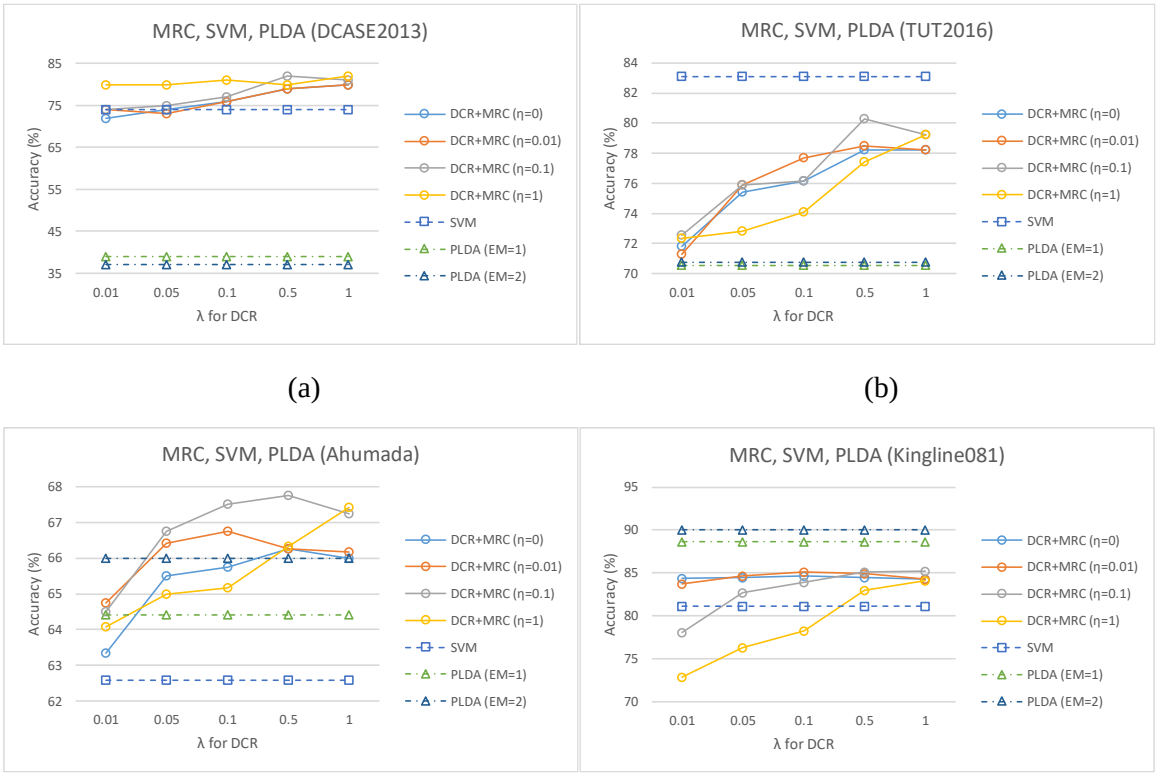


Figure 6.9 Classification accuracy of DCR+MRC, SVM and PLDA on different datasets. (a) Results on DCASE2013. (b) Results on TUT2016. (c) Results on Ahumada. (d) Results on Kingline081.

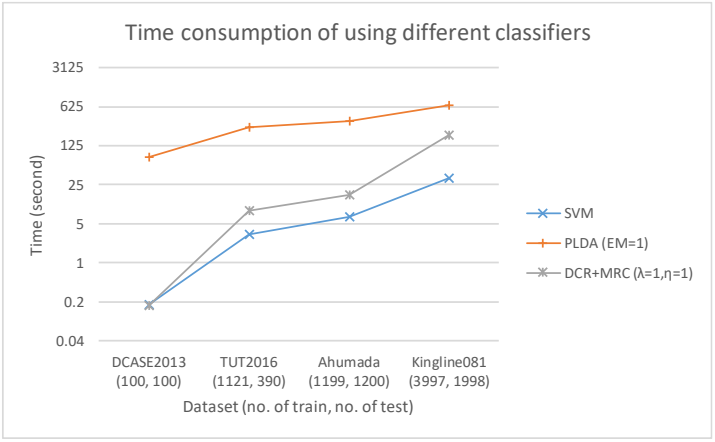


Figure 6.10 Time consumption of DCR+MRC, SVM and PLDA.

Ahumada. Therefore, in general, there is no one classifier always outperforming the others. It is then important to choose a suitable classifier for a specific pattern recognition task or a specific dataset.

PLDA seems to work well on speech signals, as it gives the best performance on the Kingline081 speech dataset and approaches the best performance on the Ahumada speech dataset, but work poorly on non-

speech acoustic signals. PLDA assumes the feature representations to follow a Gaussian distribution and conform to a factor analysis model, which may not be satisfied by non-speech acoustic signals. Although PLDA is suitable for speech signals, its time consumption is high even if there is only 1 EM iteration as shown in Figure 6.10. More EM iterations will then consume more time.

DCR+MRC generally works well on different datasets. It gives the best performance on DCASE2013 and Ahumada, and approaches the best performance on TUT2016 and Kingline081. Unlike SVM and PLDA which require estimating the model parameters, after having the DCR for each feature representation, MRC directly makes predictions using DCR instead of building any classification model. This results in high computational efficiency. As the MATLAB codes for DCR are not optimized, the computation takes more time than SVM, but is still more efficient than PLDA.

SVM is generally applicable to different tasks such as speaker identification and acoustic scene classification, and is very fast in computation when using open source tools, such as LIBSVM. It assumes the data are separable either in the original feature space or in the kernel space, but this assumption is difficult to fulfill in real scenarios. It may not work very well, but will not work very badly either.

Chapter 7 Feature Projection Techniques: Theoretical Analysis

7.1 Fisher Discriminant Analysis

This section discusses Fisher discriminant analysis (FDA) and its variants, including the linear discriminant analysis (LDA) and the kernel discriminant analysis (KDA). Specifically, two versions of LDA will be explained, and their corresponding kernel-based formulations are derived. A brief comparison between the two versions is also presented.

7.1.1 Linear Discriminant Analysis

Given a set of training vectors denoted as $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots \boldsymbol{x}_N\}$, where N is the number of training data, and the dimensionality of a training vector is $D \times 1$. Suppose these training vectors belong to K different classes, and there are N_k training vectors belonging to class k, denoted as c_k. The objective of LDA is to find a projection matrix $\boldsymbol{V}$ with a size of $D \times P$ such that the projected vectors in the same class are closer to their center, while the centers of different classes are farther from each other. Denoting the corresponding projected vectors as $\{\boldsymbol{x}'_1, \boldsymbol{x}'_2, \dots \boldsymbol{x}'_N\}$, the objective of LDA can then be achieved by reducing the within-class covariance $\boldsymbol{C}_W$ and increasing the between-class covariance $\boldsymbol{C}_B$ as defined in (7.1), where $\boldsymbol{\mu}$ is the mean of all the projected vectors, and $\boldsymbol{\mu}_k$ is the mean of the projected vectors belonging to class k [104].

$$\begin{aligned} \boldsymbol{\mu}_k &= \frac{1}{N_k}\sum_{\boldsymbol{x}_n \in c_k} \boldsymbol{x}'_n \\ \boldsymbol{\mu} &= \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}'_n \\ \boldsymbol{C}_W &= \sum_{k=1}^{K}\sum_{\boldsymbol{x}_n \in c_k}(\boldsymbol{x}'_n - \boldsymbol{\mu}_k)(\boldsymbol{x}'_n - \boldsymbol{\mu}_k)^T \\ \boldsymbol{C}_B &= \sum_{k=1}^{K} N_k(\boldsymbol{\mu}_k - \boldsymbol{\mu})(\boldsymbol{\mu}_k - \boldsymbol{\mu})^T \end{aligned} \tag{7.1}$$

Considering that $\boldsymbol{x}'_n = \boldsymbol{V}^T\boldsymbol{x}_n$, $\boldsymbol{C}_W$ and $\boldsymbol{C}_B$ can be expressed in terms of $\boldsymbol{V}$ as given by (7.2), where $\boldsymbol{S}_W$ is the within-class scatter matrix and $\boldsymbol{S}_B$ is the between-class scatter matrix [104].

$$\begin{aligned} \boldsymbol{C}_W &= \boldsymbol{V}^T\boldsymbol{S}_W\boldsymbol{V} \\ \boldsymbol{C}_B &= \boldsymbol{V}^T\boldsymbol{S}_B\boldsymbol{V} \end{aligned} \tag{7.2}$$

where

$$\begin{aligned} \boldsymbol{m}_k &= \frac{1}{N_k}\sum_{\boldsymbol{x}_n \in c_k} \boldsymbol{x}_n \\ \boldsymbol{m} &= \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n \\ \boldsymbol{S}_W &= \sum_{k=1}^{K}\sum_{\boldsymbol{x}_n \in c_k}(\boldsymbol{x}_n - \boldsymbol{m}_k)(\boldsymbol{x}_n - \boldsymbol{m}_k)^T \\ \boldsymbol{S}_B &= \sum_{k=1}^{K} N_k(\boldsymbol{m}_k - \boldsymbol{m})(\boldsymbol{m}_k - \boldsymbol{m})^T \end{aligned} \tag{7.3}$$

7.1.1.1 LDA Version 1

The objective of LDA can be realized by maximizing the objective function $Q(\boldsymbol{V})$ given by (7.4), where tr{.} computes the trace of the matrix.

$$Q(\boldsymbol{V}) = \text{tr}\{(\boldsymbol{V}^T\boldsymbol{S}_W\boldsymbol{V})^{-1}(\boldsymbol{V}^T\boldsymbol{S}_B\boldsymbol{V})\} \tag{7.4}$$

Maximizing $Q(\boldsymbol{V})$ defined in (7.4) is equivalent to the optimization problem given by (7.5) [104][105], where $\boldsymbol{v}_p$ is the p-th column vector of $\boldsymbol{V}$, which is a projection direction.

$$\begin{aligned} &\max \text{tr}\{\boldsymbol{V}^T\boldsymbol{S}_B\boldsymbol{V}\} \\ &s.t. \\ &\boldsymbol{V}^T\boldsymbol{S}_W\boldsymbol{V} = \boldsymbol{I} \end{aligned} \quad \Leftrightarrow \quad \begin{aligned} &\max \sum_{p=1}^{P} \boldsymbol{v}_p^T\boldsymbol{S}_B\boldsymbol{v}_p \\ &s.t. \\ &\boldsymbol{v}_p^T\boldsymbol{S}_W\boldsymbol{v}_p = 1 \;\; for \;\; p = 1,2 \dots P \end{aligned} \tag{7.5}$$

The constrained optimization problem in (7.5) can be reformulated using a Lagrangian function $L(\boldsymbol{V}, \lambda)$ as given by (7.6), where $\{\lambda_1, \lambda_2 \dots \lambda_P\}$ are Lagrange multipliers.

$$L(\boldsymbol{V}, \lambda) = \sum_{p=1}^{P} \boldsymbol{v}_p^T\boldsymbol{S}_B\boldsymbol{v}_p - \sum_{p=1}^{P} \lambda_p(\boldsymbol{v}_p^T\boldsymbol{S}_W\boldsymbol{v}_p - 1) \tag{7.6}$$

Setting the derivative of $L(\boldsymbol{V}, \lambda)$ to zero gives (7.7), which means $\boldsymbol{v}_p$ is an eigenvector of $\boldsymbol{S}_W^{-1}\boldsymbol{S}_B$.

$$\frac{\partial L(\boldsymbol{V},\lambda)}{\partial \boldsymbol{v}_p} = 2\boldsymbol{S}_B\boldsymbol{v}_p - 2\lambda_p\boldsymbol{S}_W\boldsymbol{v}_p = \boldsymbol{0} \quad \Longrightarrow \quad \boldsymbol{S}_B\boldsymbol{v}_p = \lambda_p\boldsymbol{S}_W\boldsymbol{v}_p \quad \Longrightarrow \quad \boldsymbol{S}_W^{-1}\boldsymbol{S}_B\boldsymbol{v}_p = \lambda_p\boldsymbol{v}_p \tag{7.7}$$

7.1.1.2 LDA Version 2

Besides the objective function defined in (7.4), another expression of the objective function can be adopted to realize the target that the within-class covariance of the projected vectors is reduced and the between-class covariance is increased. The new objective function is defined in (7.8), where α is a non-negative weighting coefficient balancing the importance of the within-class covariance and the between-class covariance.

$$Q(\boldsymbol{V}) = \mathrm{tr}\{\boldsymbol{V}^T \boldsymbol{S}_B \boldsymbol{V}\} - \mathrm{tr}\{\alpha \boldsymbol{V}^T \boldsymbol{S}_W \boldsymbol{V}\} \tag{7.8}$$

In order to prevent the solution from being infinity, it is necessary to apply the unit-length constraint. The optimization problem then becomes (7.9).

$$\begin{matrix} \max \mathrm{tr}\{\boldsymbol{V}^T \boldsymbol{S}_B \boldsymbol{V}\} - \mathrm{tr}\{\alpha \boldsymbol{V}^T \boldsymbol{S}_W \boldsymbol{V}\} \\ s.t. \\ \boldsymbol{V}^T \boldsymbol{V} = \boldsymbol{I} \end{matrix} \Leftrightarrow \begin{matrix} \max \sum_{p=1}^{P} \boldsymbol{v}_p^T (\boldsymbol{S}_B - \alpha \boldsymbol{S}_W) \boldsymbol{v}_p \\ s.t. \\ \boldsymbol{v}_p^T \boldsymbol{v}_p = 1 \ \ for \ \ p = 1,2 \dots P \end{matrix} \tag{7.9}$$

The corresponding Lagrangian function $L(\boldsymbol{V}, \lambda)$ is then given by (7.10), where $\{\lambda_1, \lambda_2 \dots \lambda_P\}$ are Lagrange multipliers.

$$L(\boldsymbol{V}, \lambda) = \sum_{p=1}^{P} \boldsymbol{v}_p^T (\boldsymbol{S}_B - \alpha \boldsymbol{S}_W) \boldsymbol{v}_p - \sum_{p=1}^{P} \lambda_p (\boldsymbol{v}_p^T \boldsymbol{v}_p - 1) \tag{7.10}$$

Setting the derivative of $L(\boldsymbol{V}, \lambda)$ to zero yields (7.11), meaning that $\boldsymbol{v}_p$ is an eigenvector of $(\boldsymbol{S}_B - \alpha \boldsymbol{S}_W)$.

$$\frac{\partial L(\boldsymbol{V}, \lambda)}{\partial \boldsymbol{v}_p} = 2(\boldsymbol{S}_B - \alpha \boldsymbol{S}_W) \boldsymbol{v}_p - 2\lambda_p \boldsymbol{v}_p = \boldsymbol{0} \quad \Longrightarrow \quad (\boldsymbol{S}_B - \alpha \boldsymbol{S}_W) \boldsymbol{v}_p = \lambda_p \boldsymbol{v}_p \tag{7.11}$$

7.1.1.3 Brief Comparison between the Two Versions of LDA

In the following, LDA (v1) refers to LDA version 1, and LDA (v2) refers to LDA version 2. We can see that LDA (v1) requires finding the eigenvectors of $\boldsymbol{S}_W^{-1} \boldsymbol{S}_B$, whereas LDA (v2) requires finding the eigenvectors of $(\boldsymbol{S}_B - \alpha \boldsymbol{S}_W)$. The number of independent eigenvectors (viz. the number of orthogonal projection directions) of LDA (v1) is limited by $\min\{\mathrm{rank}(\boldsymbol{S}_B), \mathrm{rank}(\boldsymbol{S}_W)\}$, whereas that of LDA (v2) is limited by $(\mathrm{rank}(\boldsymbol{S}_B) + \mathrm{rank}(\boldsymbol{S}_W))$. From this perspective, LDA (v2) may find out more orthogonal projection directions than LDA (v1).

