Supervised Learning Approaches for Language and Speaker Recognition

A THESIS

SUBMITTED FOR THE DEGREE OF

Doctor of Philosophy

IN THE

Faculty of Engineering

BY

Shreyas Ramoji

INDIAN INSTITUTE OF SCIENCE

भारतीय विज्ञान संस्थान

Department of Electrical Engineering

Indian Institute of Science

Bangalore – 560 012 (INDIA)

June, 2023

Declaration of Originality

I, **Shreyas Ramoji**, with SR No. **04-03-05-10-12-16-1-14119** hereby declare that the material presented in the thesis titled

Supervised Learning Approaches for Language and Speaker Recognition

represents original work carried out by me in the **Department of Electrical Engineering** at **Indian Institute of Science** during the years **2016-2022**.

With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the thesis.

Advisor Name: Advisor Signature

DEDICATED TO

My parents:

(Late) Sri. Ramoji G

and

Smt. Shashikala K N

for having supported me through thick and thin, instilling in me the importance of honesty and dedication, and always wishing me the best.

Acknowledgements

I express my sincere gratitude to my Ph.D. advisor Dr. Sriram Ganapathy for giving me the opportunity to work with him in the Learning and Extraction of Acoustic Patterns lab, and for his invaluable guidance, support, and encouragement throughout my Ph.D. journey. He has not only emphasized the importance of a healthy work-life balance, but has also been a role model in this regard, by engaging in fitness activities like running and trekking while involving the lab members as well. During the peak of the first wave of the COVID-19 pandemic, when I got infected with my whole family, he helped all of us get admitted to the hospital. When I lost my father during this time, he was highly empathetic, supportive, and gave me as much time as I needed to get back on my feet. On the research front, he gave me enough freedom to follow my interests and approach, while also providing necessary criticism and guidance when required. I honestly believe it is very hard to find an advisor like Dr. Sriram Ganapathy and I am happy to have had him as my mentor.

I thank my comprehensive examination panel members Prof. T. V. Sreenivas, Prof. P. S. Sastry, and Dr. Aditya Gopalan for their appreciation and valuable suggestions towards my work. I would also like to express my gratitude to all my course instructors. Their courses not only provided me with technical knowledge in fundamental areas, but also shaped my critical thinking skills as a researcher.

I sincerely thank Dr. Neeraj Sharma, Dr. Purvi Agrawal, and Akshara Soman for all the technical discussions during the early days of my Ph.D. which have influenced my way of thinking and approach to problem-solving, which were critical for shaping my thesis. They also provided different perspectives on various problems and interpretations which were not only invaluable to my doctoral research, but also to my overall abilities as a researcher.

At the start of my Ph.D. journey, I had no exposure to Python and shell scripting, and I struggled to understand the implementations in the KALDI toolkit despite having a good grasp of the underlying theories. I now consider myself an excellent programmer thanks to my former lab mates Bharat Padi and Anand Mohan. I am deeply grateful to them for dedicating a significant amount of time to assist me in getting started with the KALDI toolkit, shell scripting, and python programming. They were instrumental in my growth as a programmer, and I couldn't have made this progress without their help.

I would like to specially thank all my co-authors of various papers for their contributions, and my teammates in various Speaker and Language Recognition Challenges we participated in: Bharath Padi, Vaishnavi Y, Anand Mohan, Satish Kumar, Anmol Bhatia, Bhargavaram Mysore, Prachi Singh. Special thanks to Prashant Krishnan for helping me run countless experiments with various hyperparameters and acoustic modeling. I sincerely thank Dr. Harish Arsikere for being my mentor during this internship and helping me understand the challenges in scaling speech recognition training to large datasets extending to millions of hours.

I thank all the present and past members of the LEAP Lab for their love and support, and for all the memorable trips we had together that made this journey memorable. I also thank my national and international colleagues with whom I had several valuable interactions during the various conferences and workshops I have been to.

Last but not least, I would like to thank all my close friends and family for their love and encouragement. I am forever grateful to my parents for their unwavering trust in me, and for their abundant love and support in ways that can not truly be explained in words, without which this thesis would not have been complete.

Abstract

In the age of artificial intelligence, it is important for machines to figure out who is speaking automatically and in what language - a task humans are naturally capable of. Developing algorithms that automatically infer the speaker, language, or accent from a given speech segment are challenging problems that have been researched for over three decades. The main aim of this doctoral research was to explore and understand the shortcomings of existing approaches to the problems and propose novel supervised approaches to overcome these shortcomings to develop robust speaker and language recognition systems.

In the first part of this doctoral research, we developed a supervised version of a popular embedding extraction approach called the i-vector, typically used as front-end embeddings for speaker and language recognition. In this approach, a database of speech recordings (in the form of a sequence of short-term feature vectors) is modeled with a Gaussian Mixture Model, called the Universal Background Model (GMM-UBM). The deviation in the mean components is captured in a lower dimensional latent space called the i-vector space using a factor analysis framework. In our research, we proposed a fully supervised version of the i-vector model, where each label class is associated with a Gaussian prior with a class-specific mean parameter. The joint prior (marginalized over the sample space of classes) on the latent variable becomes a GMM. The choice of the prior distribution is motivated by the Gaussian back-end, where the conventional i-vectors for each language are modeled with a single Gaussian distribution. With detailed data analysis and visualization, we show that the s-vector features yield representations that succinctly capture the language (accent) label information, and do a significantly better job distinguishing the various accents of the same language. We performed language recognition experiments on the NIST Language Recognition Evaluation (LRE) 2017 challenge dataset, which has test segments ranging from 3 to 30 seconds. With the s-vector framework, we observed relative improvements between 8% to 20% in terms of the Bayesian detection cost function, 4% to 24% in terms of EER, and 9% to 18% in terms of classification accuracy over the conventional i-vector framework. We also performed language recognition experiments on the RATS dataset and Mozilla CommonVoice dataset, and speaker classification experiments using LibriSpeech, demonstrating similar improvements.

In the second part, we explored the problem of speaker verification, where a binary decision has to be made on a test speech segment as to whether it is spoken by a target speaker or not, based on a limited duration of enrollment speech. We proposed a neural network approach for back-end modeling. The likelihood ratio score of the generative PLDA model was posed as a discriminative similarity function, and the learnable parameters of the score function are optimized using a verification cost, proposed to be an approximation of the minimum detection cost function (DCF). The speaker recognition experiments using the NPLDA model are performed on the speaker verification task in the VOiCES datasets and the SITW challenge dataset. Further, we explore a fully neural approach where the neural model outputs the verification score directly, given the acoustic feature inputs. This Siamese neural network (E2E-NPLDA) model combines the embedding extraction and back-end modeling stages into a single processing pipeline. The development of the single neural Siamese model allows the joint optimization of all the modules using a verification cost. We provide a detailed analysis of the influence of hyper-parameters, choice of loss functions, and data sampling strategies for training these models. Several speaker recognition experiments were performed using Speakers in the Wild (SITW), VOiCES, and NIST SRE datasets where the proposed NPLDA and E2E-NPLDA models are shown to improve over the state-of-art significantly x-vector PLDA baseline system.

Publications based on this Thesis

Peer-reviewed Journal Papers

1. S. Ramoji, S. Ganapathy, "Supervised I-vector modeling for language and accent recognition," *Computer Speech & Language* 60, (2020): p.101030.

2. S. Ramoji, P. Krishnan, S. Ganapathy, "PLDA inspired Siamese networks for speaker verification," *Computer Speech & Language* 76, (2022): p.101383.

Peer-reviewed Conference Papers

1. S. Ramoji, S. Ganapathy, "Supervised I-vector Modeling-Theory and Applications," *Proc. Interspeech* (2018): p.1091-1095.

2. S. Ramoji et. al., "The LEAP speaker recognition system for NIST SRE 2018 challenge," *Proc. ICASSP* (2019): p.5771-5775.

3. S. Ramoji, P. Krishnan, S. Ganapathy, "NPLDA: A Deep Neural PLDA Model for Speaker Verification," in *Proc. Odyssey* (2020): p.202-209.

4. S. Ramoji, P. Krishnan, S. Ganapathy, "LEAP System for SRE 2019 CTS Challenge - Improvements and Error Analysis," in *Proc. Odyssey* (2020): p.281-288.

5. S. Ramoji, P. Krishnan, S. Ganapathy, "Neural PLDA Modeling for End-to-End Speaker Verification," *Proc. Interspeech* (2020): p.4333-4337

Contents

List of Figures

List of Tables

LIST OF TABLES

Chapter 1

Introduction

1.1 Speech - A multitude of information

Speech is a form of sound, made by the human vocal tract to communicate with each other. The information communicated by human speech is not only limited to the meaning of the words, phrases, or sentences spoken, but also contains cues about speaker, gender, age, emotion, and health status. It also carries information about the language that is spoken, and also the accent or dialect of the language, which reveals where the speaker originates from. All of this information is entangled together, and humans are capable of disentangling and discerning most of this information with varying levels of attention and effort.

Several linguists, mathematicians, physicists and engineers over the years have contributed to the understanding of speech in various ways. While linguists have systematically studied language and grammar, scientists have studied and modeled how sound is produced by the vocal tract, and how each phoneme is articulated. The signal processing and machine learning engineers have explored, and are continuing to explore various tools ranging from short-time Fourier analysis to deep learning approaches to build practical speech information processing systems like speech recognition systems, age and height estimation, emotion recognition, speaker/language recognition and diarization systems, etc.

However, given the fact that all of these aforementioned aspects are entangled in a time-varying, one dimensional continuous signal, it is challenging to build intelligent systems that perform these tasks. When we train neural networks for one or more tasks, the linear and non-linear operations performed by the various layers of the network transform the input speech to a vector space which intends to capture the relevant information and suppress the irrelevant information. For example, aspects such as speaker, content, emotion, age, etc are variables that are irrelevant for a language or accent recognition system. However, in practice, it is impossible to completely suppress all of the irrelevant information while preserving a fair amount of relevant information. Despite all these challenges, there has been quite a lot of progress in the area, and there still scope for more progress.

The focus of this thesis is on language and speaker recognition. These problems involve designing systems that learn representations from digital audio signals that capture the language or speaker information from speech recordings. Multiple speakers may speak the same language with the same accent, and multilingual speakers can speak multiple languages in multiple accents, with essentially the *same voice*. The learned representations for a speaker or language recognition systems should ideally capture only the speaker or language information respectively, but in practice, they capture a lot of irrelevant information as well. In this thesis, we report our efforts in designing and analysing systems for language and speaker recognition using supervised learning approaches.

1.2 Speech Signal Processing

Speech is a continuous-time, one-dimensional real-valued signal, that is digitized to be stored and processed using various digital processing approaches. In this section, we give a brief overview of basic signal-processing approaches to extract representations to model speech information.

1.2.1 Short-time Speech Processing

Speech signals are quasi-stationary signals containing a sequence of basic stationary sound units called phonemes, whose durations range from as low as 40 milliseconds to a few hundred milliseconds [1]. To capture such transient information, speech is processed one short-time window after another, with a fixed window length (20-40 milliseconds) and shift (10-20 milliseconds), to extract a sequence of feature vectors that can be further modeled using various machine learning models or neural networks.

Examples include short-time Fourier transform (STFT), log-magnitude STFT (spectrogram), Mel-spectrogram or Mel-filer-bank representations, Mel-frequency cepstral coefficients (MFCC), Linear predictive coding coefficients (LP coefficients) which can be in the time domain (TDLP), frequency domain (FDLP), or both (2D autoregressive features) [2, 3]. Recently, there have also been successful efforts in processing windowed versions of the raw waveform directly rather than using signal processing-based feature extraction methods [4].

1.2.2 Machine Learning for Speech Processing

Machine learning uses a wide variety of algorithms to model real-world signals to make useful predictions. In this thesis, we explore machine learning approaches to infer or validate the speaker or language from spoken audio in challenging environments. The short-time speech representations discussed in section 1.2.1 serve as the starting point for machine learning models, which can be broadly classified into *generative* and *discriminative* models.

1.2.2.1 Generative Models

Generative models are a class of machine learning algorithms that estimate the probability distribution of speech representations. This could refer to the short-time speech features discussed in section 1.2.1 or segment-level representations derived from further processing of these short-time features. This is usually done by choosing a family of probability distributions and estimating its parameters using the available data.

Apart from the speech recordings, the datasets typically include additional information such as the speaker and language labels for every recording. If we estimate the joint distribution of the features and labels or the class conditional distributions for every class, we call it a supervised generative model. If no label information is considered, we call it an unsupervised generative model.

Examples of generative models include Gaussian mixture models (GMM), Hidden Markov models (HMM), Probabilistic principal component analysis (PPCA), Probabilistic linear discriminant analysis (PLDA), variational autoencoders (VAE), Generative adversarial networks (GAN)), etc.

1.2.2.2 Discriminative Models

Discriminative models are a class of machine learning algorithms that learn to map the input speech features or embeddings to the labels by directly modeling the posterior distribution of the labels, or learning the boundaries between classes.

Examples of discriminative models include logistic regression, support vector machines (SVM), decision trees and random forest classifiers, and neural network based models.

1.3 Speaker & Language Recognition

1.3.1 Problem statements

The problem statements in the area of speaker or language recognition can be posed in many ways. This is illustrated in figure 1.1.

Classification and detection are fundamental problems of speaker and language recognition that form the basis for more complex problems such as speaker and language diarization.

The classification problem assumes that a segment of speech recording belongs to exactly one among N classes under consideration, whereas the detection problems do not make any such assumptions. A test speech segment could also be a conversational audio containing multiple

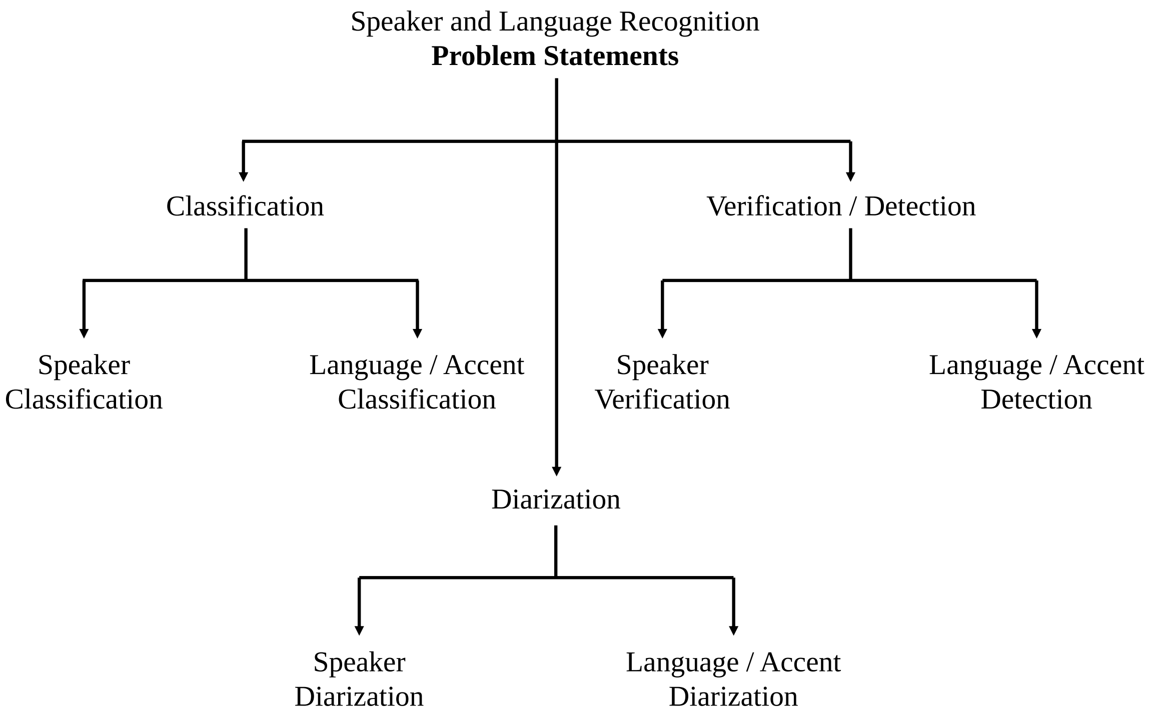


Figure 1.1: An illustration of the problem statements in speaker and language recognition.

speakers and/or multiple languages as well.

A classification system is trained using a dataset of speech recordings from the N speakers or languages of interest. A language detection system requires a set of recordings that sufficiently captures the vocabulary to model the target language, and speaker verification systems require one or more example recordings of the target speaker, called enrollment recording.

In real conversations involving people from diverse backgrounds, code mixing and code switching between multiple languages is common. In such scenarios, a more involved problem called *diarization* is of current interest. It is the task predicting the speaker or language labels as a function of time. Speaker diarization is the task of automatically determining who spoke when in a multi speaker recording, and language diarization is the task of identifying what language was spoken at what time in a multi-lingual recording.

In this thesis, we only deal with the fundamental problems of classification and detection.

1.3.2 Motivation and Applications

Algorithms for automatic speaker and language recognition have been extensively explored and studied by researchers since the 1970s. Some noteworthy publications in spoken language recognition are [5, 6], and in speaker recognition are [7, 8].

The most intuitive approach to language recognition is to use phoneme recognizers followed by language (sequence) modeling [9, 10]. This approach uses a multilingual phoneme recognizer to generate phoneme sequences which are converted to language model (n-gram) features for classification. However, the success of this approach is dependent on the performance of the phoneme decoder which is prone to noise. Moreover, the applicability of this approach is limited to distinct languages, and would perform poorly on accent recognition tasks. Most of the successful language recognition systems developed in the last two decades are similar to speaker recognition models, which are based on gross utterance level statistics computed from acoustic features. As a result, it has been a common practice to apply the algorithms and techniques from the speaker recognition literature for language recognition and vice-versa.

Speaker and language recognition have important applications in call centers, helplines, voice assistants, robotics, and also in security and defense applications. Moreover, to make speech technology like conversational AI accessible to every corner of the world, we need to explore methods to identify uncommon languages and dialects using limited training resources. It is also required to build systems that are robust to artifacts such as background noise and reverberation.

1.4 An outline of contributions

In this thesis, we present our efforts in exploring the shortcomings of existing approaches to language and speaker recognition, and give a detailed account of our proposed novel supervised approaches to develop robust language and speaker recognition systems. The contributions of this thesis can be divided in two parts, as illustrated in figure 1.2.

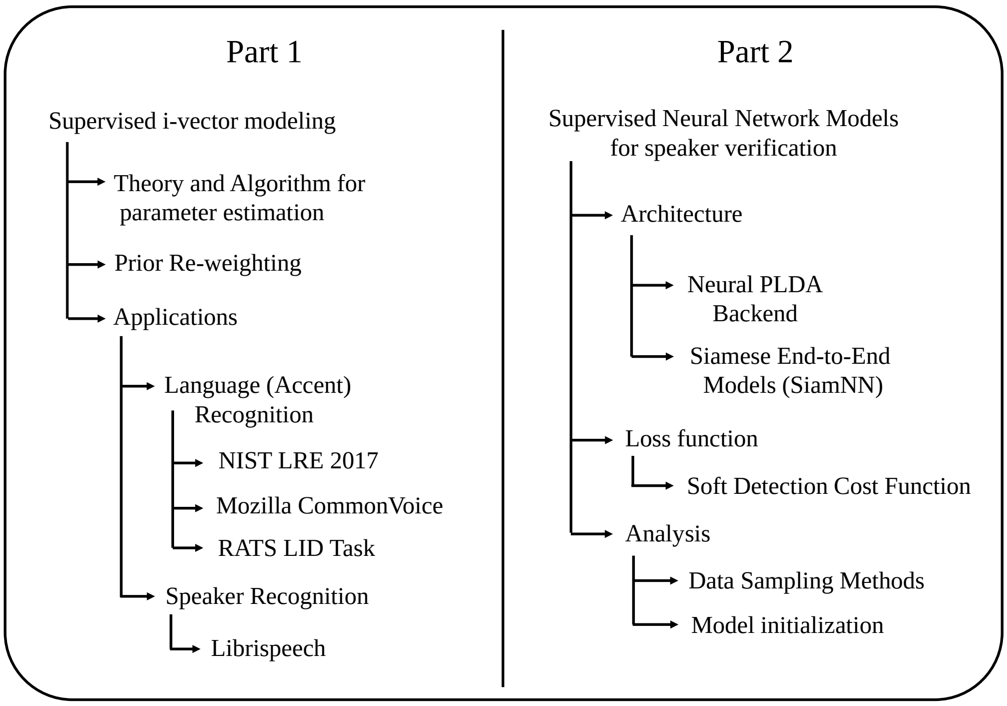

Part 1
Supervised i-vector modeling
Theory and Algorithm for parameter estimation
Prior Re-weighting
Applications
Language (Accent) Recognition
NIST LRE 2017
Mozilla CommonVoice
RATS LID Task
Speaker Recognition
Librispeech
Part 2
Supervised Neural Network Models for speaker verification
Architecture
Neural PLDA Backend
Siamese End-to-End Models (SiamNN)
Loss function
Soft Detection Cost Function
Analysis
Data Sampling Methods
Model initialization

Figure 1.2: A summary of thesis contributions.

In the first part of this doctoral research, we explore the state-of-the-art approach for language recognition in 2016, based on the i-vector approach. We identify a key limitation of this approach and propose a way to overcome the limitation by introducing the language label information into the generative model, which we refer to as the s-vector model. We rework the theory and derive the likelihood expressions for estimating the parameters of the s-vector model. We also propose a prior re-weighting factor to improve the discriminative ability of the model and study its influence. With several experiments on various datasets, we show that the s-vector approach improves significantly over the i-vector baseline for language and accent recognition tasks. The approach was also found to be effective for the speaker classification task using the librispeech dataset.

In the second part of this doctoral research, we explore the x-vector approach for speaker verification. A large neural network, designed by stacking time-delay neural network (TDNN) layers followed by statistic pooling is trained with a speaker classification objective on a large dataset with a few thousand speakers. Using this network, segment-level embeddings called x-vectors are extracted, and a back-end generative model is trained for speaker verification. We propose a discriminative back-end called the neural probabilistic linear discriminant analysis (NPLDA), an architecture inspired by the log-likelihood ratio expression of the popular PLDA backend. We also propose a loss function based on the detection cost function (DCF) which is the evaluation metric used in speaker recognition evaluations. Further, we extend the approach to train the embedding extraction and the back-end in an end-to-end manner with pairwise verification objectives. We analyse the influence of data sampling methods and model initialization for various configurations. With various experiments on challenging datasets, we show that our approach can result in highly robust speaker verification systems.

1.5 Thesis organization

In Chapter 2, we discuss the general framework for language and speaker recognition systems, the similarity between approaches used for both problems, the performance metrics used to evaluate systems, and give a detailed account of the datasets used for our experiments.

In chapter 3, we present our efforts in developing a supervised version of the i-vector model [11], termed as s-vector. With a detailed account of the assumptions and hyperparameters, we re-derive the expectation-maximization algorithm for the s-vector parameter estimation. We show that the proposed s-vectors are more discriminative than the traditional i-vectors for language and speaker recognition tasks using a variety of test datasets.

In chapter 4, we discuss our proposed neural network back-end architecture based on the probabilistic linear discriminant analysis (PLDA) for speaker verification. We define and provide an analysis of our proposed loss function used for optimizing the network with back-propagation algorithm. By discriminatively training the Neural PLDA back-end model, we show that significant performance improvements can be achieved along with robustness across several test datasets. Further, by combining the NPLDA back-end with the embedding extractor, we explore an end-to-end approach for speaker verification.

Finally, in chapter 5, we give a summary of the thesis, and identify some important research directions that can be explored in the future.

1.6 Chapter Summary

In this chapter, we have defined the problem statements of interest and given a birds-eye view of where the problem lies in the big picture of artificial intelligence, particularly dealing with speech signals. We also give a broad level overview of the signal processing and machine learning techniques used in the context of speaker and language recognition. The chapter outlined the thesis contributions and discussed the organization of the rest of the thesis.

Chapter 2

Background Study - Setting the stage

2.1 Prior work on speaker and language recognition

In this section, we give a brief overview of some noteworthy approaches to speaker and language recognition over the years.

2.1.1 Speaker Recognition

[Doddington et. al., 1985, OShaughnessy, 1986] - Speaker recognition by pattern matching [12, 13]

These review papers present various approaches to speaker recognition by a computer that were used till early 1990s. The techniques used a range of approaches like matching long-term spectral envelopes of the test utterance with the reference, dynamic time warping (DTW) aligned cepstral features for text dependent speaker recognition, and vector quantization to name a few.

[Reynolds, 1992, 1995, 2000] - Gaussian Mixture Models [14, 15]

In these works, the authors model the MFCCs of the speech utterances of each speaker by a Gaussian Mixture Model. The individual mixture components are interpreted as acoustic units like vowels, fricatives, etc. Given a test utterance, likelihoods are computed against each speaker model. For speaker classification, a maximum likelihood decision scheme was employed. For speaker verification, log-likelihood ratio scores were computed for each speaker model against the rest of the speaker models, followed by a Bayesian decision framework. Later, the approach was modified to obtain speaker specific GMMs by adapting the parameters of a universal background model. It was shown that the performance of this approach surpasses the former one with individually trained GMMs.

[Kenny et. al., 2006] - Joint factor analysis (JFA) [16, 17]

In this research, the authors propose a factor analysis approach to jointly model the speaker and session variability (a term denoting all of the phenomena which cause two recordings of a given speaker to sound different from each other). The acoustic features of an utterance are modeled using a generative model - a GMM whose means are adapted from a universal background model (UBM), modified by latent speaker and channel factors with independent Gaussian priors. This method along with probabilistic linear discriminant analysis (PLDA) [18] laid the foundation for the i-vector model [11] that was shown to be effective for both speaker and language recognition.

2.1.2 Language Recognition

[Nakagawa et. al., 1992] - Language identification by GMM and HMM [19]

In this work, the authors modeled the frame-level speech representations in four different ways. In the first approach, the speech frames were vector quantized to obtain discrete representations for each language, and decisions were made based on the vector quantization distortion from various language models. This method assumes the source to be memoryless. In the second approach, the sequences of vector-quantized representation were modeled using a hidden Markov model (HMM) to capture the sequence information. In the third approach, a Gaussian Mixture Model (GMM) was used to model the speech frames in the continuous domain, without applying vector quantization. In the last approach, a continuous ergodic HMM was used.

[Hazen, 1993, Zissman et. al., 1994, 1996] - Phoneme Recognition and Language modeling (PRLM) [20, 21, 22]

All the models explored in these research works, use one or more phoneme recognizers followed by n-gram language modeling of the target languages. There were three broad categories of PRLM systems. The first category uses a single phoneme recognizer for training n-gram language models using multiple target languages. When labelled data in multiple languages were available, the approach was modified to include multiple phoneme recognizers in parallel. The third category assumes that labelled datasets are available for the target languages, which they used to train integrated acoustic and phonotactic models.

[Torres-Carrasquilo et. al., 2002] - Shifted delta cepstral features [23]

In this work, the authors compare two GMM based language identification algorithms. The first approach utilizes GMMs trained on frame-level acoustic features which were used to compute class conditional likelihood scores. In the second approach, the GMMs were used to tokenize the frames, followed by a bank of n-gram language models. They showed that by using shifted delta cepstral (SDC) features, these models achieve comparable performance to PRLM systems. Since then, SDC features have been a popular choice for language recognition.

[Campbell et. al., 2004, 06, Van Leeuwen et. al., 2006] - Discriminative approaches to language recognition [24, 25, 26]

Following the success of discriminative models such as support vector machines (SVM) and multi-class logistic regression models for speaker recognition problems, these techniques were borrowed and applied for language recognition. Sequence kernels were initially proposed in [24] to handle sequences of SDC features. Later, SVMs were applied to the GMM-adapted supervectors. The success of these approaches formed a bridge between speaker and language recognition problems, and the trend of borrowing techniques from one to the other became a common practice.

2.2 Recent approaches to speaker and language recognition

As discussed previously in section 2.1, some of the earliest speaker recognition approaches used Gaussian Mixture Models (GMM) [14] or Joint Factor Analysis (JFA) [16] to compute the likelihoods for a sequence of short-time acoustic features of the test utterances. For Language recognition, sequence-based models such as Hidden Markov Models and n-gram models involving phoneme recognizers were used. The introduction of the total variability modeling approach [11, 27] showed that obtaining fixed-dimensional representations (embeddings) from the speech recordings as a front-end step, followed by a separate back-end model to compute the likelihoods greatly improves the performance and robustness. Moreover, in situations where computational resources are limited, such kind of a 2-stage approach greatly simplifies the implementation, as the front-end embeddings of speech segments can be extracted and stored. It also allows us to focus on one aspect of the model at a time. This two-stage pipeline of speaker and language recognition is illustrated in figure 2.1.

The front-end embedding extractor can be based on a generative or a discriminative model. These models can also be trained in supervised, unsupervised, or self-supervised frameworks. The back-end model depends on the problem statement and the evaluation criteria. For language recognition, the back-end computes log-likelihood scores for all the languages (or accents) considered. For speaker verification, the back-end model computes a log-likelihood ratio of the target and non-target hypotheses.

This section gives a brief overview of the various front-end and back-end models that have been widely used and studied in the literature.