Besides, LDA (v1) requires inverting the within-class scatter matrix $\boldsymbol{S}_W$, which may cause the singularity problem if $\boldsymbol{S}_W$ is not full-rank, whereas LDA (v2) does not have this difficulty. From this perspective, LDA (v2) tends to be more stable in terms of numerical computation. Nonetheless, we may slightly modify the objective of LDA (v1) to finding the eigenvectors of $(\boldsymbol{S}_W + \alpha \boldsymbol{I})^{-1}\boldsymbol{S}_B$ where α is a nonnegative regularization parameter. The inclusion of this regularization term helps prevent the singularity problem.

7.1.2 Kernel Discriminant Analysis

The kernel discriminant analysis (KDA) is a generalization of LDA. Assume we would like to first map the original feature vector into another dimensional space, using a mapping function $\phi: \boldsymbol{x} \longmapsto \phi(\boldsymbol{x})$, and then find the projection matrix $\boldsymbol{V}^{(\phi)}$ in the mapped space, where $\boldsymbol{v}_p^{(\phi)}$ is the p-th column vector of $\boldsymbol{V}^{(\phi)}$ and the superscript ϕ denotes the mapped space. The within-class and between-class scatter matrices in the mapped space will then have the form as given by (7.12). The relationship between the original feature vector $\boldsymbol{x}$ and the projected vector $\boldsymbol{x}'$ is $\boldsymbol{x}' = \left(\boldsymbol{V}^{(\phi)}\right)^T \phi(\boldsymbol{x}_n)$.

$$\begin{aligned} \phi(\boldsymbol{m}_k) &= \frac{1}{N_k}\sum_{\boldsymbol{x}_n \in c_k} \phi(\boldsymbol{x}_n) \\ \phi(\boldsymbol{m}) &= \frac{1}{N}\sum_{n=1}^{N} \phi(\boldsymbol{x}_n) \\ \boldsymbol{S}_W^{(\phi)} &= \sum_{k=1}^{K}\sum_{\boldsymbol{x}_n \in c_k}\big(\phi(\boldsymbol{x}_n) - \phi(\boldsymbol{m}_k)\big)\big(\phi(\boldsymbol{x}_n) - \phi(\boldsymbol{m}_k)\big)^T \\ \boldsymbol{S}_B^{(\phi)} &= \sum_{k=1}^{K} N_k\big(\phi(\boldsymbol{m}_k) - \phi(\boldsymbol{m})\big)\big(\phi(\boldsymbol{m}_k) - \phi(\boldsymbol{m})\big)^T \end{aligned} \tag{7.12}$$

Analogous to LDA, KDA should end up solving an eigenvalue problem similar to (7.7) and (7.11), which involve the computation of $\boldsymbol{S}_B^{(\phi)}\boldsymbol{v}_p^{(\phi)}$ and $\boldsymbol{S}_W^{(\phi)}\boldsymbol{v}_p^{(\phi)}$. Before we proceed, we should do some derivations on these two terms. In the derivations, we will make use of the facts given by (7.13).

$$\begin{aligned} \sum_{k=1}^{K} N_k &= N \\ N_k\phi(\boldsymbol{m}_k) &= \sum_{\boldsymbol{x}_n \in c_k} \phi(\boldsymbol{x}_n) \\ \sum_{k=1}^{K} N_k\phi(\boldsymbol{m}_k) &= N\phi(\boldsymbol{m}) \end{aligned} \tag{7.13}$$

$\boldsymbol{S}_B^{(\phi)}\boldsymbol{v}_p^{(\phi)}$ can be expanded as

$$\boldsymbol{S}_B^{(\phi)}\boldsymbol{v}_p^{(\phi)} = \sum_{k=1}^{K} N_k\big(\phi(\boldsymbol{m}_k) - \phi(\boldsymbol{m})\big)\big(\phi(\boldsymbol{m}_k) - \phi(\boldsymbol{m})\big)^T\boldsymbol{v}_p^{(\phi)}$$

$$= \sum_{k=1}^{K} N_k \phi(\boldsymbol{m}_k)\phi(\boldsymbol{m}_k)^T \boldsymbol{v}_p^{(\phi)} - \sum_{k=1}^{K} N_k \phi(\boldsymbol{m}_k)\phi(\boldsymbol{m})^T \boldsymbol{v}_p^{(\phi)}$$

$$- \sum_{k=1}^{K} N_k \phi(\boldsymbol{m})\phi(\boldsymbol{m}_k)^T \boldsymbol{v}_p^{(\phi)} + \sum_{k=1}^{K} N_k \phi(\boldsymbol{m})\phi(\boldsymbol{m})^T \boldsymbol{v}_p^{(\phi)}$$

$$= \sum_{k=1}^{K} \sum_{\boldsymbol{x}_n \in c_k} \phi(\boldsymbol{x}_n)\, \phi(\boldsymbol{m}_k)^T \boldsymbol{v}_p^{(\phi)} - \sum_{k=1}^{K} \sum_{\boldsymbol{x}_n \in c_k} \phi(\boldsymbol{x}_n)\, \phi(\boldsymbol{m})^T \boldsymbol{v}_p^{(\phi)}$$

$$- N\phi(\boldsymbol{m})\phi(\boldsymbol{m})^T \boldsymbol{v}_p^{(\phi)} + N\phi(\boldsymbol{m})\phi(\boldsymbol{m})^T \boldsymbol{v}_p^{(\phi)}$$

$$= \sum_{k=1}^{K} \sum_{\boldsymbol{x}_n \in c_k} \phi(\boldsymbol{x}_n)\, \phi(\boldsymbol{m}_k)^T \boldsymbol{v}_p^{(\phi)} - \sum_{k=1}^{K} \sum_{\boldsymbol{x}_n \in c_k} \phi(\boldsymbol{x}_n)\, \phi(\boldsymbol{m})^T \boldsymbol{v}_p^{(\phi)}$$

$$= \sum_{k=1}^{K} \sum_{\boldsymbol{x}_n \in c_k} \left(\phi(\boldsymbol{m}_k)^T \boldsymbol{v}_p^{(\phi)} - \phi(\boldsymbol{m})^T \boldsymbol{v}_p^{(\phi)} \right) \cdot \phi(\boldsymbol{x}_n) \tag{7.14}$$

By introducing a new coefficient variable $\eta_n^{(B)}$ such that

$$\eta_n^{(B)} = \sum_{k=1}^{K} \eta_{nk}^{(B)} \quad \text{where} \quad \eta_{nk}^{(B)} = \begin{cases} \phi(\boldsymbol{m}_k)^T \boldsymbol{v}_p^{(\phi)} - \phi(\boldsymbol{m})^T \boldsymbol{v}_p^{(\phi)}, & if\ \boldsymbol{x}_n \in c_k \\ 0 \quad , & otherwise \end{cases} \tag{7.15}$$

$\boldsymbol{S}_B^{(\phi)} \boldsymbol{v}_p^{(\phi)}$ can then be expressed as a linear combination of $\phi(\boldsymbol{x}_n)$ as given by (7.16), where the coefficients are $\eta_n^{(B)}$.

$$\boldsymbol{S}_B^{(\phi)} \boldsymbol{v}_p^{(\phi)} = \sum_{n=1}^{N} \eta_n^{(B)} \cdot \phi(\boldsymbol{x}_n) \tag{7.16}$$

Similarly, $\boldsymbol{S}_W^{(\phi)} \boldsymbol{v}_p^{(\phi)}$ can be expanded as

$$\boldsymbol{S}_W^{(\phi)} \boldsymbol{v}_p^{(\phi)} = \sum_{k=1}^{K} \sum_{\boldsymbol{x}_n \in c_k} \big(\phi(\boldsymbol{x}_n) - \phi(\boldsymbol{m}_k)\big)\big(\phi(\boldsymbol{x}_n) - \phi(\boldsymbol{m}_k)\big)^T \boldsymbol{v}_p^{(\phi)}$$

$$= \sum_{k=1}^{K} \sum_{\boldsymbol{x}_n \in c_k} \phi(\boldsymbol{x}_n)\phi(\boldsymbol{x}_n)^T \boldsymbol{v}_p^{(\phi)} - \sum_{k=1}^{K} \sum_{\boldsymbol{x}_n \in c_k} \phi(\boldsymbol{x}_n)\phi(\boldsymbol{m}_k)^T \boldsymbol{v}_p^{(\phi)}$$

$$- \sum_{k=1}^{K} \sum_{\boldsymbol{x}_n \in c_k} \phi(\boldsymbol{m}_k)\phi(\boldsymbol{x}_n)^T \boldsymbol{v}_p^{(\phi)} + \sum_{k=1}^{K} \sum_{\boldsymbol{x}_n \in c_k} \phi(\boldsymbol{m}_k)\phi(\boldsymbol{m}_k)^T \boldsymbol{v}_p^{(\phi)}$$

$$= \sum_{n=1}^{N} \phi(\boldsymbol{x}_n)\phi(\boldsymbol{x}_n)^T \boldsymbol{v}_p^{(\phi)} - \sum_{k=1}^{K} \sum_{\boldsymbol{x}_n \in c_k} \phi(\boldsymbol{x}_n)\phi(\boldsymbol{m}_k)^T \boldsymbol{v}_p^{(\phi)}$$

$$- \sum_{k=1}^{K} \phi(\boldsymbol{m}_k) N_k \phi(\boldsymbol{m}_k)^T \boldsymbol{v}_p^{(\phi)} + \sum_{k=1}^{K} N_k \phi(\boldsymbol{m}_k)\phi(\boldsymbol{m}_k)^T \boldsymbol{v}_p^{(\phi)}$$

$$= \sum_{n=1}^{N} \phi(\boldsymbol{x}_n)\phi(\boldsymbol{x}_n)^T \boldsymbol{v}_p^{(\phi)} - \sum_{k=1}^{K} \sum_{\boldsymbol{x}_n \in c_k} \phi(\boldsymbol{x}_n)\phi(\boldsymbol{m}_k)^T \boldsymbol{v}_p^{(\phi)}$$

$$= \sum_{k=1}^{K} \sum_{\boldsymbol{x}_n \in c_k} \left(\phi(\boldsymbol{x}_n)^T \boldsymbol{v}_p^{(\phi)} - \phi(\boldsymbol{m}_k)^T \boldsymbol{v}_p^{(\phi)} \right) \cdot \phi(\boldsymbol{x}_n) \tag{7.17}$$

By introducing a new coefficient variable $\eta_n^{(W)}$ such that

$$\eta_n^{(W)} = \sum_{k=1}^{K} \eta_{nk}^{(W)} \quad \text{where} \quad \eta_{nk}^{(W)} = \begin{cases} \phi(\boldsymbol{x}_n)^T \boldsymbol{v}_p^{(\phi)} - \phi(\boldsymbol{m}_k)^T \boldsymbol{v}_p^{(\phi)}, & if\ \boldsymbol{x}_n \in c_k \\ 0 \quad , & otherwise \end{cases} \tag{7.18}$$

$\boldsymbol{S}_W^{(\phi)} \boldsymbol{v}_p^{(\phi)}$ can then be expressed as a linear combination of $\phi(\boldsymbol{x}_n)$ as given by (7.19), with the coefficients given by $\eta_n^{(W)}$.

$$\boldsymbol{S}_W^{(\phi)} \boldsymbol{v}_p^{(\phi)} = \sum_{n=1}^{N} \eta_n^{(W)} \cdot \phi(\boldsymbol{x}_n) \tag{7.19}$$

The facts that $\boldsymbol{S}_B^{(\phi)} \boldsymbol{v}_p^{(\phi)}$ and $\boldsymbol{S}_W^{(\phi)} \boldsymbol{v}_p^{(\phi)}$ are the linear combination of the mapped training vectors as given by (7.16) and (7.19) respectively will be useful for understanding KDA.