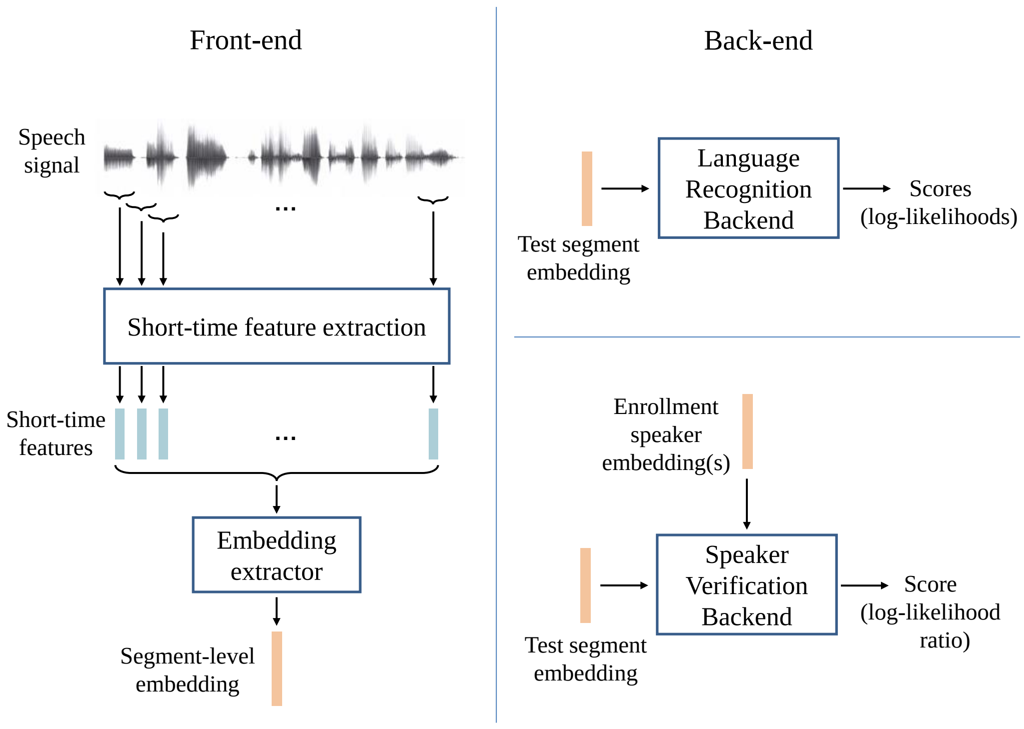


Figure 2.1: Illustration of speaker and language recognition systems

2.3 An overview of Front-end Models

2.3.1 The i-vector Model

One of the earliest approaches to front-end embedding extraction is the i-vector model [11, 27]. The short-time acoustic features of the speech utterances in a large dataset are modeled using a Gaussian Mixture Model - Universal Background Model (GMM-UBM). The GMM-UBM parameters (mean components) are adapted to fit a speech utterance. The difference between the GMM-UBM means and the adapted means is assumed to be generated in a lower dimensional space with a Gaussian prior. Estimating the latent variables in this lower dimensional space gives a fixed dimensional representation for each utterance called an i-vector. These i-vectors represent the total variability of a speach utterance from the GMM-UBM, arising due to various factors such as speaker, language, etc. The i-vectors are extracted and further processed using a back-end model trained for speaker or language recognition.

2.3.2 Neural Network Based Models

As neural networks became popular, over the last decade, researchers explored various architectures that could capture the speaker or language information of speech utterances of arbitrary duration in emeddings of fixed dimensions. In this section, we discuss a few approaches in the literature.

2.3.2.1 d-vector Models

One of the earliest neural network based models that was used for text-dependent speaker verification is the d-vector model [28]. In this approach, a feed-forward DNN is trained to classify speakers at the frame level. The average of the speaker features from the last hidden layer of the DNN is called the d-vector. The approach was later modified and adopted for text-independent speaker recognition [29] and also for language recognition [30].

2.3.2.2 X-vector Models

Snyder et. al [31, 32] proposed a neural network architecture involving a time-delay neural network (TDNN) architecture followed by a segment-level statistics temporal pooling layer where the mean and standard deviations along the time dimension are computed and stacked as a single vector. The temporal pooling layer is followed by fully connected layers and an output softmax layer to learn a classification objective. Embeddings can be chosen from any one of the segment-level hidden layers, typically called x-vectors. Various modifications to the architecture include the factorized TDNN [33], self-attention weighted statistics pooling [34], and residual networks (ResNet) [35].

2.3.2.3 Recurrent Models

For language identification, it is advantageous to capture the sequence information in the embeddings. With this motivation, there have been attempts to use recurrent architectures like Long Short Term Memory (LSTM) [36, 37], Bidirectional LSTM (Bi-LSTM) and Gated Recurrent Units (GRU) [38]. Incorporating recurrent layers into TDNN based speaker embedding extractors have been explored in [39, 40].

2.4 An overview of Back-End Models

A back-end model is a generative or discriminative model of the embeddings coming from the front-end model. It is used to compute the log-likelihoods or the log-likelihood ratio scores required to make a decision. Typically, the embeddings are processed with operations such as centering (mean subtraction), whitening (covariance normalization), within-class covariance normalization, unit length normalization, and dimensionality reduction using standard approaches such as the principle component analysis (PCA) or linear discriminant analysis (LDA). This is followed by one of the following backend models.

2.4.1 Probabilistic Linear Discriminant Analysis (PLDA)

The PLDA [18, 41] is a generative model where the embeddings are factored into class-specific and channel/session-specific components. The model lets us compute the likelihood of embeddings given the class, making it a useful tool for class inference, classification, and clustering problems.

If the j^{th} embedding of class i is denoted as $\boldsymbol{x}_{ij}$, the PLDA generative process is given by

$$\boldsymbol{x}_{ij} = \boldsymbol{\mu} + F\boldsymbol{h}_i + G\boldsymbol{w}_{ij} + \boldsymbol{\epsilon}_{ij} \tag{2.1}$$

where $\boldsymbol{\mu}$ is the mean, F and G are the class and channel subspace matrices respectively, $\boldsymbol{h}_i \sim \mathcal{N}(0, I)$ and $\boldsymbol{w}_{ij} \sim \mathcal{N}(0, I)$ are latent variables representing the class and channel respectively, and $\boldsymbol{\epsilon}_{ij} \sim \mathcal{N}(0, \Sigma)$ is a residual term. $F\boldsymbol{h}_i$ is the speaker factor that is shared by all the embeddings of speaker i, and $G\boldsymbol{w}_{ij}$ is the channel factor that captures the within speaker variability.

Given a pair of speech embeddings $\boldsymbol{x}^{(e)}$ and $\boldsymbol{x}^{(t)}$, the log-likelihood ratio (LLR) for speaker verification is expressed as follows:

$$LLR = \log \frac{\mathrm{p}(\boldsymbol{x}^{(e)}, \boldsymbol{x}^{(t)}|H_t)}{\mathrm{p}(\boldsymbol{x}^{(e)}, \boldsymbol{x}^{(t)}|H_{nt})} = \log \frac{\int \mathrm{p}(\boldsymbol{w}_e|\boldsymbol{h})\mathrm{p}(\boldsymbol{w}_t|\boldsymbol{h})\mathrm{p}(\boldsymbol{h})d\boldsymbol{h}}{\int \mathrm{p}(\boldsymbol{w}_e|\boldsymbol{h}_e)\mathrm{p}(\boldsymbol{h}_e)d\boldsymbol{h}_e \int \mathrm{p}(\boldsymbol{w}_t|\boldsymbol{h}_t)\mathrm{p}(\boldsymbol{h}_t)d\boldsymbol{h}_t} \tag{2.2}$$

Variants of the PLDA with a student's t distribution as a prior instead of the standard normal prior were also proposed in [42].

2.4.2 Gaussian Back-End Models

The Gaussian Back-end is a generative model that considers the embeddings of each class l as a Gaussian Distribution with a class-specific mean and covariance matrix: $\mathcal{N}(\boldsymbol{\mu}_l, \Sigma_l)$. For a test embedding, the log-likelihood for each class can be computed as

$$\mathcal{L}_l = \mathrm{p}(\boldsymbol{x}|l) = -\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_l)^{\intercal}\Sigma_l^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_l) - \frac{1}{2}\log|\Sigma_l| + c \tag{2.3}$$

where c is the constant factor. In some cases, certain other constraints are imposed while estimating the class-specific Gaussian parameters. For example, in [43], a shared covariance matrix $\Sigma_l = \Sigma \;\forall\; l$ (the within-class covariance matrix) is used for language recognition.

For speaker verification problems, where the log-likelihood ratio has to be computed for a pair of utterances, the enrollment and test embeddings ($\boldsymbol{x}_e$ and $\boldsymbol{x}_t$) are centered and concatenated as:

$$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}_e \\ \boldsymbol{x}_t \end{pmatrix} \tag{2.4}$$

The concatenated embeddings belonging to the target and non-target hypotheses are modeled as Gaussian distributions $\mathcal{N}(\boldsymbol{\mu}_t, \Sigma_t)$ and $\mathcal{N}(\boldsymbol{\mu}_{nt}, \Sigma_{nt})$ respectively, and the log-likelihood ratio for a test trial is computed as:

$$LLR = -(\boldsymbol{x} - \boldsymbol{\mu}_t)^{\intercal}\Sigma_t^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_t) + (\boldsymbol{x} - \boldsymbol{\mu}_{nt})^{\intercal}\Sigma_{nt}^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_{nt}) \tag{2.5}$$

2.4.3 Support Vector Machines

A support vector Machine is a discriminative two-class classifier construced from sums of kernel functions as

$$f(\boldsymbol{x}) = \sum_i a_i t_i K(\boldsymbol{x}, \boldsymbol{x}_i) + b_i \tag{2.6}$$

where $t_i \in \{-1, 1\}$ are the target values corresponding to the two classes, $\boldsymbol{x}_i$ is a subset of the training set called the support vectors that are obtained by an optimization process.

The kernel function K is constrained to have certain properties such that it can be expressed as an inner product

$$K(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{\phi}(\boldsymbol{x})^{\intercal} \boldsymbol{\phi}(\boldsymbol{y}) \tag{2.7}$$

where $\boldsymbol{\phi}$ is a non-linear mapping from the embedding space to a higher dimensional space.

For speaker verification, the concatenated vectors of the enrollment and test embeddings are used to train a pair-wise SVM [44]. For speaker or language classification problems, one-versus-rest SVMs are trained for every class, whose outputs are processed to obtain log-likelihoods.

2.4.4 Logistic Regression and Neural Network Back-ends

Logistic regression is a discriminative model that linearly transforms the embeddings followed by a logistic function to estimate the posterior probabilities of every class. Deep neural network back-ends with various architectures involving fully connected layers and choices of non-linear activations are used to apply non-linear transformations to allow the classes to be well separated. The parameters of the neural network are learnt via gradient descent of a discriminative loss function, using the backpropagation algorithm.

2.5 Language and Accent Recognition

The problem of language recognition involves the development of algorithms that automatically infer the language and aspects of language, such as the dialect and accent from a given speech recording. The dialect is a particular and unique form of the language that includes general vocabulary and grammar. In contrast, accent refers to how the people of a region or a social group pronounce the words of the language. Such pronunciation could also be influenced by the native language of the speaker, such as Tamil accented Kannada.

To set the stage for the language recognition experiments, we begin by discussing the motivation, some evaluation metrics for the language recognition problems, and popular approaches in the literature for language recognition.

2.5.1 Evaluation of Language and Accent Recognition Systems

As introduced in section 1.3.1, we can view the problem of language recognition as a classification of a detection problem. Based on the nature of the requirement, one can decide on the metrics to evaluate the systems. The most common evaluation metrics are discussed in this section.

In terms of the conditional probability density for the observed data (O) given a target language model (L_i), the log-likelihood score (l_i) is defined as:

$$l_i = \log(O|L_i) \tag{2.8}$$

For language classification, we employ either a maximum likelihood decision rule or a maximum *aposteriori* decision rule to decide a test utterance as belonging to one out of the N_L language or accent classes.

For language detection, the decision is made by applying a threshold to the log-likelihood ratios of each language. If a LID system outputs a vector of the log-likelihood scores for the N_L

languages/accents under consideration, we compute the log-likelihood *ratio* for every language L_i as

$$LLR(O, L_i) = \log \frac{P(O|L_i)}{P(O|\text{Not } L_i)} = -\log \left[\frac{1}{N_l - 1} \sum_{j \neq i} \exp(l_j - l_i) \right] \tag{2.9}$$

The evaluation metrics for language detection are defined as functions of the log-likelihood ratio scores.

2.5.1.1 Classification Accuracy

Classification accuracy is the simplest evaluation metric for any classification problems, such as speaker and language classification. It is defined as the percentage of correct predictions to the total test examples evaluated.

2.5.1.2 Average Detection Cost

The average detection cost is an evaluation metric for language detection, and is used as the primary metric for the NIST Language Recognition Evaluations [45]. It is based on the Bayes risk which is defined for a pair of target and non target languages (L_T and L_N) as:

$$C^{(L_T,L_N)}(\theta) = C_{Miss} P_{Target} P_{Miss}^{(L_T)}(\theta) + C_{FA}(1 - P_{Target}) P_{FA}^{(L_T,L_N)}(\theta) \tag{2.10}$$

where C_{Miss} and C_{FA} are the predetermined application dependent costs for missed detection and false alarm respectively, P_{Target} is the *apriori* probability of the specified target language. $P_{Miss}^{(L_T)}$ is the probability of missing detection of the target language, and $P_{FA}^{(L_T,L_N)}$ is the probability of falsely detecting an example of language L_N as the target language L_T, obtained by applying a decision threshold of θ to the log-likelihood ratios.

For better interpretability, the cost function is normalized by the best cost obtained by always rejecting the test audio. This yields:

$$C_{Norm}^{(L_T,L_N)}(\theta) = P_{Miss}^{(L_T)}(\theta) + \beta P_{FA}^{(L_T,L_N)}(\theta) \tag{2.11}$$

where β is defined as

$$\beta = \frac{C_{FA}(1 - P_{Target})}{C_{Miss}P_{Target}} \tag{2.12}$$

The average Bayes risk for a set of N_L languages or accents is defined as

$$C_{avg}(\theta) = \frac{1}{N_L}\left\{\sum_{L_T} P_{Miss}^{(L_T)}(\theta) + \frac{\beta}{N_L - 1}\sum_{L_T}\sum_{L_N} P_{FA}^{(L_T,L_N)}(\theta)\right\} \tag{2.13}$$

The theoretical optimal average detection cost is obtained by applying a threshold of log β to the log-likelihood ratio scores. We refer to this as simply C_{avg}.

2.5.1.3 Minimum detection cost

The minimum detection cost is computed by the thresholds that minimize the average detection cost function.

$$minC_{avg} = \min_{\theta_1,\ldots,\theta_{N_L}} \frac{1}{N_L}\left\{\sum_{L_T} P_{Miss}^{(L_T)}(\theta_{L_T}) + \frac{\beta}{N_L - 1}\sum_{L_T}\sum_{L_N} P_{FA}^{(L_T,L_N)}(\theta_{L_T})\right\} \tag{2.14}$$

If the system outputs well represent log-likelihoods, the $minC_{avg}$ will be very close to the actual C_{avg} computed using the threshold of log β to the LLRs.

2.5.2 Datasets

In this section, we give an account of the datasets we used for our Language Recognition experiments.

Table 2.1: LRE 2017 training set : Target languages, language clusters, total number of files per language and total duration.

Cluster	Target Languages	#files	Total Duration (hours)
Arabic	Egyptian Arabic (ara-arz)	440	190.9
	Iraqi Arabic (ara-acm)	1406	130.8
	Levantine Arabic (ara-apc)	3509	440.7
	Maghrebi Arabic (ara-ary)	919	81.8
Chinese	Mandarin (zho-cmn)	3331	379.4
	Min Nan (zho-nan)	95	13.3
English	British English (eng-gbr)	98	4.8
	General American English (eng-usg)	2448	327.7
Slavic	Polish (qsl-pol)	587	59.3
	Russian (qsl-rus)	1221	69.5
Iberian	Caribbean Spanish (spa-car)	688	166.3
	European Spanish (spa-eur)	121	24.7
	Latin American Spanish (spa-lac)	898	175.9
	Brazilian Portuguese (por-brz)	444	4.1

2.5.2.1 The NIST Language Recognition Evaluation 2017 Datasets

The training dataset consisted of 16, 205 files from 14 closely related languages and dialects grouped into five clusters. The details of the dataset mentioning the number of files and total duration for each language are given in Table 2.1. The total duration of the train files is about 2069 hours.

The development dataset has 3661 files, and the evaluation dataset has 25451 files. Both dev and eval datasets contain files of duration 3, 10 and 30 seconds from the MLS14 corpus. We trained all the systems using only the LRE 2017 train dataset (LDC2017E22), and we report the performances on the development and evaluation datasets (LDC2017E23).

2.5.2.2 The Mozilla Common Voice Dataset

The Mozilla Common Voice is a corpus of speech data read by users [46] based upon text from a number of public domain sources like blog posts, old books, movies, and other public

Table 2.2: The Mozilla CommonVoice dataset: Accents chosen, total number of files used for training, development and testing and total duration.

Accent	#files			Total Duration (mins)		
	Train	Dev	Test	Train	Dev	Test
African	926	98	196	61	7.61	14.8
Australian	4182	104	196	239	6.83	12.6
Canada	3778	104	202	229	8.02	14.5
England	15266	96	202	868	6.48	14.2
Indian	4366	103	201	259	7.32	13.8
Ireland	679	102	200	38	6.65	19.3
NewZealand	886	101	204	50	6.68	13.3
Scotland	1323	95	201	75	6.06	13.0
USA	31966	103	198	1845	6.96	13.2

speech corpora. Its primary purpose is to enable the training and testing of automatic speech recognition (ASR) systems. The dataset contains several different accents of English with a total of approximately 64k speech files (sentences) along with the accent labels (Table 2.2). The average duration per file was 3.5 seconds.

The accents with less than 1 hour were discarded, and 9 accents of English were used in training/test. For this purpose, approximately 100 files from each accent were selected randomly for the development set, and approximately 200 files from each accent were selected randomly for the evaluation set. The remaining files were used for training the accent recognition systems.

2.5.2.3 The DARPA RATS Dataset

The DARPA Robust Automatic Transcription of Speech (RATS) [47] program targets the development of speech systems operating on highly distorted speech recorded over "degraded" radio channels. The data used here consists of recordings obtained from re-transmitting a clean signal over eight different radio channel types, where each channel introduces a unique degradation mode specific to the device and modulation characteristics [47]. For the language identification (LID) task, the performance is degraded due to the short segment duration of the speech recordings in addition to the significant amount of channel noise [48]. The training data for the RATS experiments consist of 20000 recordings (about 1600 hours of audio) from five target languages (Arabic, Pashto, Dari, Farsi and Urdu) as well as from several other non-target languages. The development and the evaluation data consists of 5663 and 14757 recordings respectively, from the noisy channels.

For our language recognition experiments, we extract 3 sec, 10 sec and 30 sec chunks from the full length evaluation files to create an identical test scenario as the NIST SRE dataset.

2.6 Speaker Recognition

The key challenges in speaker verification that degrade the performance of these systems are language mismatch in the two trials that constitute the pair, shorter duration of the recordings, and noise/reverberation artifacts. In the recent past, several challenges have been organized for benchmarking the progress of speaker verification technology in these scenarios. The prominent ones include the NIST speaker recognition Evaluation (SRE) challenges [49],[50], the short duration speaker verification challenge (SDSVC) [51], the speakers in the wild (SITW) challenge [52], and the VOiCES challenge [53].

2.6.1 Evaluation of Speaker Recognition Systems

For speaker classification with N_S target speakers, we employ either a maximum likelihood decision rule or a maximum *aposteriori* decision rule to decide a test utterance as belonging to one out of the N_S speakers.

For open set speaker verification or detection, we utilize example recordings of a target speaker called the enrollment recordings to model the target speaker hypothesis H_t (a given test segment is spoken by the target speaker) and the null hypothesis H_{nt} (the test segment is not spoken by the target speaker). Given a test speech segment, speaker verification is essentially a hypothesis testing task intended to infer whether the test segment is spoken by the enrolled speaker (H_t) or a different speaker (H_{nt}). Equivalently, the target and non-target hypotheses can be stated as:

- H_t: The given two speech segments X_e and X_t are spoken by the same speaker.
- H_{nt}: The given two speech segments X_e and X_t are spoken by two different speakers.

Speaker Verification systems compute a log-likelihood ratio (score) for a *trial* (pair of speech utterances X_e, X_t), to which a threshold is applied to make a decision.

$$LLR(X_e, X_t) = \log \frac{P(X_e, X_t|H_t)}{P(X_e, X_t|H_{nt})} \tag{2.15}$$

The evaluation metrics discussed for the verification problems are defined as functions of the log likelihood ratios.

2.6.1.1 Classification Accuracy

Speaker classification accuracy is defined in a similar manner as in language classification. In this case, it is the ratio of the number of test examples for which the speaker is correctly predicted to the total number of test examples.

2.6.1.2 Equal Error Rate

Equal error rate is the location on the Detection Error Trade-off curve where the false alarm rate (P_{FA}) and missed detection rate (P_{miss}) become equal.

2.6.1.3 Detection Cost Functions

The detection cost function (DCF) is an evaluation metric for detection and verification problems, mainly used in speaker recognition challenges such as NIST SRE. It is based on the Bayes risk, which is defined as

$$C_{Det}(\theta) = C_{Miss}P_{Target}P_{Miss}(\theta) + C_{FA}(1 - P_{Target})P_{FA}(\theta) \tag{2.16}$$

where C_{Miss} and C_{FA} are the costs assigned to the missed detection and false alarm events respectively, P_{Target} is the *apriori* probability of observing a target trial, P_{Miss} and P_{FA} are the miss-rate and false alarm probabilities obtained by applying a decision threshold of θ on the log-likelihood ratio scores.

For interpretability, the cost function is normalized by the best cost obtained by always rejecting the test segments ($C_{Miss}P_{Target}$). This yields:

$$C_{Norm}(\theta) = P_{Miss}(\theta) + \beta \; P_{FA}(\theta) \tag{2.17}$$

where β is defined as

$$\beta = \frac{C_{FA}(1 - P_{Target})}{C_{Miss}P_{Target}} \tag{2.18}$$

The theoretical optimal threshold to minimize the Bayes risk is log β. This is often referred to as the actual detection cost or actDCF in speaker verification literature.

However, in practice, the true minimum of the detection cost function is not obtained at the threshold of log β. The minimum detection cost (minDCF or C_{Min}) can be computed numerically as

$$C_{Min} = \min_{\theta} P_{Miss}(\theta) + \beta \; P_{FA}(\theta) \tag{2.19}$$

2.6.1.4 Log-Likelihood Ratio Cost Function

An ideal speaker verification system that outputs true log-likelihood ratios will have an application independent quality, which means it would give optimal performance for a wide range of choices for C_{Miss}, C_{FA} and P_{Target} (applications). In practice, speaker verification systems are usually designed with a specific application in mind (Fixed values of C_{Miss}, C_{FA} and P_{Target}) to obtain better *discriminability* for that application. The detection cost functions discussed above measure the *discriminability* of the system, but not how well the output scores represent log-likelihood ratios (*calibration* of the system). Brummer et.al. [54] proposed an information theoretic evaluation metric that quantifies both the discriminability of the system as well the calibration. This is called the log-likelihood ratio cost function and is defined as:

$$L_{GBCE} = \frac{1}{2\log 2}\left\{\frac{1}{|\mathcal{T}|}\sum_{i\in\mathcal{T}}\log(1+e^{-s_i}) + \frac{1}{|\mathcal{N}|}\sum_{i\in\mathcal{N}}\log(1+e^{s_i})\right\} \tag{2.20}$$

2.6.2 Datasets

In this section, we give an account of the datasets used for speaker recognition experiments as reported in this thesis.

2.6.2.1 The NIST Speaker Recognition Evaluation Datasets (1996-2019)

The NIST Speaker Recognition Evaluation datasets consist of conversational telephone speech (CTS) data collected from a variety of handsets and mobile devices (PSTN and VoIP) and interview recordings from participants both within and outside the United States of America, for evaluating and benchmarking speaker verification systems. These datasets consist of conversations in multiple languages and also English spoken by several non-native speakers. In addition to conversational telephone speech, the development and evaluation sets from NIST SRE 2018 included audio from video (AFV) from the VAST corpus that contained speech extracted from open-source media (videos). The CTS data has a sampling rate of 8kHz and the AfV has a sampling rate of 44KHz, which is down-sampled to the desired sampling rate in our experiments depending on the model.

2.6.2.2 Comprehensive Switchboard Corpora

The Comprehensive Switchboard Corpora includes Switchboard-1, Switchboard-2 (Phases 1-3), and Switchboard Cellular datasets collected by Texas Instruments under DARPA sponsorship and the Linguistic Data Consortium (LDC). These datasets contain telephone conversations carried out in the English language by hundreds of volunteers on specific topics. The data is collected in a similar fashion as the NIST SRE datasets.

2.6.2.3 The Mixer 6 Corpus

The Mixer 6 Corpus is another dataset of telephone and interview recordings in American English collected by the LDC for the purpose of speaker recognition research.

2.6.2.4 Speakers In The Wild (SITW) Dataset

The Speakers in the Wild (SITW) speaker recognition database contains hand-annotated speech samples from open-source media for the purpose of benchmarking text-independent speaker recognition technology on single and multi-speaker audio acquired across unconstrained or ”wild” conditions. In particular, the dataset consists of recordings from 299 speakers that were extracted from various open-source media.

2.6.2.5 VoxCeleb 1 and 2 Datasets

The Visual Geometry Group (VGG) from the University of Oxford developed a large-scale open source speech database by extracting speech from YouTube videos of various celebrities across the world, covering various scenarios and a diverse set of channel conditions. This was done in two parts. VoxCeleb-1 contains 1251 speakers, and VoxCeleb-2 contains 6112 speakers. Furthermore, both these datasets are split into development and test portions for speaker verification.

2.6.2.6 VOiCES speaker recognition Dataset

The VOiCES corpus [53] was developed to promote speech and signal processing research of speech recorded by far-field microphones in noisy room conditions. This was done by creating challenging natural scenarios where background noise played in conjunction with foreground speech selected from the LibriSpeech corpus were played and recorded by distant microphones in furnished rooms. Multiple sessions were recorded in each room to accommodate for all foreground speech-background noise combinations. The audio was recorded using twelve microphones placed throughout the room, resulting in 120 hours of audio per microphone.

2.7 Chapter Summary

In this chapter, we deep dive into the problems of speaker and language recognition, draw a parallel between the various approaches used in the literature so far highlighting the similarity. We give a skeletal view of the various components of speaker and language recognition systems, and divide the implementation into two parts: front-end and back-end. We give examples of the various approaches used for embedding extraction and the back-end systems.

We then expand on how the speaker and language recognition systems are evaluated, define the various evaluation metrics, and also give an account of the datasets used in our experiments.

Chapter 3

Supervised i-vector Modelling for Language Recognition

Before the widespread adoption of deep learning approaches, a factor analysis based modeling was used to represent a variable length speech utterance in the form of a fixed dimensional vector (termed as i-vector). This approach has been prominently used for many tasks like speaker recognition, language recognition and speech recognition. The conventional i-vector approach to speaker and language recognition constitutes an unsupervised learning paradigm where a variable length speech utterance is converted into a fixed dimensional feature vector (termed as i-vector). The i-vector approach belongs to the broader family of factor analysis models where the utterance level adapted means of a Gaussian Mixture Model - Universal Background Model (GMM-UBM) are assumed to lie in a low rank subspace. The latent variables in the low rank model are assumed to have a standard Gaussian prior distribution. In the first phase of this doctoral research, we reworked the theory of i-vector modeling in a supervised framework, where the class labels (like language or accent) of the speech recordings are introduced directly into the i-vector model using a mixture Gaussian prior. In this chapter, we provide the mathematical formulation of the supervised i-vector (s-vector) model, and framework for s-vector extraction using the minimum mean squared error estimation (MMSE) approach. A detailed analysis of the s-vector model is given, and contrasted with the traditional i-vector framework. The proposed model is used for language recognition tasks using the NIST Language Recognition Evaluation (LRE) 2017 dataset as well as an accent recognition task using the Mozilla common voices dataset. In these experiments, we show that the s-vector model provides significant improvements over the conventional i-vector model (relative improvements of up to 24% for the NIST LRE task in terms of primary detection cost metric).

3.1 Introduction

In the last decade, one of the most popular approach for language (accent) and speaker recognition consisted of modeling a database of speech recordings (in the form of a sequence of short-term feature vectors) with a Gaussian Mixture Model - Universal Background Model (GMM-UBM) [14, 15]. The initial approaches for speaker recognition using log-likelihood scores were replaced with the factor analysis models [16], where the adapted Gaussian mean components (spliced as a single high dimensional vector called the supervector) are expressed as a sum of speaker and session factors. The parameters in this model were derived using a maximum likelihood (ML) framework with an iterative expectation maximization (EM) approach. The approach of joint factor analysis (JFA) was further simplified by total variability modeling (i-vector modeling), where all variabilities were captured by a single fixed dimensional latent vector [27]. With a prior of standard normal distribution (having zero mean and identity covariance), the latent variables were called i-vectors. The i-vector features, extracted using an EM framework with a maximum likelihood objective, were used for further processing. For example, the speaker verification systems use a probabilistic linear discriminant analysis (PLDA) [41] to model channel variability [55]. The language (accent) recognition systems with i-vectors used a cosine based scoring [56] or a support vector machine (SVM) model for language classification [24]. A modification of the prior density was attempted in [57], however the changes to standard normal prior did not show consistent improvements. The baseline system for the NIST LRE 2017 evaluations [58] used the i-vector features with length normalization [59] and linear discriminant analysis [56] followed by a SVM classifier.

In the last decade, there have been attempts to incorporate i-vector features for speaker adaptation [60] in speech recognition systems. The replacement of GMM-UBM with a deep neural network (DNN) acoustic model, for computing the statistics for i-vector extraction has shown improvements in speaker/language recognition tasks [61, 62]. While normalization methods like length normalization have been proposed in the post processing of the i-vectors [59], all the efforts outlined above use the unsupervised ML framework for the training the i-vector models.