7.1.2.1 KDA Version 1

Analogous to LDA (v1), KDA (v1) aims at solving the optimization problem defined in (7.20).

$$\begin{gathered} \max \sum_{p=1}^{P} \left(\boldsymbol{v}_p^{(\phi)} \right)^T \boldsymbol{S}_B^{(\phi)} \boldsymbol{v}_p^{(\phi)} \\ s.t. \\ \left(\boldsymbol{v}_p^{(\phi)} \right)^T \boldsymbol{S}_W^{(\phi)} \boldsymbol{v}_p^{(\phi)} = 1 \ for\ p = 1,2 \ldots P \end{gathered} \tag{7.20}$$

Consequently, KDA (v1) is simplified to the eigenvalue problem given by (7.21), and the solution is then given by the eigenvectors of $\left(\boldsymbol{S}_W^{(\phi)} \right)^{-1} \boldsymbol{S}_B^{(\phi)}$.

$$\boldsymbol{S}_B^{(\phi)} \boldsymbol{v}_p^{(\phi)} = \lambda_p \boldsymbol{S}_W^{(\phi)} \boldsymbol{v}_p^{(\phi)} \tag{7.21}$$

Reformulating (7.21) using the expanded expressions of $\boldsymbol{S}_B^{(\phi)} \boldsymbol{v}_p^{(\phi)}$ or $\boldsymbol{S}_W^{(\phi)} \boldsymbol{v}_p^{(\phi)}$ given by (7.16) or (7.19) respectively, we obtain

$$\begin{cases} \boldsymbol{v}_p^{(\phi)} = \lambda_p \left(\boldsymbol{S}_B^{(\phi)} \right)^{-1} \boldsymbol{S}_W^{(\phi)} \boldsymbol{v}_p^{(\phi)} = \sum_{n=1}^{N} \left(\eta_n^{(W)} \lambda_p \left(\boldsymbol{S}_B^{(\phi)} \right)^{-1} \right) \phi(\boldsymbol{x}_n) \\ \boldsymbol{v}_p^{(\phi)} = \left(\lambda_p \boldsymbol{S}_W^{(\phi)} \right)^{-1} \boldsymbol{S}_B^{(\phi)} \boldsymbol{v}_p^{(\phi)} = \sum_{n=1}^{N} \left(\eta_n^{(B)} \lambda_p^{-1} \left(\boldsymbol{S}_W^{(\phi)} \right)^{-1} \right) \phi(\boldsymbol{x}_n) \end{cases} \tag{7.22}$$

It is claimed in [71] that $\boldsymbol{v}_p^{(\phi)}$ can be a linear combination of all the mapped training vectors as given by (7.23), where $\boldsymbol{u}_p^{(\phi)}$ is the coefficient vector in the mapped space, $\left(u_p^{(\phi)}\right)_n$ is the n-th element of $\boldsymbol{u}_p^{(\phi)}$, and $\phi(\boldsymbol{X}) = [\phi(\boldsymbol{x}_1)\ \ \phi(\boldsymbol{x}_2)\ \ ...\ \ \phi(\boldsymbol{x}_N)]$ is the data matrix containing all the mapped training vectors.

$$\boldsymbol{v}_p^{(\phi)} = \sum_{n=1}^{N} \left(u_p^{(\phi)}\right)_n \phi(\boldsymbol{x}_n) = \phi(\boldsymbol{X})\boldsymbol{u}_p^{(\phi)} \tag{7.23}$$

However, as can be seen from (7.22), $\boldsymbol{v}_p^{(\phi)}$ will be a linear transformation of the linear combination of all the mapped training vectors, instead of a direct linear combination. Therefore, (7.23) may not always hold unless $\phi(\boldsymbol{X})$ spans the whole mapped space, viz. the covariance of $\phi(\boldsymbol{X})$ is full-rank. Nevertheless, even if (7.23) does not hold in some circumstances, the only risk is that we may not be able to find all possible $\boldsymbol{v}_p^{(\phi)}$, but any solution $\boldsymbol{u}_p^{(\phi)}$ will correspond to a solution $\boldsymbol{v}_p^{(\phi)}$.

Using the relationship given by (7.23), the optimization problem of KDA (v1) becomes

$$\begin{gathered} \max \sum_{p=1}^{P} \left(\phi(\boldsymbol{X})\boldsymbol{u}_p^{(\phi)}\right)^T \boldsymbol{S}_B^{(\phi)} \phi(\boldsymbol{X})\boldsymbol{u}_p^{(\phi)} \\ s.t. \\ \left(\phi(\boldsymbol{X})\boldsymbol{u}_p^{(\phi)}\right)^T \boldsymbol{S}_W^{(\phi)} \phi(\boldsymbol{X})\boldsymbol{u}_p^{(\phi)} = 1 \ \ for\ \ p = 1,2 \dots P \end{gathered} \tag{7.24}$$

It is noted that $\phi(\boldsymbol{X})^T \boldsymbol{S}_B^{(\phi)} \phi(\boldsymbol{X})$ and $\phi(\boldsymbol{X})^T \boldsymbol{S}_W^{(\phi)} \phi(\boldsymbol{X})$ can be reformulated as given by (7.25) and (7.26) respectively, where the inner products of mapped vectors are expressed in terms of a kernel function $\ker(.,.)$ as given by (7.27). In (7.27), the dimensionality of $\boldsymbol{\Phi}_{mk}$, $\boldsymbol{\Phi}_m$ and $\boldsymbol{\Phi}_n$ is $N \times 1$ where N is the number of training vectors, and the subscript i represents the i-th element.

$$\begin{aligned} \phi(\boldsymbol{X})^T \boldsymbol{S}_B^{(\phi)} \phi(\boldsymbol{X}) &= \phi(\boldsymbol{X})^T \left(\sum_{k=1}^{K} N_k \big(\phi(\boldsymbol{m}_k) - \phi(\boldsymbol{m})\big)\big(\phi(\boldsymbol{m}_k) - \phi(\boldsymbol{m})\big)^T\right) \phi(\boldsymbol{X}) \\ &= \sum_{k=1}^{K} N_k \big(\phi(\boldsymbol{X})^T \phi(\boldsymbol{m}_k) - \phi(\boldsymbol{X})^T \phi(\boldsymbol{m})\big)\big(\phi(\boldsymbol{X})^T \phi(\boldsymbol{m}_k) - \phi(\boldsymbol{X})^T \phi(\boldsymbol{m})\big)^T \\ &= \sum_{k=1}^{K} N_k (\boldsymbol{\Phi}_{mk} - \boldsymbol{\Phi}_m)(\boldsymbol{\Phi}_{mk} - \boldsymbol{\Phi}_m)^T \end{aligned} \tag{7.25}$$

$$\begin{aligned} \phi(\boldsymbol{X})^T \boldsymbol{S}_W^{(\phi)} \phi(\boldsymbol{X}) &= \phi(\boldsymbol{X})^T \left(\sum_{k=1}^{K} \sum_{\boldsymbol{x}_n \in c_k} \big(\phi(\boldsymbol{x}_n) - \phi(\boldsymbol{m}_k)\big)\big(\phi(\boldsymbol{x}_n) - \phi(\boldsymbol{m}_k)\big)^T\right) \phi(\boldsymbol{X}) \\ &= \sum_{k=1}^{K} \sum_{\boldsymbol{x}_n \in c_k} \big(\phi(\boldsymbol{X})^T \phi(\boldsymbol{x}_n) - \phi(\boldsymbol{X})^T \phi(\boldsymbol{m}_k)\big)\big(\phi(\boldsymbol{X})^T \phi(\boldsymbol{x}_n) - \phi(\boldsymbol{X})^T \phi(\boldsymbol{m}_k)\big)^T \end{aligned}$$

$$= \sum_{k=1}^{K} \sum_{\boldsymbol{x}_n \in c_k} (\boldsymbol{\Phi}_n - \boldsymbol{\Phi}_{mk})(\boldsymbol{\Phi}_n - \boldsymbol{\Phi}_{mk})^T \tag{7.26}$$

where

$$\begin{gathered} (\Phi_{mk})_i = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{m}_k) = \frac{1}{N_k} \sum_{\boldsymbol{x}_n \in c_k} \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_n) = \frac{1}{N_k} \sum_{\boldsymbol{x}_n \in c_k} \ker(\boldsymbol{x}_i, \boldsymbol{x}_n) \\ (\Phi_m)_i = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{m}) = \frac{1}{N} \sum_{n=1}^{N} \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_n) = \frac{1}{N} \sum_{n=1}^{N} \ker(\boldsymbol{x}_i, \boldsymbol{x}_n) \\ (\Phi_n)_i = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_n) = \ker(\boldsymbol{x}_i, \boldsymbol{x}_n) \end{gathered} \tag{7.27}$$

If we further define $\boldsymbol{R}_B^{(\phi)} = \phi(\boldsymbol{X})^T \boldsymbol{S}_B^{(\phi)} \phi(\boldsymbol{X})$ and $\boldsymbol{R}_W^{(\phi)} = \phi(\boldsymbol{X})^T \boldsymbol{S}_W^{(\phi)} \phi(\boldsymbol{X})$, the optimization problem of KDA (v1) is simplified to (7.28), whose solution is then the eigenvectors of $\left(\boldsymbol{R}_W^{(\phi)}\right)^{-1} \boldsymbol{R}_B^{(\phi)}$.

$$\begin{gathered} \max \sum_{p=1}^{P} \left(\boldsymbol{u}_p^{(\phi)}\right)^T \boldsymbol{R}_B^{(\phi)} \boldsymbol{u}_p^{(\phi)} \\ s.t. \\ \left(\boldsymbol{u}_p^{(\phi)}\right)^T \boldsymbol{R}_W^{(\phi)} \boldsymbol{u}_p^{(\phi)} = 1 \ \ for \ \ p = 1,2 \dots P \end{gathered} \tag{7.28}$$

Having found all the coefficient vector $\boldsymbol{u}_p^{(\phi)}$ for $p = 1,2 \dots P$, for a testing vector $\boldsymbol{y}$, its corresponding projected vector, denoted as $\boldsymbol{y}'$, can be computed using (7.29), where y_p' denotes the p-th element of $\boldsymbol{y}'$, $\left(u_p^{(\phi)}\right)_i$ is the i-th element of $\boldsymbol{u}_p^{(\phi)}$, and N is the number of training vectors. The dimensionality of $\boldsymbol{y}'$ will be $P \times 1$.

$$y_p' = \left(\boldsymbol{v}_p^{(\phi)}\right)^T \phi(\boldsymbol{y}) = \left(\phi(\boldsymbol{X}) \boldsymbol{u}_p^{(\phi)}\right)^T \phi(\boldsymbol{y}) = \left(\boldsymbol{u}_p^{(\phi)}\right)^T \phi(\boldsymbol{X})^T \phi(\boldsymbol{y}) = \sum_{i=1}^{N} \left(u_p^{(\phi)}\right)_i \ker(\boldsymbol{x}_i, \boldsymbol{y}) \tag{7.29}$$

7.1.2.2 KDA Version 2

Analogous to LDA (v2), KDA (v2) aims at solving the optimization problem defined in (7.30).

$$\begin{gathered} \max \sum_{p=1}^{P} \left(\boldsymbol{v}_p^{(\phi)}\right)^T \left(\boldsymbol{S}_B^{(\phi)} - \alpha \boldsymbol{S}_W^{(\phi)}\right) \boldsymbol{v}_p^{(\phi)} \\ s.t. \\ \left(\boldsymbol{v}_p^{(\phi)}\right)^T \boldsymbol{v}_p^{(\phi)} = 1 \ \ for \ \ p = 1,2 \dots P \end{gathered} \tag{7.30}$$

It is then simplified to the eigenvalue problem in (7.31), whose solution is given by the eigenvectors of $(\boldsymbol{S}_B^{(\phi)} - \alpha \boldsymbol{S}_W^{(\phi)})$.

$$\left(\boldsymbol{S}_B^{(\phi)} - \alpha \boldsymbol{S}_W^{(\phi)}\right) \boldsymbol{v}_p^{(\phi)} = \lambda_p \boldsymbol{v}_p^{(\phi)} \tag{7.31}$$

Using the equations given by (7.16) and (7.19), (7.31) becomes

$$\sum_{n=1}^{N} \eta_n^{(B)} \cdot \phi(\boldsymbol{x}_n) - \alpha \sum_{n=1}^{N} \eta_n^{(W)} \cdot \phi(\boldsymbol{x}_n) = \lambda_p \boldsymbol{v}_p^{(\phi)} \tag{7.32}$$

Rearranging (7.32) gives (7.33), which implies that as long as $\boldsymbol{v}_p^{(\phi)}$ is a solution to (7.30), $\boldsymbol{v}_p^{(\phi)}$ should be a linear combination of all the mapped training vectors.

$$\boldsymbol{v}_p^{(\phi)} = \frac{1}{\lambda_p} \sum_{n=1}^{N} \left(\eta_n^{(B)} - \alpha \eta_n^{(W)}\right) \cdot \phi(\boldsymbol{x}_n) \tag{7.33}$$

This enables us to express $\boldsymbol{v}_p^{(\phi)}$ as (7.34), where $\boldsymbol{u}_p^{(\phi)}$ is the coefficient vector, and $\phi(\boldsymbol{X})$ is the data matrix comprising all the mapped training vectors, the same as that used in KDA (v1).

$$\boldsymbol{v}_p^{(\phi)} = \phi(\boldsymbol{X}) \boldsymbol{u}_p^{(\phi)} \tag{7.34}$$

Using (7.34), KDA (v2) can be reformulated as (7.35), where $\boldsymbol{R}_B^{(\phi)}$ and $\boldsymbol{R}_W^{(\phi)}$ are the same as those used in KDA (v1). In order to obtain a unique solution, the unit-length constraint is applied to $\boldsymbol{u}_p^{(\phi)}$. The unit-length constraint of $\boldsymbol{v}_p^{(\phi)}$ can be satisfied by normalizing the projected vector.

$$\begin{gathered} \max \sum_{p=1}^{P} \left(\boldsymbol{u}_p^{(\phi)}\right)^T \left(\boldsymbol{R}_B^{(\phi)} - \alpha \boldsymbol{R}_W^{(\phi)}\right) \boldsymbol{u}_p^{(\phi)} \\ s.t. \\ \left(\boldsymbol{u}_p^{(\phi)}\right)^T \boldsymbol{u}_p^{(\phi)} = 1 \;\; for \;\; p = 1,2 \ldots P \end{gathered} \tag{7.35}$$

Similar to LDA (v2), solving (7.35) gives (7.36), whose solution is the eigenvectors of $\left(\boldsymbol{R}_B^{(\phi)} - \alpha \boldsymbol{R}_W^{(\phi)}\right)$.

$$\left(\boldsymbol{R}_B^{(\phi)} - \alpha \boldsymbol{R}_W^{(\phi)}\right) \boldsymbol{u}_p^{(\phi)} = \lambda_p \boldsymbol{u}_p^{(\phi)} \tag{7.36}$$

Having found all the coefficient vector $\boldsymbol{u}_p^{(\phi)}$ for $p = 1,2 \ldots P$, for a testing vector $\boldsymbol{y}$, its corresponding projected vector, denoted as $\boldsymbol{y}'$, can be computed using (7.29). Having obtained the projected vector, we may implicitly normalize the projection vector $\boldsymbol{v}_p^{(\phi)}$ through normalizing y_p' as given by (7.37),

where y'_p denotes the p-th element of $\boldsymbol{y}'$, $\left(u_p^{(\phi)}\right)_i$ is the i-th element of $\boldsymbol{u}_p^{(\phi)}$, and N is the number of training vectors.

$$\frac{y'_p}{\left\|\boldsymbol{v}_p^{(\phi)}\right\|_2} = \frac{\left(\boldsymbol{v}_p^{(\phi)}\right)^T \phi(\boldsymbol{y})}{\sqrt{\left(\boldsymbol{v}_p^{(\phi)}\right)^T \boldsymbol{v}_p^{(\phi)}}} = \frac{\left(\boldsymbol{u}_p^{(\phi)}\right)^T \phi(\boldsymbol{X})^T \phi(\boldsymbol{y})}{\sqrt{\left(\boldsymbol{u}_p^{(\phi)}\right)^T \phi(\boldsymbol{X})^T \phi(\boldsymbol{X}) \boldsymbol{u}_p^{(\phi)}}} = \frac{\sum_{i=1}^N \left(u_p^{(\phi)}\right)_i \ker(\boldsymbol{x}_i, \boldsymbol{y})}{\sqrt{\sum_{i=1}^N \sum_{j=1}^N \left(u_p^{(\phi)}\right)_i \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j) \left(u_p^{(\phi)}\right)_j}}$$

$$= \frac{\sum_{i=1}^N \left(u_p^{(\phi)}\right)_i \ker(\boldsymbol{x}_i, \boldsymbol{y})}{\sqrt{\sum_{i=1}^N \sum_{j=1}^N \left(u_p^{(\phi)}\right)_i \ker(\boldsymbol{x}_i, \boldsymbol{x}_j) \left(u_p^{(\phi)}\right)_j}} \tag{7.37}$$

7.1.2.3 Brief Comparison between the Two Versions of KDA

KDA (v1) requires finding the eigenvectors of $\left(\boldsymbol{R}_W^{(\phi)}\right)^{-1} \boldsymbol{R}_B^{(\phi)}$, while KDA (v2) requires finding the eigenvectors of $\left(\boldsymbol{R}_B^{(\phi)} - \alpha \boldsymbol{R}_W^{(\phi)}\right)$. When compared to KDA (v2), KDA (v1) may have the singularity problem if $\boldsymbol{R}_W^{(\phi)}$ is not full-rank. Nevertheless, this problem can be avoided by introducing a regularization parameter α such that KDA (v1) aims at finding the eigenvectors of $\left(\boldsymbol{R}_W^{(\phi)} + \alpha \boldsymbol{I}\right)^{-1} \boldsymbol{R}_B^{(\phi)}$. Besides, the kernel-based formulations for KDA (v1) given by (7.20) and (7.28) do not have a one-to-one correspondence. One $\boldsymbol{u}_p^{(\phi)}$ solved from (7.28) always corresponds to one $\boldsymbol{v}_p^{(\phi)}$ solved from (7.20), but not vice versa. In contrast, the kernel-based formulations for KDA (v2) given by (7.30) and (7.35) have a one-to-one correspondence.

7.2 Nuisance Attribute Projection

This section deals with the technique originating from the nuisance attribute projection (NAP), which is widely used in speaker verification [74]. The role of NAP is to remove the unwanted information (i.e., the nuisance attribute) from the extracted feature vector; for example, the channel information embedded in a GSV. Nevertheless, the technique of NAP can be extended to a more general case, making it a general projection technique. The relationship between NAP and LDA will also be discussed.

7.2.1 Basic Formulation

Suppose there is a training set of speech samples denoted as $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_N\}$, where N is the number of training samples, and $\boldsymbol{x}_n$ is a $D \times 1$ vector (such as GSV or i-vector) representing the n-th speech sample. Assume that these speech samples come from different channels, such as different recording microphones or different telephone channels. Then, the channel information embedded in the training vectors will be a type of noise that will degrade the quality of the training vectors.

The objective of NAP is to find a projection matrix $\boldsymbol{V}$ with a size of $D \times P$, such that after performing this projection, most channel information can be removed from the original vector. This objective can be achieved by minimizing the objective function $Q(\boldsymbol{V})$ given by (7.38), where W_{ij} is the ij-th element of $\boldsymbol{W}$, which is an $N \times N$ symmetric weight matrix. Thus, in the projected space, the channel information is more concentrated. The projected vector $\boldsymbol{x}'_n$ corresponding to the vector $\boldsymbol{x}_n$ is given by $\boldsymbol{x}'_n = (\boldsymbol{I} - \boldsymbol{V}\boldsymbol{V}^T)\boldsymbol{x}_n$, where the channel information is expected to be projected out [76].

$$Q(\boldsymbol{V}) = \sum_{i=1}^{N} \sum_{j=1}^{N} W_{ij} \left\| \boldsymbol{V}^T \boldsymbol{x}_i - \boldsymbol{V}^T \boldsymbol{x}_j \right\|_2^2 \tag{7.38}$$

where

$$W_{ij} = \begin{cases} 1, & if\ \boldsymbol{x}_i\ and\ \boldsymbol{x}_j\ come\ from\ different\ channels \\ 0, & otherwise \end{cases} \tag{7.39}$$

NAP works well as a channel compensation technique for speaker verification. However, the original idea may not be directly applicable to general pattern recognition tasks, as the feature vectors for different tasks may not contain the so-called channel information. Fortunately, by simply modifying the meaning of the weight matrix given by (7.39) to that given by (7.40), a general projection technique is obtained, which can be used to improve the discrimination ability of the original feature vector. In (7.40), γ is a nonnegative regularization parameter explained later. The projected vector is then given as $\boldsymbol{x}'_n = \boldsymbol{V}^T \boldsymbol{x}_n$.

$$W_{ij} = \begin{cases} 1, & if\ \boldsymbol{x}_i\ and\ \boldsymbol{x}_j\ come\ from\ the\ same\ class \\ -\gamma, & otherwise \end{cases} \tag{7.40}$$

Using the weight matrix as given by (7.40), $Q(\boldsymbol{V})$ can also be written as (7.41), where K is the number of classes and c_k denotes the k-th class.

$$Q(\boldsymbol{V}) = \sum_{k=1}^{K} \sum_{\boldsymbol{x}_i \in c_k} \sum_{\boldsymbol{x}_j \in c_k} \left\| \boldsymbol{V}^T \boldsymbol{x}_i - \boldsymbol{V}^T \boldsymbol{x}_j \right\|_2^2 - \gamma \sum_{k=1}^{K} \sum_{\boldsymbol{x}_i \in c_k} \sum_{\boldsymbol{x}_j \notin c_k} \left\| \boldsymbol{V}^T \boldsymbol{x}_i - \boldsymbol{V}^T \boldsymbol{x}_j \right\|_2^2 \quad (7.41)$$

The expression of the objective function in (7.41) shows that one goal is reducing the pairwise distance if the projected vectors come from the same class, and another goal is increasing the pairwise distance if the projected vectors come from different classes. This makes the projected vectors closer to each other if they belong to the same class, and farther from each other if they belong to different classes. The regularization parameter γ controls the tradeoff between the two goals. We keep referring to this projection technique as NAP, even though the original meaning has been changed.

It is reasonable to restrict the projection directions to have a unit length, i.e., the column vectors in $\boldsymbol{V}$ should have a unit length. The projection matrix can then be obtained by solving the constrained optimization problem defined in (7.42), where $\boldsymbol{v}_p$ is the p-th column vector in $\boldsymbol{V}$, and P is the number of columns in $\boldsymbol{V}$.

$$\begin{gathered} \min Q(\boldsymbol{V}) \\ s.t. \\ \boldsymbol{v}_p^T \boldsymbol{v}_p = 1 \;\; for \;\; p = 1,2 \ldots P \end{gathered} \quad (7.42)$$

The objective function $Q(\boldsymbol{V})$ can be expanded as

$$Q(\boldsymbol{V}) = \sum_{i=1}^{N} \sum_{j=1}^{N} W_{ij} \left(\boldsymbol{x}_i - \boldsymbol{x}_j\right)^T \boldsymbol{V}\boldsymbol{V}^T \left(\boldsymbol{x}_i - \boldsymbol{x}_j\right)$$

$$= \sum_{i=1}^{N} \sum_{j=1}^{N} W_{ij} (\boldsymbol{x}_i - \boldsymbol{x}_j)^T \left(\sum_{p=1}^{P} \boldsymbol{v}_p \boldsymbol{v}_p^T\right) (\boldsymbol{x}_i - \boldsymbol{x}_j)$$

$$= \sum_{p=1}^{P} \sum_{i=1}^{N} \sum_{j=1}^{N} W_{ij} (\boldsymbol{x}_i - \boldsymbol{x}_j)^T \boldsymbol{v}_p \boldsymbol{v}_p^T (\boldsymbol{x}_i - \boldsymbol{x}_j)$$

$$= \sum_{p=1}^{P} \boldsymbol{v}_p^T \left(\sum_{i=1}^{N} \sum_{j=1}^{N} W_{ij} (\boldsymbol{x}_i - \boldsymbol{x}_j)(\boldsymbol{x}_i - \boldsymbol{x}_j)^T\right) \boldsymbol{v}_p$$

$$= \sum_{p=1}^{P} \boldsymbol{v}_p^T \left(\sum_{i=1}^{N} \sum_{j=1}^{N} W_{ij} (\boldsymbol{x}_i \boldsymbol{x}_i^T + \boldsymbol{x}_j \boldsymbol{x}_j^T - \boldsymbol{x}_i \boldsymbol{x}_j^T - \boldsymbol{x}_j \boldsymbol{x}_i^T)\right) \boldsymbol{v}_p$$

$$= \sum_{p=1}^{P} \boldsymbol{v}_p^T \left(\sum_{i=1}^{N} \sum_{j=1}^{N} W_{ij} \boldsymbol{x}_i \boldsymbol{x}_i^T\right) \boldsymbol{v}_p + \sum_{p=1}^{P} \boldsymbol{v}_p^T \left(\sum_{i=1}^{N} \sum_{j=1}^{N} W_{ij} \boldsymbol{x}_j \boldsymbol{x}_j^T\right) \boldsymbol{v}_p$$

$$-\sum_{p=1}^{P}\boldsymbol{v}_p^T\left(\sum_{i=1}^{N}\sum_{j=1}^{N}W_{ij}\boldsymbol{x}_i\boldsymbol{x}_j^T\right)\boldsymbol{v}_p-\sum_{p=1}^{P}\boldsymbol{v}_p^T\left(\sum_{i=1}^{N}\sum_{j=1}^{N}W_{ij}\boldsymbol{x}_j\boldsymbol{x}_i^T\right)\boldsymbol{v}_p$$

$$=2\sum_{p=1}^{P}\boldsymbol{v}_p^T\left(\sum_{i=1}^{N}\sum_{j=1}^{N}W_{ij}\boldsymbol{x}_i\boldsymbol{x}_i^T\right)\boldsymbol{v}_p-2\sum_{p=1}^{P}\boldsymbol{v}_p^T\left(\sum_{i=1}^{N}\sum_{j=1}^{N}W_{ij}\boldsymbol{x}_i\boldsymbol{x}_j^T\right)\boldsymbol{v}_p \tag{7.43}$$

For simplicity, we may use a data matrix $\boldsymbol{X}$ to represent all the training vectors, namely $\boldsymbol{X}=[\boldsymbol{x}_1\ \boldsymbol{x}_2\ \ldots\ \boldsymbol{x}_N]$, which is a $D\times N$ matrix. We may also define a matrix $\boldsymbol{Z}_W$ as given by (7.44), where $\boldsymbol{e}$ is an $N\times 1$ vector with all elements being one, and diag{.} is an operation that puts the input vector on the principal diagonal of the output diagonal matrix [76].

$$\boldsymbol{Z}_W=\text{diag}\{\boldsymbol{We}\}-\boldsymbol{W} \tag{7.44}$$

Using the notations $\boldsymbol{X}$ and $\boldsymbol{Z}_W$, $Q(\boldsymbol{V})$ can be simplified to

$$Q(\boldsymbol{V})=2\sum_{p=1}^{P}\boldsymbol{v}_p^T\boldsymbol{X}\boldsymbol{Z}_W\boldsymbol{X}^T\boldsymbol{v}_p \tag{7.45}$$

The constrained optimization problem in (7.42) can be reformulated using a Lagrangian function $L(\boldsymbol{V},\lambda)$ as given by (7.46), where the constant factor 2 in (7.45) is dropped, and $\{\lambda_1,\lambda_2\ldots\lambda_P\}$ are Lagrange multipliers.

$$L(\boldsymbol{V},\lambda)=\sum_{p=1}^{P}\boldsymbol{v}_p^T\boldsymbol{X}\boldsymbol{Z}_W\boldsymbol{X}^T\boldsymbol{v}_p-\sum_{p=1}^{P}\lambda_p(\boldsymbol{v}_p^T\boldsymbol{v}_p-1) \tag{7.46}$$

Setting the derivative of $L(\boldsymbol{V},\lambda)$ with respect to $\boldsymbol{v}_p$ to zero yields (7.47), meaning that $\boldsymbol{v}_p$ is the p-th eigenvector of $\boldsymbol{X}\boldsymbol{Z}_W\boldsymbol{X}^T$. The unit-length constraint of $\boldsymbol{v}_p$ can then be satisfied by normalization. The number of projection directions, i.e., P, is then determined by the number of eigenvectors of $\boldsymbol{X}\boldsymbol{Z}_W\boldsymbol{X}^T$.

$$\frac{\partial L(\boldsymbol{V},\lambda)}{\partial\boldsymbol{v}_p}=2\boldsymbol{X}\boldsymbol{Z}_W\boldsymbol{X}^T\boldsymbol{v}_p-2\lambda_p\boldsymbol{v}_p=\boldsymbol{0}\quad\Longrightarrow\quad\boldsymbol{X}\boldsymbol{Z}_W\boldsymbol{X}^T\boldsymbol{v}_p=\lambda_p\boldsymbol{v}_p \tag{7.47}$$

7.2.2 Kernel Extension

From (7.47), it is noticed that if $\boldsymbol{v}_p$ is a solution to the eigenvalue problem, it should have the expression as given by (7.48), which implies that $\boldsymbol{v}_p$ is a linear combination of all the training vectors. For simplicity, a coefficient vector $\boldsymbol{u}_p$ is defined, whose dimensionality is $N\times 1$.

$$\boldsymbol{v}_p=\boldsymbol{X}\frac{\boldsymbol{Z}_W\boldsymbol{X}^T\boldsymbol{v}_p}{\lambda_p}=\boldsymbol{X}\boldsymbol{u}_p \tag{7.48}$$

Substituting $\boldsymbol{v}_p$ in (7.47) by $\boldsymbol{X}\boldsymbol{u}_p$ gives

$$\boldsymbol{X}\boldsymbol{Z}_W\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{u}_p = \lambda_p\boldsymbol{X}\boldsymbol{u}_p \tag{7.49}$$

Multiplying both sides of (7.49) by $\boldsymbol{X}^T$ then yields

$$\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{Z}_W\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{u}_p = \lambda_p\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{u}_p \tag{7.50}$$

Since only the inner products of the training vectors are involved in computation, the kernel trick is applicable. Suppose we would like to map the original vector $\boldsymbol{x}$ to another dimensional space using a mapping function $\phi: \boldsymbol{x} \longmapsto \phi(\boldsymbol{x})$, (7.50) becomes (7.51), where $\phi(\boldsymbol{X}) = [\phi(\boldsymbol{x}_1)\ \phi(\boldsymbol{x}_2)\ \ldots\ \phi(\boldsymbol{x}_N)]$, and $\boldsymbol{u}_p^{(\phi)}$ is the coefficient vector in the mapped space ϕ.

$$\phi(\boldsymbol{X})^T\phi(\boldsymbol{X})\boldsymbol{Z}_W\phi(\boldsymbol{X})^T\phi(\boldsymbol{X})\boldsymbol{u}_p^{(\phi)} = \lambda_p\phi(\boldsymbol{X})^T\phi(\boldsymbol{X})\boldsymbol{u}_p^{(\phi)} \tag{7.51}$$