In this chapter, we give a detailed account of our fully supervised version of the i-vector model that was published in [63, 64], where each label class is associated with a Gaussian prior with a class-specific mean parameter. The joint prior (marginalized over the sample space of classes) on the latent variable is a GMM. In this chapter, we give a detailed account of the EM algorithm for this choice of prior. Specifically, we show that the GMM forms a conjugate prior for this framework (given the statistics, the posterior distribution of the latent vectors is also a GMM). This choice of prior is motivated by the use of a Gaussian back-end approach [43], where the conventional i-vectors for each language are modeled with a single Gaussian distribution. In the proposed model, the posterior distribution of the i-vectors is a GMM. Thus, the maximum aposteriori (MAP) estimates are not very useful for the multi-modal GMM posterior distribution. We resort to the minimum mean square error (MMSE) estimate of the latent variables, which we refer as the supervised i-vector (s-vector) for a given test recording. Furthermore, the use of class dependent prior also allows us to weigh the importance of the prior with a factor (the belief on the prior can be varied based on the duration of the recording).

With detailed data analysis and visualization, we show that the s-vector features yield representations that succinctly capture the language (accent) label information. We also show that the conventional i-vectors are a special case of the more generic s-vectors proposed in this work, where all the label information is assumed to belong to one class.

The proposed s-vectors are used for a language recognition task in the LRE 2017 dataset. We use the same setup as in the baseline i-vector system [58], except for the replacement of the unsupervised i-vectors with the s-vectors. The subsequent steps, including the SVM training for language classification, are also performed in a similar fashion for all the systems. In our experiments comparing the supervised and unsupervised i-vector features, the proposed approach provides significant improvements in LRE task (relative improvements of up to 24% in terms of the primary cost metric). We also show that the proposed approach yields consistent gains for RATS language recognition experiments [47] on short-duration conditions, and the Mozilla CommonVoices dataset [46] involving short-duration utterances of accented English speech. We also experiment with the s-vector approach for speaker classification, which typically contains a larger number of classes than in language recognition. Using one-hundred speakers from the Librispeech dataset, we demonstrate that the s-vector approach improves the performance of speaker classification over the i-vector baseline.

The rest of the chapter is organized as follows. In Sec. 3.3, we provide the mathematical derivation of the proposed supervised EM framework for s-vector model parameter estimation. We emphasize primarily the difference in the formulation compared to the traditional i-vectors. The minimum divergence re-estimation step is also discussed here. In Sec. 3.3.2, we derive the expressions for s-vector extraction for a given test utterance. The language recognition experiments are reported in Sec. 3.4. A detailed discussion of the results with confusion matrices and data visualization is provided in Sec. 3.5, along with a discussion of the application to speaker recognition in Sec. 3.5.7. This is followed by a brief summary in Sec. 3.6.

3.2 The mathematical framework of the i-vector model

We provide the mathematical derivation of the conventional i-vector modeling [16]. This is required as the s-vector modeling is built on this model. The notations used here follow those from the work of Kenny and Dehak et. al. [16, 27].

Given a dataset of S recordings, let $X(s) = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{H(s)}]$ denote the sequence of F dimensional front-end feature vectors, where $H(s)$ is the number of frames in recording s. Let $\Lambda = \{\pi_c, \boldsymbol{\mu}_c, \Sigma_c\}_{c=1}^{C}$ denote the parameters of a C component Gaussian Mixture Model - Universal Background Model (GMM-UBM). The UBM mean supervector is denoted by $\boldsymbol{M}_0 = [{\boldsymbol{\mu}_1}^{\intercal}, \ldots, {\boldsymbol{\mu}_c}^{\intercal}]^{\intercal}$. It is assumed that for each recording s, an adapted mean supervector $\boldsymbol{M}(s)$ was used to generate $X(s) = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{H(s)}]$. The i-vector model (also known as the total variability model) is a generative model for $\boldsymbol{M}(s)$ and is given by

$$\boldsymbol{M}(s) = \boldsymbol{M}_0 + T\boldsymbol{y}(s) \tag{3.1}$$

where T is a matrix of dimension $CF \times R$ called the total variability matrix, and $\boldsymbol{y}(s)$ is a latent vector of dimension $R \times 1$. A standard normal distribution is used as the prior density for $\boldsymbol{y}(s)$. The i-vector of recording s is defined as the MAP estimate of $\boldsymbol{y}(s)$ given $X(s)$. The Baum-Welch statistics of recording s for mixture c are given by

$$N_c(s) = \sum_{i=1}^{H(s)} \mathrm{p}_\Lambda(c \mid \boldsymbol{x}_i) \tag{3.2}$$

$$\boldsymbol{F}_{X,c}(s) = \sum_{i=1}^{H(s)} \mathrm{p}_\Lambda(c \mid \boldsymbol{x}_i)(\boldsymbol{x}_i - \boldsymbol{\mu}_c) \tag{3.3}$$

$$S_{XX,c}(s) = \sum_{i=1}^{H(s)} \mathrm{p}_\Lambda(c \mid \boldsymbol{x}_i)(\boldsymbol{x}_i - \boldsymbol{\mu}_c)(\boldsymbol{x}_i - \boldsymbol{\mu}_c)^{\intercal} \tag{3.4}$$

The BW statistics are sufficient statictics for estimation of the model parameters. In matrix form, the zeroth and first order statistics are written as:

$$N(s) = \begin{pmatrix} N_1(s)I & & 0 \\ & \ddots & \\ 0 & & N_C(s)I \end{pmatrix}, \qquad \boldsymbol{F}_X(s) = \begin{pmatrix} \boldsymbol{F}_{X,1}(s) \\ \vdots \\ \boldsymbol{F}_{X,C}(s) \end{pmatrix}$$

where I is an identity matrix of size $F \times F$. In [65], it was shown that the log-likelihood function of the frame-level features of recording s is

$$\log \mathrm{p}_T(X(s) \mid \boldsymbol{y}(s)) = G(s) + H_T(s, \boldsymbol{y}(s)) \tag{3.5}$$

where $G(s) = \sum_{c=1}^{C} N_c(s) \log \left((2\pi)^{-\frac{F}{2}} |\Sigma_c|^{-\frac{1}{2}} \right) - \frac{1}{2} \operatorname{tr} \left(\Sigma^{-1} S_{XX}(s) \right)$ is a term independent of T and $\boldsymbol{y}(s)$. Hence it will not play a role in estimating T and $\boldsymbol{y}(s)$. The second term is

$$H_T(s, \boldsymbol{y}(s)) \ = \ \boldsymbol{y}(s)^{\intercal} T^{\intercal} \Sigma^{-1} \boldsymbol{F}_X(s) - \frac{1}{2} \boldsymbol{y}(s)^{\intercal} T^{\intercal} \Sigma^{-1} N(s) T \boldsymbol{y}(s) \tag{3.6}$$

Using the Bayes' formula, it can be shown that the *aposteriori* density function of $\boldsymbol{y}(s)$ given $X(s)$ is Gaussian with covariance $\mathcal{L}(s)^{-1}$ and mean $\mathcal{L}(s)^{-1} T^{\intercal} \Sigma^{-1} \boldsymbol{F}_X(s)$ [27] where

$$\mathcal{L}(s) = I + T^{\intercal} \Sigma^{-1} N(s) T \tag{3.7}$$

The proofs for the likelihood function and the posterior can be found in [65]. For easy reference, we also present the proofs in Appendix 5.3.4. Estimating T by maximum likelihood is done using the Expectation-Maximization (EM) algorithm [66].

3.2.1 Expectation (E) step:

The Q function is given by:

$$\begin{aligned} Q\left(T \mid T^{(t)}\right) &= \sum_{s=1}^{S} \mathbb{E}_{\boldsymbol{y}(s)\mid X(s),T^{(t)}} \log \mathrm{p}_T(X(s), \boldsymbol{y}(s)) \\ &= \sum_{s=1}^{S} \mathbb{E}_{\boldsymbol{y}(s)\mid X(s),T^{(t)}} \log \mathrm{p}_T(X(s) \mid \boldsymbol{y}(s)) \\ &\qquad + \sum_{s=1}^{S} \mathbb{E}_{\boldsymbol{y}(s)\mid X(s),T^{(t)}} \log \mathrm{p}(\boldsymbol{y}(s)) \end{aligned} \tag{3.8}$$

The second term is independent of T and doesn't play a role in the M-Step, and hence it can be ignored. Substituting for the likelihood term from Eq (3.5), ignoring the irrelevant terms and simplifying yields:

$$\begin{aligned} Q\left(T \mid T^{(t)}\right) = \sum_{s=1}^{S} \Bigg[&\operatorname{tr}\left\{ T^{\intercal} \Sigma^{-1} \boldsymbol{F}_X(s) \hat{\boldsymbol{y}}^{(t)}(s)^{\intercal} \right\} \\ &- \frac{1}{2} \operatorname{tr}\left\{ \Sigma^{-1} N(s) T E_{yy}^{(t)}(s) T^{\intercal} \right\} \Bigg] \end{aligned} \tag{3.9}$$

where

$$\mathcal{L}^{(t)}(s) = I + T^{(t)\intercal} \Sigma^{-1} N(s) T^{(t)} \tag{3.10}$$

$$\hat{\boldsymbol{y}}^{(t)}(s) = \mathcal{L}^{(t)}(s)^{-1} T^{(t)\intercal} \Sigma^{-1} \boldsymbol{F}_X(s) \tag{3.11}$$

$$E_{yy}^{(t)}(s) = \mathcal{L}^{(t)}(s)^{-1} + \hat{\boldsymbol{y}}^{(t)}(s) \hat{\boldsymbol{y}}^{(t)}(s)^{\intercal} \tag{3.12}$$

The E-step can be summarized as computing the quantities in Eq (3.10 - 3.12). The vector $\hat{\boldsymbol{y}}^{(t)}(s)$ is the MAP estimate of the latent variable computed using the total variability matrix $T^{(t)}$, which is usually called as the i-vector.

3.2.2 Maximization (M) step:

The update equation for the matrix T is obtained by maximizing the Q function w.r.t T.

$$T^{(t+1)} = \underset{T}{\operatorname{argmax}} \;\; Q\left(T \mid T^{(t)}\right) \tag{3.13}$$

Differentiating Eq (3.9), equating it to zero and simplifying yields the system of linear equations in every row of $T^{(t+1)}$:

$$\sum_{s=1}^{S} N(s) T^{(t+1)} \left(\mathcal{L}^{(t)}(s)^{-1} + \hat{\boldsymbol{y}}^{(t)}(s) \hat{\boldsymbol{y}}^{(t)}(s)^{\intercal}\right) = \sum_{s=1}^{S} \boldsymbol{F}_X(s) \hat{\boldsymbol{y}}^{(t)}(s)^{\intercal} \tag{3.14}$$

$T^{(t+1)}$ is obtained by solving the above systems of linear equations.

3.2.3 Minimum Divergence Re-estimation

For i-vector modeling, another update step called the minimum divergence re-estimation of the T matrix is done after every M-step to ensure that the model is consistent with the prior on the latent variable $\boldsymbol{y}$. Assume that the current estimate T results in i-vector estimates with covariance $K_{\boldsymbol{yy}}$ instead of identity covariance, the TVM equation can be written as

$$\begin{aligned} \boldsymbol{M}(s) &= \boldsymbol{M}_0 + T\boldsymbol{y}(s) && (3.15) \\ &= \boldsymbol{M}_0 + \underbrace{T K_{\boldsymbol{yy}}^{\frac{1}{2}}}_{T'} \underbrace{\left(K_{\boldsymbol{yy}}^{-\frac{1}{2}} \boldsymbol{y}(s)\right)}_{\boldsymbol{y}'(s)} && (3.16) \end{aligned}$$

Hence, by making the modifications $T' \leftarrow T K_{\boldsymbol{yy}}^{\frac{1}{2}}$ and $\boldsymbol{y}'(s) \leftarrow K_{\boldsymbol{yy}}^{-\frac{1}{2}} \boldsymbol{y}(s)$, and re-estimating $\{\boldsymbol{m}_l\}_{l=1}^{L}$ using $\boldsymbol{y}'(s)$, the KL divergence of the distribution of i-vectors using updated model from the prior on $\boldsymbol{y}(s)$ will be minimized. The quantity $K_{\boldsymbol{yy}}$ is computed as

$$K_{\boldsymbol{yy}} = \frac{1}{S}\sum_{s=1}^{S}\mathbb{E}_{\boldsymbol{y}(s)|X(s),T}\left[\boldsymbol{y}(s)\boldsymbol{y}(s)^{\intercal}\right] \tag{3.17}$$

It can be shown that by making these updates, the total data log likelihood also improves [16].

$$\sum_{s=1}^{S}\log \mathrm{p}_{T'}(X(s)) \geq \sum_{s=1}^{S}\log \mathrm{p}_{T}(X(s)) \tag{3.18}$$

3.3 The s-vector model

The i-vector model that is popularly used in speaker and language recognition is outlined in 3.2. We follow notations similar to the conventional i-vectors to derive the s-vector features. In the traditional i-vector approach, the Total Variability Model (TVM) (Eq. (3.1)) together with the Gaussian Mixture Model - Universal Background Model (GMM-UBM) constitute a latent variable based generative model for the short-time sequence of features. In the s-vector model, we incorporate the class label information in this generative modeling framework. We denote the feature sequence of recording s by

$$X(s) = \{\boldsymbol{x}_1(s), \ldots \boldsymbol{x}_{H(s)}(s)\} \tag{3.19}$$

where $H(s)$ denotes the length of recording s. The corresponding label is denoted as $l(s)$. Note that the values of $l(s)$ are discrete ($l(s) = 1, ..., L$), where L denotes the total number of classes. The adapted means of the GMM-UBM are modeled as:

$$\boldsymbol{M}(s) = \boldsymbol{M}_0 + T\boldsymbol{y}(s) \tag{3.20}$$

where $\boldsymbol{M}(s)$, $\boldsymbol{M}_0$ and T are similar to the definitions used in the conventional i-vector model (Eq. 3.1). Though this equation is identical to the conventional i-vector model, the key difference is that the latent vector $\boldsymbol{y}(s)$ depends on the class label $l(s)$. The prior on $\boldsymbol{y}(s)$ conditioned on the class l is modeled as a Gaussian with mean $\boldsymbol{m}_l$, and a shared, identity covariance matrix for all classes $l \in \{1, \ldots, L\}$, i.e.,

$$p(\boldsymbol{y}(s)|l(s) = l) \sim \mathcal{N}(\boldsymbol{y}(s); \boldsymbol{m}_l, I) \tag{3.21}$$

For recordings without a known label, the prior distribution is then a Gaussian mixture model,

$$\mathrm{p}(\boldsymbol{y}(s)) = \sum_{l=1}^{L} \mathrm{p}(l)\mathcal{N}(\boldsymbol{y} \mid \boldsymbol{m}_l, I) \tag{3.22}$$

By parameterizing the class conditioned means $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_L$ along with the other model parameters, we make use of the labels of the train recordings to estimate the model parameters, thereby introducing supervision. If $\boldsymbol{m}_l = \boldsymbol{0}$ is set for all $l \in \{1, \ldots, L\}$, the proposed model reverts back to the standard i-vector model. In the s-vector model, we make the following statistical assumptions:

1. The *apriori* probability of observing label l is uniform, i.e., $\mathrm{p}(l) = \frac{1}{L} \quad \forall l \in \{1, \ldots, L\}$

2. Given the latent variable $\boldsymbol{y}(s)$, $X(s)$ is conditionally independent on the class label $l(s)$

$$\mathrm{p}_T(X(s) \mid \boldsymbol{y}(s), l(s)) = \mathrm{p}_T(X(s) \mid \boldsymbol{y}(s)) \tag{3.23}$$

While the full covariance model for each label class is feasible in the proposed framework, the shared covariance model allows for simplicity in model estimation (and greatly reduces the memory requirements). As some of the classes have a very small number of recordings, the class conditioned covariance matrices may not be well estimated (for example, the language classes like British-English in the NIST LRE 2017 dataset have a very small number of recordings). The popularly used Gaussian backend model [43] for log likelihood computation also doesn't model the covariance of each class separately. The reason for using an Identity covariance matrix is because a model with a shared covariance matrix W can be converted to an equivalent model with identity covariance matrix by the following transformation:

$$\begin{aligned} \boldsymbol{M}(s) &= \boldsymbol{M}_0 + T\boldsymbol{y}(s) \\ &= \boldsymbol{M}_0 + TW^{\frac{1}{2}}W^{-\frac{1}{2}}\boldsymbol{y}(s) \\ &= \boldsymbol{M}_0 + T'\boldsymbol{y}'(s) \end{aligned}$$

where $\boldsymbol{y}'$ follows a GMM distribution with Identity mixture covariances. This is achieved through the minimum divergence re-estimation procedure in each iteration (3.2.3). A similar approach has been used in the conventional i-vector framework where a standard Gaussian distribution with identity covariance is used instead of a model with full covariance.

We derive the steps involved in the estimation of s-vector model parameters in the following subsection.

3.3.1 EM Algorithm for Parameter Estimation

The class conditioned means $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_L$ can be parameterized along with the matrix T. The set of parameters of the s-vector model to be estimated are denoted as,

$$\Theta = \{T, \boldsymbol{m}_1, \ldots, \boldsymbol{m}_L\}. \tag{3.24}$$

The likelihood function of Θ in terms of $X(s), l(s)$ and $\boldsymbol{y}(s)$ is

$$\begin{aligned} \mathrm{p}_\Theta(X(s), \boldsymbol{y}(s), l(s)) &= \mathrm{p}_\Theta(X(s), \boldsymbol{y}(s) \mid l(s))\mathrm{p}(l(s)) \\ &= \frac{1}{L}\mathrm{p}_\Theta(X(s) \mid \boldsymbol{y}(s))\mathrm{p}_\Theta(\boldsymbol{y}(s) \mid l(s)) \end{aligned} \tag{3.25}$$

The complete data log-likelihood is

$$\sum_{s=1}^{S} \mathrm{logp}_{\Theta}(X(s), \boldsymbol{y}(s), l(s))$$
$$= S \log \frac{1}{L} + \sum_{s=1}^{S} \log \mathrm{p}_{\Theta}(X(s) \mid \boldsymbol{y}(s)) + \sum_{s=1}^{S} \log \mathrm{p}_{\Theta}(\boldsymbol{y}(s) \mid l(s)) \tag{3.26}$$

Here, the middle term $\mathrm{p}_{\Theta}(X(s) \mid \boldsymbol{y}(s))$ is identical to that of Eq. (3.5). The *aposteriori* density function of $\boldsymbol{y}(s)$ given $X(s)$ and $l(s)$ is Gaussian with covariance $\mathcal{L}^{-1}(s)$ and mean given by $\mathcal{L}^{-1}(s)\big\{\boldsymbol{m}_{l(s)} + T^{\intercal}\Sigma^{-1}\boldsymbol{F}_X(s)\big\}$, where $\mathcal{L}(s)$ is same as in Eq. (3.7). This result is obtained by applying the Bayes' formula as shown in Appendix 5.3.4.

The EM algorithm (supervised EM) is used to solve for the parameters $\Theta = \{T, \boldsymbol{m}_1, \ldots, \boldsymbol{m}_L\}$ which maximize the joint likelihood function $\sum\limits_{s=1}^{S} \log \mathrm{p}_{\Theta}(X(s), l(s))$.

3.3.1.1 Expectation (E) step

In the expectation step, we obtain an expression for the Q function, defined in terms of $X(s)$ and $l(s)$ as

$$\begin{aligned} Q\left(\Theta \mid \Theta^{(t)}\right) &= \sum_{s=1}^{S} \mathbb{E}_{\boldsymbol{y}(s)|X(s),l(s),\Theta^{(t)}} \log \mathrm{p}_{\Theta}(X(s), l(s), \boldsymbol{y}(s)) \\ &= \sum_{s=1}^{S} \mathbb{E}_{\boldsymbol{y}(s)|X(s),\Theta^{(t)}} \log \mathrm{p}_{\Theta}(X(s) \mid \boldsymbol{y}(s)) \\ &\qquad + \sum_{s=1}^{S} \mathbb{E}_{\boldsymbol{y}(s)|X(s),\Theta^{(t)}} \log \mathrm{p}(\boldsymbol{y}(s) \mid l(s)) \end{aligned} \tag{3.27}$$

Here, we have used the independence assumption (Eq 3.23). The term $S \log \frac{1}{L}$ has been ignored here, as it is independent of the model parameters and does not play a role in the EM algorithm. Substituting the required terms from Eq (3.5, 3.22 & 3.26) and simplifying by ignoring the constant terms gives

$$Q\left(\Theta|\Theta^{(t)}\right) = \sum_{s=1}^{S}\left[\operatorname{tr}\left\{T^{\intercal}\Sigma^{-1}\boldsymbol{F}_X(s)\hat{\boldsymbol{y}}_{l(s)}^{(t)}(s)^{\intercal}\right\} - \frac{1}{2}\operatorname{tr}\left\{\Sigma^{-1}N(s)TE_{yy,l(s)}^{(t)}(s)T^{\intercal}\right\}\right] + \sum_{l=1}^{L}\sum_{\substack{s=1\\ l(s)=l}}^{S}\left(\hat{\boldsymbol{y}}_{l(s)}^{(t)}(s)^{\intercal}\boldsymbol{m}_l - \frac{1}{2}\boldsymbol{m}_l{}^{\intercal}\boldsymbol{m}_l\right) \tag{3.28}$$

For each recording s, using the parameter estimates $\Theta^{(t)}$ at iteration t, we compute the following quantities

$$\mathbb{E}_{\Theta^{(t)}}\left[\boldsymbol{y}(s) \mid X(s), l(s)\right] = \hat{\boldsymbol{y}}_{l(s)}^{(t)}(s) = \mathcal{L}^{(t)}(s)^{-1}\left\{\boldsymbol{m}_{l(s)} + T^{(t)\intercal}\Sigma^{-1}\boldsymbol{F}_X(s)\right\} \tag{3.29}$$

$$\mathbb{E}_{\Theta^{(t)}}\left[\boldsymbol{y}(s)\boldsymbol{y}(s)^{\intercal} \mid X(s), l(s)\right] = E_{yy,l(s)}^{(t)}(s) = \mathcal{L}^{(t)}(s)^{-1} + \hat{\boldsymbol{y}}_{l(s)}^{(t)}(s)\hat{\boldsymbol{y}}_{l(s)}^{(t)}(s)^{\intercal} \tag{3.30}$$

where $\boldsymbol{F}_X(s)$ and $N(s)$ are the Baum-Welch statistics in matrix form (defined in 3.2) and $\mathcal{L}^{(t)}(s) = I + T^{(t)\intercal}\Sigma^{-1}N(s)T^{(t)}$

3.3.1.2 Maximization Step

The update equation for the matrix T is obtained by maximizing the Q function w.r.t T.

$$T^{(t+1)} = \underset{T}{\operatorname{argmax}}\ Q\left(\Theta \mid \Theta^{(t)}\right) \tag{3.31}$$

As the Q function is convex in T, partially differentiating Eq (3.9) w.r.t T, equating it to zero and simplifying yields the system of linear equations in every row of $T^{(t+1)}$:

$$\sum_{s=1}^{S} N(s)T^{(t+1)}E_{yy,l(s)}^{(t)}(s) = \sum_{s=1}^{S}\boldsymbol{F}_X(s)\hat{\boldsymbol{y}}^{(t)}(s)^{\intercal} \tag{3.32}$$

$T^{(t+1)}$ is obtained by solving the above systems of linear equations. The update equation for the T matrix is

$$T_c^{(t+1)} = \left(\sum_{s=1}^{S} N_c(s) E_{yy,l(s)}^{(t)}(s)\right)^{-1} \left(\sum_{s=1}^{S} \boldsymbol{F}_{X,c}(s) \hat{\boldsymbol{y}}_{l(s)}^{(t)}(s)^{\intercal}\right) \tag{3.33}$$

where T_c is the sub-matrix of T corresponding to c^{th} mixture of GMM-UBM.

Similarly, the update equations for the class conditioned means are obtained by partially differentiating Eq (3.9) w.r.t $\boldsymbol{m}_l$, equating it to zero and solving for $\boldsymbol{m}_l$. This yields the update equations:

$$\boldsymbol{m}_l^{(t+1)} = \frac{1}{S_l} \sum_{\substack{s=1 \\ l(s)=l}}^{S} \hat{\boldsymbol{y}}_{l(s)}^{(t)}(s) \tag{3.34}$$

where S_l is the number of training recordings with class label l.

The update equation for the T matrix is numerically identical to the update equations of the conventional i-vector model. However, the difference is in how the quantities $\hat{\boldsymbol{y}}_{l(s)}^{(t)}(s)$ and $E_{yy,l(s)}^{(t)}(s)$ are computed in the E-step (equations 3.29 and 3.30). The term $\boldsymbol{m}_{l(s)}^{(t)}$ is introduced in the s-vector model, which is different for each class l and updated in every EM iteration, allowing extra degrees of freedom for the model such that the latent vectors for each class are confined to a Gaussian distribution around the class center. Setting $\boldsymbol{m}_{l(s)}^{(t)}$ to zero for all classes and not having it gives us the conventional i-vector model. Here the latent vectors are forced to confine to a standard normal distribution irrespective of the class. Hence, we rely on the inherent differences in the acoustic feature frames between each language to be encoded in the latent i-vector space, rather than explicitly forcing it.

3.3.1.3 Minimum divergence re-estimation

The idea of minimum divergence estimation [16] is to model the T matrix in such a way as to force the empirical distribution to conform to the GMM prior as assumed. Specifically, it is required that the class conditioned covariances are shared among all classes and equal to the identity matrix. We require the within class covariance matrix to be identity. The minimum divergence update equation is given by $T^{(t)} \longleftarrow T^{(t)}L$, where $LL^{\intercal}$ is the Cholesky decomposition of the within-class covariance matrix K_{yy}, given below as,

$$K_{\boldsymbol{yy}} = \frac{1}{S}\sum_{s=1}^{S} \mathbb{E}_{\boldsymbol{y}(s)|X(s),l(s),T}\left[(\boldsymbol{y}(s) - \boldsymbol{m}_{l(s)})(\boldsymbol{y}(s) - \boldsymbol{m}_{l(s)})^{\intercal}\right] \tag{3.35}$$

$$= \frac{1}{S}\sum_{s=1}^{S} \mathbb{E}_{\boldsymbol{y}(s)|X(s),l(s),T}\left[\boldsymbol{y}(s)\boldsymbol{y}(s)^{\intercal} - \boldsymbol{y}(s)\boldsymbol{m}_{l(s)}{}^{\intercal} - \boldsymbol{m}_{l(s)}\boldsymbol{y}(s)^{\intercal} + \boldsymbol{m}_{l(s)}\boldsymbol{m}_{l(s)}{}^{\intercal}\right] \tag{3.36}$$

Expanding and simplifying using the parameters at time step t,

$$K_{yy}^{(t)} = \frac{1}{S}\sum_{s=1}^{S}\Big(\mathcal{L}^{(t)}(s)^{-1} + \hat{\boldsymbol{y}}_{l(s)}^{(t)}(s)\hat{\boldsymbol{y}}_{l(s)}^{(t)}(s)^{\intercal} - \hat{\boldsymbol{y}}_{l(s)}^{(t)}(s)\boldsymbol{m}_{l(s)}^{(t)}{}^{\intercal} - \boldsymbol{m}_{l(s)}^{(t)}\hat{\boldsymbol{y}}_{l(s)}^{(t)}(s)^{\intercal} + \boldsymbol{m}_{l(s)}^{(t)}\boldsymbol{m}_{l(s)}^{(t)}{}^{\intercal}\Big) \tag{3.37}$$

3.3.2 S-vector extraction

The conventional i-vectors are simply the maximum *aposteriori* (MAP) estimates of $\boldsymbol{y}(s)$ given the Baum-Welch statistics $N(s)$ and $\boldsymbol{F}_X(s)$. In the proposed model, for an unlabeled test recording, the posterior distribution of $\boldsymbol{y}(s)$ turns out to be a GMM (similar to the prior). This model belongs to the broad class of Bayesian models with conjugate prior (The conventional i-vector model is also a conjugate prior Bayesian model, where the prior and posterior are Gaussian distributed). The posterior distribution of $\boldsymbol{y}(s)$ is given by

$$\mathrm{p}(\boldsymbol{y}(s) \mid X(s)) = \sum_{l=1}^{L} \mathrm{p}(l \mid X(s))\,\mathrm{p}(\boldsymbol{y}(s) \mid X(s), l) \tag{3.38}$$

For the i-vector model, the posterior distribution of $\boldsymbol{y}(s)$ is Gaussian, and the i-vector is defined as the MAP estimate (mode of the posterior distribution), which is also equal to its mean (MMSE estimate). Unlike the i-vector case, the posterior distribution of $y(s)$ in the s-vector model is multimodal in nature, and hence, there is no one obvious way to use define the s-vector. We explore multiple methods for deriving s-vector representations from this model as follows.

3.3.2.1 aMAP s-vectors

Due to the fact that exact mode of a GMM does not have a closed-form expression, and must be obtained by iterative numerical methods, finding the MAP estimate is numerically expensive. The label conditioned $\boldsymbol{y}_l$ vector corresponding to the class with the largest language posterior may be a close approximation of the MAP estimate, which we define as approximate MAP (aMAP) s-vectors.

3.3.2.2 PCA s-vectors

The first approach we tried is to splice all the label-conditioned MAP estimates to form the recording level representation that can be further dimensionality reduced using principal component analysis (PCA) [67].

3.3.2.3 Averaged label-conditioned s-vectors

The second approach is to compute a raw average of the individual label-conditioned MAP estimates.

3.3.2.4 Minimum mean-squared (MMSE) s-vectors

In the third approach, we use the minimum mean-square-error (MMSE) estimate of $\boldsymbol{y}(s)$. The MMSE estimate (s-vector) is defined by the posterior mean of $\boldsymbol{y}(s)$, given the input features $X(s)$. It is computed using the following expression:

$$\begin{aligned}\hat{\boldsymbol{y}}_{MMSE}(s) &= \mathbb{E}(\boldsymbol{y}(s) \mid X(s)) \\ &= \sum_{l=1}^{L} \mathrm{p}(l \mid X(s))\hat{\boldsymbol{y}}_l(s) \qquad (3.39)\end{aligned}$$

We refer to $\hat{\boldsymbol{y}}_l(s)$ as a class conditioned s-vector (conditioned on class l) and it is given by

$$\hat{\boldsymbol{y}}_l(s) = \left(I + T^{\intercal}\Sigma^{-1}N(s)T\right)^{-1}\left(\boldsymbol{m}_l + T^{\intercal}\Sigma^{-1}\boldsymbol{F}_X(s)\right) \tag{3.40}$$

and the posteriors $\mathrm{p}(l \mid X(s))$ are calculated from the class conditioned likelihoods $\mathrm{p}_{\Theta}(X(s) \mid l)$. The expressions for language posteriors $\mathrm{p}(l|X(s)$ are given below.