Define a kernel matrix $\boldsymbol{\Phi}$, whose ij-th element is the inner product of $\phi(\boldsymbol{x}_i)$ and $\phi(\boldsymbol{x}_j)$, and is expressed using a kernel function ker $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ as given by (7.52),

$$\Phi_{ij} = \phi(\boldsymbol{x}_i)^T\phi\big(\boldsymbol{x}_j\big) = \text{ker}\ (\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{7.52}$$

Then (7.51) can be simplified to (7.53), which is the kernel-based NAP (KNAP).

$$\boldsymbol{\Phi}\boldsymbol{Z}_W\boldsymbol{\Phi}\boldsymbol{u}_p^{(\phi)} = \lambda_p\boldsymbol{\Phi}\boldsymbol{u}_p^{(\phi)} \tag{7.53}$$

When the mapping function is the simple identity mapping, namely $\phi(\boldsymbol{x}) = \boldsymbol{x}$, (7.53) then becomes (7.50). Specifying a mapping ϕ or a kernel function ker $(.,.)$, the coefficient vector $\boldsymbol{u}_p^{(\phi)}$ can be obtained by solving the generalized eigenvalue problem in (7.53). The number of projection directions, namely, P, is then the number of eigenvectors. In the mapped space ϕ, the relationship between the projection direction $\boldsymbol{v}_p^{(\phi)}$ and the coefficient vector $\boldsymbol{u}_p^{(\phi)}$ is given by (7.54).

$$\boldsymbol{v}_p^{(\phi)} = \phi(\boldsymbol{X})\boldsymbol{u}_p^{(\phi)} \tag{7.54}$$

Having found $\boldsymbol{u}_p^{(\phi)}$ for $p = 1,2 \ldots P$, for a testing vector $\boldsymbol{y}$, its corresponding projected vector, denoted as $\boldsymbol{y}'$, can be computed using (7.55), where y_p' denotes the p-th element of $\boldsymbol{y}'$, $\left(u_p^{(\phi)}\right)_i$ is the i-th element of $\boldsymbol{u}_p^{(\phi)}$, and N is the number of training vectors. The dimensionality of $\boldsymbol{y}'$ will be $P \times 1$.

$$y_p' = \left(\boldsymbol{v}_p^{(\phi)}\right)^T \phi(\boldsymbol{y}) = \left(\phi(\boldsymbol{X})\boldsymbol{u}_p^{(\phi)}\right)^T \phi(\boldsymbol{y}) = \left(\boldsymbol{u}_p^{(\phi)}\right)^T \phi(\boldsymbol{X})^T \phi(\boldsymbol{y}) = \sum_{i=1}^{N} \left(u_p^{(\phi)}\right)_i \ker(\boldsymbol{x}_i, \boldsymbol{y}) \tag{7.55}$$

In order to fulfill the unit-length constraint of the projection directions, although it may be difficult to normalize the projection vector $\boldsymbol{v}_p^{(\phi)}$ in the mapped space ϕ, it is feasible to implicitly normalize $\boldsymbol{v}_p^{(\phi)}$ through normalizing y_p' as given by (7.56), which requires $\boldsymbol{u}_p^{(\phi)}$ instead of $\boldsymbol{v}_p^{(\phi)}$.

$$\frac{y_p'}{\left\|\boldsymbol{v}_p^{(\phi)}\right\|_2} = \frac{\left(\boldsymbol{v}_p^{(\phi)}\right)^T \phi(\boldsymbol{y})}{\sqrt{\left(\boldsymbol{v}_p^{(\phi)}\right)^T \boldsymbol{v}_p^{(\phi)}}} = \frac{\left(\boldsymbol{u}_p^{(\phi)}\right)^T \phi(\boldsymbol{X})^T \phi(\boldsymbol{y})}{\sqrt{\left(\boldsymbol{u}_p^{(\phi)}\right)^T \phi(\boldsymbol{X})^T \phi(\boldsymbol{X}) \boldsymbol{u}_p^{(\phi)}}} = \frac{\sum_{i=1}^{N} \left(u_p^{(\phi)}\right)_i \ker(\boldsymbol{x}_i, \boldsymbol{y})}{\sqrt{\left(\boldsymbol{u}_p^{(\phi)}\right)^T \boldsymbol{\Phi} \boldsymbol{u}_p^{(\phi)}}} \tag{7.56}$$

7.2.3 Relationship with LDA

The objective function of NAP is based on the pairwise Euclidean distance, which is the distance between two vectors. In contrast, the objective function of LDA is based on the cluster-center distance, which is the distance between two clusters of vectors. Although NAP and LDA seem uncorrelated, there does exist some relationship between them.

Given a set of training vectors $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_N\}$ belonging to K classes, where the k-th class, denoted as c_k, comprises N_k training vectors. Denote $\boldsymbol{x}_n'$ as the projected version of $\boldsymbol{x}_n$, $\boldsymbol{\mu}$ as the mean of all the projected vectors, and $\boldsymbol{\mu}_k$ as the mean of the projected vectors belonging to class k, we have

$$\begin{aligned} \boldsymbol{x}_n' &= \boldsymbol{V}^T \boldsymbol{x}_n \\ \boldsymbol{\mu} &= \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{x}_n' \\ \boldsymbol{\mu}_k &= \frac{1}{N_k} \sum_{\boldsymbol{x}_n \in c_k} \boldsymbol{x}_n' \end{aligned} \tag{7.57}$$

The objective function of NAP can then be expressed as (7.58), which comprises two parts denoted as Δ_1 and Δ_2.

$$Q_{NAP}(\boldsymbol{V}) = \sum_{k=1}^{K} \sum_{\boldsymbol{x}_i \in c_k} \sum_{\boldsymbol{x}_j \in c_k} \left\| \boldsymbol{x}_i' - \boldsymbol{x}_j' \right\|_2^2 - \gamma \sum_{k=1}^{K} \sum_{\boldsymbol{x}_i \in c_k} \sum_{\boldsymbol{x}_j \notin c_k} \left\| \boldsymbol{x}_i' - \boldsymbol{x}_j' \right\|_2^2 = \Delta_1 - \gamma \Delta_2 \tag{7.58}$$

In the following, we show how to relate NAP to LDA through reformulating Δ_1 and Δ_2. Δ_1 can be reformulated in terms of $\boldsymbol{\mu}_k$ as given by (7.59).

$$\begin{aligned}
\Delta_1 &= \sum_{k=1}^{K} \sum_{\boldsymbol{x}_i \in c_k} \sum_{\boldsymbol{x}_j \in c_k} \left\| \boldsymbol{x}_i' - \boldsymbol{x}_j' \right\|_2^2 = \sum_{k=1}^{K} \sum_{\boldsymbol{x}_i \in c_k} \sum_{\boldsymbol{x}_j \in c_k} \left\| (\boldsymbol{x}_i' - \boldsymbol{\mu}_k) - (\boldsymbol{x}_j' - \boldsymbol{\mu}_k) \right\|_2^2 \\
&= \sum_{k=1}^{K} \sum_{\boldsymbol{x}_i \in c_k} \sum_{\boldsymbol{x}_j \in c_k} \left(\| \boldsymbol{x}_i' - \boldsymbol{\mu}_k \|_2^2 + \left\| \boldsymbol{x}_j' - \boldsymbol{\mu}_k \right\|_2^2 - 2 (\boldsymbol{x}_i' - \boldsymbol{\mu}_k)^T (\boldsymbol{x}_j' - \boldsymbol{\mu}_k) \right) \\
&= \sum_{k=1}^{K} N_k \sum_{\boldsymbol{x}_i \in c_k} \| \boldsymbol{x}_i' - \boldsymbol{\mu}_k \|_2^2 + \sum_{k=1}^{K} N_k \sum_{\boldsymbol{x}_j \in c_k} \left\| \boldsymbol{x}_j' - \boldsymbol{\mu}_k \right\|_2^2 \\
&\quad - 2 \sum_{k=1}^{K} \sum_{\boldsymbol{x}_i \in c_k} \sum_{\boldsymbol{x}_j \in c_k} (\boldsymbol{x}_i' - \boldsymbol{\mu}_k)^T (\boldsymbol{x}_j' - \boldsymbol{\mu}_k) \\
&= 2 \sum_{k=1}^{K} N_k \sum_{\boldsymbol{x}_i \in c_k} \| \boldsymbol{x}_i' - \boldsymbol{\mu}_k \|_2^2 - 2 \sum_{k=1}^{K} \sum_{\boldsymbol{x}_i \in c_k} (\boldsymbol{x}_i' - \boldsymbol{\mu}_k)^T \sum_{\boldsymbol{x}_j \in c_k} (\boldsymbol{x}_j' - \boldsymbol{\mu}_k) \\
&= 2 \sum_{k=1}^{K} N_k \sum_{\boldsymbol{x}_i \in c_k} \| \boldsymbol{x}_i' - \boldsymbol{\mu}_k \|_2^2
\end{aligned} \tag{7.59}$$

Similarly, Δ_2 can be reformulated in terms of $\boldsymbol{\mu}_k$ and $\boldsymbol{\mu}$ as given by (7.60).

$$\begin{aligned}
\Delta_2 &= \sum_{k=1}^{K} \sum_{\boldsymbol{x}_i \in c_k} \sum_{\boldsymbol{x}_j \notin c_k} \left\| \boldsymbol{x}_i' - \boldsymbol{x}_j' \right\|_2^2 = \sum_{i=1}^{N} \sum_{j=1}^{N} \left\| \boldsymbol{x}_i' - \boldsymbol{x}_j' \right\|_2^2 - \Delta_1 \\
&= \sum_{i=1}^{N} \sum_{j=1}^{N} \left\| (\boldsymbol{x}_i' - \boldsymbol{\mu}) - (\boldsymbol{x}_j' - \boldsymbol{\mu}) \right\|_2^2 - \Delta_1 \\
&= \sum_{i=1}^{N} \sum_{j=1}^{N} \| \boldsymbol{x}_i' - \boldsymbol{\mu} \|_2^2 + \sum_{i=1}^{N} \sum_{j=1}^{N} \left\| \boldsymbol{x}_j' - \boldsymbol{\mu} \right\|_2^2 - 2 \sum_{i=1}^{N} \sum_{j=1}^{N} (\boldsymbol{x}_i' - \boldsymbol{\mu})^T (\boldsymbol{x}_j' - \boldsymbol{\mu}) - \Delta_1 \\
&= 2N \sum_{i=1}^{N} \| \boldsymbol{x}_i' - \boldsymbol{\mu} \|_2^2 - 2 \sum_{i=1}^{N} (\boldsymbol{x}_i' - \boldsymbol{\mu})^T \sum_{j=1}^{N} (\boldsymbol{x}_j' - \boldsymbol{\mu}) - \Delta_1 \\
&= 2N \sum_{i=1}^{N} \| \boldsymbol{x}_i' - \boldsymbol{\mu} \|_2^2 - \Delta_1 \\
&= 2N \sum_{k=1}^{K} \sum_{\boldsymbol{x}_i \in c_k} \| (\boldsymbol{x}_i' - \boldsymbol{\mu}_k) + (\boldsymbol{\mu}_k - \boldsymbol{\mu}) \|_2^2 - \Delta_1 \\
&= 2N \sum_{k=1}^{K} \sum_{\boldsymbol{x}_i \in c_k} \| (\boldsymbol{x}_i' - \boldsymbol{\mu}_k) \|_2^2 + 2N \sum_{k=1}^{K} \sum_{\boldsymbol{x}_i \in c_k} \| (\boldsymbol{\mu}_k - \boldsymbol{\mu}) \|_2^2 \\
&\quad + 4N \sum_{k=1}^{K} \sum_{\boldsymbol{x}_i \in c_k} (\boldsymbol{x}_i' - \boldsymbol{\mu}_k)^T (\boldsymbol{\mu}_k - \boldsymbol{\mu}) - \Delta_1
\end{aligned}$$

$$= 2N\sum_{k=1}^{K}\sum_{\boldsymbol{x}_i\in c_k}\|(\boldsymbol{x}_i'-\boldsymbol{\mu}_k)\|_2^2 + 2N\sum_{k=1}^{K}\sum_{\boldsymbol{x}_i\in c_k}\|(\boldsymbol{\mu}_k-\boldsymbol{\mu})\|_2^2$$

$$+4N\sum_{k=1}^{K}\Big(\sum_{\boldsymbol{x}_i\in c_k}(\boldsymbol{x}_i'-\boldsymbol{\mu}_k)^T\Big)(\boldsymbol{\mu}_k-\boldsymbol{\mu}) - \Delta_1$$

$$= 2N\sum_{k=1}^{K}\sum_{\boldsymbol{x}_i\in c_k}\|(\boldsymbol{x}_i'-\boldsymbol{\mu}_k)\|_2^2 + 2N\sum_{k=1}^{K}\sum_{\boldsymbol{x}_i\in c_k}\|(\boldsymbol{\mu}_k-\boldsymbol{\mu})\|_2^2 - \Delta_1$$

$$= 2N\sum_{k=1}^{K}\sum_{\boldsymbol{x}_i\in c_k}\|(\boldsymbol{x}_i'-\boldsymbol{\mu}_k)\|_2^2 + 2N\sum_{k=1}^{K}\sum_{\boldsymbol{x}_i\in c_k}\|(\boldsymbol{\mu}_k-\boldsymbol{\mu})\|_2^2 - 2\sum_{k=1}^{K}N_k\sum_{\boldsymbol{x}_i\in c_k}\|\boldsymbol{x}_i'-\boldsymbol{\mu}_k\|_2^2$$

$$= 2\sum_{k=1}^{K}(N-N_k)\sum_{\boldsymbol{x}_i\in c_k}\|\boldsymbol{x}_i'-\boldsymbol{\mu}_k\|_2^2 + 2N\sum_{k=1}^{K}\sum_{\boldsymbol{x}_i\in c_k}\|(\boldsymbol{\mu}_k-\boldsymbol{\mu})\|_2^2$$

$$= 2\sum_{k=1}^{K}(N-N_k)\sum_{\boldsymbol{x}_i\in c_k}\|\boldsymbol{x}_i'-\boldsymbol{\mu}_k\|_2^2 + 2N\sum_{k=1}^{K}N_k\|(\boldsymbol{\mu}_k-\boldsymbol{\mu})\|_2^2 \tag{7.60}$$