The log likelihood of $X(s)$ for class l can be by marginalizing over the distribution of $\boldsymbol{y}(s)$ as follows:

$$\begin{aligned}
\log \mathrm{p}(X(s) \mid l) &= \log \int\limits_{<\boldsymbol{y}(s)>} \mathrm{p}(X(s), \boldsymbol{y}(s) \mid l(s))\ \mathrm{d}\boldsymbol{y}(s) && (3.41)\\
&= \log \int\limits_{<\boldsymbol{y}(s)>} \mathrm{p}(X(s) \mid \boldsymbol{y}(s), l(s))\ \mathrm{p}(\boldsymbol{y}(s) \mid l(s))\ \mathrm{d}\boldsymbol{y}(s) && (3.42)\\
&= \log \int\limits_{<\boldsymbol{y}(s)>} \mathrm{p}(X(s) \mid \boldsymbol{y}(s))\ \mathrm{p}(\boldsymbol{y}(s) \mid l(s))\ \mathrm{d}\boldsymbol{y}(s) && (3.43)
\end{aligned}$$

The $l(s)$ term vanishes due to the independence assumption (Eq 3.23). Substituting for the log-likelihood from Eq. (3.5) and the expression for the class conditioned prior of $\boldsymbol{y}(s)$, we get

$$\log \mathrm{p}(X(s) \mid l) = \log \int\limits_{<\boldsymbol{y}(s)>} e^{(G(s)+H_T(s,\boldsymbol{y}(s))}\frac{1}{(2\pi)^{\frac{R}{2}}}e^{-\frac{1}{2}(\boldsymbol{y}(s)-\boldsymbol{m}_l)^{\intercal}(\boldsymbol{y}(s)-\boldsymbol{m}_l)}\mathrm{d}\boldsymbol{y}(s) \tag{3.44}$$

The integral term can be simplified by separating out the terms independent of $\boldsymbol{y}(s)$. As $H_T(s, \boldsymbol{y}(s))$ is a quadratic function in $\boldsymbol{y}(s)$, the argument of the integral has a Gaussian form in $\boldsymbol{y}(s)$. Solving for the integral and simplifying, we get

$$\begin{aligned}
\log \mathrm{p}(X(s) \mid l) = G(s) &- \frac{1}{2}\boldsymbol{m}_l^{\intercal}\boldsymbol{m}_l - \frac{1}{2}\log|\mathcal{L}(s)|\\
&+ \frac{1}{2}(\boldsymbol{m}_l + T^{\intercal}\Sigma^{-1}\boldsymbol{F}_X(s))^{\intercal}\mathcal{L}(s)^{-1}(\boldsymbol{m}_l + T^{\intercal}\Sigma^{-1}\boldsymbol{F}_X(s)) && (3.45)
\end{aligned}$$

The intermediate steps involved in solving the integral term in Eq. 3.44 to obtain Eq. 3.45 is given in Appendix 5.3.4. The class posteriors $\mathrm{p}(l|X(s))$ can be calculated using the Bayes' formula, and terms $G(s)$ and $-\frac{1}{2}\log|\mathcal{L}(s)|$ are independent of l and can be ignored while calculating the language posteriors.

As shown in the experiments, the MMSE-based s-vectors perform the best when the posteriors, $\mathrm{p}(l \mid X(s))$, are well estimated. Setting $\mathrm{p}(l \mid X(s)) = 1$ for the true class l and 0 for the other classes (true label based one-hot encoding) gives us the *oracle* s-vectors (cheat experiment). Using the oracle s-vectors, we are able to show the performance upper-bounds of the proposed s-vector model. On the other hand, the worst case situation would be when the class posterior distribution is assumed to be uniform ($\mathrm{p}(l|X(s)) = \frac{1}{L}$), we obtain the average s-vectors mentioned in Sec. 3.3.2.3.

3.3.3 Re-weighting the priors

Although the proposed model incorporates label information, the algorithm only tries to maximize the joint likelihood. This does not ensure that the model is discriminative. In an attempt to make the proposed model more discriminative, the class conditioned prior covariance (Eq (3.21)) can be scaled by a factor $\frac{1}{\lambda}$, where $\lambda \geq 1$. The covariance matrix is a measure of the spread of a vector distribution around its mean. Scaling down the covariance matrix reduces this spread by forcing the vectors to be closer to the mean. When the class means are fixed, but the shared(identity) covariance matrix is scaled down, the separation between the classes is forced to increase.

Using the modified prior distribution, it can be shown that the E-step will be modified as follows:

$$\mathcal{L}^{(t)}(s) = \lambda I + T^{(t)\intercal}\Sigma^{-1}N(s)T^{(t)} \tag{3.46}$$

$$\hat{\boldsymbol{y}}_{l(s)}^{(t)}(s) = \mathcal{L}^{(t)}(s)^{-1}\left(\lambda \boldsymbol{m}_{l(s)}^{(t)} + T^{(t)\intercal}\Sigma^{-1}\boldsymbol{F}_X(s)\right) \tag{3.47}$$

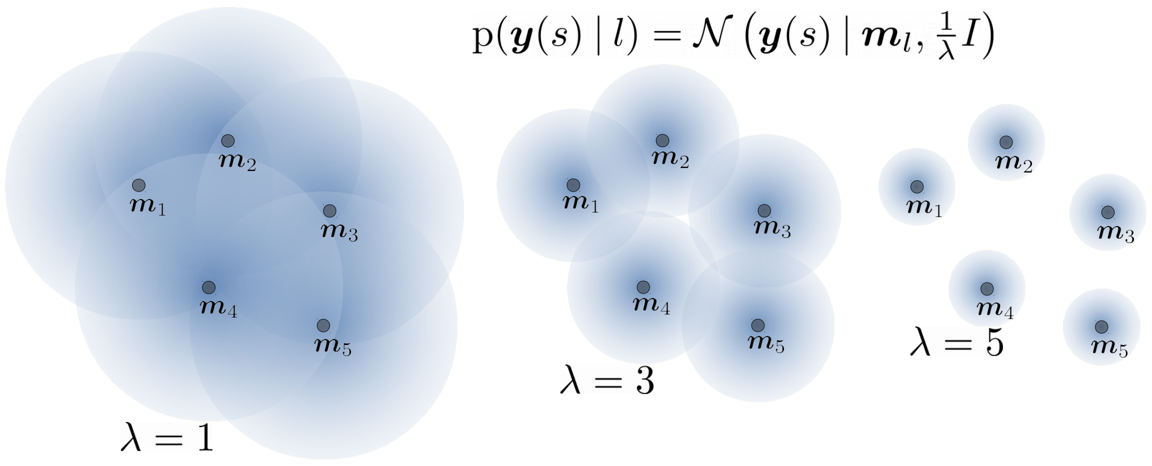


Figure 3.1: Illustration of the variation of prior distribution with varying λ. In this example, the means of Gaussian mixtures $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_5$ represent five language classes.

With a prior re-weighting factor of λ, the class conditioned s-vector of Eq.(3.40) will be modified as

$$\hat{\boldsymbol{y}}_l(s) = \left(\lambda I + T^{\intercal}\Sigma^{-1}N(s)T\right)^{-1}\left(\lambda \boldsymbol{m}_l + T^{\intercal}\Sigma^{-1}\boldsymbol{F}_X(s)\right) \tag{3.48}$$

Figure 1 highlights the effect of λ on the prior density. When the value of λ is increased, the GMM means are unchanged, and the mixture component covariance around the means is reduced by a factor of $\frac{1}{\lambda}$. This forces the latent variables to be more concise around the language means, thereby enhancing the model's discriminative ability. However, when a high value of λ is chosen (and in noisy conditions), the embedding could be more concise around the wrong language class mean. Thus, we find that for shorter durations (and in noisy conditions), a lower value of λ is preferred, while for longer durations in cleaner conditions (like in NIST LRE 2017), a higher value of λ improves the LID performance. By using the hyper-parameter λ, we can control the degree of confidence on the prior distribution.

3.4 Experiments and Results

We demonstrate the advantages of the s-vector model by applying it to language and accent recognition problems using the NIST LRE 2017 dataset [58] and the Mozilla CommonVoice dataset [46] respectively.

3.4.1 Performance Metrics

We use three different metrics for evaluating the language and accent recognition tasks. The first two metrics, namely the NIST LRE 2017 primary detection cost ($C_{primary}$) and the equal error rate (EER) report the performance on a language detection setting. For this purpose, likelihood ratios are computed for each language versus the rest, and a threshold of β is applied to obtain the false alarm (P_{FA}) and miss (P_{Miss}) probabilities. The NIST LRE 2017 primary cost metric ($C_{Primary}$) is defined as:

$$C_{Primary} = \frac{C_{avg}(\beta_1 = 1) + C_{avg}(\beta_2 = 9)}{2} \tag{3.49}$$

where C_{avg} is the average detection cost over all languages defined in Eq. 2.13.

The equal error rate (EER) is the P_{FA} (or P_{Miss}) computed at the threshold where P_{FA} and P_{Miss} become equal. The classification accuracy is used to report the performance in a language identification setting (closed set language classification).

3.4.2 Experiments conducted

We performed three sets of language recognition experiments, based on three different datasets: The NIST LRE 2017 dataset, RATS language identification dataset and the Mozilla Common-Voice dataset. Each of these datasets have a train, validation and test partition. The validation and test partitions are further divided into 3, 10 and 30 seconds groups based on their duration. Figure 3.2 depicts the pipeline of the baseline i-vector language/accent recognition system that

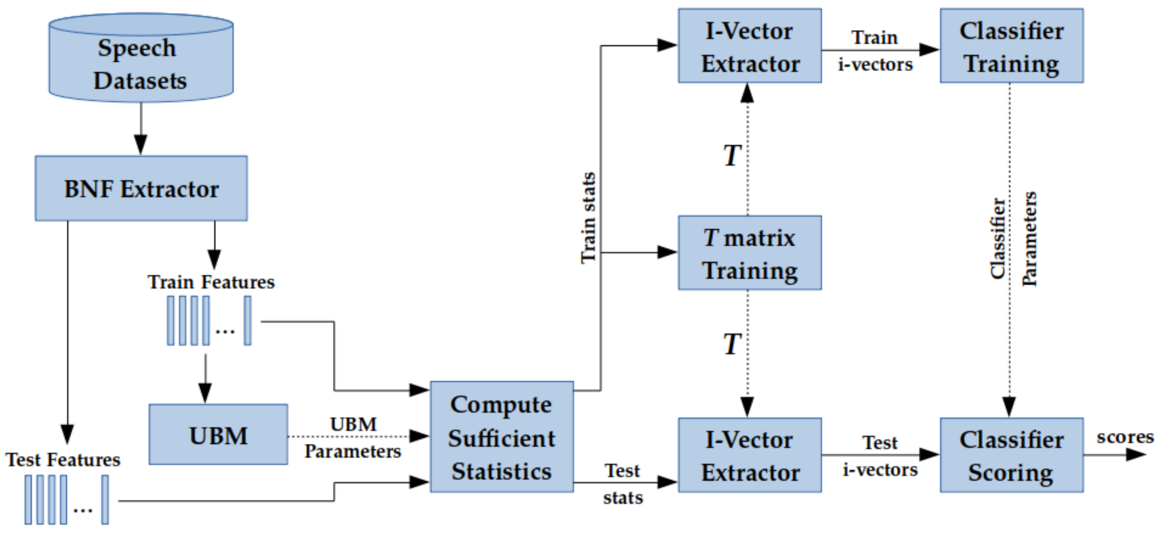


Figure 3.2: I-vector based language/accent recognition pipeline

uses bottleneck features from a DNN acoustic model. In our experiments involving s-vectors, all the components in this pipeline remain the same as the baseline, except for the T matrix training and s-vector extraction steps (described in Sec. 3.3.2).

A deep neural network (DNN) based acoustic model trained on switchboard and Fisher corpora for an automatic speech recognition (ASR) task was used to extract frame level bottleneck features of 80 dimensions as a front end. Speech activity detection (SAD) is applied to retain only the voiced frames. We use the implementation of Sohn's statistical model based VAD from the Voicebox toolkit [68]. A GMM-UBM of 2048 mixtures is trained using the bottleneck features of the training set. For the total variability training with random initialization, the T matrix is estimated using the EM algorithm for 6 iterations. We set the i-vector/s-vector dimension to 500.

Following the i-vector/s-vector extraction, the representations are centered, within class covariance normalized [56], and dimensionality reduced with LDA to $L-1$ dimensions ($L = 14$ for the NIST LRE 2017 and 9 for the Mozilla accent recognition task). We then use two different back-end models for obtaining the language log-likelihood scores, namely the Gaussian back-end (GB) [43] and Support Vector Machines (SVM) [24] with radial basis function (RBF) kernel.

3.4.3 Results on NIST LRE experiments

In Tables 3.1 and 3.2, we report the $C_{primary}$, EER, and classification accuracies for the baseline systems, namely the unsupervised i-vectors systems, and our various s-vector systems. As noted previously, the oracle s-vectors (cheat) and the average s-vectors represent performance bounds. For the MMSE s-vectors, we experimented with various value of λ, to see how $C_{primary}$ varies with λ for the development dataset, for each duration (3 sec, 10 sec and 30 sec) separately. The value of λ that gave the best performance on the development set was then used to compare our s-vector systems with the baseline on the evaluation set.

On the NIST LRE evaluation dataset, the proposed MMSE s-vector improves over the baseline relatively by [19%, 20%, 8%] in terms of $C_{primary}$, [16%, 24%, 4%] in terms of EER, and [16%, 18%, 9%] in terms of accuracy for [3 sec, 10 sec, 30 sec] durations.

The aMAP s-vectors (defined in Sec. 3.3.2.1) perform poorly when compared to the i-vector baseline approach, particularly for the 3 sec condition. For longer durations, the difference in performance is minimal. In terms of overall performance, the aMAP s-vectors perform marginally worse than the baseline i-vector approach. This may be because the errors in the estimated posteriors introduce noise in the s-vectors used for back-end training, as well as during inference.

The PCA s-vectors (defined in Sec. 3.3.2.2) also performs worse on the 3 sec condition than the i-vector baseline. However, for the 10 sec and 30 sec condition, it is at least as good or better than the i-vector baseline. However, the PCA s-vectors perform better than the aMAP s-vectors on average.

The MMSE approach utilizes a full Bayesian estimation of the embedding vector. It is more elegant mathematically while it also simplifies the computation over the PCA method. As seen in Table 3.1 and Table 3.2, the MMSE s-vector approach performs significantly better than both aMAP and PCA s-vectors on both the development and evaluation datasets. The average s-vectors and oracle s-vectors represent two extreme cases of the weighted average of

Table 3.1: Results on the NIST LRE 2017 development dataset for the i-vector and the various s-vector approaches, using SVM back-end for scoring.

	Dev Performances : $100C_{primary}$ [EER (%)] {Accuracy(%)}		
Model config.	3 sec	10 sec	30 sec
Unsupervised i-vector [58]	52.7 [16.6] {51.8}	27.1 [7.5] {74.0}	13.1 [3.6] {87.8}
aMAP s-vector	57.1 [17.9] {50.4}	27.6 [7.7] {74.0}	13.2 [3.7] {87.7}
Average s-vector	47.8 [14.3] {56.3}	22.4 [6.2] {78.3}	12.2 [**3.3**] {88.0}
PCA s-vector [63]	57.6 [18.2] {50.4}	27.0 [7.4] {74.6}	12.1 [3.4] {88.2}
MMSE s-vector	**44.4** [**14.7**] {**63.5**}	**19.5** [**5.9**] {**83.7**}	**11.7** [3.4] {**89.4**}
Oracle s-vector (cheat)	13.0 [3.9] {88.3}	5.6 [1.7] {94.4}	5.0 [1.1] {94.9}

Table 3.2: Results on the NIST LRE 2017 evaluation dataset for the i-vector and the various s-vector approaches, using SVM back-end for scoring.

	Eval Performances : $100C_{primary}$ [EER (%)] {Accuracy(%)}		
Model config.	3 sec	10 sec	30 sec
Unsupervised i-vector [58]	53.6 [16.1] {53.8}	29.9 [8.6] {72.4}	16.7 [3.9] {83.0}
aMAP s-vector	58.4 [18.0] {51.5}	30.1 [8.5] {72.7}	16.2 [3.9] {83.7}
Average s-vector	49.7 [13.9] {58.5}	27.0 [7.0] {75.3}	15.7 [**3.7**] {84.0}
PCA s-vector [63]	58.2 [17.7] {54.1}	30.5 [8.2] {73.7}	15.8 [3.8] {83.9}
MMSE s-vector	**43.7** [**13.5**] {**61.2**}	**23.7** [**6.5**] {**77.4**}	**15.4** [3.8] {**84.5**}
Oracle s-vector (cheat)	12.6 [3.6] {86.9}	6.5 [1.6] {92.9}	5.7 [1.4] {93.6}

label-conditioned s-vectors. It is interesting to note that although the raw average s-vector approach utilizes uniform weighting, it still outperforms the baseline i-vector model, and also the aMap and PCA s-vectors.

3.4.4 Results on the Mozilla CommonVoice Accent Recognition Task

The following figure shows the variation of $C_{primary}$ with λ for MMSE s-vectors on the development dataset using GB classifiers. In this case, the MMSE s-vector ($\lambda = 7$) gave the best performance on the development data. This configuration is used on the evaluation set and the results are reported in Table 3.3. On the evaluation dataset, the proposed s-vectors provide only moderate improvements (over the baseline), with average relative improvements of 2% in

Table 3.3: Results on accent recognition experiments with Mozilla CommonVoice Datasets using Gaussian backend for scoring

Model config.	Eval Performances: $100C_{primary}$ [EER (%)] {Accuracy (%)}
Unsupervised i-vector [58]	77.2 [21.2] {52.5}
Average s-vector	**75.9** [20.7] {**53.3**}
MMSE s-vector	76.0 [**20.7**] {53.2}
Oracle s-vector (cheat)	61.5 [15.3] {62.2}

Table 3.4: Results of the RATS Language identification task

	Performances : $100C_{primary}$ [EER (%)] {Accuracy(%)}		
Model config.	3 sec	10 sec	30 sec
	Dev.		
Unsup. i-vector [58]	94.0 [26.3] {66.1}	63.8 [17.6] {77.1}	**40.5** [10.9] {86.9}
MMSE s-vector	**86.7** [25.2] {70.2}	**59.5** [16.6] {80.2}	41.3 [11.3] {87.5}
	Eval		
Unsup. i-vector [58]	89.0 [25.1] {64.8}	63.0 [17.9] {76.2}	40.3 [11.6] {85.7}
MMSE s-vector	**79.5** [22.5] {65.8}	**56.3** [16.3] {77.5}	**39.5** [11.4] {85.4}

terms of $C_{primary}$, 2.4% in terms of EER, and 4% in terms of accuracy. As the recordings are very short in duration and the accent classes are highly overlapping, the posterior distribution is not well estimated. Hence, the average s-vector also performs as well as the MMSE s-vector for this task.

3.4.5 Results on RATS Language Recognition Task

The RATS LID results are reported in Table 3.4. The RATS dataset involves language recognition on 5 target languages along with several other imposter classes. The proposed s-vector approaches show consistent improvements over the baseline system on short duration conditions (3 sec and 10 sec) over the baseline i-vector approach. For example, on the RATS evaluation set for the 3 sec and 10 sec condition, the proposed s-vector approach improves the baseline system by about 11 % relative in terms of $C_{primary}$ metric. These results are also consistent with the NIST LRE 2017 results.

3.4.6 Computational Complexity

With R being the dimension of the i-vectors/s-vectors and if L labels are included in the model, the complexity of both i-vector and s-vector extraction is of order $O(R^3)$. The s-vector extraction involves an additional step of computing the posterior probability $\mathrm{p}(l \mid X)$ and the language specific posterior mean for each langauge l. Both of these steps are of the order $O(RL)$. In order to analyze the computation time, 50 files of 30 sec duration from the NIST LRE task were selected and their i-vectors and s-vectors were extracted sequentially in a single threaded mode on an Intel CPU with 256 GB of RAM. The feature extraction and zeroth/first order statistic computation was performed as a pre-processing step before the i-vector/s-vector estimation. The total computation time for 50 recordings was about 24 sec and 28 sec respectively for the i-vector and s-vector estimation procedure. Thus, the s-vector estimation involves 15 % more computation time than the i-vector estimation. However, we find that the effect of this increased computation impacts the overall processing pipeline involving voice activity detection, feature extraction, embedding estimation and SVM scoring by only about 2 % relative.

3.5 Discussion

3.5.1 Influence of the prior re-weighting factor λ

Figure 3.3 shows the variation of $C_{primary}$ with λ for MMSE s-vector systems on the NIST LRE 2017 development dataset. As seen in the figure, the optimal choice of λ was 3, 5 and 7 for 3 sec, 10 sec, and 30 sec conditions respectively. These configurations are used on the evaluation dataset.

For the NIST LRE task, the s-vectors show significant improvement over the i-vector baseline on the development data for most choices of λ (Figure 3.3). As seen in Figure 3.1, increasing

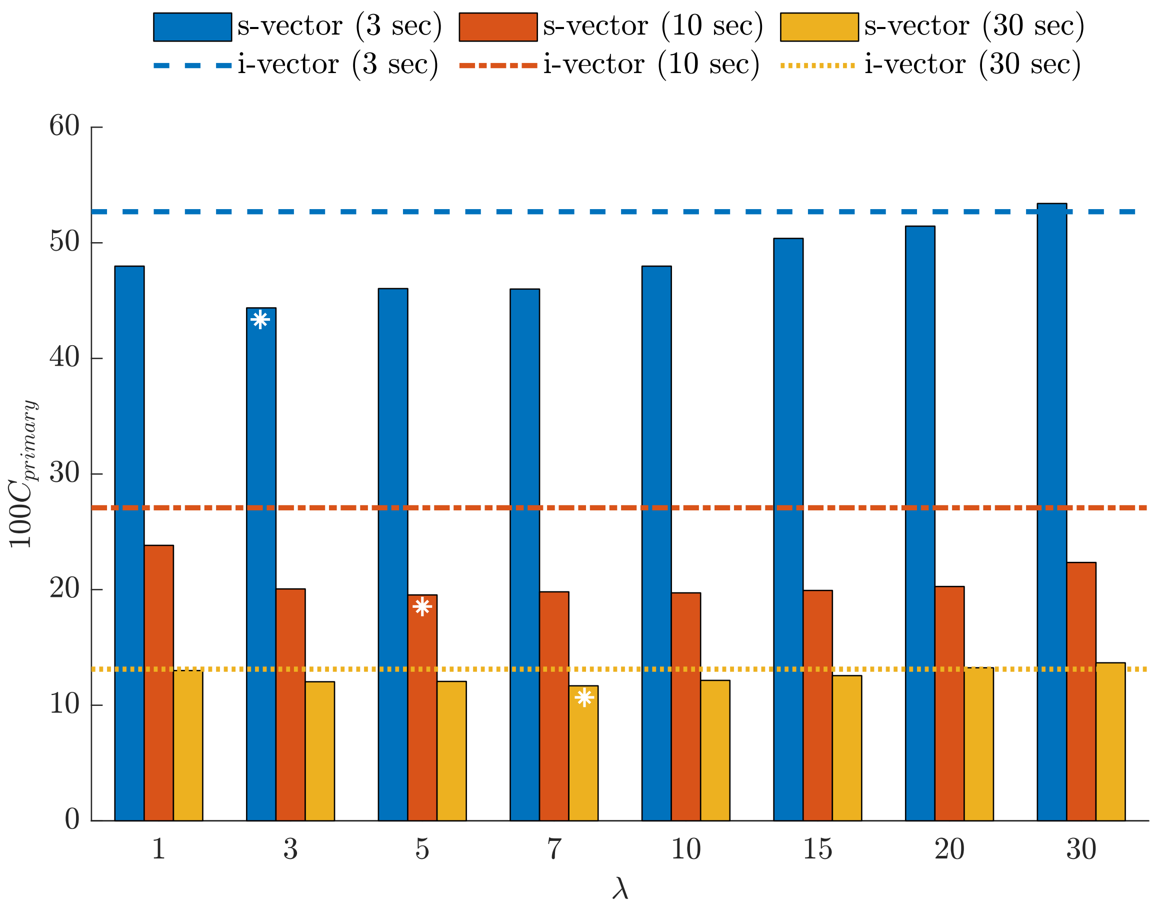


Figure 3.3: Variation of $C_{primary}$ with λ for MMSE s-vectors on the NIST LRE development dataset for each of the durations [3 sec, 10 sec, and 30 sec]. The dotted line denotes the unsupervised i-vector baseline.

the value of λ improves the belief on the class distribution chosen by the posterior ($\mathrm{p}(l|X(s))$) (the Gaussian clusters are more concise around the mean in Figure 3.1). For short-duration conditions (3 sec), the posterior distribution $\mathrm{p}(l|X(s))$ is not well estimated and has errors. Hence, increasing the value of λ makes the distribution concise on poorly estimated posteriors which degrades the performance. On the other hand, for the longer duration condition of 30 sec, as the posterior $\mathrm{p}(l|X(s))$ is well estimated, having a concise distribution (by increasing the value of λ) improves the performance. Thus, there is a trade-off in the choice of λ that can provide the optimal performance based on the duration of the utterance. For shorter duration utterances like 3 sec, the best performance is achieved at $\lambda = 3$, for moderately long duration

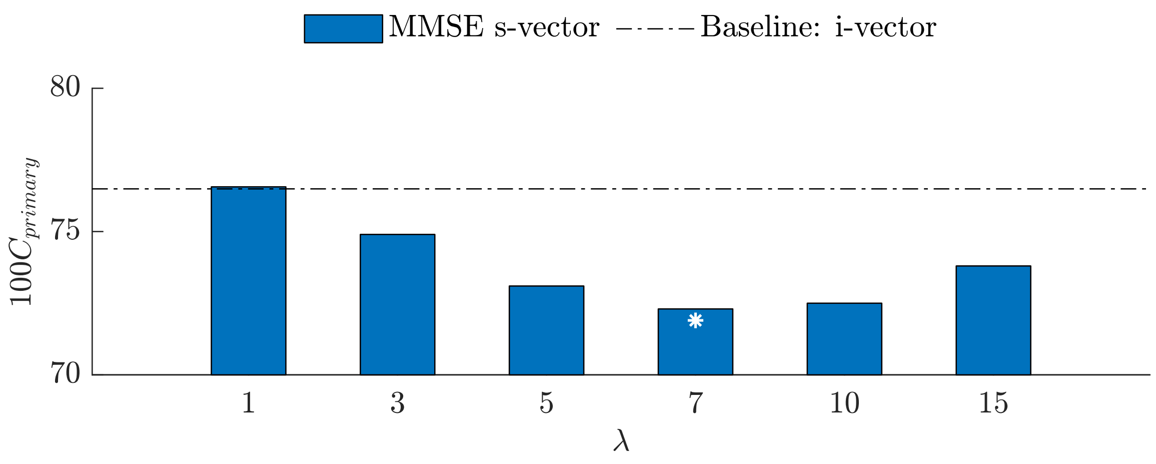


Figure 3.4: Variation of $C_{primary}$ with λ for the accent recognition task on the Mozilla CommonVoice Development Dataset.

utterances (10 sec), the best performance is achieved by using $\lambda = 5$, and for long recordings of 30 sec, the best choice is $\lambda = 7$. Also, the relative improvements for the proposed s-vector approach over the conventional i-vector system are more significant on the 3 sec and 10 sec conditions, which are more challenging.

The variation of $C_{primary}$ with λ for the Mozilla CommonVoice accent recognition task, using the development dataset is shown in Fig. 3.4. As the average duration of files in this dataset is 3.5 seconds, we do not categorize this analysis into multiple duration bins like in the case of NIST LRE 2017. For this dataset, $\lambda = 7$ was found to be optimal.

3.5.2 Comparison with other approaches

In the initial phase of this doctoral research [63], we had proposed to use an external model such as a neural network trained with categorical cross-entropy objective, in order to extract the language posteriors $\mathrm{p}(l \mid X(s))$. We trained a fully connected feed-forward neural network with two hidden layers of 512 dimensions and ReLU non-linearities and a 14-dimensional softmax output layer (same specifications as "System B" in [69]). We utilized the outputs from this model as approximations to the posteriors $\mathrm{p}(l \mid X(s))$. The embeddings extracted using these posteriors are referred to as "Approx. MMSE s-vectors" in Tables 3.5 and 3.6. While this

Table 3.5: Comparison of results of the MMSE s-vector using SVM back-end for scoring with other approaches on the NIST LRE 2017 development dataset

	Dev Performances : $100C_{primary}$ [EER (%)] {Accuracy(%)}		
Model config.	3 sec	10 sec	30 sec
Unsupervised i-vector [58]	52.7 [16.6] {51.8}	27.1 [7.5] {74.0}	13.1 [3.6] {87.8}
Sup. i-vector [70]	47.2 [15.1] {61.1}	20.2 [5.7] {80.9}	14.8 [4.1] {85.1}
Simplified Sup. i-vector [70]	58.2 [19.6] {47.8}	27.3 [7.8] {73.6}	13.4 [3.7] {87.7}
LSTM [71]	53.7 [15.39] {52.7}	33.8 [9.7] {69.2}	32.0 [8.81] {70.9}
HGRU [72]	53.1 [15.09] {57.5}	27.6 [6.6] {76.9}	25.5 [6.1] {78.6}
Approx. MMSE s-vector [63]	51.1 [15.3] {54.1}	23.9 [6.1] {77.4}	12.1 [3.3] {88.3}
MMSE s-vector	**44.4** [**14.7**] {**63.5**}	**19.5** [**5.9**] {**83.7**}	**11.7** [3.4] {**89.4**}

Table 3.6: Comparison of results of the MMSE s-vector using SVM back-end for scoring with other approaches on the NIST LRE 2017 evaluation dataset

	Eval Performances : $100C_{primary}$ [EER (%)] {Accuracy(%)}		
Model config.	3 sec	10 sec	30 sec
Unsupervised i-vector [58]	53.6 [16.1] {53.8}	29.9 [8.6] {72.4}	16.7 [3.9] {83.0}
Sup. i-vector [70]	46.1 [14.7] {59.6}	25.6 [7.3] {76.4}	19.9 [5.1] {80.9}
Simplified Sup. i-vector [70]	57.2 [19.1] {49.8}	29.8 [8.4] {71.4}	16.7 [4.1] {82.8}
LSTM [71]	55.2 [15.4] {54.7}	35.4 [8.7] {72.1}	28.1 [7.3] {76.1}
HGRU [72]	55.4 [15.3] {55.1}	32.3 [7.5] {74.1}	23.3 [4.9] {83.0}
Approx. MMSE s-vector [63]	52.2 [14.6] {56.8}	27.8 [7.1] {74.7}	15.8 [3.6] {84.0}
MMSE s-vector	**43.7** [**13.5**] {**61.2**}	**23.7** [**6.5**] {**77.4**}	**15.4** [3.8] {**84.5**}

approach provides better performance than the PCA s-vectors, the exact expressions of label posteriors for MMSE s-vectors perform significantly better over all other configurations and across all durations.