Using the reformulations of Δ_1 and Δ_2, $Q_{NAP}(\boldsymbol{V})$ can be expressed as

$$Q_{NAP}(\boldsymbol{V}) = \Delta_1 - \gamma\Delta_2$$

$$= 2\sum_{k=1}^{K}N_k\sum_{\boldsymbol{x}_i\in c_k}\|\boldsymbol{x}_i'-\boldsymbol{\mu}_k\|_2^2$$

$$-\gamma\Big(2\sum_{k=1}^{K}(N-N_k)\sum_{\boldsymbol{x}_i\in c_k}\|\boldsymbol{x}_i'-\boldsymbol{\mu}_k\|_2^2 + 2N\sum_{k=1}^{K}N_k\|(\boldsymbol{\mu}_k-\boldsymbol{\mu})\|_2^2\Big)$$

$$= \sum_{k=1}^{K}\big(2N_k - 2\gamma(N-N_k)\big)\sum_{\boldsymbol{x}_i\in c_k}\|\boldsymbol{x}_i'-\boldsymbol{\mu}_k\|_2^2 - \sum_{k=1}^{K}2\gamma NN_k\|(\boldsymbol{\mu}_k-\boldsymbol{\mu})\|_2^2 \tag{7.61}$$

Recalling that the objective function of LDA (v2) is given by

$$Q_{LDA}(\boldsymbol{V}) = \mathrm{tr}\{\boldsymbol{C}_B\} - \mathrm{tr}\{\alpha\boldsymbol{C}_W\}$$

$$= \mathrm{tr}\Big\{\sum_{k=1}^{K}N_k(\boldsymbol{\mu}_k-\boldsymbol{\mu})(\boldsymbol{\mu}_k-\boldsymbol{\mu})^T\Big\} - \mathrm{tr}\Big\{\alpha\sum_{k=1}^{K}\sum_{\boldsymbol{x}_i\in c_k}(\boldsymbol{x}_i'-\boldsymbol{\mu}_k)(\boldsymbol{x}_i'-\boldsymbol{\mu}_k)^T\Big\}$$

$$= \mathrm{tr}\Big\{\sum_{k=1}^{K}N_k(\boldsymbol{\mu}_k-\boldsymbol{\mu})^T(\boldsymbol{\mu}_k-\boldsymbol{\mu})\Big\} - \mathrm{tr}\Big\{\alpha\sum_{k=1}^{K}\sum_{\boldsymbol{x}_i\in c_k}(\boldsymbol{x}_i'-\boldsymbol{\mu}_k)^T(\boldsymbol{x}_i'-\boldsymbol{\mu}_k)\Big\}$$

$$= \sum_{k=1}^{K}N_k(\boldsymbol{\mu}_k-\boldsymbol{\mu})^T(\boldsymbol{\mu}_k-\boldsymbol{\mu}) - \alpha\sum_{k=1}^{K}\sum_{\boldsymbol{x}_i\in c_k}(\boldsymbol{x}_i'-\boldsymbol{\mu}_k)^T(\boldsymbol{x}_i'-\boldsymbol{\mu}_k) \tag{7.62}$$

Considering that NAP aims at minimizing $Q_{NAP}(\boldsymbol{V})$ whereas LDA (v2) aims at maximizing $Q_{LDA}(\boldsymbol{V})$, NAP and LDA (v2) are equivalent under the condition given by (7.63), which demonstrates the relationship between the two projection techniques.

$$\frac{2N_k-2\gamma(N-N_k)}{2\gamma NN_k} = \frac{\alpha}{N_k} \quad\Longrightarrow\quad \gamma = \frac{N_k}{(1+\alpha)N-N_k} \tag{7.63}$$

According to (7.63), when the value of α varies within $[0, \infty)$, in order to maintain the equivalence, γ will vary within $\left[\frac{N_k}{N-N_k}, 0\right)$.

7.3 Brief Comparison between LDA and NAP

Both LDA and NAP aim at finding a projection matrix that realizes two targets. The first target is to shorten the within-class distances (i.e., shorten the distances between the projected vectors belonging to the same class), and the second target is to increase the between-class distances (i.e., increase the distances between the projected vectors belonging to different classes). Specifically, LDA realizes the first target by shortening the distances between the projected vectors and their class centers, whereas NAP realizes the first target by shortening the distances between pairs of projected vectors belonging to the same class. LDA realizes the second target by increasing the distances between different class centers, whereas NAP realizes the second target by increasing the distances between pairs of projected vectors belonging to different classes.

As the objective function of NAP is based on pairwise distances, a good way of determining the weight of each pairwise distance may lead to a good performance. From this perspective, the design of the objective function of NAP is more flexible than that of LDA. Nonetheless, the weight parameter can also be included in the expression of the scatter matrices in LDA, which makes the design of LDA flexible, too.

Chapter 8 Feature Projection Techniques: Experimental Comparison

This chapter evaluates the effectiveness of NAP, KNAP, LDA, and KDA as a projection technique for improving the discrimination ability of the raw feature representation. A speaker identification task is done using the Ahumada dataset. The raw feature representations are 1280-dimensional GSV with $\beta = 0$ and 1280-dimensional i-vector estimated with 5 EM iterations, based on a 64-mixture UBM. The kernel-based projection (i.e., KNAP and KDA) adopts a Gaussian kernel defined by (8.1), where d is the kernel parameter being varied in the experiments. The property of the Gaussian kernel is that it implicitly maps the raw feature representation from its original feature space to an infinite-dimensional space (i.e., $\phi(\boldsymbol{x}_i)$ has a dimensionality of infinity) [106].

$$\ker(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j) = \exp\left\{-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{d}\right\} \tag{8.1}$$

Since the projection technique ends up solving an eigenvalue problem, the total number of possible projection directions are the total number of eigenvectors. In our experiments, all the projection directions are used instead of choosing several most significant ones (e.g., choosing the eigenvectors with large eigenvalues), as we believe the more the projection directions, the more the class information the projected vectors will carry. This results in a dimensionality of 1280×1 (i.e., the dimensionality of the raw feature representation) for NAP and LDA, and a dimensionality of 1199×1 (i.e., the number of training data) for KNAP and KDA.

The experimental results using GSV as the raw feature representation are shown in Figures 8.1 ~ 8.3. Figure 8.1 compares the linear SVM and the kernel SVM. Different values of the parameter C and the kernel parameter d are tried. Figure 8.2 shows the effectiveness of applying NAP and KNAP to GSV.

Different values of the regularization parameter γ and the kernel parameter d are tried. Figure 8.3 shows the effectiveness of applying LDA and KDA to GSV. Both versions of LDA and KDA are investigated, and the corresponding regularization parameter α and the kernel parameter d vary. After applying the projection techniques, a linear SVM with $C = 1$ is employed as the classifier.

As shown in Figure 8.1, linear SVM is insensitive to different values of C, so it is safe to set $C = 1$ in general. However, this is not true for SVM with the Gaussian kernel. Gaussian kernel maps GSV to an infinite-dimensional space. In that extremely high dimensional space, the mapped feature vectors should be linearly separable. Therefore, a large value of C should be chosen to ensure a small value of ξ_n, so that there are few training data lying on the wrong side of the separating hyperplanes in the infinite-dimensional space.

As shown in Figure 8.2, NAP does not take effect at all, and KNAP gives even worse performance. Albeit NAP resembles LDA in some sense, the way it finds out the projection directions is ineffective. Although both NAP and LDA find the projection directions by finding the eigenvectors, the choices of the eigenvectors are not unique. Thus, the way of constructing the objective function does matter. In addition, the choice of the value of the regularization parameter γ does not exert any influence. Nevertheless, when compared to LDA, NAP is more flexible in that the weight coefficient W_{ij} can be dependent on each pair of training data $(\boldsymbol{x}_i, \boldsymbol{x}_j)$, such that different pairs of data are weighted differently.

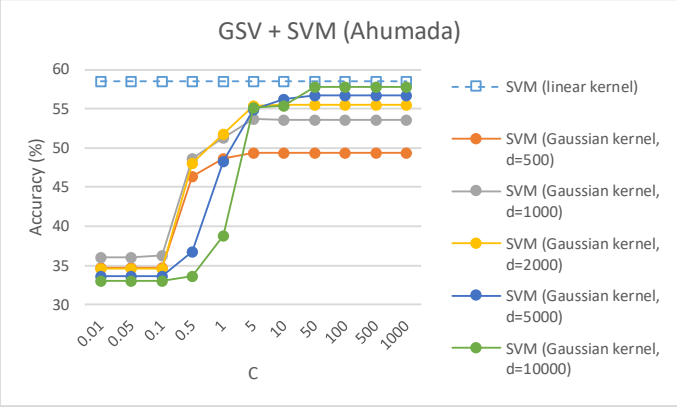


Figure 8.1 Linear SVM vs. kernel SVM using GSV as the raw feature representation.

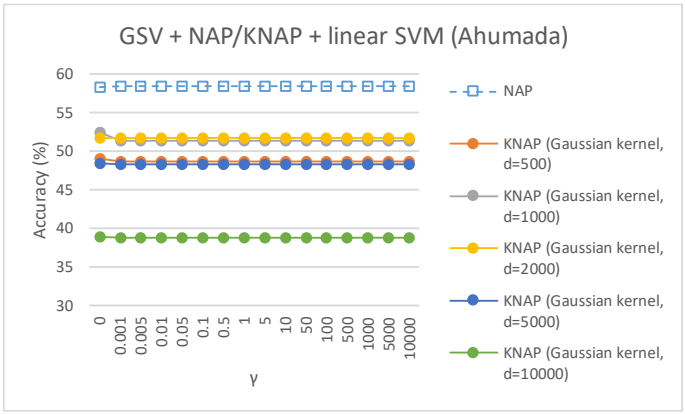
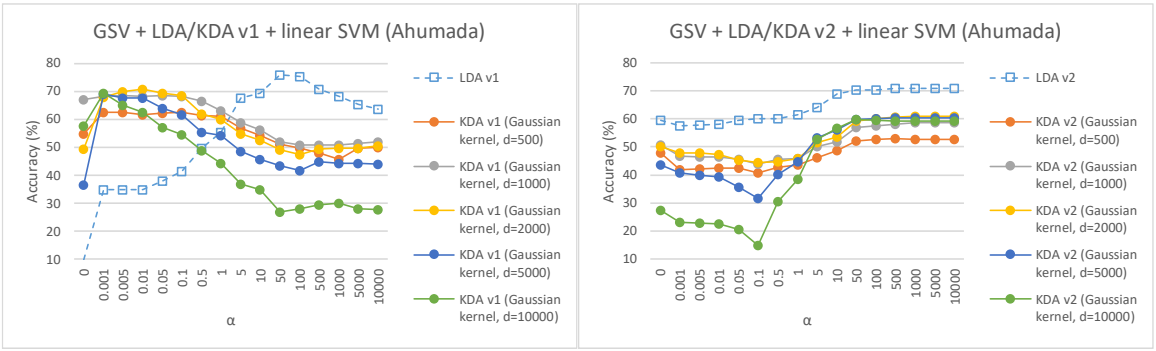


Figure 8.2 NAP vs. kernel NAP using GSV as the raw feature representation.



(a) (b)

Figure 8.3 LDA vs. KDA using GSV as the raw feature representation. (a) LDA v1 and KDA v1. (b) LDA v2 and KDA v2.

As shown in Figure 8.3, both versions of LDA can significantly improve the performance of GSV (more than 10% in accuracy). LDA (v1) seems to be more favorable than LDA (v2), but is more sensitive to the choice of the regularization parameter. KDA is worse than LDA, which is similar to the observation on NAP and KNAP, indicating that an implicit feature mapping before the projection seems unnecessary, as GSV already has a high dimensionality.

The large variation of the performance for different values of α indicates the importance of the regularization parameter. Actually, α has different meanings for LDA (v1) and LDA (v2). Both formulations include 1) the scatter matrix $\boldsymbol{S}_W$ representing the within-class aggregation, and 2) the scatter matrix $\boldsymbol{S}_B$ representing the between-class separation. Specifically, LDA (v1) aims at finding the eigenvectors of $(\boldsymbol{S}_W + \alpha \boldsymbol{I})^{-1}\boldsymbol{S}_B$, where the larger the value of α, the weaker the influence of $\boldsymbol{S}_W$, and the stronger the influence of $\boldsymbol{S}_B$. So, for LDA (v1), the larger the value of α, the more it focuses on the

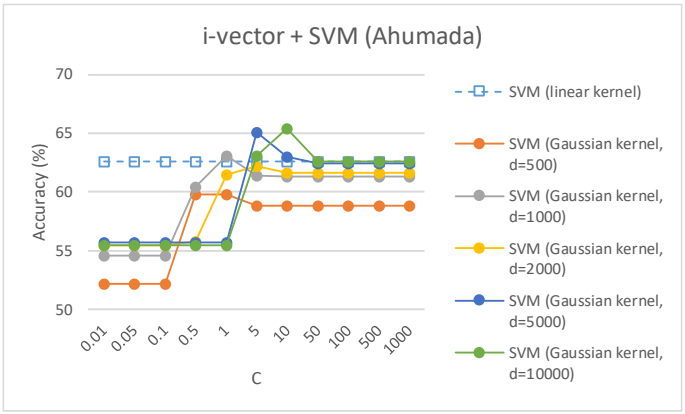
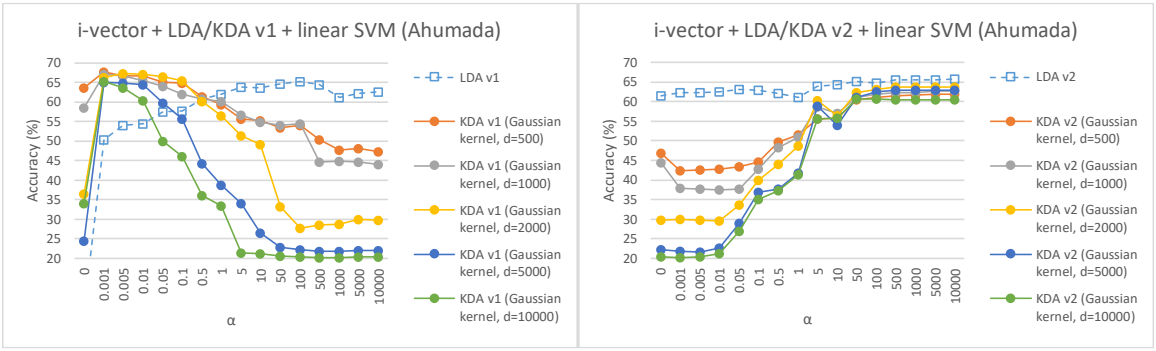


Figure 8.4 Linear SVM vs. kernel SVM using i-vector as the raw feature representation.



(a) (b)

Figure 8.5 LDA vs. KDA using i-vector as the raw feature representation. (a) LDA v1 and KDA v1. (b) LDA v2 and KDA v2.

separation of different classes in the projected space. In contrast, LDA (v2) aims at finding the eigenvectors of $(\boldsymbol{S}_B - \alpha \boldsymbol{S}_W)$, where the larger the value of α, the stronger the influence of $\boldsymbol{S}_W$. So, for LDA (v2), the larger the value of α, the more it focuses on the aggregation of the projected vectors within the same class.