We also compare the performance of the proposed s-vectors with supervised i-vectors and simplified supervised i-vectors introduced in [70] (Table 3.5, 3.6). This previous approach of using language labels fails to statistically model the label distribution as the label information (in the form of one-hot encoded vectors) is appended to the adapted means of the GMM (without any change in the i-vector modeling framework). In the proposed work, the labels are handled as discrete symbols, and the label information impacts the choice of the prior distribution in the EM framework. The results indicate that the proposed approach of using labels is superior to the previous work [70] for all durations.

The results for neural network approaches for language recognition in NIST LRE 2017 development and evaluation data [71, 73, 72] are also reported in Table 3.2). The long short term memory (LSTM) recurrent neural network (RNN) based LID system [71, 73] uses an end-to-end LSTM model for language recognition. An end-to-end hierarchical model for language recognition [72] using gated recurrent units (GRU) improved the LSTM-based model for longer duration speech recordings. Both these models, use the same training and test data compared to the proposed s-vector model. As seen in Table 3.5 and Table 3.6, the s-vector model improves over the baseline neural network models in all test duration conditions.

3.5.3 Data Visualization

In order to analyze the improvements obtained using the proposed approach, we use a data visualization approach using the t-distributed stochastic neighborhood embedding (tSNE) [74]. The tSNE is an unsupervised dimensionality reduction method that preserves the local neighborhood of the data space in the lower dimensional subspace. We perform tSNE dimensionality reduction to two dimensions on the unsupervised i-vectors and the proposed s-vectors (on the NIST LRE 2017 development set for 3 sec recordings). The two-dimensional scatter plot is separately shown for each of the five language clusters (Arabic, Chinese, English, Slavic, and Iberian). This is because most of the confusion in language classification happens within the broad language cluster. The tSNE plots are shown in Figure 3.5. As seen here, for most of the language clusters, the s-vectors have a reduced within-class variance in the cluster distribution. For English and Chinese languages, the between-class separability is also improved for the proposed s-vectors. The tSNE plots illustrate that the s-vectors provide representations that are better suited for language recognition compared to the unsupervised i-vectors.

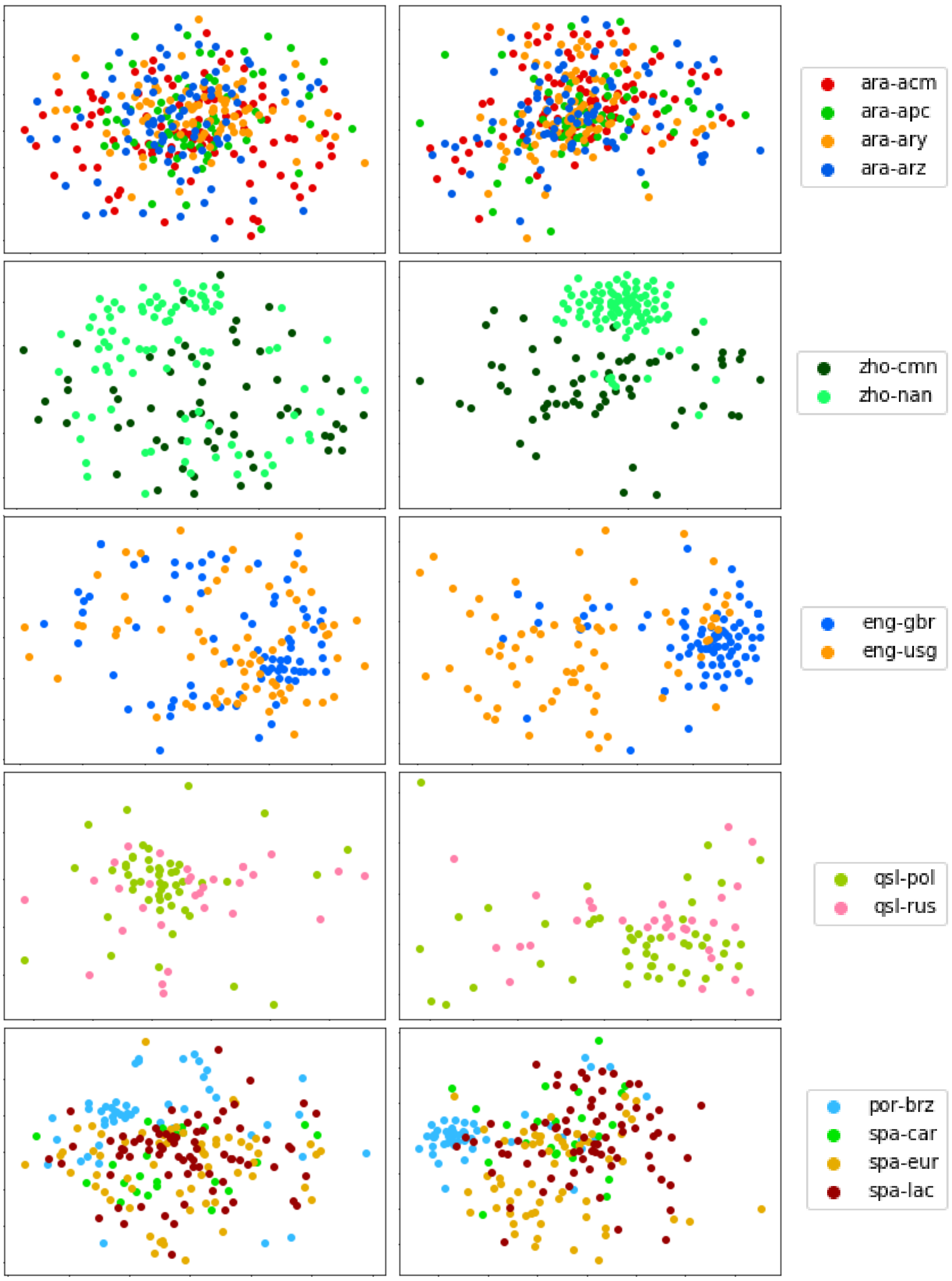


Figure 3.5: t-SNE scatter plots of unsupervised i-vectors (left) and MMSE s-vectors with $\lambda = 3$ (right) for the NIST LRE 2017 development dataset with 3 sec recordings. The language clusters belong to Arabic, Chinese, English, Slavic and Iberian (from top to bottom).

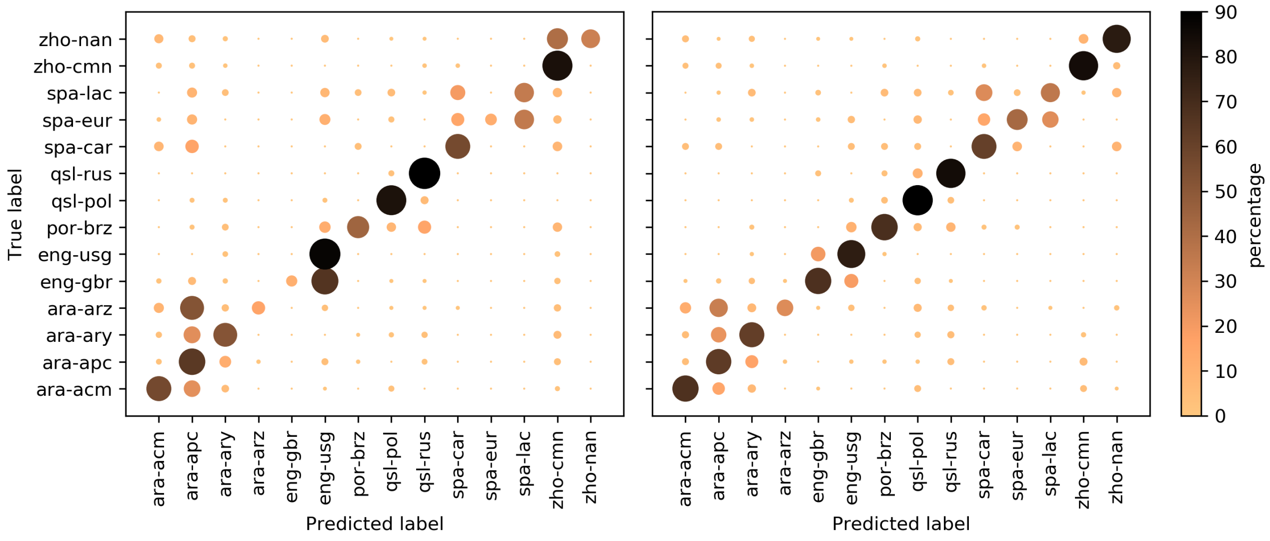


Figure 3.6: Row-normalized Confusion Matrices of i-vector system (left) and s-vector system with $\lambda = 3$ (right) on the NIST LRE 2017 development dataset for the 3 sec condition.

3.5.4 Confusion Matrix Analysis

The row-normalized confusion matrix plots for 3 sec recordings in the NIST LRE 2017 development set are shown in Figure 3.6. The ideal confusion matrix plot is an identity matrix where all the non-diagonal entries are zeros and the matrix is diagonally dominant. The confusion matrix plot of the baseline system (left side plot) indicates that for many languages like eng-gbr, ara-arz, and spa-eur, the diagonal entries are not the highest in the row (indicating that the majority of the examples of the particular language class are confused as another class within the same broad language cluster). While this issue persists for the ara-arz class in the proposed s-vector model (right side plot), all the other language classes have the desired diagonal dominance. In particular, the two language clusters that showed a good class separation in the tSNE plots (Figure 3.5) for the s-vector model (eng-usg versus eng-gbr and zho-cmn versus zho-nan) also showed significantly reduced confusions (Figure 3.6). Part of the degradation of the baseline system on these dialects of English and Chinese language cluster may be attributed to the highly imbalanced training data for these languages (Table 2.1). Thus, the confusion plots highlight that the proposed model not only improves the overall performance, but is able to improve the class-specific performances on challenging cases where the training data is somewhat limited.

3.5.5 Relationship to Prior Work

The idea behind the s-vector model is very similar to the PLDA and JFA models [16, 18], in the sense that all of these are supervised generative models. All three of these models can be viewed as a 2-step generative process. In the first step, a class-specific latent vector $\boldsymbol{v}$ (speaker/language factor) is drawn according to some distribution, and in the second step, the utterance-specific latent vector representing an example of the class is drawn from a normal distribution centered at $\boldsymbol{v}$, which encodes the class information. In the case of PLDA and JFA, the class-specific latent vector is modeled as a Gaussian distribution, whereas in the s-vector model, the class-specific vectors are assumed to come from a finite set $\{\boldsymbol{m}_1, \ldots, \boldsymbol{m}_l\}$. However, there is a key difference in the roles these models play in speaker and language recognition literature. The PLDA and JFA models are typically used in speaker verification to obtain the verification scores (Log-likelihood ratios) for a pair of recordings, whereas the s-vector model is proposed as a better embedding extractor than the i-vector model, particularly for language recognition.

The s-vector model is based on MMSE estimation of latent representations that have a GMM prior density in the total-variability factor analysis model. In the original i-vector model [11], a standard Gaussian density is used. The motivation of standard Gaussian density in the factor model is two fold - i) the Gaussian prior density results in a conjugate prior where the posterior density is also Gaussian distributed, ii) the MAP estimate is simply the posterior mean and that allows efficient estimation of latent representation for speaker/language recognition. However, the standard i-vector model is unsupervised and does not use the language labels of the training dataset even when they are available, for the language recognition task. The proposed approach overcomes this limitation by using a GMM prior density for the latent representation. The mixture components correspond to the target language classes. Comparing with the standard i-vector model, the proposed approach also yields a conjugate posterior density. However, the simple MAP estimation is no longer feasible and the more involved minimum-mean square error (MMSE) estimate is required to obtain the latent s-vector representations. Thus, with moderate increase in computational complexity, we show that the proposed approach is able to efficiently incorporate the label information in the training data for the embedding extractor.

A noteworthy attempt to utilize supervision in the i-vector model was made in [70], where the label information (in the form of one-hot encoding or mean speaker embedding) are appended with the adapted supervector, followed by the factor analysis model. The label regression loss and the supervector reconstruction error are jointly minimized to train the model. The i-vectors extracted from this model are also found to be more discriminative than the conventional i-vector model.

The use of GMM prior density has been attempted in the past in [57, 75, 76]. While the modeling strategy in these works is similar to the proposed approach, they make approximations to simplify the posterior estimation that defy the underlying Bayesian factor analysis model assumptions. In [57], the authors use the GMM prior density with each mixture component corresponding to one of the language class labels, similar to the proposed approach. The posterior density of the latent variable is written as,

$$\begin{aligned} p(\boldsymbol{y}(s)|X(s)) &= \sum_{l=1}^{L} p(\boldsymbol{y}(s), l|X(s)) \\ &= \sum_{l=1}^{L} p(l|X(s))p(\boldsymbol{y}(s)|l) \end{aligned} \tag{3.50}$$

However, the authors make an approximation by setting $p(l|X(s))$ to p(l), which is the prior probability of the language label [57]. Strictly speaking, this violates the Bayesian posterior probability model. The resulting embedding used in [57] ($E(\boldsymbol{y}(s)|X(s))$ with this approximation is not the MMSE estimate of the latent vector. In our work, we derive the exact expression for p($l|X(s)$) under the specific case, where the prior covariances of each class (mixture component) are shared and equal to $\frac{1}{\lambda}I$ (Eq. 3.45). Then, we proceed to use the posterior to find the MMSE estimate $E(\boldsymbol{y}(s)|X(s))$.

The methods developed in [75, 76], use a GMM prior density on the latent representations where the mixture components correspond to phonetic groups. The authors use an external phonetic recognizer (Hungarian phoneme recognition system) to obtain frame-level phoneme posteriors. These frame level posteriors are converted to utterance-level posteriors using an accumulation of frame level posterior statistics. In our approach, we do not employ an external model for posterior estimation as they are directly obtained from the s-vector model itself. In addition, the proposed approach has utterance level language labels which is different from the frame level phoneme labels used in [75, 76]. Hence, there is no approximation needed to convert posterior information from frame level to utterance level.

3.5.6 Estimating the Prior Weight Using Posterior Covariance

In the language recognition experiments reported in Table 3.5 and Table 3.6, the hyper-parameter λ that controls the weighting of the prior (covariance matrix of the prior density is $\frac{1}{\lambda}I$) is chosen based on the performance on the development data. In a subsequent analysis, we attempt the estimation of the hyper-parameter λ as a function of the trace of the posterior covariance $(I + T^{\intercal}\Sigma^{-1}N(s)T)^{-1}$. The diagonal entries of the posterior covariance matrix contain the variances along each dimension. Hence, its trace serves as a measure of *uncertainty* of $\boldsymbol{y}(s)$. Intuitively, lower uncertainty is associated with a higher level of confidence in the data. As we saw that a higher value of lambda is beneficial for cases with higher uncertainty, such as in shorter and noisy recordings, allowing λ to vary with the posterior covariance may be a reasonable experiment to try. We also note that the posterior covariance matrices of longer recordings typically have a smaller trace, implying lower uncertainty. As $N(s)$ contains the frame counts for each GMM component, its entries are directly proportional to the test utterance duration. The approach of tying the λ value to the posterior covariance is partly motivated by previous efforts on uncertainty propagation in factor analysis [77].

Table 3.7: Results on NIST LRE 2017 evaluation dataset for two conditions where the λ is fixed and when the λ is tied to the trace of the posterior covariance matrix.

	Performances : $100C_{primary}$ [EER (%)] {Accuracy(%)}		
Model config.	3 sec	10 sec	30 sec
MMSE s-vector (λ fixed)	43.7 [13.5] {61.2}	23.7 [6.5] {77.4}	**15.4** [3.8] {84.5}
MMSE s-vector (λ tied)	**43.6** [13.8] {61.5}	24.7 [6.8] {77.2}	15.5 [3.9] {84.5}

We use a second-order polynomial (obtained using the development set) to find the value of λ for each utterance. Note that, the prior-weighting changes for each utterance in this case and it is tied to the posterior covariance, unlike the fixed choice of λ per duration used in the previous experiments.

Table 3.7 compares the results on NIST LRE 2017 evaluation dataset for the two cases 1) with fixed λ that is duration specific, 2) with utterance level choice of λ that is tied to the trace of the posterior covariance matrix. As seen in this Table, the language recognition performance is similar in both cases indicating that prior density covariance parameter λ can be chosen based on the statistics of the data for each utterance.

3.5.7 Application to Closed Set Speaker Recognition

One of the potential drawbacks of the proposed approach is the reliance on the supervised labels in the embedding extraction. For language recognition tasks, the number of class labels are typically small thereby allowing the modeling of each language class with a GMM component. In tasks such as speaker recognition, where i-vector approaches are dominantly used, the number of class labels (speakers) can be significantly high. In order to test the limits of the proposed approach for cases with large number of classes, we perform a closed set speaker recognition task on the Librispeech dataset [78]. Here, we train a background model and the total variability matrix from a set of background speakers (from the Librispeech dataset). We created three test sets consisting of variable number of speakers (50, 100 and 200). A multi-class

Table 3.8: Performance in terms of equal error rate (EER) % for a closed set speaker recognition experiment on the Librispeech dataset.

Model config.	50 spk.	100 spk.	200 spk.
Unsup. i-vector	0.122	0.156	0.215
MMSE s-vector	0.122	0.158	0.218

SVM is used as back-end model for speaker classification. The i-vector/s-vector embeddings are used in these experiments and the performance is measured in terms of equal error rate (EER). Table 3.8 reports the results for speaker recognition experiment on Librispeech dataset. As seen in Table 3.8, the s-vector system did not improve over the i-vector approach in the closed set speaker recognition task. However, even with 200 speaker classes, the performance of the proposed s-vector model does not degrade compared to the i-vector approach. One potential future research direction for speaker verification would be to use an unsupervised speaker clustering approach to generate the label classes for the s-vector model. This may reduce the number of classes while still preserving the speaker discriminability used in the s-vector model.

3.6 Chapter Summary

In this chapter, we began by giving a detailed mathematical account of the popular i-vector approach, which was the state-of-the-art in speaker and language recognition. We then modified the prior distribution of the latent variables to introduce label information into the model, to make it supervised. We derived the expectation-maximization (EM) algorithm to estimate the model parameters of the proposed s-vector model. We introduced a hyperparameter to re-weight the prior covariances of each class, to make the model more discriminative.

With several experiments using the NIST LRE 2017 datasets, we showed that the proposed s-vector model performs much better than the conventional i-vector model. With data visualization techniques and confusion matrices, we showed that the s-vectors perform well in distinguishing between accents of a common language cluster. We also analysed how the hyperparameters influence the performance and how to appropriately choose them.

Chapter 4

Supervised Neural-Network Models for Speaker Verification

The deep learning methodologies in state-of-the-art speaker recognition systems are predominantly limited to the extraction of recording level embeddings. This is usually followed by generative modeling of the embeddings to output the verification score. In this chapter, we give an account of a neural network based pairwise discriminative back-end inspired by the PLDA model [79], followed by an end-to-end approach where the neural model outputs the verification score directly, given the acoustic feature inputs [80, 81]. These models, termed as E2E-NPLDA, combines the embedding extraction and back-end modeling into a single processing pipeline. The back-end modeling is achieved using a neural approach to PLDA modeling, called neural probabilistic linear discriminant analysis (NPLDA). In the NPLDA model, the verification score is computed as a discriminative similarity function. The development of the single neural E2E-NPLDA model allows the joint optimization of all the modules using a verification cost. Several speaker recognition experiments are performed using SITW, VOiCES, and NIST SRE datasets, where the proposed E2E-NPLDA model is shown to significantly improve over the state-of-art x-vector PLDA baseline system (relative improvements of up to 35 % in the primary cost metric). We also provide a detailed analysis of the influence of hyper-parameters, choice of loss functions, and data sampling strategies for training the model. In particular, we highlight that the proposed soft detection cost function based fine-tuning improves over other loss functions considered.

4.1 Introduction

The recent developments in the field of speaker and language recognition have mirrored the advancement in deep learning to derive speaker embeddings from time-delay neural networks (TDNN). The TDNN models are trained using large amounts of data on a speaker discrimination task and consist of a layer that generates recording level embeddings called x-vectors [32]. The x-vector embeddings have shown promising improvements over the i-vector embeddings for many speaker recognition tasks [82]. The embeddings, like i-vectors/x-vectors, are commonly processed with various transformations such as linear discriminant analysis (LDA) [27], unit length normalization [59] and within-class covariance normalization (WCCN) [83]. The transformed vectors are further modeled using the probabilistic linear discriminant analysis (PLDA) [42]. The PLDA model, typically formulated using Gaussian assumptions, computes a log-likelihood ratio from a pair of enrollment and test embeddings. The state-of-art systems use a neural model to extract x-vector embeddings of fixed dimension followed by a generative Gaussian PLDA back-end model [32].

First, we describe our proposed a neural approach to PLDA modeling [79, 80]. This approach consists of a back-end modeling framework that integrates pre-processing and scoring, called NPLDA. The advantages of a neural back-end are twofold.

- The neural back-end model can be directly optimized for the detection cost function (DCF) which is used as the performance metric in speaker verification systems. This is in contrast to the multitude of optimization functions in LDA, WCCN, and PLDA in the state-of-art systems.

- The neural back-end model will allow the integration of the front-end neural embedding extractor with the back-end model to form a single deep neural model for ASV system design. The joint training of this fully neural model will further allow the embedding extractor to be optimized with a verification cost function as opposed to the classification cost used in current embedding extractors [32].

Further, we extend our efforts on NPLDA to develop a single end-to-end Siamese neural network model (E2E-NPLDA). The builds on the advantages mentioned above to construct a neural network model consisting of two parallel threads of processing - one for enrollment utterance and the other for the test utterance. These threads share the weights and combine the embedding extraction and pre-processing steps. The final outputs from the two processing threads are used in a quadratic score function emulating the PLDA score computation. We use an approximation to the minimum detection cost (C_{Min} or minDCF) [84] to optimize the E2E-NPLDA model. Thus, the E2E-NPLDA framework offers an elegant processing pipeline where the input acoustic features of the enrollment and test recordings are fed to the model which outputs the verification score. We also provide a comprehensive analysis of the initialization aspects of the E2E-NPLDA model, the choice of cost function used in the optimization, data sampling strategies in training the model, and the computational efficiency of the model.

The rest of the chapter is organized as follows. The related prior work is discussed in Section 4.2. The back-end modeling approaches are detailed in Section 4.3. The Neural PLDA back-end is introduced in Section 4.4. The cost functions that are plausible in neural ASV systems are described in Section 4.4.1. The approach to end-to-end neural modeling using Siamese neural network architecture (E2E-NPLDA) is discussed in Section 4.4.2. Section 4.5 reports the experiments and results. Section 4.6 presents various model considerations and their impact on the performance. This is followed by a summary of the work in Section 4.7.

4.2 Related Prior Work

The pairwise generative and discriminative modeling approaches to ASV back-end design were investigated by Cumani et. al. [44, 85, 86]. The discriminative version of PLDA with logistic regression and support vector machine (SVM) kernels was explored by Burget et. al. [55]. In SRE experiments, the discriminative PLDA (DPLDA) was seen to over-fit on the training speakers [87]. For deriving speaker embeddings, Ferrer et. al. [88] explored the regularization of embedding extractor networks using GB. Similarly, Mingote et. al. [89] proposed an approximate DCF metric for text-dependent speaker verification.

The earliest work on end-to-end Siamese modeling for signature verification used time-delay neural network models was proposed by Bromley et. al [90]. Heigold et.al. [91] proposed an end-to-end text-dependent speaker verification system using LSTM architecture to derive embeddings of enrollment and test segments, followed by cosine similarity scoring. The model is trained by minimizing the binary cross-entropy (BCE). Zhang et. al. [92] proposed a siamese network with sequence to sequence attention mechanism to achieve text-dependent speaker verification. Wan et. al. [93] explored a generalized end-to-end loss by minimizing the centroid means of within speaker distances while maximizing across speaker distances. Snyder et. al. [31] proposed the use of a network-in-network based Siamese end-to-end architecture for text-independent speaker verification. For the ASV training with a few thousand speakers ($5-15$k speakers), the i-vector baseline was consistently better than the E2E approach [31], whereas, with a large number of training speakers (102k speakers), the proposed neural ASV approach outperformed the i-vector baseline. Rohdin et. al. [94] developed the joint modeling of the PLDA scoring with the i-vector extraction using a deep neural network architecture. In another E2E effort, the use of triplet loss was explored by Zhang et. al. [95]. An unsupervised approach to train Siamese networks for speaker verification using triplet loss was proposed by Khan et. al. [96]. In spite of these efforts, the state-of-the-art ASV system uses the x-vector embeddings, followed by pre-processing of the embeddings and the generative Gaussian PLDA back-end model to generate the verification score [32].

During the same time as we published the work detailed in this chapter, several other publications closely related to this work were also published. In the following paragraphs, we briefly discuss some of these publications and connect them to our contributions in this chapter.

[Chung et. al., 2020] - Metric learning for ASV [97]] - In this paper, the authors showed that *metric learning* objectives for open-set speaker verification can generate results that are comparable to, and in some cases even outperform the traditional approach of using classification objectives to obtain embeddings. Unlike classification objectives, metric learning objectives are functions of similarity distance measures between pairs of embeddings. These metric learning objectives allow speaker verification systems to be trained end-to-end, and the similarity measure can directly act as a verification score without having to train a separate back-end model. The loss functions we use for our neural network approaches also fall under the category of metric learning.

[Ferrer et. al., 2020-2022] - Condition aware discriminative back-end for speaker verification [98, 99, 100]] - In this series of publications, Ferrer et. al. propose a very similar approach to our Neural PLDA. Just like our approach, they too, implement the PLDA log-likelihood score in a neural network architecture and optimize the parameters using the binary cross-entropy loss function. However, their main objective is not to improve the discriminative performance but to achieve robust calibration across several conditions. In their initial work [98, 99], they achieve this by augmenting the neural PLDA loss function with another branch that estimates the calibration parameters to scale and shift the raw scores output by the Neural PLDA module. In their latest publication [100], they call their back-end model discriminative condition aware PLDA (DCA-PLDA), where they augment another branch to the existing pipeline to use the duration information to perform *duration-dependent calibration*, apart from the side information dependent calibration discussed in their previous publications [98, 99].

4.3 An account of back-end models

The back-end modeling of speaker embeddings typically involves centering, dimensionality reduction using LDA, and length normalization as pre-processing steps. In this section, we present the key mathematical details of popular methods used for back-end modeling of the pre-processed embeddings for speaker verification. These include the generative Gaussian PLDA modeling [42, 101], discriminative PLDA modeling in SVM framework [55], the pairwise Gaussian back-end model [44]. These models serve as baselines for our experiments.

4.3.1 Generative Gaussian PLDA (GPLDA)

The GPLDA model on the embeddings (x-vector with post-processing) for a recording is given by,

$$\boldsymbol{\eta}_r = \boldsymbol{\Phi}\boldsymbol{w} + \boldsymbol{\epsilon}_r \tag{4.1}$$

Here, $\boldsymbol{w}$ is the latent speaker vector having a standard Gaussian prior density, $\boldsymbol{\eta}_r$ is the x-vector embedding after post-processing, $\boldsymbol{\Phi}$ is the speaker variability matrix and $\boldsymbol{\epsilon}_r$ is the residual vector having a Gaussian prior density with zero mean and covariance denoted as $\boldsymbol{\Sigma}$, that models the within speaker covariance. The across-speaker covariance is given by $\boldsymbol{\Sigma}_{ac} = \Phi\Phi^{\intercal}$, and the total covariance is given by $\boldsymbol{\Sigma}_{tot} = \Phi\Phi^{\intercal} + \boldsymbol{\Sigma}$.