The experimental results using i-vector as the raw feature representation are shown in Figure 8.4 and Figure 8.5. Figure 8.4 compares the linear SVM and the kernel SVM, while Figure 8.5 shows the effectiveness of applying LDA and KDA to i-vector. On using i-vector, the kernel SVM may offer small improvements over the linear SVM with suitably chosen values of C and d as shown in Figure 8.4. Actually, the combination of i-vector and SVM is better than the combination of GSV and SVM by comparing Figure 8.1 and Figure 8.4. Still, i-vector leaves less room for LDA to further improve its performance (about 3% improvement in accuracy) as shown in Figure 8.5, when compared with GSV

(more than 10% improvement in accuracy) as shown in Figure 8.3. The model assumption of i-vector states that the latent vector follows a Gaussian distribution, which implies the elements in different dimensions of an i-vector are supposed to be independent of each other. However, the projection technique finds the projected space by exploiting the relationship between different elements in the raw feature representation, and thus may fail when i-vector is used, whose elements are in principle decorrelated. This phenomenon reveals that the projection techniques highly depend on the characteristics of the raw feature representation.

Chapter 9 Conclusion

9.1 Major Findings

In this thesis, we focus on acoustic and speech signal classification. Three fundamental parts composing a classification system are investigated, which are 1) feature representations, 2) feature transformation techniques, and 3) classifiers. All these parts are important and influencing one another.

The feature representation captures the characteristics of an acoustic sample. The information it carries should be abundant, which requires its dimensionality to be high. Example high-dimensional feature representations include GSV and i-vector. GSV is obtained by MAP adapting the GMM-based UBM, so its computation is quite efficient. However, its dimensionality is determined by the number of mixture components in the GMM, which is unchangeable when the GMM is fixed. I-vector is obtained by estimating an FA model from the GMM, which requires estimating extra model parameters. As the size of the factor-loading matrix in the FA can be large, the computation can be inefficient. Nonetheless, the dimensionality of i-vector depends on the dimensionality of the latent vector in the FA, which is independent of the number of mixture components in the GMM, and thus is changeable.

To make a compromise between the efficiency in computation and the flexibility in dimensionality, we propose the MFALV, which is obtained by estimating an MFA from the GMM. Estimating the model parameters of the MFA is more efficient than that of the FA, as the size of the factor-loading matrix in the MFA is small. Nonetheless, the dimensionality of MFALV is proportional to the number of mixture components, so it is changeable but not as flexible as that of i-vector. The experimental results on two speaker identification tasks show that the performance of MFALV is comparable to or even better than that of i-vector. In brief, GSV has the highest computation efficiency, i-vector has the best flexibility in dimensionality, while MFALV lies in between. Different feature representations have their own strengths and weaknesses.

Observing that GSV, i-vector and MFALV are all based on a GMM-based universal background model, we propose the generic supervector, which is the generalization of GSV and MFALV. I-vector can then be obtained by post-processing the generic supervector. The formulation of the generic supervector provides a general design framework, which helps design new types of feature representations. Since the generic supervector is the weighted sum of a mapped vector and a calibration vector, it can be used to interpret the feature representation produced by a convolutional layer in a classic CNN. The inclusion of the calibration vector also helps intuitively explain the robustness of the residual network.

SVM and PLDA are two widely used classification models for acoustic and speech signal classification tasks. While SVM has been a mature classification model, PLDA has a scalability issue that hinders it from efficiently handling large numbers of training data, especially when the data have high dimensionality. Facing this issue, we propose a scalable formulation such that PLDA can efficiently make predictions even if the number of training data is large. Despite its superior performance for doing speaker identification tasks, PLDA does a bad job on acoustic scene classification tasks, probably owing to the violation of the model assumption. Besides, building a PLDA model consumes much more time than building an SVM.

The dictionary-based classifier, such as the SR-based classifier and the CR-based classifier, is a model-free classifier. When compared to SR, CR is more computationally efficient. In order to further boost the discrimination ability of CR, we propose the DCR, which provides improvements over CR in both speaker identification and acoustic scene classification tasks. The time consumption of DCR is much lower than that of PLDA and slightly higher than that of SVM. Four datasets have been used to evaluate the performance of different classifiers. The experimental results demonstrate that, in general, no one classifier always surpasses the others in all tasks.

We have also investigated two probabilistic models, viz. GMM and RBM. The model assumption determines that GMM is good at handling low-dimensional decorrelated feature vectors, whereas RBM is good at handling high-dimensional correlated feature vectors. As a probability estimator, GMM is better than RBM, which is demonstrated by our experiments on acoustic scene classification. Nevertheless, the most important use of RBM is working as the building block for a DBN. By adding

a softmax layer at the end of a DBN, the DBN-DNN is formed, which can then be fine-tuned for doing classification tasks. It is noteworthy that tuning the softmax layer only cannot offer a good performance. The improvement is observed only by tuning all the layers of the DBN-DNN. This observation indicates that the DBN trained in an unsupervised manner does not produce an expressive feature representation. In addition, high dimensionality of the input feature vector is very important for RBM and DBN-DNN to effectively learn the relationship between the elements inside the input feature vector. If the dimensionality of the input feature vector is low, increasing the depth of DBN-DNN may even degrade its performance.

The discriminative classification model, such as SVM, assumes the data to be separable. So, if the data are inseparable, SVM may fail. The generative classification models, such as GMM, do not require the data to be separable. The model is constructed based on the characteristics of all the training data in a class, so the decision of whether a vector should belong to a class, is made by the joint distribution of the vector and all the training data in that class. In this case, the boundaries of different classes can be overlapped.

Two widely used projection techniques, viz. LDA and NAP, are analyzed in this thesis. As the generalization of LDA and NAP, the kernel-based formulations for LDA and NAP, viz. KDA and KNAP respectively, are derived in detail. We then analyze the relationship between LDA and NAP, proving that NAP is equivalent to a formulation of LDA under some condition. It is also found that using a non-linear kernel may not improve the effectiveness of the projection technique, as sometimes the implicit feature mapping provided by the kernel is unnecessary. Nevertheless, the kernel-based formulation generalizes the original formulation and provides the possibility of trying different kernel functions for implicit feature mappings.

As a projection technique, both LDA and KDA can significantly improve the performance of GSV for speaker identification tasks, but NAP and KNAP do not take any effect at all. Although the objective function of NAP is equivalent to that of LDA under some condition, the way NAP finds the projection directions is not as effective as LDA. However, both LDA and KDA fail on i-vector, as the elements in an i-vector are approximately independent of each other, which probably hinders the projection

techniques to exploit the relationship between different elements. This phenomenon reveals that the effectiveness of the projection techniques highly depends on the structure of the feature representation. Some feature representations work well but leave little space for the projection technique to offer improvements, while some feature representations work poorly but make room for the projection technique to take effect.

In summary, different feature representations have different characteristics in terms of effectiveness and efficiency. It is indeed difficult for a feature representation to hold the highest effectiveness and the highest efficiency at the same time. All have their own strengths and weaknesses. The effectiveness of the feature transformation techniques also highly depends on the characteristics of the feature representation. Similarly, no one classifier can always surpass the others in all cases. Different classifiers make different model assumptions on the feature representation, so the most important thing is to choose the most suitable combination of the feature representation and the classifier, such that the assumptions of both are matched.

9.2 Future Directions

The formulation of the generic supervector provides a general design framework. Using different background models or choosing different feature mapping functions may yield different types of feature representations. For instance, the background model can be a GMM, an MFA, a DBN, a DNN or the combination of them. Future research can then be focused on the design of heterogenous supervectors, where the background model is composed of different types of models. The generic supervector may also have multiple levels, similar to the structure of a DNN. However, the major difference between a deep supervector and a DNN is that different levels in a deep supervector can be based on different types of background models and different construction mechanisms.

Including a kernel in SVM has been shown to be effective for long. The kernel trick can also be combined with other existing techniques, as long as the objective function only involves the inner products of the feature vectors, such as the sparse representation and the collaborative representation. It is also feasible to apply the kernel trick to a neural network, as a neural network performs matrix

multiplications layer by layer. We may also apply the kernel trick to a multivariate GMM if different variables share the same variance [108], which enables GMM to model feature distributions in the kernel space instead of the original feature space. If using the Gaussian kernel, the GSV, which is based on a GMM, will have infinite dimensions.

References

[1] J. H. L. Hansen and T. Hasan, "Speaker recognition by machines and humans: a tutorial review," *IEEE Signal Processing Magazine*, vol. 32, no. 6, pp. 74-99, 2015.

[2] C. Hanilci, F. Ertas, T. Ertas, and O. Eskidere, "Recognition of brand and models of cell-phones from recorded speech signals," *IEEE Trans. on Information Forensics and Security*, vol. 7, no. 2, pp. 625-634, 2012.

[3] S. Gupta, S. Cho, and C. C. J. Kuo, "Current developments and future trends in audio authentication," *IEEE Multimedia*, vol. 19, pp. 50-59, Jan. 2012.

[4] R. Garg, A. L. Varna, and M. Wu, "Modeling and analysis of electric network frequency signal for timestamp verification," in *Proc. IEEE Int. Workshop on Information Forensics and Security*, 2012, pp. 67-72.

[5] H. Zhao and H. Malik, "Audio recording location identification using acoustic environment signature," *IEEE Trans. on Information Forensics and Security*, vol. 8, no. 11, pp. 1746-1759, 2013.

[6] D. Barchiesi, D. Giannoulis, D. Stowell, and M. D. Plumbley, "Acoustic scene classification: classifying environments from the sounds they produce," *IEEE Signal Processing Magazine*, vol. 32, no. 3, pp. 16-34, 2015.

[7] W. M. Campbell, D. E. Sturim, and D. A. Reynolds, "Support vector machines using GMM supervectors for speaker verification," *IEEE Signal Processing Letters*, vol. 13, no. 5, pp. 308-311, 2006.

[8] N. Dehak, P. J. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, "Front-end factor analysis for speaker verification," *IEEE Trans. on Audio, Speech, and Language Processing*, vol. 19, no. 4, pp. 788-798, 2011.

[9] D. Reynolds, "Gaussian mixture models," in *Encyclopedia of Biometrics*, 2015, pp. 827-832.

[10] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2007, pp. 153-160.

[11] T. Kinnunen and H. Li, “An overview of text-independent speaker recognition: from features to supervectors,” *Speech Communication*, vol. 52, no. 1, pp. 12-40, 2010.

[12] X. Huang, A. Acero, and H. W. Hon, “Speech signal representations,” in *Spoken Language Processing: A Guide to Theory, Algorithm and System Development*. Upper Saddle River, NJ: Prentice Hall PTR, 2001, ch. 6, pp. 273-333.

[13] G. Hinton *et al.*, “Deep neural networks for acoustic modelling in speech recognition: the shared views of four research groups,” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82-97, 2012.

[14] D. Luo, P. Korus, and J. Huang, “Band energy difference for source attribution in audio forensics,” *IEEE Trans. on Information Forensics and Security*, vol. 13, no. 9, pp. 2179-2189, 2018.

[15] M. Todisco *et al.*, “Integrated presentation attack detection and automatic speaker verification: Common features and Gaussian back-end fusion,” in *Proc. Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2018, pp. 77-81.

[16] H. Lee, P. Pham, Y. Largman, and A. Y. Ng, “Unsupervised feature learning for audio classification using convolutional deep belief networks,” in *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2009, pp. 1096-1104.

[17] Y. Li *et al.*, “Mobile phone clustering from speech recordings using deep representation and spectral clustering,” *IEEE Trans. on Information Forensics and Security*, vol. 13, no. 4, pp. 965-977, 2018.

[18] F. Richardson, D. Reynolds, and N. Dehak, “Deep neural network approaches to speaker and language recognition,” *IEEE Signal Processing Letters*, vol. 22, no. 10, pp. 1671-1675, 2015.

[19] J. Li, W. Dai, F. Metze, S. Qu, and S. Das, “A comparison of deep learning methods for environmental sound detection,” in *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 126-130.

[20] Y. Zhang and J. R. Glass, “Unsupervised spoken keyword spotting via segmental DTW on Gaussian posteriorgrams,” in *Proc. IEEE Workshop on Automatic Speech Recognition & Understanding*, 2009, pp. 398-403.

[21] S. Chachada and C. C. J. Kuo, “Environmental sound recognition: a survey,” in *IEEE Proc. Asia Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, 2013, pp. 1-9.

[22] D. A. Reynolds, “Robust text-independent speaker identification using Gaussian mixture speaker models,” *IEEE Trans. on Speech and Audio Processing*, vol. 3, no. 1, pp. 72-83, 1995.

[23] D. A. Reynolds, T. F. Quatieri, and R. B. Dunn, “Speaker verification using adapted Gaussian mixture models,” *Digital Signal Processing*, vol. 10, pp. 19-41, 2000.

[24] A. D. Dileep and C. C. Sekhar, “GMM-based intermediate matching kernel for classification of varying length patterns of long duration speech using support vector machines,” *IEEE Trans. on Neural Networks and Learning Systems*, vol. 25, no. 8, pp. 1421-1432, 2014.

[25] D. Garcia-Romero and C. Y. Espy-Wilson, “Automatic acquisition device identification from speech recordings,” in *Proc. IEEE Int. Conf. on Acoustic, Speech and Signal Processing (ICASSP)*, 2010, pp. 1806-1809.

[26] C. L. Kotropoulis, “Source phone identification using sketches of features,” *IET Biometric*, vol. 3, no. 2, pp. 75-83, 2014.

[27] L. Jing *et al.*, “DCAR: a discriminative and compact audio representation for audio processing,” *IEEE Trans. on Multimedia*, vol. 19, no. 12, pp. 2637-2650, 2017.

[28] V. Vestman, B. Soomro, A. Kanervisto, V. Hautamaki, and T. Kinnunen, “Who do I sound like? Showcasing speaking recognition technology by youtube voice search,” in *Proc. IEEE Int. Conf. on Acoustic, Speech and Signal Processing (ICASSP)*, 2019, pp. 5781-5785.

[29] A. Mesaros *et al.*, “Detection and classification of acoustic scenes and events: outcome of the dcase 2016 challenge,” *IEEE/ACM Trans. on Audio, Speech, and Language Processing*, vol. 26, no. 2, pp. 379-393, 2018.

[30] H. Delgado *et al.*, “ASVspoof 2017 version 2.0: meta-data analysis and baseline enhancements,” in *Proc. Odyssey*, 2018, pp. 296-303.

[31] Y. Li, M. Liu, W. Wang, Y. Zhang, and Q. He, “Acoustic scene clustering using joint optimization of deep embedding learning and clustering iteration,” *IEEE Trans. on Multimedia*, early access, 2019.

[32] L. Xu, K. A. Lee, H. Li, and Z. Yang, "Generalizing i-vector estimation for rapid speaker recognition," *IEEE/ACM Trans. on Audio, Speech, and Language Processing*, vol. 26, no. 4, pp. 749-759, 2018.

[33] V. Vestman and T. Kinnunen, "Supervector compression strategies to speed up i-vector system development," in *Proc. Odyssey*, 2018, pp. 357-364.

[34] S. Cumani and P. Laface, "E-vectors: JFA and i-vectors revisited," in *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 5435-5439.

[35] F. Perronnin and C. Dance, "Fisher kernels on visual vocabularies for image categorization," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2007, pp. 1-8.

[36] J. Sanchez, F. Perronnin, T. Mensink, and J. Verbeek, "Image classification with the fisher vector: Theory and practice," *International Journal of Computer Vision*, vol. 105, no. 3, pp. 222-245, 2013.

[37] T. S. Jaakkola and D. Haussler, "Exploiting generative models in discriminative classifiers," in *Proc. Advances in Neural Information Processing Systems (NIPS)*, 1999, pp. 487-493.

[38] W. M. Campbell, "Using deep belief networks for vector-based speaker recognition," in *Proc. Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2014, pp. 676-680.

[39] Y. Lei, N. Scheffer, L. Ferrer, and M. McLaren, "A novel scheme for speaker recognition using a phonetically-aware deep neural network," in *Proc. IEEE Int. Conf. on Acoustic, Speech and Signal Processing (ICASSP)*, 2014, pp. 1695-1699.

[40] E. Variani, X. Lei, E. McDermott, I. L. Moreno, and J. Gonzalez-Dominguez, "Deep neural networks for small footprint text-dependent speaker verification," in *Proc. Int. Conf. on Acousitc, Speech and Signal Processing (ICASSP)*, 2014, pp. 4052-4056.

[41] D. Snyder, D. Garcia-Romero, D. Povey, and S. Khudanpur, "Deep neural network embeddings for text-independent speaker verification," in *Proc. Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2017, pp. 999-1003.

[42] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, and S. Khudanpur, "X-vectors: robust dnn embeddings for speaker recognition," in *Proc. Int. Conf. on Acoustic, Speech and Signal Processing (ICASSP)*, 2018, pp. 5329-5333.

[43] R. Hyder, S. Ghaffarzadegan, Z. Feng, J. H. L. Hansen, and T. Hasan, "Acoustic scene classification using a CNN-supervector system trained with auditory and spectrogram image features," in *Proc. Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2017, pp. 3073-3077.

[44] G. Lavrentyeva *et al.*, "Audio replay attack detection with deep learning frameworks," in *Proc. Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2017, pp. 82-86.

[45] O. Eskidere, "Source microphone identification from speech recordings based on a Gaussian mixture model," *Turkish Journal of Electrical Engineering & Computer Sciences*, vol. 22, pp. 754-767, 2014.

[46] N. Peters, H. Lei, and G. Friedland, "Name that room: room identification using acoustic features in a recording," in *Proc. 20th ACM Int. Conf. on Multimedia*, 2012, pp. 841-844.

[47] A. J. Eronen *et al.*, "Audio-based context recognition," *IEEE Trans. on Audio, Speech, and Language Processing*, vol. 14, no. 1, pp. 321-329, 2006.

[48] G. E. Hinton, "A practical guide to training restricted Boltzmann machines," in *Neural Networks: Tricks of the Trade*, Springer, 2012, pp. 599-619.

[49] O. Ghahabi and J. Hernando, "Restricted Boltzmann machine supervectors for speaker recognition," in *Proc. Int. Conf. on Acoustic, Speech and Signal Processing (ICASSP)*, 2015, pp. 4804-4808.

[50] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527-1554, 2006.

[51] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504-507, 2006.

[52] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Trans. on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 30-42, 2012.

[53] C. M. Bishop, "Sparse kernel machines," in *Pattern Recognition and Machine Learning*, Springer, 2006, ch. 7, pp. 325-358.

[54] S. Chandrakala and S. L. Jayalakshmi, "Environmental audio scene and sound event recognition for autonomous surveillance: A survey and comparative studies," *ACM Computing Surveys*, vol. 52, no. 3, 2019.

[55] L. Ferrer and M. McLaren, "Joint plda for simultaneous modeling of two factors," *Journal of Machine Learning Research*, vol. 20, no. 24, pp. 1-29, 2019.

[56] B. Desplanques, K. Demuynck, and J. P. Martens, "Factor analysis for speaker segmentation and improved speaker diarization," in *Proc. Annual Conf. of the International Speech Communication Association (INTERSPEECH)*, 2015, pp. 3081-3085.

[57] A. Nagrani, J. S. Chung, and A. Zisserman, "VoxCeleb: A large-scale speaker identification dataset," in *Proc. Annual Conf. of the International Speech Communication Association (INTERSPEECH)*, 2017, pp. 2616-2620.

[58] J. S. Chung, A. Nagrani, and A. Zisserman, "VoxCeleb2: Deep speaker recognition," *Proc. Annual Conf. of the International Speech Communication Association (INTERSPEECH)*, 2018, pp. 1086-1090.

[59] J. Wright *et al.*, "Sparse representation for computer vision and pattern recognition," *Proceedings of the IEEE*, vol. 98, no. 6, pp. 1031-1044, 2010.

[60] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma, "Robust face recognition via sparse representation," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 31, no. 2, pp. 210-227, 2009.

[61] K. Huang and S. Aviyente, "Sparse representation for signal classification," in *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2007, pp. 609-616.

[62] Y. Panagakis, C. L. Kotropoulos, and G. R. Arce, "Music genre classification via joint sparse low-rank representation of audio features," *IEEE/ACM Trans. on Audio, Speech, and Language Processing*, vol. 22, no. 12, pp. 1905-1917, 2014.

[63] J. F. Gemmeke, T. Virtanen, and A. Hurmalainen, "Exemplar-based sparse representations for noise robust automatic speech recognition," *IEEE Trans. on Audio, Speech, and Language Processing*, vol. 19, no. 7, pp. 2067-2080, 2011.

[64] M. Li, X. Zhang, Y. Yan, and S. Narayanan, "Speaker verification using sparse representations on total variability i-vectors," in *Proc. 12th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2011, pp. 2729-2732.

[65] T. Lin and Y. Zhang, "Speaker recognition based on long-term acoustic features with analysis sparse representation," *IEEE Access*, vol. 7, pp. 87439-87447, 2019.

[66] L. Zou, Q. He, and J. Wu, "Source cell phone verification from speech recordings using sparse representation," *Digital Signal Processing*, vol. 62, pp. 125-136, 2017.

[67] I. Naseem, R. Togneri, and M. Bennamoun, "Sparse representation for speaker identification," in *Proc. IEEE Int. Conf. on Pattern Recognition (ICPR)*, 2010, pp. 4460-4463.

[68] Y. H. Chin, J. C. Wang, C. L. Huang, K. Y. Wang, and C. H. Wu, "Speaker identification using discriminative features and sparse representation," *IEEE Trans. on Information Forensics and Security*, vol. 12, no. 8, pp. 1979-1987, 2017.

[69] L. Zhang, M. Yang, and X. Feng, "Sparse representation or collaborative representation: which helps face recognition," in *Proc. IEEE Int. Conf. on Computer Vision (ICCV)*, 2011, pp. 471-478.

[70] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, "Eigenfaces vs. fisherfaces: recognition using class specific linear projection," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 711-720, 1997.

[71] S. Mika, G. Ratsch, J. Weston, B. Scholkopf, and K. R. Muller, "Fisher discriminant analysis with kernels," in *Proc. IEEE Signal Processing Society Workshop*, 1999, pp. 41-48.

[72] G. Baudat and F. Anouar, "Generalized discriminant analysis using a kernel approach," *Neural Computation*, vol. 12, no. 10, pp. 2385-2404, 2000.

[73] R. K. Das, A. B. Manam, and S. R. M. Prasanna, “Exploring kernel discriminant analysis for speaker verification with limited test data,” *Pattern Recognition Letters*, vol. 98, pp. 26-31, 2017.

[74] A. Solomonoff, W. M. Campbell, and I. Boardman, “Advances in channel compensation for SVM speaker recognition,” in *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, 2005, pp. 629-632.

[75] V. Struc, B. Vesnicer, F. Mihelic, and N. Pavesic, “Removing illumination artifacts from face images using the nuisance attribute projection,” in *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, 2010, pp. 846-849.

[76] A. Solomonoff, W. M. Campbell, and C. Quillen, “Nuisance attribute projection,” *Speech Communication*, pp. 1-73, 2007.

[77] X. Zhao *et al.*, “Nonlinear kernel nuisance attribute projection for speaker verification,” in *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, 2008, pp. 4125-4128.

[78] Y. Jiang, K. A. Lee, Z. Tang, B. Ma, A. Larcher, and H. Li, “PLDA modeling in i-vector and supervector space for speaker verification,” in *Proc. Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2012, pp. 1680-1683.

[79] L. E. Shafey, C. McCool, R. Wallace, and S. Marcel, “A scalable formulation of probabilistic linear discriminant analysis: applied to face recognition,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 35, no. 7, pp. 1788-1794, 2013.

[80] KingLine Data Center, American English Speech Recognition Corpus (King-ASR-L-081), Speechocean, 2013.

[81] J. O. Garcia, J. G. Rodriguez, and V. M. Aguiar, “AHUMADA: a large speech corpus in Spanish for speaker characterization and identification,” *Speech Communication*, vol. 31, no. 2, pp. 255-264, 2000. Available at: http://atvs.ii.uam.es/atvs/ahumada25.html.

[82] D. Stowell, D. Giannoulis, E. Benetos, M. Lagrange, and M. D. Plumbley, “Detection and classification of acoustic scenes and events,” *IEEE Trans. on Multimedia*, vol. 17, no. 10, pp. 1733-1746, 2015. Available at: http://c4dm.eecs.qmul.ac.uk/sceneseventschallenge/.

[83] A. Mesaros, T. Heittola, and T. Virtanen, "TUT database for acoustic scene classification and sound event detection," in *Proc. IEEE 24th European Signal Processing Conference (EUSIPCO)*, 2016, pp. 1128-1132. Available at: http://dcase.community/challenge2016/task-acoustic-scene-classification.

[84] M. N. Do, "Fast approximation of Kullback-Leibler distance for dependence trees and hidden Markov models," *IEEE Signal Processing Letters*, vol. 10, no. 4, pp. 115-118, 2003.

[85] C. C. Chang and C. J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. on Intelligent Systems and Technology*, vol. 2, no. 3, pp. 1-27, 2011. Available at: http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[86] P. Kenny, G. Boulianne, and P. Dumouchel, "Eigenvoice modeling with sparse training data," *IEEE Trans. on Speech and Audio Processing*, vol. 13, no. 3, pp. 345-354, 2005.

[87] C. M. Bishop, "Continuous latent variables," in *Pattern Recognition and Machine Learning*, Springer, 2006, ch. 12, pp. 559-603.

[88] Z. Ghahramani and G. E. Hinton, "The EM algorithm for mixtures of factor analyzers," *Technical Report CRG-TR-96-1*, University of Toronto, 1996.

[89] S. Young *et al.*, *The HTK book (v3.4)*, Cambridge: Cambridge University Press, 2006, pp. 156-157.

[90] Y. Jiang and F. H. F. Leung, "The scalable version of probabilistic linear discriminant analysis and its potential as a classifier for audio signal classification," in *Proc. IEEE Int. Joint Conf. on Neural Networks (IJCNN)*, 2018, pp. 1-7.

[91] C. M. Bishop, "Mixture models and EM," in *Pattern Recognition and Machine Learning*, Springer, 2006, ch. 9, pp. 423-459.

[92] Y. Bengio, "Energy-based models and Boltzmann machines," in *Learning Deep Architectures for AI*, Foundations and Trends® in Machine Learning, 2009, ch. 5, pp. 48-67.

[93] R. B. Palm, "Prediction as a candidate for learning deep hierarchical models of data," *Technical University of Denmark*, 2012.
Available at: https://github.com/rasmusbergpalm/DeepLearnToolbox.

[94] M. Brookes, Voicebox, 2003.

Available at: http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html.

[95] C. W. Hsu and C. J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE Trans. on Neural Networks*, vol. 13, no. 2, pp. 415-425, 2002.

[96] X. Yang, Q. Song, and Y. Wang. "A weighted support vector machine for data classification." *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 21, no. 5, pp. 961-976, 2007.

[97] S. J. D. Prince and J. H. Elder, "Probabilistic linear discriminant analysis for inferences about identity," in *Proc. IEEE Int. Conf. on Computer Vision (ICCV)*, 2007, pp. 1-8.

[98] P. Li, Y. Fu, U. Mohammed, J. H. Elder, and S. J. D. Prince, "Probabilistic models for inference about identity," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 34, no. 1, pp. 144-157, 2012.

[99] S. Ioffe, "Probabilistic linear discriminant analysis," in *Proc. European Conf. on Computer Vision (ECCV)*, 2006, pp. 531-542.

[100] C. M. Bishop, "Probability distributions," in *Pattern Recognition and Machine Learning*, Springer, 2006, ch. 2, pp. 67-136.

[101] A. Y. Yang, Z. Zhou, A. G. Balasubramanian, S. S. Sastry, and Y. Ma, "Fast l1-minimization algorithms for robust face recognition," *IEEE Trans. on Image Processing*, vol. 22, no. 8, pp. 3234-3246, 2013.

[102] Y. Xu, Z. Zhong, J. Yang, J. You and D. Zhang, "A new discriminative sparse representation method for robust face recognition via l2 regularization," *IEEE Trans. on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2233-2242, 2017.

[103] D. Donoho, V. Stodden, and Y. Tsaig, *About SparseLab(v2.1)*, Stanford, 2007.

[104] C. M. Bishop, "Linear models for classification," in *Pattern Recognition and Machine Learning*, Springer, 2006, ch. 4, pp. 179-224.

[105] K. Fukunaga, "Feature Extraction and Linear Mapping for Classification," in *Introduction to Statistical Pattern Recognition*, 2nd Ed., San Diego, California: Academic Press, 1990, ch. 10, pp. 441-507.

[106] C. J. C. Burges, “A tutorial on support vector machines for pattern recognition,” *Data mining and knowledge discovery*, vol. 2, no. 2, pp. 121-167, 1998.

[107] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770-778.

[108] Y. Jiang and F. H. F. Leung, “Gaussian mixture model and Gaussian supervector for image classification,” in *Proc. IEEE Int. Conf. on Digital Signal Processing (DSP)*, 2018, pp. 1-5.

[109] R. Rigamonti, M. A. Brown, and V. Lepetit, “Are sparse representations really relevant for image classification,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2011, pp. 1545-1552.

[110] Q. Shi, A. Eriksson, A. V. D. Hengel, and C. Shen, “Is face recognition really a compressive sensing problem,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2011, pp. 553-560.