If enrollment and test x-vectors (after post-processing) are denoted as $\boldsymbol{\eta}_e$ and $\boldsymbol{\eta}_t$ respectively [59], the GPLDA score (log-likelihood ratio) is defined as,

$$\begin{aligned} l(\boldsymbol{\eta}_e, \boldsymbol{\eta}_t) = \log \frac{p(\boldsymbol{\eta}_e, \boldsymbol{\eta}_t | H_t)}{p(\boldsymbol{\eta}_e, \boldsymbol{\eta}_t | H_{nt})} &= \log \mathcal{N}\left(\begin{pmatrix}\boldsymbol{\eta}_e \\ \boldsymbol{\eta}_t\end{pmatrix}; \begin{pmatrix}\mathbf{0} \\ \mathbf{0}\end{pmatrix}, \begin{pmatrix}\boldsymbol{\Sigma}_{tot} & \boldsymbol{\Sigma}_{ac} \\ \boldsymbol{\Sigma}_{ac} & \boldsymbol{\Sigma}_{tot}\end{pmatrix}\right) \\ &- \log \mathcal{N}\left(\begin{pmatrix}\boldsymbol{\eta}_e \\ \boldsymbol{\eta}_t\end{pmatrix}; \begin{pmatrix}\mathbf{0} \\ \mathbf{0}\end{pmatrix}, \begin{pmatrix}\boldsymbol{\Sigma}_{tot} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_{tot}\end{pmatrix}\right) \end{aligned} \tag{4.2}$$

where H_t and H_t are the target and non-target hypotheses respectively. Substituting for the Gaussian density expressions and expanding, the log-likelihood ratio evaluates to a quadratic function of $\boldsymbol{\eta}_e$ and $\boldsymbol{\eta}_t$, as follows:

$$l(\boldsymbol{\eta}_e, \boldsymbol{\eta}_t) = \boldsymbol{\eta}_e^\intercal \boldsymbol{Q} \boldsymbol{\eta}_e + \boldsymbol{\eta}_t^\intercal \boldsymbol{Q} \boldsymbol{\eta}_t + 2\boldsymbol{\eta}_e^\intercal \boldsymbol{P} \boldsymbol{\eta}_t + \text{const} \tag{4.3}$$

where,

$$\boldsymbol{Q} = \boldsymbol{\Sigma}_{tot}^{-1} - (\boldsymbol{\Sigma}_{tot} - \boldsymbol{\Sigma}_{ac}\boldsymbol{\Sigma}_{tot}^{-1}\boldsymbol{\Sigma}_{ac})^{-1} \tag{4.4}$$

$$\boldsymbol{P} = \boldsymbol{\Sigma}_{tot}^{-1}\boldsymbol{\Sigma}_{ac}(\boldsymbol{\Sigma}_{tot} - \boldsymbol{\Sigma}_{ac}\boldsymbol{\Sigma}_{tot}^{-1}\boldsymbol{\Sigma}_{ac})^{-1} \tag{4.5}$$

4.3.2 Discriminative PLDA (DPLDA)

The discriminative PLDA [55] uses the expanded vector $\boldsymbol{\varphi}(\boldsymbol{\eta}_e, \boldsymbol{\eta}_t)$, where $(\boldsymbol{\eta}_e, \boldsymbol{\eta}_t)$ are enrollment and test embeddings.

$$\boldsymbol{\varphi}(\boldsymbol{\eta}_e, \boldsymbol{\eta}_t) = \begin{bmatrix} 1 \\ \text{vec}(\boldsymbol{\eta}_e + \boldsymbol{\eta}_t) \\ \text{vec}(\boldsymbol{\eta}_e\boldsymbol{\eta}_t^\intercal + \boldsymbol{\eta}_t\boldsymbol{\eta}_e^\intercal) \\ \text{vec}(\boldsymbol{\eta}_e\boldsymbol{\eta}_e^\intercal + \boldsymbol{\eta}_t\boldsymbol{\eta}_t^\intercal) \end{bmatrix} \tag{4.6}$$

Here, "vec" corresponds to the operation of vectorizing (flatenning) a 2-D matrix into a single column vector. Using the expanded vector, the PLDA score is computed as,

$$s = \boldsymbol{\omega}^\intercal \boldsymbol{\varphi}(\boldsymbol{\eta}_e, \boldsymbol{\eta}_t) \tag{4.7}$$

The weight vector $\boldsymbol{\omega}$ is trained using SVM with a quadratic kernel. The verification score on the test trials is generated as the inner product of the weight vector $\boldsymbol{\omega}$ with the expanded vector $\boldsymbol{\varphi}$. For an embedding dimension of N, the dimension of $\boldsymbol{\phi}$ is $2N^2 + N + 1$. This large dimensionality of $\boldsymbol{\phi}$ is one of the most crucial drawbacks of the DPLDA model which is overcome by the quadratic layer implementation of the proposed NPLDA back-end.

4.3.3 Pairwise Gaussian back-end (GB)

In this modeling, two Gaussian distributions (mean and covariance matrices) are learned from the training data corresponding to target trial condition and non-target trial condition. The Gaussian distributions learned from the training are used to generate a likelihood ratio score on the given test trial. The ratio of the log-likelihood (target likelihood to non-target likelihood ratio) is used as the final score from the model. Given a trial of enrollment and test embeddings $\boldsymbol{\eta_e}$ and $\boldsymbol{\eta_t}$, the pairwise GB [102, 86] models the concatenated vector of enrollment and test embeddings, $\boldsymbol{\eta} = [\boldsymbol{\eta}_e{}^\intercal \ \boldsymbol{\eta}_t{}^\intercal]^\intercal$ as a Gaussian distribution with parameters $(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ for target trials, and $(\boldsymbol{\mu}_{nt}, \boldsymbol{\Sigma}_{nt})$ for non-target trials.

The log-likelihood ratio score (l) for a trial ($\boldsymbol{\eta} = [\boldsymbol{\eta}_e{}^\intercal \ \boldsymbol{\eta}_t{}^\intercal]^\intercal$) is computed as,

$$l = (\boldsymbol{\eta} - \boldsymbol{\mu}_{nt})^\intercal \boldsymbol{\Sigma}_{nt}^{-1} (\boldsymbol{\eta} - \boldsymbol{\mu}_{nt}) - (\boldsymbol{\eta} - \boldsymbol{\mu}_t)^\intercal \boldsymbol{\Sigma}_t^{-1} (\boldsymbol{\eta} - \boldsymbol{\mu}_t) \tag{4.8}$$

4.4 The Neural PLDA (NPLDA) Approach

The neural PLDA (NPLDA) is our novel approach to converting the probabilistic linear discriminant analysis (PLDA) back-end to a neural network model. Fig. 4.1 shows a Siamese neural network architecture depicting a shared embedding extractor for the enrollment and test branches with the NPLDA back-end. While the embedding extractors can have any architecture, in this thesis, we restrict ourselves to two popular neural network architectures, namely, the E-TDNN and F-TDNN. Recently, several advances in the architecture front have been used for speaker verification, such as residual networks [97] and ECAPA-TDNN [103], and the NPLDA approach can be extended to these architectures as well.

In the GPLDA approach, the standard operations that are done on the speaker embeddings are centering, dimensionality reduction with linear discriminant analysis (LDA), and unit length

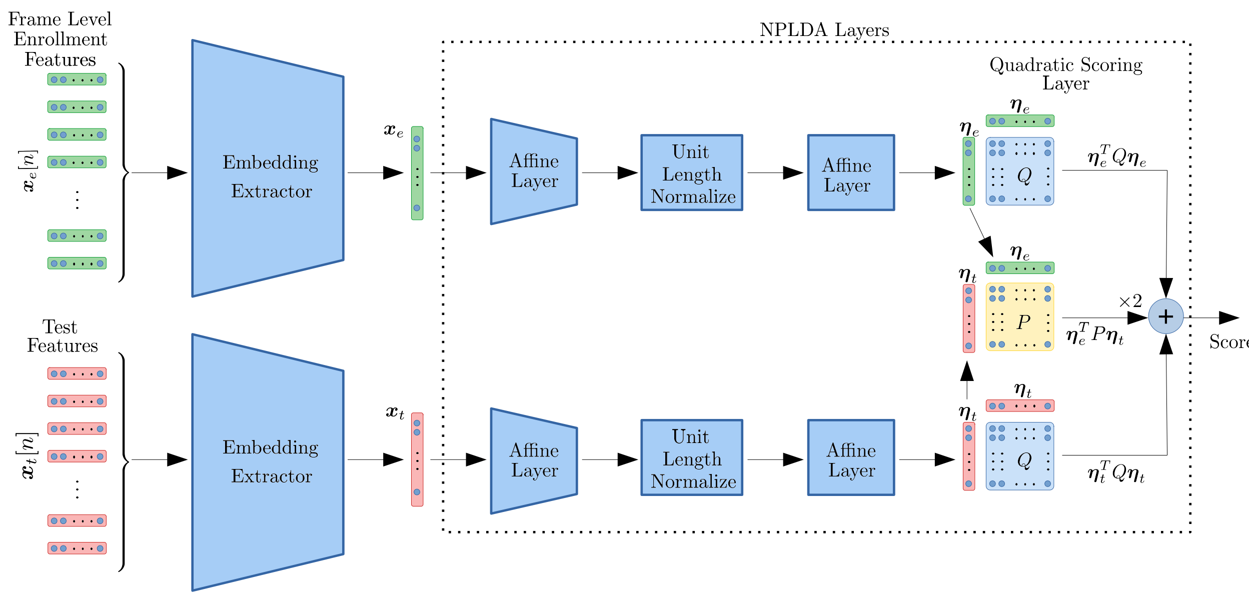


Figure 4.1: The E2E-NPLDA architecture for speaker verification. The network parameters (shaded in blue) in the top (enrollment) and bottom panel (test) are shared.

normalization [59]. These operations are performed in the first layer of the NPLDA model by posing the centering and LDA as an affine transformation followed by the length normalization non-linearity. Note that the length normalization is a smooth differentiable non-linearity that is typically used with cosine similarity losses in neural networks.

The second layer of the NPLDA model replicates the quadratic score computation in PLDA. The PLDA log-likelihood score given in Eq. (4.2) is implemented as a quadratic neural layer. The Kaldi implementation of PLDA applies a linear transformation that centers the embeddings (after unit length normalization) and simultaneously diagonalizes the within and between class covariance matrices [59, 104]. This is replicated as the affine layer after the unit length normalization layer. The NPLDA model parameters are learned in a backpropagation setting with a suitable loss function. In order to train the NPLDA model for speaker verification, pairs of x-vector embeddings representing target (from the same speaker) and non-target hypotheses (from different speakers) are sampled in batches.

In the following section, we discuss various objective function choices for training neural

ASV models. Apart from the quadratic score functions, these objective functions can also be applied to other similarity or distance metrics, such as the cosine similarity or L2 distance between embeddings.

4.4.1 Objective Functions for Neural ASV Models

4.4.1.1 Binary cross-entropy (BCE) and its weighted versions

A generalized version of the popular BCE loss can be obtained by scaling the terms corresponding to the target and non-target hypotheses with weights λ_1 and λ_2 respectively. This can be expressed as

$$L_{GBCE} = \lambda_1 \sum_{i \in \mathcal{T}} \log(1 + e^{-(s_i - \theta)}) + \lambda_2 \sum_{i \in \mathcal{N}} \log(1 + e^{(s_i - \theta)}) \tag{4.9}$$

Here, s_i is the output of the model (score) for trial i, θ is the threshold applied, $\mathcal{T}$ and $\mathcal{N}$ are the sets of indices corresponding to target and non-target trials respectively.

- **Vanilla BCE loss:** The simple BCE loss is a special case of this generalized version where $\lambda_1 = \lambda_2$, and $\theta = 0$. This is used in Heigold et. al. [91] Wan et. al. [93].

- **Log-likelihood ratio cost (Cllr):** The log-likelihood ratio cost function (C_{llr}) is a proper scoring rule typically used as a metric to evaluate the discrimination as well as calibration capabilities of a speaker verification system [54]. It is also a special case of , where $\lambda_1 = \frac{1}{|\mathcal{T}|}$ and $\lambda_2 = \frac{1}{|\mathcal{N}|}$. A scaled version of the C_{llr} metric was used as the loss function in Snyder et. al. [31].

- **Prior weighted C_{llr} (WCllr):** The standard objective used in likelihood ratio calibration is a prior weighted version of C_{llr} [105]. This loss function is used for optimizing the back-end in Ferrer et. al. [98, 99]. Here, $\lambda_1 = \frac{C_{Miss}\pi}{|\mathcal{T}|}$, $\lambda_2 = \frac{C_{FA}(1-\pi)}{|\mathcal{N}|}$, and $\theta = \log \frac{C_{Miss}(1-\pi)}{C_{FA}\pi}$, where π is the prior probability of target trials, and $CMiss$ and C_{FA} are the Bayesian detection costs associated with the miss and false alarms respectively.

4.4.1.2 Proposed soft detection cost

In ASV systems, the normalized detection cost function (DCF) [84] is :

$$C_{Norm}(\beta, \theta) = P_{Miss}(\theta) + \beta P_{FA}(\theta) \tag{4.10}$$

where β is defined as

$$\beta = \frac{C_{FA}(1-\pi)}{C_{Miss}\pi} \tag{4.11}$$

and θ is the threshold value on the score. The cost associated with a miss and false alarms are C_{Miss} and C_{FA} respectively, and the prior probability of target trials is denoted as π.

The probability of miss/false-alarms is computed by applying the threshold θ to the score s_i.

$$P_{Miss}(\theta) = \frac{1}{|\mathcal{T}|} \sum_{i \in \mathcal{T}} \mathbb{1}(s_i < \theta) \quad ; \quad P_{FA}(\theta) = \frac{1}{|\mathcal{N}|} \sum_{i \in \mathcal{N}} \mathbb{1}(s_i \geq \theta) \tag{4.12}$$

Here, $\mathbb{1}$ is the indicator function, $\mathcal{T}$ denotes the set of target trials, and $\mathcal{N}$ denotes the set of non-target trials.

The minimum detection cost (C_{min} or minDCF) is achieved at the threshold where the DCF is at the minimum value.

$$C_{Min} = \text{minDCF} = \min_{\theta} C_{Norm}(\beta, \theta) \tag{4.13}$$

The optimization using Eq. (4.10) is infeasible due to the step discontinuity in the indicator function $\mathbb{1}$. We propose a differentiable approximation of the normalized detection cost using a warped sigmoid function.

We define the soft detection cost function (SoftDCF) as:

$$C_{Norm}^{(\text{soft})}(\beta, \theta) = P_{Miss}^{soft}(\theta) + \beta P_{FA}^{soft}(\theta) \tag{4.14}$$

Here, the soft probabilities of miss and false-alarm are computed as:

$$P_{Miss}^{(\text{soft})}(\theta) = \frac{1}{\mathcal{T}} \sum_{i \in \mathcal{T}} [1 - \sigma(\alpha(s_i - \theta))] \;\; ; \;\; P_{FA}^{(\text{soft})}(\theta) = \frac{1}{\mathcal{N}} \sum_{i \in \mathcal{N}} \sigma(\alpha(s_i - \theta)) \tag{4.15}$$

A large value for α, would make the warping function closely approximate the true detection cost function for a wide range of thresholds. In our experiments, we choose $\alpha = 15$. Further, the threshold θ is also considered as a model parameter.

With the neural models for embedding extraction and back-end, we explore the construction of a Siamese neural network model that can be input with the acoustic features to directly generate the verification score. The key advantage of such an approach will be the joint optimization of all the parameters of the network using a speaker verification cost. Further, this approach would also build upon the prior research advancements made in the field on embedding extraction, pre-processing of embeddings, and the PLDA modeling for verification score. We call the proposed family of models E2E-NPLDA. The proposed E2E-NPLDA architecture is shown in Fig. 4.1. The early layers of the model are identical to the embedding extraction framework while the later layers implement the back-end score modeling.

4.4.2 The E2E-NPLDA model for speaker Verification

The learnable parameters in the top panel (enrollment) and bottom panel (test) of the E2E-NPLDA model (Fig 4.1) are tied. This sharing makes the architecture Siamese. The front-end model parameters are initialized with the embedding extractor parameters while the PLDA model parameters are used to initialize the back-end layers. The entire Siamese model is trained using the speaker verification cost (soft detection cost function given in Eq. (4.14)).

The trainable parameters include the components of the quadratic cost function as well.

The trials for training the model consist of recordings from several thousand speakers. This results in millions of trials (enrollment-test combinations). Hence, a data sub-sampling strategy that accounts for the available GPU memory is required to enable the efficient implementation of the proposed model[1].

4.4.2.1 Brute force trial sampling

In the brute force implementation, each training batch of N trials consists of $2N$ unique utterances assuming there are no repetitions. We estimate the memory required for a batch update of training parameters as the sum of memory required to store the network parameters, gradients, forward and backward activations. Let each utterance contain T acoustic frames. Let k_i denote the dimension of the input to the i^{th} TDNN layer in the embedding extractor network. With a context of c_i frames, the total memory required by the TDNN embedding extractor alone is $2NT\sum_i k_i c_i \times 8$ bytes. A batch-size of 2048 trials would consist of 4096 unique utterances. Assuming each utterance to consist of 20 second segments ($T = 2000$) along with the 10 layers of TDNN architecture, the brute force implementation results in GPU memory load of 120 GB. Hence, we propose a trial sub-sampling strategy that involves sharing the enrollment/test sides of the pairs which form a trial.

Reducing the batch size or the segment length has adverse effects on the model training/performance. A reduced batch size means that the speaker variability within a batch was insufficient to train the model properly and this resulted in the divergence of the training loss. A reduced segment length makes the audio chunks too small to extract meaningful speaker information from the audio segment in the embeddings. The training and testing loss performance degrades with the reduced segment duration. In essence, this means that developing a novel memory-efficient sampling strategy is the only option for improving the performance of the model.

[1]The implementation of the proposed model can be found in https://github.com/iiscleap/E2E-NPLDA

4.4.2.2 Low memory trial sampling

We propose to construct trials for training the E2E-NPLDA with significant sharing of the speech segments. For ease of implementation, we fix the number of frames from each recording by creating a modified dataset. This involves splitting longer utterances into multiple segments with a fixed number of frames. The maximum duration of the segment is 20 seconds. With this setting, a batch of 64 segments leads to a total duration of $NT = 1280$ seconds. The memory requirement for the embedding extraction on these segments is about 15 GB.

The next strategy is to form as many trials as possible from these $N = 64$ audio segments. Let the $N = 64$ segments contain m speakers, with approximately equal number of $\frac{64}{m}$ segments per speaker. The embeddings generated for these 64 segments (through the embedding extractor) are used to form (64 choose 2) ${}^{64}C_2 = 2016$ trials and these trials form the batch of training for the E2E-NPLDA model. This strategy reduces the memory footprint significantly. Each of these trials is assigned a label of target or non-target and the model training is performed using the cost function defined in Eq. (4.10). The gradients are back-propagated to update all the E2E-NPLDA model parameters. To avoid cross-gender trials in training, we also make use of gender labels.

It is worth noting that the batch statistics of the two sampling methods are significantly different. A batch of trials in the brute force sampling method (Algo. 1) can contain trials from multiple gender/domains, whereas, in the proposed low memory sampling method (Algo. 2), all the trials in a batch are from the same gender/domain. Further, the segments that form the trials are shared. Hence, the number of unique speakers within a batch of trials is much smaller for the low memory sampling method (Algo. 2) compared to the brute force sampling approach. In spite of these statistical differences, our experiments on the NPLDA model show that the training of the model is not impacted by the low memory trial sampling (Sec. 4.6.2).

Table 4.1: Description of the evaluation datasets and test conditions from SITW, VOiCES and SRE18 VAST corpora.

Dataset	Recording Condition	Avg. dur (Sec)		Trials	Evaluation parameters		
		Enrol.	Test		π	C_{Miss}	C_{FA}
SITW [52]	Multi-media	427	80.6	721k	0.01	1	1
VOiCES Dev [53]	Reverb/Noise	14.5	15.6	4M	0.01	1	1
VOiCES Eval [53]	Reverb/Noise	14.4	15.6	3.6M	0.01	1	1
SRE18 Dev VAST [49]	Multi-media	188.8	257	270	0.05	1	1
SRE18 Eval VAST [49]	Multi-media	152.2	231	31k	0.05	1	1

4.5 Experiments and Results

We present experiments to evaluate the NPLDA and E2E-NPLDA models using two popular embedding extractor architectures. We perform several experiments on data sampling strategies, neural network parameter initialization approaches training utterance duration and various loss functions. The proposed E2E-NPLDA model is compared with baseline systems involving the x-vector PLDA approach.

All the systems are trained using the VoxCeleb 1 [106] and VoxCeleb 2 corpora [35] with a sampling rate of 16 kHz. These datasets contain English language speech extracted from celebrity interview videos available on YouTube, spanning a wide range of different ethnicities, accents, professions, and ages. The entire VoxCeleb 2 dataset with 6112 speakers and the Dev portion of VoxCeleb 1 verification split with 1211 speakers constitute the training set. After removing utterances that are less than 5 seconds and removing speakers with less than eight utterances, we are left with $1,276,392$ utterances of 7320 speakers, of which 4423 are male and 2897 are female. The total duration of the training dataset is 2779 hours. The test portion of VoxCeleb 1 with 40 speakers is used as a validation set for hyperparameter selection and to choose the model checkpoint for evaluation on various datasets.

We present ASV results on Speakers in the Wild (SITW) [52] eval, VOiCES [53], and the NIST SRE 2018 VAST evaluation [49] datasets. The details of these datasets like number of trials, and evaluation parameters are given in Table 4.1.

4.5.1 Embedding extractors

For all our experiments, we experiment with two popular neural network architectures for embedding extraction. Both the embedding extractors are trained with 30-dimensional MFCC features from 25 ms frames shifted every 10 ms using a 40-channel Mel-scale filterbank spanning the frequency range of 20 Hz - 7600 Hz. The pre-processing includes an energy based voice activity detection (VAD) and cepstral mean-variance normalization (CMVN). A 5-fold augmentation was used to generate 6.38M training segments for the combined VoxCeleb set.

4.5.1.1 E-TDNN

For the first set of experiments, we train the extended time-delay neural network (E-TDNN) architecture described in [82] to extract the x-vector embeddings. The network contains 10 layers of time-delay neural networks (TDNN) with ReLU non-linearity and batch normalization, followed by a statistics pooling layer, which computes the sample mean and standard deviations across the time frames. The pooled output is connected to two feed-forward layers with ReLU non-linearity and batch normalization, and the output layer has 7320 dimensions. The network is trained with a softmax cross-entropy objective using the Kaldi toolkit [107]. The x-vectors of 512 dimensions are extracted from the affine component (layer 12).

4.5.1.2 F-TDNN

For the second set of experiments, we train the embedding extractor with the architecture described in [87]. The architecture uses factorized TDNN (F-TDNN) [33] layers. The F-TDNN reduces the number of parameters of the network by factorizing the weight matrix of each TDNN layer as the product of two low-rank matrices. The semi-orthogonalization step of the weight matrices is done after every 5 steps of backpropagation. The architecture consists of an input TDNN layer, followed by eight F-TDNN layers with ReLU non-linearity, batch normalization, and dropouts. Out of these F-TDNN layers, three layers have a skip connection input. This is followed by a dense ReLU layer and a stats pooling layer which accumulates the sample means and standard deviations across time frames. The segment level processing consists of two fully connected layers, and the output layer is identical to the E-TDNN architecture. The x-vectors of 512 dimensions are extracted from the affine layer following the pooling layer.

4.5.2 Experiments with the NPLDA Back-End and E2E-NPLDA Models

We perform experiments for the two embedding extractor networks and report results for three baseline systems - GPLDA, GB, and DPLDA. We compare the baseline systems with the NPLDA back-end and the E2E-NPLDA models. The common post-processing steps applicable to all the back-end models following the extraction of x-vectors include centering, dimensionality reduction using LDA, followed by unit length normalization [59]. Our initial experiments during the SRE 2018 evaluation [102, 108] gave us the best baseline results when the LDA dimension was set to 170. All the further experiments conducted thereafter towards this thesis used an LDA dimension of 170. The first affine layer of the NPLDA model that corresponds to centering and LDA has an output size of 170 to match this choice. The GPLDA model is trained using the recipe from Kaldi [107]. The parameters of the model are estimated using the expectation-maximization (EM) algorithm.

For neural-network experiments using the softDCF loss, the hyperparameters such as the warping factor α, learning rate, and batch size, were chosen based on several initial experiments with the NPLDA back-end whose results are not reported in this thesis. We used the brute force sampling algorithm (Sec. 4.4.2.1) with a fixed batch size of 1024 trials, and ran several experiments with various combinations of warping factors (α) and learning rates, with α ranging from 1 to 200, and a (fixed) learning rate ranging from 10^{-2} to 10^{-9}. The best performance on the VoxCeleb 1 validation set was achieved for $\alpha = 15$, and learning rate 10^{-4}. Further improvements were achieved by reducing the learning rate by 50% if the validation loss increased

Table 4.2: Performance comparisons of baseline systems with the best performing NPLDA and E2E-NPLDA models

(a) ETDNN Results

Model	Init.	SITW-E		VOiCES-D		VOiCES-E		SRE18 D-V		SRE18 E-V	
		EER	C_{Min}	EER	C_{Min}	EER	C_{Min}	EER	C_{Min}	EER	C_{Min}
GPLDA	-	2.87	0.301	2.56	0.300	6.45	0.521	7.41	0.461	15.56	0.641
GB	-	4.08	0.411	3.66	0.377	9.41	0.666	11.11	0.527	16.55	0.685
DPLDA	-	3.06	0.313	2.63	0.324	6.62	0.558	8.23	0.556	14.24	0.592
NPLDA	GPLDA	2.10	0.209	1.79	0.223	5.69	0.450	7.41	0.377	**12.55**	0.533
E2E-NPLDA	GPLDA	2.36	0.216	2.04	0.231	**5.60**	**0.425**	**6.58**	0.379	12.83	0.530
E2E-NPLDA	NPLDA	**1.92**	**0.198**	**1.67**	**0.211**	5.70	0.438	**6.58**	**0.300**	13.65	**0.502**
Relative improvements for E2E-NPLDA over GPLDA in %		33	34	35	30	13	18	11	35	18	22
Relative improvements for E2E-NPLDA over NPLDA in %		9	5	7	5	0	3	11	20	-9	6

(b) FTDNN results

Model	Init.	SITW-E		VOiCES-D		VOiCES-E		SRE18 D-V		SRE18 E-V	
		EER	C_{Min}	EER	C_{Min}	EER	C_{Min}	EER	C_{Min}	EER	C_{Min}
GPLDA	-	2.68	0.276	2.15	0.268	6.01	0.490	7.41	0.449	15.82	0.610
GB	-	3.47	0.346	2.94	0.315	7.28	0.548	12.76	0.486	17.46	0.647
DPLDA	-	2.90	0.280	2.21	0.273	6.00	0.480	7.82	0.444	13.02	0.553
NPLDA	GPLDA	1.78	0.178	1.49	0.176	5.18	0.432	**6.17**	0.337	12.38	0.467
E2E-NPLDA	GPLDA	1.96	0.177	1.48	0.181	5.45	**0.396**	**6.17**	**0.226**	**12.19**	0.497
E2E-NPLDA	NPLDA	**1.67**	**0.165**	**1.36**	**0.166**	**4.99**	0.407	7.00	0.263	12.68	**0.455**
Relative improvements for E2E-NPLDA over GPLDA in %		38	40	37	38	17	19	17	50	23	25
Relative improvements for E2E-NPLDA over NPLDA in %		6	7	9	6	4	6	-13	19	-2	3

for two consecutive epochs. With a warping factor $\alpha = 15$, and an initial learning rate of 10^{-4}, we conducted a few experiments by varying the batch size from 128 to 4096 trials. The best results were obtained for a batch size of 2048 trials.

The results of these experiments are reported in Table 4.2. The evaluation metrics reported

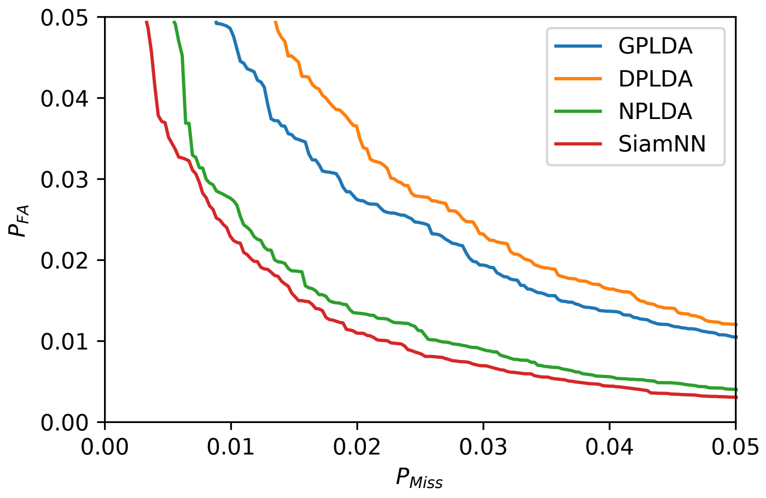


Figure 4.2: DET plots for the various models using the FTDNN architecture on the SITW evaluation set.

are the equal error rate (EER) and the minimum detection cost (C_{Min}) defined in Eq. (4.13). The value of β for the softDCF loss was computed separately for each test dataset, based on the values of π, C_{FA} and C_{Min} defined in Table. 4.1. For the NPLDA back-end, we used a warping factor $\alpha = 15$, initial learning rate 10^{-4}, reducing it by a factor of 0.5 if the validation loss increased for two successive epochs. For the E2E-NPLDA model, a higher initial learning rate of 10^{-6} was required. The low-memory trial sampling (Algo.2, as discussed in Sec. 4.4.2.2), with a batch size of 64 unique utterances per batch. To sample these 64 utterances, we first select k speakers at random for each batch, where k can range from 3 to 8. Then, we choose $\lfloor \frac{64}{k} \rfloor + 1$ utterances from ($64 \mod k$) speakers, and $\lfloor \frac{64}{k} \rfloor$ utterances from the remaining speakers. The trials sampled using this algorithm were also used for the GB and DPLDA baselines.

Both the NPLDA [79] and E2E-NPLDA models used the soft detection cost function (Soft-DCF) for optimization, whereas the BCE loss was used for the DPLDA baseline.

The detection error trade-off (DET) plots for the methods based on the FTDNN x-vectors are shown in Figure 4.2.

The results using both the embedding extractors show that, amongst the baseline systems, the GPLDA and DPLDA models perform better than the GB model. With the ETDNN x-vectors, the GPLDA model fares better than the DPLDA on the SITW, VOiCES, and SRE18 Dev (VAST) datasets. However, with the FTDNN x-vectors, the GPLDA model is better than DPLDA only for SITW and VOiCES Dev datasets. We report results for the NPLDA model initialized with the respective GPLDA model and two E2E-NPLDA models initialized with the respective GPLDA and NPLDA models. We consistently observe that the proposed E2E-NPLDA model provides significant improvements over the baseline systems.

The results using the E-TDNN model as the embedding extractor are reported in Table 4.2 (a). We observe the NPLDA models show relative improvements over the GPLDA model in the range of $14\% - 31\%$ in terms of C_{Min} and $12\% - 30\%$ in terms of EER. The E2E-NPLDA models show relative improvements over the GPLDA model in the range of $18\% - 35\%$ in terms of C_{Min} and $11\% - 35\%$ in terms of EER. In terms of relative improvements over the NPLDA model, the E2E-NPLDA improves by $0 - 20\%$ for the C_{Min} metric using the E-TDNN model and by $3 - 19$ % for the C_{Min} metric using the F-TDNN model. In terms of EER metric, the relative improvements over the NPLDA model for the E2E-NPLDA model range from -13 to +19% in the FTDNN architecture. The relative degradation in EER may potentially due to the optimization targeting the C_{Min} metric in the proposed approach.

The results are consistent when we use the F-TDNN model (Table 4.2 (b)) as the embedding extractor with relative gains for the NPLDA model over the GPLDA model in the range of $12\% - 35\%$ and $14\% - 34\%$ in terms of C_{Min} and EER, respectively. The E2E-NPLDA model shows gains over the GPLDA model in the range of $19\% - 50\%$ in terms of C_{Min} and $17\% - 38\%$ in terms of EER.

4.6 Discussion

4.6.1 Influence of training utterance duration

As discussed in Section 4.4.2.1, we create a modified dataset by splitting longer utterances into 5,10, and 20 second segments (2000 frames) after voice activity detection (VAD) and

Table 4.3: Performance on the evaluation datasets for different choice of training utterance duration. Here, chunks refer to 5,10 and 20 second segments

Model	Dur. of Utts.	SITW-E		VOiCES-D		VOiCES-E		SRE18 D-V		SRE18 E-V	
		EER	C_{Min}	EER	C_{Min}	EER	C_{Min}	EER	C_{Min}	EER	C_{Min}
GPLDA	Full	2.40	0.256	2.08	0.257	5.91	0.480	7.41	0.486	15.24	0.597
GPLDA	Chunks	2.68	0.276	2.15	0.268	6.01	0.490	7.41	0.449	15.82	0.610
NPLDA	Full	1.75	0.171	1.49	0.167	5.36	0.442	7.41	0.374	12.38	0.498
NPLDA	Chunks	1.78	0.178	1.49	0.176	5.18	0.432	6.17	0.337	12.38	0.467

mean normalization. We compare the performances of the models on the modified dataset and the original one. The results are reported in Table 4.3. We observe that the performance of the systems are quite comparable for training with full duration utterances and with $5 - 20$ second segments. While a slight degradation in EER results is observed, an improvement in performance in terms of detection cost function C_{Min} is obtained for both the GPLDA model as well as the proposed NPLDA model. These results allow us to proceed using short segments for the E2E-NPLDA model. All subsequent results are reported using $5 - 20$ second segments for the training of GPLDA, NPLDA, GB, and E2E-NPLDA models.

4.6.2 Comparison of data sampling algorithms for NPLDA

The data sampling methods are discussed in Sec. 4.4.2.2. The performance comparison of the two sampling techniques with NPLDA models trained on the SRE18 Evaluation dataset can be seen in Table 4.4. Although the statistics of batch-wise trials have changed significantly (Algo. 2), we see that its performance is better than our previous sampling method (Algo. 1) [79]. The comparable performance of (Algo. 2) allows us to use this method for E2E-NPLDA model as this is not memory intensive.

4.6.3 Comparison of loss functions for NPLDA

We discuss our experiments using the NPLDA model comparing the different loss functions discussed in Section 4.3. We perform similar pre-processing steps as discussed in the previous

Table 4.4: Performance on the evaluation datasets for NPLDA model with different data sampling strategies.

Model	Sampling Algorithm	SITW-E		VOiCES-D		VOiCES-E		SRE18 D-V		SRE18 E-V	
		EER	C_{Min}	EER	C_{Min}	EER	C_{Min}	EER	C_{Min}	EER	C_{Min}
NPLDA	Algo 1	1.89	0.191	1.65	0.205	5.41	0.498	7.41	0.337	13.33	0.504
	Algo 2	1.78	0.178	1.49	0.176	5.18	0.432	6.17	0.337	12.38	0.467

Table 4.5: Comparison of NPLDA models for various loss functions

Embedding Extractor	Init.	Loss Fn.	SITW-E		VOiCES-D		VOiCES-E		SRE18 D-V		SRE18 E-V	
			EER	C_{Min}	EER	C_{Min}	EER	C_{Min}	EER	C_{Min}	EER	C_{Min}
ETDNN	GPLDA	BCE [93]	2.11	0.237	2.52	0.298	6.93	0.585	7.41	0.379	13.02	0.542
		Cllr [54]	2.08	0.241	2.56	0.318	7.30	0.64	8.64	0.449	13.85	0.616
		WCllr [98]	2.19	0.223	2.21	0.26	6.67	0.498	7.41	0.416	**12.38**	0.513
		SoftDCF	2.10	0.209	1.79	0.223	5.69	0.450	7.41	0.377	12.55	0.533
FTDNN	GPLDA	BCE [93]	1.89	0.199	2.10	0.229	6.40	0.538	10.70	**0.337**	**12.38**	0.545
		Cllr [54]	1.95	0.214	2.06	0.246	6.71	0.605	11.11	0.370	13.02	0.561
		WCllr [98]	1.94	0.195	1.95	0.214	5.80	0.474	9.88	0.374	**12.38**	0.500
		SoftDCF	**1.78**	**0.178**	**1.49**	**0.176**	**5.18**	**0.432**	**6.17**	**0.337**	**12.38**	**0.467**

subsections and report the results for BCE [93], Cllr [54], WCllr [98], and our proposed SoftDCF cost functions. The experiments are performed for the two embedding extractor network models: The E-TDNN and F-TDNN. The NPLDA models are initialized with the respective GPLDA models. These results are reported in Table 4.5.

We observe that the proposed softDCF loss performs significantly better than the other cost functions for both the embedding extractor network models for all the datasets.

4.6.4 Comparison of Initialization Methods

It is evident from Table. 4.2 that in most cases, the E2E-NPLDA model performs better when its back-end is initialized with the pre-trained NPLDA parameters than the GPLDA parameters. Moreover, there is a significant difference induced due to the difference in initialization itself. When the NPLDA/E2E-NPLDA parameters are initialized randomly and optimized with the softDCF loss, it quickly converges to a saddle point where the $C_{Min} = 1$, and the EER is around 50%. This is merely chance-level performance which implies that the models are not learning

Table 4.6: Comparison of NPLDA models for different initialization choices using the BCE loss.

Model	Embedding Extractor	Init.	SITW-E		VOiCES-D		VOiCES-E		SRE18 D-V		SRE18 E-V	
			EER	C_{Min}	EER	C_{Min}	EER	C_{Min}	EER	C_{Min}	EER	C_{Min}
NPLDA	ETDNN	Random	3.53	0.362	4.11	0.466	8.96	0.680	11.11	0.486	14.98	0.598
NPLDA	ETDNN	GPLDA	2.11	0.237	2.52	0.298	6.93	0.585	**7.41**	0.379	13.02	**0.542**
NPLDA	FTDNN	Random	2.81	0.286	3.42	0.412	8.14	0.667	11.11	0.416	13.33	0.558
NPLDA	FTDNN	GPLDA	**1.89**	**0.199**	**2.10**	**0.229**	**6.40**	**0.538**	10.70	**0.337**	**12.38**	0.545

anything meaningful. However, using the other loss functions, such as the BCE and WCllr, the models can be trained successfully with randomly initialized parameters. We explain the reason for this stark difference between the two loss functions in the following section (4.6.5). In this section, we present a discussion on the effect of initialization for the cases where meaningful improvements are achieved.

We performed experiments using the NPLDA model with different initializations for both the ETDNN and FTDNN embedding extractors with the BCE loss. The results for these experiments are reported in Table 4.6. We report the results for the GPLDA model along with two NPLDA models, one initialized randomly and the other with the GPLDA model for the respective embedding extractor. We observe that the NPLDA model initialized with the GPLDA parameters performs significantly better than with random initialization. The trend is similar for models using embeddings from ETDNN/FTDNN models. The initialization with a generative model boosts the performance. This also illustrates that initialization is a crucial step in verification style problems using deep learning.

The performance of the proposed ASV system on the SITW task is further analyzed as a function of the training epoch in Fig. 4.3. In Fig. 4.3(a), we compare the performance of the NPLDA model for different initialization choices (random versus GPLDA model). The GPLDA initialization leads to faster learning and convergence to better local minima. Further, the use of the SoftDCF cost function improves the performance of the NPLDA model, where the C_{Min} value achieved is always better than GPLDA model initialization. The comparison of

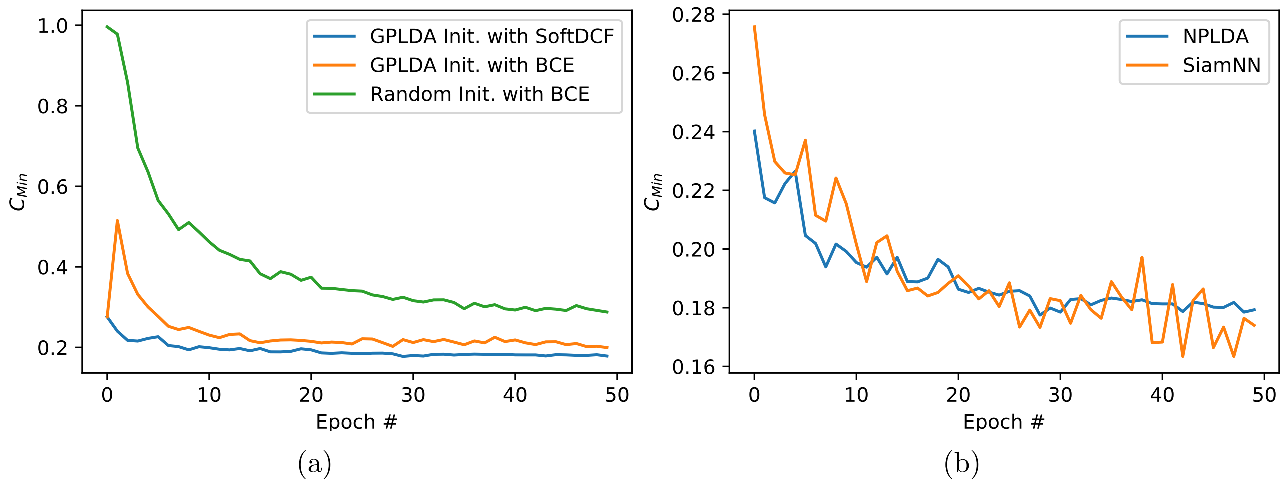


Figure 4.3: Plots of C_{min} vs training epochs on SITW evaluation set for various models and loss functions: (a) NPLDA models trained with softDCF and BCE for two types of initialization - GPLDA initialization or random initialization, (b) NPLDA vs E2E-NPLDA using soft detection loss with GPLDA initialization

NPLDA training and the E2E-NPLDA training using the same GPLDA initialization is shown in Fig. 4.3(b). The E2E-NPLDA model has more learning capacity as the entire model is trained in this case. This results in improved C_{Min} value at end of 50 epochs of training. The stopping criterion was based on the VoxCeleb 1 validation data. The network is not optimized on the SITW eval set. The model weights stored after every epoch are used to generate the plot of C_{min} vs epochs on an unseen SITW eval set in Fig. 4.3(b).

4.6.5 A comparative analysis between SoftDCF and WCllr losses

In this section, we perform a comparative analysis of the SoftDCF and the WCllr loss functions with the intent of understanding how they fundamentally differ in their role of optimizing the model performance. To do this, we express them in a comparable format and highlight some similarities and differences.

Recall that the softDCF loss is defined as:

$$softDCF = \frac{1}{|\mathcal{T}|}\sum_{i\in\mathcal{T}}[1-\sigma(\alpha(s_i-\theta))] + \frac{\beta}{|\mathcal{N}|}\sum_{i\in\mathcal{N}}\sigma(\alpha(s_i-\theta)) \tag{4.16}$$

Recall that the WCllr loss function can be expressed as a special case of the generalized BCE function (Eq. 4.9) as follows:

$$C_{llr}^{(w)} = \frac{\pi C_{Miss}}{|\mathcal{T}|} \sum_{i \in \mathcal{T}} \log(1 + e^{-(s_i - \theta)}) + \frac{(1-\pi) C_{FA}}{|\mathcal{N}|} \sum_{i \in \mathcal{N}} \log(1 + e^{(s_i - \theta)}) \tag{4.17}$$

which can be normalized by πC_{Miss} and re-written as

$$C_{llr}^{(w)} = -\frac{1}{|\mathcal{T}|} \sum_{i \in \mathcal{T}} \log(\sigma(s_i - \theta)) - \frac{\beta}{|\mathcal{N}|} \sum_{i \in \mathcal{N}} \log(1 - \sigma(s_i - \theta)) \tag{4.18}$$

Here, s_i is the system output for a trial (score), θ is the decision threshold[1] applied to s_i, α is the sigmoid warping factor, $\mathcal{T}$ denotes the set of target trials and $\mathcal{N}$, the non-target trials.

Now, equations 4.16 and 4.18 are in a comparable format, involving sums over different functions of the target and non-target scores output by the system, which can be written in a generalized format as follows:

$$L = \sum_{i \in \mathcal{T}} F(s_i) + \sum_{i \in \mathcal{T}} G(s_i) \tag{4.19}$$

For the softDCF loss,

$$F(s_i) = \frac{1}{|\mathcal{T}|} [1 - \sigma(\alpha(s_i - \theta))] \qquad G(s_i) = \frac{\beta}{|\mathcal{N}|} \sigma(\alpha(s_i - \theta)) \tag{4.20}$$

and for the WCllr loss:

$$F(s_i) = -\frac{1}{|\mathcal{T}|} \log(\sigma(s_i - \theta)) \qquad G(s_i) = -\frac{\beta}{|\mathcal{N}|} \log(1 - \sigma(s_i - \theta)) \tag{4.21}$$

[1]Note that in our softDCF loss definition, θ is a model parameter. On the other hand, the WCllr loss computes a term $l_i = as_i + b$, interpreted as a calibrated log-likelihood ratio, to which a fixed threshold of $\log \beta$ is applied. Here, a and b are learnable parameters [54, 98]. For the purpose of comparison with softDCF, we can assume $a = 1$ to be fixed, and consider $\theta = -b + \log \beta$ as the *threshold* applied over the raw scores s_i.

Note that both these functions penalize both types of errors (miss and false alarms). The function F returns a *high* value if a target score is lesser than the decision threshold, thereby penalizing false negatives (miss), and a low value otherwise. Similarly, the function G returns a high value if a non-target score is greater than the threshold, penalizing the false alarms, and a low value otherwise. Both functions considered here will achieve a desirable minimum value if the target scores predominantly lie above the threshold, and almost all the non-target scores lie below the threshold. By analyzing the derivatives, we describe how each of these cost functions differ, when used in the back-propagation algorithm.

Differentiating equation 4.19 with respect to the network parameters Λ, we get

$$\frac{\partial L}{\partial \Lambda} = \sum_{i \in \mathcal{T}} \frac{\partial F(s_i)}{\partial s_i} \frac{\partial s_i}{\partial \Lambda} + \sum_{i \in \mathcal{N}} \frac{\partial G(s_i)}{\partial s_i} \frac{\partial s_i}{\partial \Lambda} \tag{4.22}$$

Denoting $\frac{\partial F(s_i)}{\partial s_i}$ as $f(s_i)$ and $\frac{\partial G(s_i)}{\partial s_i}$ as $g(s_i)$ and rewriting the equation, we get:

$$\frac{\partial L}{\partial \Lambda} = \sum_{i \in \mathcal{T}} f(s_i) \frac{\partial s_i}{\partial \Lambda} + \sum_{i \in \mathcal{N}} g(s_i) \frac{\partial s_i}{\partial \Lambda} \tag{4.23}$$

Equation 4.23 is a weighted sum of the gradient of the network outputs w.r.t the parameters. These weights are a function of the output s_i, and the ground truth. The "gradient descent" update is given by:

$$\Lambda^{(new)} \longleftarrow \Lambda^{(old)} - \eta \frac{\partial L}{\partial \Lambda} \tag{4.24}$$

where η is a sufficiently small enough positive learning rate. Substituting for $\frac{\partial L}{\partial \Lambda}$, we can write

$$\Lambda^{(new)} \longleftarrow \Lambda^{(old)} - \left(\sum_{i \in \mathcal{T}} \eta f(s_i) \frac{\partial s_i}{\partial \Lambda} + \sum_{i \in \mathcal{N}} \eta g(s_i) \frac{\partial s_i}{\partial \Lambda} \right) \tag{4.25}$$

The gradient descent algorithm has to effectively cause the target scores to increase above the threshold, and the non-target scores to decrease below the threshold. For this to happen, $f(s_i)$ has to be non-positive and $g(s_i)$ should be non-negative. By examining how $f(s_i)$ and $g(s_i)$ vary with the network output s_i, for both the loss functions, we can comment on how different they are from each other in achieving the objective of separating the target and non-target distributions.

For the WCllr loss, the functions F, G, and their respective derivatives f and g are:

$$F(s_i) = -\frac{1}{|\mathcal{T}|}\log(\sigma(s_i - \theta)) \qquad G(s_i) = -\frac{\beta}{|\mathcal{N}|}\log(1 - \sigma(s_i - \theta)) \tag{4.26}$$

$$f(s_i) = -\frac{1}{|\mathcal{T}|}\left[1 - \sigma(s_i - \theta)\right] \qquad g(s_i) = \frac{\beta}{|\mathcal{N}|}\sigma(s_i - \theta) \tag{4.27}$$

The plots of $f(s_i)$ and $g(s_i)$ are shown in Figure 4.4. The WCllr loss function satisfies the above-mentioned criterion that $f(s_i)$ should be non-positive and $g(s_i)$ should be non-negative. The function $f(s_i)$ has a value of $-\frac{0.5}{|\mathcal{T}|}$ at the threshold, and quickly converges to $-\frac{1}{|\mathcal{T}|}$ asymptotically as s_i reduces below the threshold, and quickly converges to 0 asymptotically as s_i increases above the threshold. Similarly, $g(s_i)$ has a value of $\frac{0.5\beta}{|\mathcal{N}|}$ at the threshold, and quickly converges to $\frac{\beta}{|\mathcal{N}|}$ asymptotically as s_i increases above the threshold, and quickly converges to 0 asymptotically as s_i decreases below the threshold.

This indicates that the contributors to the total gradients being back-propagated are from the target training examples whose outputs lie below the threshold (miss), and the non-target training examples with output scores predominantly above the threshold (false alarms).

On the contrary, in the case of the softDCF loss, we have

$$F(s_i) = \frac{1}{|\mathcal{T}|}\left[1 - \sigma(\alpha(s_i - \theta))\right] \qquad G(s_i) = \frac{\beta}{|\mathcal{N}|}\sigma(\alpha(s_i - \theta)) \tag{4.28}$$

$$f(s_i) = -\frac{\alpha}{|\mathcal{T}|}\sigma(\alpha(s_i - \theta))\left[1 - \sigma(\alpha(s_i - \theta))\right] \qquad g(s_i) = \frac{\alpha\beta}{|\mathcal{N}|}\sigma(\alpha(s_i - \theta))\left[1 - \sigma(\alpha(s_i - \theta))\right] \tag{4.29}$$

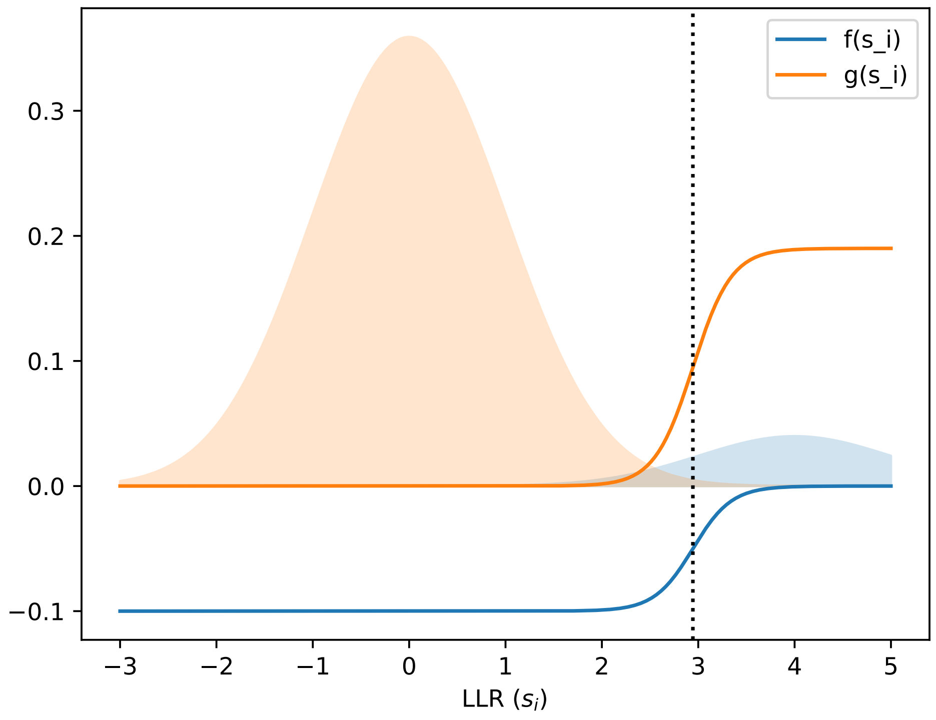


Figure 4.4: The plots of $f(s_i)$ and $g(s_i)$ for the WCllr loss. The vertical dotted black line is used to denote the threshold θ. For the purpose of illustration, these plots are generated considering $C_{FA} = C_{Miss} = 1$, $\pi = 0.05$ as the evaluation parameters. Further, we assume a batch size of 100 with $|\mathcal{T}| = 10$ and $|\mathcal{N}| = 90$. We also show the non-target score distribution in orange(on the left) and the target score distribution in blue (on the right).

As illustrated in Fig. 4.5, the functions $f(s_i)$ and $g(s_i)$ are bell-shaped curves around the threshold, whose width and height are controlled by the warping factor α. This means only the training examples whose output scores are within the support of the bell shaped curves (between 2 and 4 in the illustrated plot) are the ones that contribute to the total gradients, and hence, to the parameter updates. The magnitude of gradients contributed by training examples whose outputs are close to the threshold is smaller in the case of WCllr, as compared to the softDCF loss.

To summarize, the gradient update rule using WCllr loss updates the parameters in a direction that tries to force *all* the scores of the target training examples below the threshold to

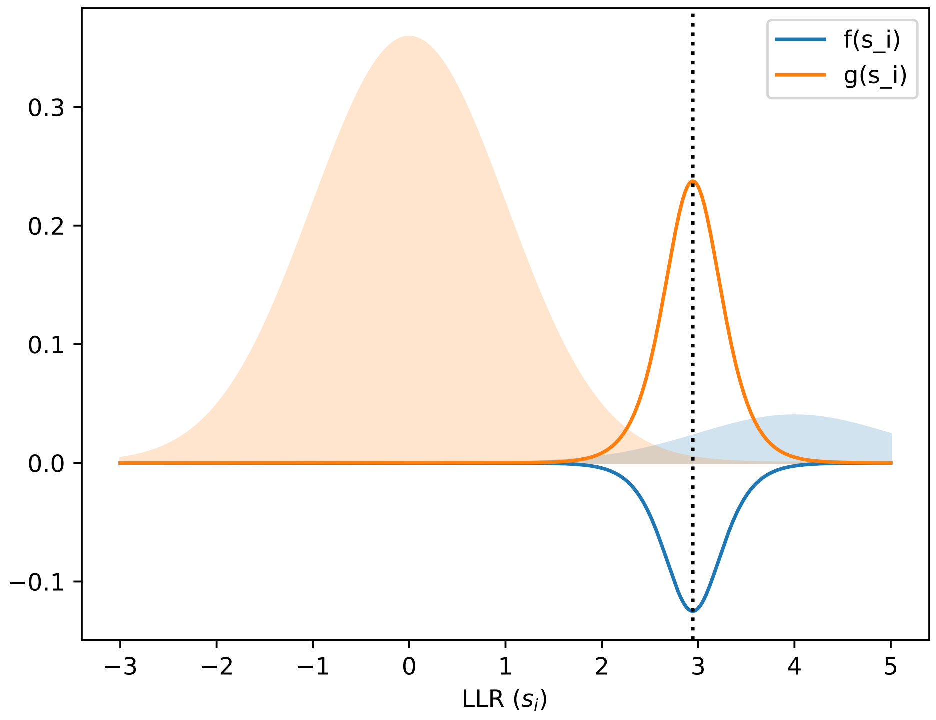


Figure 4.5: The plots of $f(s_i)$ and $g(s_i)$ for the softDCF loss. The vertical dotted black line is used to denote the threshold θ. For the purpose of illustration, these plots are generated considering $C_{FA} = C_{Miss} = 1$, $\pi = 0.05$ as the evaluation parameters. Further, we assume a batch size of 100 with $|\mathcal{T}| = 10$ and $|\mathcal{N}| = 90$. We also show the non-target score distribution in orange(on the left) and the target score distribution in blue (on the right).

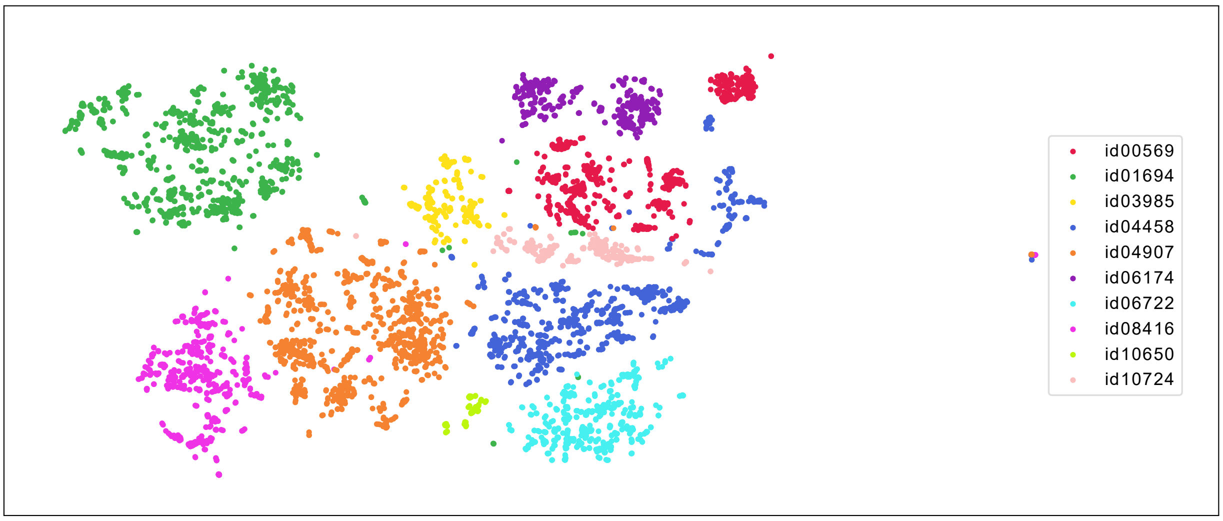
increase, and *all* the scores of the non-target training examples above the threshold to decrease. This happens repeatedly, batch after batch and epoch after epoch, eventually causing almost all the target and non-target score distributions to distinctively separate out at the threshold. On the other hand, the gradient update rule using softDCF loss updates the parameters in a direction such that the target scores in a small neighbourhood around the threshold are encouraged to increase, and the non-target scores in the same region to decrease. This causes the errors in a small neighborhood around the threshold to be corrected, while giving up on *larger* errors.

This observation sheds light on the question as to why the softDCF loss does not work when the parameters are initialized randomly. In the randomly initialized state, the model does not have any ability to distinguish a target trial from a non-target trial. If we evaluate such a model, we would only get chance-level performance. The target and non-target score distributions would be highly overlapping with each other, and not well separated like that of a trained model. They would also have a larger loss with large gradients compared to a pre-trained network. In such cases, as the P_{FA} term has a higher weight (by a factor of β), all the scores (target and non-target) are found to quickly move to the left side of the bell curve where $P_{FA}^{(soft)} \approx 0$ and $P_{miss}^{(soft)} \approx 1$. This is a saddle point of the softDCF loss, where the gradient magnitudes are approximately zero, and does not ensure that the target and non-target distributions are separated. Thus, most of the samples end up outside the support of the bell-shaped $f(s_i)$ or $g(s_i)$ curves. Hence, when the network is randomly initialized, the network converges to a saddle point where the target and non-target distributions are highly overlapping.

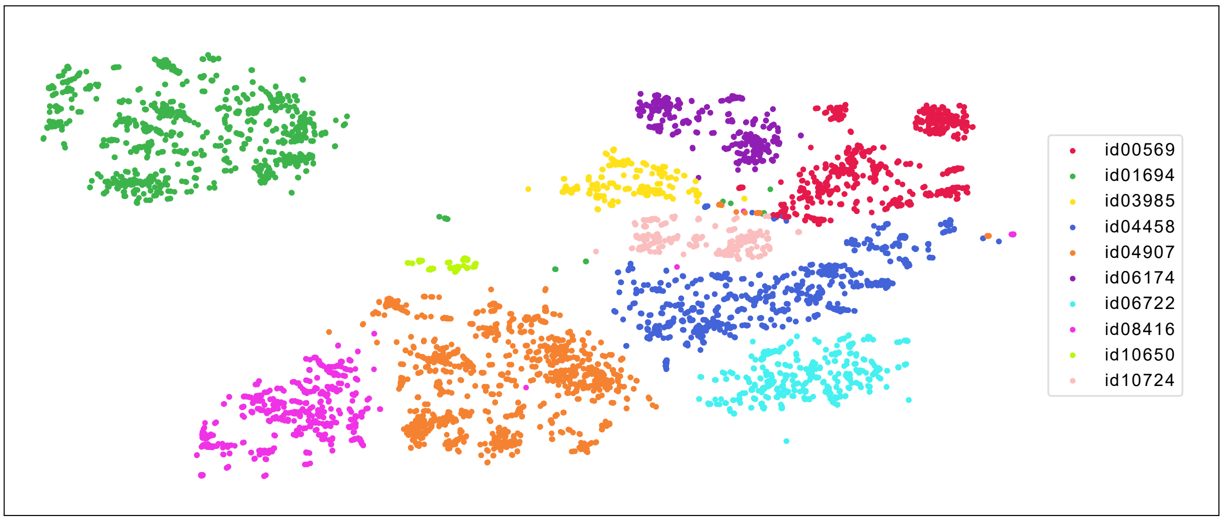
This analysis highlights how the softDCF loss complements the BCE or WCllr loss functions for the task of speaker verification. It also explains the need to initialize a verification network with pre-trained model parameters such as the GPLDA, for the softDCF loss to work. As the training loss converges, the WCllr loss keeps trying to correct *larger* errors, whereas the softDCF loss focuses on correcting minor errors where the scores are very close to the threshold, while giving up on larger errors. As our experiments have shown that the softDCF performs slightly better than WCllr, we hypothesize that this may be because trying to correct larger errors (outliers and mislabeled training examples) may introduce more errors, but it takes a smaller effort to try correcting errors that are only in a small neighbourhood of the threshold, without introducing new errors.

4.6.6 Visualization of x-vectors using tSNE

We visualize the representations using the t-distributed stochastic neighborhood embedding (tSNE) [74]. The tSNE is an unsupervised dimensionality reduction method that preserves the local neighborhood of the data space in the lower-dimensional subspace. We performed tSNE



(a) Baseline X-vectors (Kaldi)



(b) E2E-NPLDA X-vectors

Figure 4.6: The tSNE visualization of 10 randomly selected female speakers from the Voxceleb development dataset for baseline x-vector embeddings and embeddings from layer-12 of the proposed E2E-NPLDA system. The F-ratio of the 2-D t-SNE embeddings for the x-vector extractor and the E2E-NPLDA system are 8.7 and 15.4 respectively.

dimensionality reduction to two dimensions on the baseline x-vectors, and the embeddings from layer-10 of the E2E-NPLDA system at the same processing step. In Fig. 4.6, we show the tSNE visualization of embeddings from 10 female speakers selected from the VoxCeleb development dataset. While the difference may not be apparent at the first glance, we can see that the speaker clusters appear to be better separated at the boundaries.

A quantitative measure of the improvement in the separability is obtained by measuring the F-ratio of the 2-D tSNE embeddings in both these cases. F-ratio is defined as the ratio of the between-class (speaker) variance to the within-class (speaker) variance. For the baseline x-vector embeddings, we observe an F-ratio of 8.7 and for the embeddings from the corresponding layer in the E2E-NPLDA network, we observe an F-ratio of 15.4. The improved separability obtained by the proposed system may be significant when the embeddings from the E2E-NPLDA model are extended for tasks like speaker diarization.

4.6.7 Comparison with related works

In this section, we train the models only on VoxCeleb-2 data and report the results on the original Voxceleb-1 test set, in order to enable the comparison with other works reported in the literature. The work by Nagrani et. al. [109] makes a detailed comparison of models trained with various architectures, temporal pooling/aggregation methods, loss functions, and scoring methods. Further, Chung et.al. [97] explored a range of metric learning and classification objectives by conducting careful experiments worth over 20000 GPU hours and consolidated their results. We highlight the best results corresponding to each architecture reported in these works and compare them with our proposed model (E2E-NPLDA).

For these experiments, we retrain our models using only the Voxceleb2 (development) portion containing 5994 speakers for training, as done in prior works. The other results reported in this table use models based on residual networks and with various loss function choices. We test the publicly released model trained with the angular-prototypical loss [97] on the Voxceleb1-test, SITW and the VOiCES evaluation trials[1]. We also extract the embeddings from the ResNet model and process them with centering and LDA. Further, we train the GPLDA/NPLDA backends. The results are reported in Table 4.7. While the GPLDA model trained on the ResNet embeddings performs worse than the cosine scoring approach, the NPLDA model on the ResNet

[1]The trained model was downloaded from http://www.robots.ox.ac.uk/~joon/data/baseline_lite_ap.model

Table 4.7: Comparison of the results using the FTDNN models trained on VoxCeleb-2 dataset. The models are tested with VoxCeleb-1 original test set and other evaluation sets. The table also provides comparison with other published works.

Model	Training loss	Back End	Vox1 Test		SITW E		Voices E	
			EER	C_{min}	EER	C_{min}	EER	C_{min}
FTDNN Xvector	Softmax	GPLDA	2.67	0.18	3.2	0.21	6.68	0.38
E2E-NPLDA	Softmax	NPLDA	**2.27**	**0.14**	**2.65**	**0.16**	6.61	**0.35**
VGG-M [35]	Softmax+contrastive	-	5.94	-	-	-	-	-
Resnet34 [35]	Softmax+contrastive	-	4.83	-	-	-	-	-
Resnet50 [35]	Softmax+contrastive	-	3.95	-	-	-	-	-
ResNet34L [97]	Angular	Cosine	2.39	0.17	3.55	0.25	11.1	0.71
ResNet34L [97]	Angular	GPLDA	3.58	0.22	4.16	0.27	10.22	0.54
ResNet34L [97]	Angular	NPLDA	2.46	0.17	3.06	0.21	8.58	0.47

embeddings gives an improvement in performance over the cosine scoring rule (particularly on the VOiCES dataset).

Comparing the prior works with our proposed E2E-NPLDA model (Table 4.7), the E2E-NPLDA results improve over the other architectures and loss functions on all the tasks considered (Vox1 test data, SITW, and VOiCES). These results indicate that the joint optimization of all the modules in a speaker verification model using the soft detection loss function in a Siamese style neural network achieves considerable improvements in some of the challenging speaker verification datasets.

4.7 Chapter Summary

This chapter gave a detailed account of a novel approach to speaker verification back-end modeling by posing the generative PLDA modeling framework as neural layers. The proposed neural PLDA (NPLDA) model is optimized directly for a verification cost function which approximates the detection cost function, which is the primary metric used in speaker verification evaluation. The NPLDA model allows the direct integration of the neural embedding extractor as well as the pre-processing steps in a single neural pipeline, referred to as the E2E-NPLDA. The E2E-

NPLDA system provides significant performance improvements over the state-of-art speaker verification systems for a variety of datasets like SITW, VOiCES, and NIST SRE datasets. We have also compared various considerations like initialization, loss function, model architecture for the embedding extractor, and data sampling strategies. The additional analysis shows that the performance benefits are also achieved with improved computational efficiency. Further, the embeddings from the E2E-NPLDA model show improved speaker discrimination which may be beneficial for tasks like speaker diarization.

Chapter 5

Summary and Future Perspectives

Developing algorithms that automatically infer the speaker, language, or accent from a given segment of speech are challenging tasks that have been researched for more than three decades. Researchers in the field have explored several approaches in signal processing, pattern recognition, and machine learning. Speaker and language recognition have a variety of applications in call centers, AI-based helplines, voice assistants, robotics, and also in security and defense applications. Moreover, to make speech technology like conversational AI accessible to every corner of the world, we need to explore methods to identify uncommon languages and dialects using limited training resources. It is also required to build systems that are robust to artifacts such as background noise and reverberation. The main theme of this doctoral research was to explore the shortcomings of existing approaches to these challenges, and explore novel directions to tackle these shortcomings for developing robust speaker and language recognition systems in a supervised setting.

Speaker and Language Recognition can be broadly classified into two types:

1. **Detection or Verification problem:** Is speaker X (or Language Y) present in a given segment of speech?

2. **Classification problem:** Who among a given set of speakers is speaking? Which

among a given set of languages is being spoken?

While the classification problem assumes that the test recording belongs to exactly one of the N classes, the detection or verification problem neither assumes that there are a fixed number of classes, nor that the test recording belongs to exactly one speaker or language class. It is important to note that for speaker verification, an enrollment utterance is used to obtain a model of the target speaker to be detected or verified. Since the task is to say whether a given test segment is from the enrollment speaker or not, it can also be viewed as the task of identifying whether a *pair* of speech segments are from the same speaker or not. The detection/verification systems are evaluated by metrics such as equal error rates (EER) and the Bayesian detection cost function (DCF, minDCF) [54]. These metrics are widely used for speaker and language recognition as they account for biases and imbalances in the test datasets, which make them excellent indicators of the discriminative ability of the speaker and language recognition systems.

In this chapter, we summarize the key contributions of the thesis, discuss some limitations of our approaches, and identify some important unexplored directions that can address the drawbacks of the existing methods.

5.1 Key contributions of the thesis

We have explored two major directions in this thesis. In the first part of our research, we developed a supervised version of a popular i-vector approach, which we call as s-vector. In the second part of the research, we proposed a neural network version of a popular generative back-end model for speaker verification called Probabilistic Linear Discriminant Analysis (PLDA). We call this Neural PLDA or NPLDA. Furthermore, we extended the neural network back-end to an end-to-end approach, where a Siamese neural-network based front-end and the back-end are jointly trained (E2E-NPLDA).

The two major directions pursued in this thesis are linked by the applications they can be used in, that is, for speaker and language recognition. Additionally, both the directions pursued are based on highly related factor analysis frameworks - the i-vector and PLDA. It has been a standard practice to use the approaches for speaker recognition to language recognition, and vice-versa, and such approaches have shown to be very effective. The approaches explored in the first part of the thesis are more suitable for language recognition, and the approaches explored in the second part are more suitable for speaker verification.

In the following subsections, we give a brief summary of each of the directions pursued.

5.1.1 Supervised i-vector approach for Language Recognition

In the last decade, one of the most popular approaches to language (accent) and speaker recognition consisted of modeling a database of speech recordings (in the form of a sequence of short-term feature vectors) with a Gaussian Mixture Model - Universal Background Model (GMM-UBM) [14, 15]. Given a speech segment from a particular speaker or language, the GMM-UBM mean components can be modified to fit the speech segment using maximum aposteriori (MAP) adaptation. The deviation in the mean components is captured in a lower dimensional latent space called the i-vector space [11, 27]. The parameters in this model were derived using a maximum likelihood (ML) framework with an iterative expectation maximization (EM) approach. The latent vector is assumed to have a standard normal prior (having zero mean and identity covariance). The MAP estimate of the latent vector given a speech segment is called the i-vector. These i-vectors serve as front-end embeddings for both speaker and language recognition.

The i-vector model is an unsupervised generative model, as it doesn't require any speaker or language information to train. In chapter 3, we discussed our fully supervised version of the i-vector model that we published in [63, 64], where each label class is associated with a Gaussian prior with a class-specific mean parameter. The joint prior on the latent variable becomes a GMM. The choice of prior is motivated by the Gaussian back-end approach [56], where the conventional i-vectors for each language are modeled with a single Gaussian distribution. Hence, introducing a GMM prior made the assumptions consistent with the back-end, which we hypothesized as one of the main reasons for the proposed model to improve the performance significantly.

We gave a detailed derivation of the EM algorithm to estimate the model parameters, and discussed various approaches to extracting the s-vector embeddings in section 3.3. We performed language recognition experiments on the NIST Language Recognition Evaluation (LRE) 2017 challenge dataset [58], which has test segments ranging from 3 to 30 seconds. With the s-vector framework, we observe relative improvements between 8% to 20% in terms of the Bayesian detection cost function, 4% to 24% in terms of EER, and 9% to 18% in terms of classification accuracy over the conventional i-vector framework. While the performance improved for all the durations, more improvements were observed for shorter duration test cases. We also perform language recognition experiments showing similar improvements on the RATS dataset and Mozilla CommonVoice dataset. All the language and accent recognition experiments are reported in section 3.4.

We concluded the chapter with a detailed discussion of the proposed approach in section 3.5. With detailed data analysis and visualization, we showed that the s-vector features yield representations succinctly capture the language (accent) label information, and do a significantly better job distinguishing the various accents of the same language. To test our approach on speaker classification, we also performed experiments using the LibriSpeech dataset, which we reported in section 3.5.7.

5.1.2 Supervised Neural Network models for Speaker Verification

In the second part of this thesis (Chapter 4), we discussed our research on speaker verification. The state-of-art approach for speaker verification consisted of a neural network based embedding extractor called x-vectors [32] and a back-end model such as the Probabilistic Linear Discriminant Analysis (PLDA) [41] to compute the log-likelihood ratio scores.

In this part of our research, we began by proposing a neural network approach for back-end modeling, based on the PLDA approach. The likelihood ratio score of the generative PLDA model was posed as a discriminative similarity function and the learnable parameters of the score function were optimized using a verification cost. The proposed model, termed as neural PLDA (NPLDA), was initialized using the generative PLDA model parameters. An approximation of the minimum detection cost function (DCF) was proposed to be used as the loss function to train the NPLDA model parameters. This was discussed in sections 4.4 and 4.4.1. The speaker recognition experiments using the NPLDA model were performed on the speaker verification task in the VOiCES datasets as well as the SITW challenge dataset.

Further, we explored a fully end-to-end neural network approach where the model outputs the verification score directly, given the acoustic feature inputs. This Siamese neural network (E2E-NPLDA) model combines the embedding extraction and back-end modeling into a single processing pipeline. The development of the single neural Siamese model allows the joint optimization of all the modules using a verification cost. As these models had to be trained on pairs of speech segments, the approaches we used to sample trials to train the NPLDA back-end model was no longer feasible in the end-to-end setting, as they would easily exhaust the GPU memory. Hence, we explored low-memory trial sampling methods, where many utterances were repeated within a batch. This was explained in detail in section 4.4.2.

Several speaker recognition experiments were performed using Speakers in the Wild (SITW), VOiCES, and NIST SRE datasets, where the proposed NPLDA and E2E-NPLDA models were shown to improve significantly over the x-vector PLDA baseline system (relative improvements of up to 35 % in the primary cost metric). In particular, we showed that the proposed soft detection cost function based optimization improves significantly over other loss functions considered on challenging test conditions, indicating robustness. The details of experiments and results are provided in section 4.5.

We concluded the chapter with a detailed discussion in section 4.6, analyzing the influence of training duration, providing comparative analysis of the data sampling approaches, loss functions and initialization methods. To understand the results better, we visualized the E2E-NPLDA embeddings and compared it with the baseline x-vector embedding using t-SNE plots. With the help of equations and plots, we analyzed the differences between our proposed softDCF loss and the BCE loss functions. We showed that the two loss functions complement each other for the task of speaker verification.

5.2 Limitations of the work

Although it is possible to use the s-vector approaches for speaker recognition, and the Siamese neural network approaches for language recognition, there are certain inherent aspects of speaker and language itself, that limit the extent to which these approaches can be shared between speaker and language embedding models. For example, speaker verification is typically an out-of-set problem, where an enrollment speaker during test time is typically not a part of the training data, and the enrollment duration is typically less than a minute. However, in language/accent recognition, the verification/detection problem is typically a closed-set problem, where the entire training data corresponding to the target language is typically used for enrollment. It is not easy to capture the language identity of unseen languages using short audio segments. This limits the extent to which Siamese neural network approaches are used for language detection. Similarly, the sheer diversity and the huge population of the world make it challenging to employ the s-vector approach for speaker recognition in a scalable way.

We mention the drawbacks of the two major directions pursued in the following subsections.

5.2.1 Supervised i-vector approach for Language Recognition

One major drawback of the proposed s-vector approach is the limit on number of labels in the embedding extraction. For language recognition tasks, the number of class labels are typically small thereby allowing the modeling of each language class with a GMM component. In tasks such as speaker recognition, where the i-vector approaches are dominantly used, the number of class labels (speakers) can become significantly high. The i-vector computation involves a $N \times N$ matrix inversion operation, where N is the i-vector dimension. The s-vector computation involves L such matrix inversion operations, where L is the number of class labels. Hence the memory requirements and computation time increases linearly with the number of labels considered. For tasks like speaker recognition, it is common to have a few thousand speaker labels. In such cases, we would require much higher memory with a large number of processors running in parallel or high end graphics processing units to extract the s-vectors in reasonable time. This limits the applications of s-vectors to cases where there are smaller number of speaker or language labels.

5.2.2 Supervised Neural Network models for Speaker Verification

The major disadvantage of the explored direction is the limitation of softDCF loss function. While this loss achieves significant improvements over other loss functions considered, it fails to work when the model parameters are initialized randomly. As discussed in 4.6.5, the softDCF loss works by correcting errors using training examples whose output scores lie close to the decision threshold. For a randomly initialized E2E-NPLDA model, the output score distributions of the target and non-target examples are highly overlapping to begin with, and optimizing such a network with the softDCF loss using the backpropagation algorithm drives it to a saddle point, where the gradients are close to zero, but the target and non-target score distributions are still highly overlapping. Hence, to use the softDCF loss, we require the model parameters to be initialized using the generative PLDA, or pre-trained with any other objective, such as the binary cross-entropy loss or the weighted C_{llr}. However, it can still be combined with other loss functions as it results in robust performance across a wide range of test datasets.

5.3 Future directions

All the existing literature on speaker and language recognition, along with the directions explored in this thesis, have demonstrated the efficacy of supervised learning approaches and neural networks. However, there are several directions in which future research can build upon the approaches pursued in this thesis. In this section, we discuss some future directions that can be inspired by this thesis.

5.3.1 Neural network approaches for language recognition

Of late, neural network approaches have been used for language recognition as well, and they have been shown to outperform the i-vector approaches [110, 111]. Using the Siamese network approaches and loss functions presented in this thesis for language recognition can be particularly useful for problems such as language diarization. However, as there can be a huge within-language variability due to speaker and content, the Siamese networks optimized using a binary verification loss might not converge as well as a regular language classifier that uses softmax loss. A potential solution in such cases is to combine the binary verification losses with the softmax loss, so that the network explicitly learns a similarity metric that can help in problems such as language diarization.

5.3.2 Neural generative models for language and accent recognition

As the focus of the research community shifted from the i-vector approach to Neural language embeddings like x-vectors, all approaches have been focused on discriminative modeling. One major disadvantage of discriminative models is that they are prone to overfitting, particularly in low-resource conditions. There are several languages with extremely limited training data, which have limited speaker variability, and do not sufficiently cover all the vocabulary, which makes it hard for discriminative models to generalize well in practical scenarios. In contrast, generative models can generalize better with limited training data, making them suitable for low-resource accent recognition. Moreover, humans who speak multiple languages and accents are naturally better at recognizing those accents. This is primarily because they can repeat or imagine an utterance in the target accents, and make comparisons with the utterance to recognize the accent. This fact also reinforces the intuition that generative models can be advantageous for language and accent recognition.

One could explore deep generative models for languages and accents, such as variational autoencoders (VAE) or generative adversarial networks (GANs). Similar to the i-vector approach, one can explore various neural network architectures that can be used as the encoders of VAE, and the latent representations can be used for language and accent recognition. Traditionally in VAE, the KL divergence between the encoded distribution and the standard normal distribution is added to the reconstruction loss for regularization. This approach can be reformulated in a supervised setting similar to the s-vector approach, by using label-conditioned Gaussian distributions to regularize the VAE, instead of a standard normal distribution.

5.3.3 Self-supervised Learning for speaker and language recognition

In this thesis, we have explored discriminative models and supervised generative models for speaker and language recognition. Today, there is a widespread interest in the machine learning community to leverage large amounts of unlabelled data to derive meaningful representations. Examples include HuBERT [112], wav2vec [113], and their several variants. In HuBERT, the speech feature frames are clustered using k-means, and a transformer-based model is trained to predict the clusters of masked hidden units. The wav2vec model is a CNN-based neural network encoder trained to predict the encoder representations of the next k frames, using the as a function of previous n frames, by optimizing a contrastive loss. There are several ways in which the approaches proposed in this thesis can be combined with these self-supervised learning approaches.

Firstly, the representations learnt from HuBERT/wav2vec models can replace the MFCC or BNF features used in the i-vector/s-vector approaches for language recognition. Further, as Lei et.al. suggested in [61], the senone posteriors from a DNN acoustic model can be used instead of a GMM to compute the Baum-Welch statistics of an i-vector model. Similar approaches can be explored using the phonetic cluster posteriors from a HuBERT/wav2vec model, with the s-vector framework. It may also be beneficial to use the Siamese speaker verification approaches discussed in this thesis on the self-supervised representations learnt from HuBERT/wav2vec models.

In speaker recognition, there have been successful implementations of self-supervised approaches for speaker verification with the VoxCeleb datasets [114]. As speaker and language are utterance-level information, we may benefit from self-supervised learning techniques applied for longer segments. The supervised approaches explored in this thesis can be used with pseudo-speaker labels obtained from various clustering algorithms.

5.3.4 Architectures for speaker detection in conversational settings

Although the theme of speaker detection assumes that a test segment can contain multiple speakers, the test datasets primarily consist of single-speaker test segments. If there are other distracting speakers in the test segment, it can result in a missed detection. To solve this problem, we need the speaker verification system to be aware of the enrollment speaker, while processing the test segment. We can achieve this by exploring attention-based neural architectures that remember the enrollment recording while processing the test, and pay attention to the regions that are similar to the enrollment speaker.

We explored several architectures along these lines, but however, none of these architectures performed better than the baseline. We hypothesize that this is because both our train and test datasets (VoxCeleb 2 and 1) contained only single speaker segments. However, this is an area worth investigating in future, with more challenging speaker verification datasets containing multi-speaker test segments. In such cases, it would not be meaningful to use any of the classification based approaches, as the ground truth will have multiple speaker labels, and loss functions such as the softmax loss and its angular margin versions require one-hot label vectors for training. The directions pursued in this thesis can be used in these scenarios, as Siamese architectures with metric learning based loss functions do not have the drawback of the classification approaches.

Appendix A

In this appendix, we present a detailed step-by-step derivation of the log-likelihood expression (Equation. 3.5) of the i-vector/s-vector model, and derive the expressions for the posterior density of the latent variable $\boldsymbol{y}$, for both the i-vector and s-vector models. The proof is adopted from [65], with some minor modifications. For this, we again define all the terms that are involved.

- The short-time frame-level features of a speech utterance s (MFCC/BNFs), are denoted as:

$$X(s) = \left[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{H(s)}\right] \tag{A.1}$$

  Here, $\boldsymbol{x}_i$ is an F-dimensional vector, and $H(s)$ is the number of frames for utterance s.

- The GMM-UBM parameters are given by

$$\Lambda = \{\pi_c, \boldsymbol{\mu}_c, \Sigma_c\}_{c=1}^{C} \tag{A.2}$$

- The GMM mean supervector and covariance matrix, along with their dimensions, can be expressed as:

$$\boldsymbol{M}_0 = \begin{pmatrix} \boldsymbol{\mu}_1 \\ \vdots \\ \boldsymbol{\mu}_C \end{pmatrix}_{CF\times 1} \tag{A.3}$$

$$\Sigma = \begin{pmatrix} \begin{bmatrix} \Sigma_1 \end{bmatrix} & & 0 \\ & \ddots & \\ 0 & & \begin{bmatrix} \Sigma_C \end{bmatrix} \end{pmatrix}_{CF\times CF} \tag{A.4}$$

- **Baum Welch statistics**: For each GMM mixture component c,

$$N_c(s) = \sum_{i=1}^{H(s)} \mathrm{p}_\Lambda(c \mid \boldsymbol{x}_i) \tag{A.5}$$

$$\boldsymbol{F}_{X,c}(s) = \sum_{i=1}^{H(s)} \mathrm{p}_\Lambda(c \mid \boldsymbol{x}_i)(\boldsymbol{x}_i - \boldsymbol{\mu}_c) \tag{A.6}$$

$$S_{X,X,c}(s) = \sum_{i=1}^{H(s)} \mathrm{p}_\Lambda(c \mid \boldsymbol{x}_i)(\boldsymbol{x}_i - \boldsymbol{\mu}_c)(\boldsymbol{x}_i - \boldsymbol{\mu}_c)^\intercal \tag{A.7}$$

- BW stats in matrix form:

$$N(s) = \begin{pmatrix} N_1(s)\begin{bmatrix} I_{F\times F} \end{bmatrix} & & 0 \\ & \ddots & \\ 0 & & N_C(s)\begin{bmatrix} I_{F\times F} \end{bmatrix} \end{pmatrix}_{CF\times CF} \tag{A.8}$$

$$\boldsymbol{F}_X(s) = \begin{pmatrix} \boldsymbol{F}_{X,1}(s) \\ \vdots \\ \boldsymbol{F}_{X,C}(s) \end{pmatrix}_{CF\times 1} \tag{A.9}$$

$$S_{X,X}(s) = \begin{pmatrix} S_{X,X,1}(s) & & 0 \\ & \ddots & \\ 0 & & S_{X,X,C}(s) \end{pmatrix}_{CF\times CF} \tag{A.10}$$

The i-vector model assumes that the feature sequence $X(s)$ is generated from an adapted GMM with a new (utterance specific) mean supervector $\boldsymbol{M}(s)$, according to the generative model:

$$\boldsymbol{M}(s) = \boldsymbol{M}_0 + T\boldsymbol{y}(s) \tag{A.11}$$

T is the total variability matrix of dimensions $CF \times R$, and can be written as:

$$T = \begin{pmatrix} [T_1]_{F\times R} \\ \vdots \\ [T_C]_{F\times R} \end{pmatrix}_{CF\times R} \tag{A.12}$$

where T_c can be considered as the submatrix of T corresponding to the c^{th} mixture of the GMM.

Now, with the assumption that if $\boldsymbol{x}_i$ is aligned to mixture component c, its likelihood can be written as the Gaussian log-likelihood of micture component c, the complete log-likelihood of an observed feature sequence $X(s)$, given the latent vector $\boldsymbol{y}$ can be written as:

$$\log \mathrm{p}_T\left(X(s)|\boldsymbol{y}(s)\right) = \sum_{c=1}^{C} \sum_{\substack{i=1 \\ \boldsymbol{x}_i \text{ aligned to} \\ \text{mixture } c}}^{H(s)} \log\left(\frac{1}{(2\pi)^{\frac{F}{2}}\left|\Sigma_c\right|^{\frac{1}{2}}}\right)$$

$$-\frac{1}{2}\sum_{c=1}^{C}\sum_{\substack{i=1\\ \boldsymbol{x}_i \text{ aligned to}\\ \text{mixture } c}}^{H(s)} \left(\boldsymbol{x}_i-\boldsymbol{\mu}_c-T_c\boldsymbol{y}(s)\right)^{\intercal}\Sigma_c^{-1}\left(\boldsymbol{x}_i-\boldsymbol{\mu}_c-T_c\boldsymbol{y}(s)\right) \tag{A.13}$$

Now, The sum over the frames aligned to mixture c are given by the Baum-Welch statistics. The first summation term can be written in terms of the zeroth order statistics, and the second summation containing a second order term can be rearranged and simplified as follows:

$$\begin{aligned}\log \mathrm{p}_T\left(X(s)|\boldsymbol{y}(s)\right) = &\sum_{c=1}^{C} N_c(s)\log\left(\frac{1}{(2\pi)^{\frac{F}{2}}\left|\Sigma_c\right|^{\frac{1}{2}}}\right)\\ &-\frac{1}{2}\sum_{c=1}^{C}\sum_{\substack{i=1\\ \boldsymbol{x}_i \text{ aligned to}\\ \text{mixture } c}}^{H(s)} \operatorname{tr}\left(\Sigma_c^{-1}\left(\boldsymbol{x}_i-\boldsymbol{\mu}_c-T_c\boldsymbol{y}(s)\right)\left(\boldsymbol{x}_i-\boldsymbol{\mu}_c-T_c\boldsymbol{y}(s)\right)^{\intercal}\right)\end{aligned} \tag{A.14}$$

$$\begin{aligned}= &\sum_{c=1}^{C} N_c(s)\log\left(\frac{1}{(2\pi)^{\frac{F}{2}}\left|\Sigma_c\right|^{\frac{1}{2}}}\right)\\ &-\frac{1}{2}\sum_{c=1}^{C}\operatorname{tr}\left(\Sigma_c^{-1}\sum_{\substack{i=1\\ \boldsymbol{x}_i \text{ aligned to}\\ \text{mixture } c}}^{H(s)}\left(\boldsymbol{x}_i-\boldsymbol{\mu}_c-T_c\boldsymbol{y}(s)\right)\left(\boldsymbol{x}_i-\boldsymbol{\mu}_c-T_c\boldsymbol{y}(s)\right)^{\intercal}\right)\end{aligned} \tag{A.15}$$

$$\begin{aligned}\log \mathrm{p}_T(X(s)\mid\boldsymbol{y}(s)) = &\sum_{c=1}^{C} N_c(s)\log\left(\frac{1}{(2\pi)^{\frac{F}{2}}\left|\Sigma_c\right|^{\frac{1}{2}}}\right)\\ &-\frac{1}{2}\sum_{c=1}^{C}\operatorname{tr}\left(\Sigma_c^{-1}\sum_{i=1}^{H(s)}\mathrm{p}_\Lambda(c\mid\boldsymbol{x}_i)\left(\boldsymbol{x}_i-\boldsymbol{\mu}_c-T_c\boldsymbol{y}(s)\right)\left(\boldsymbol{x}_i-\boldsymbol{\mu}_c-T_c\boldsymbol{y}(s)\right)^{\intercal}\right)\end{aligned} \tag{A.16}$$

$$=\sum_{c=1}^{C} N_c(s)\log\left(\frac{1}{(2\pi)^{\frac{F}{2}}\left|\Sigma_c\right|^{\frac{1}{2}}}\right)$$

$$-\frac{1}{2}\sum_{c=1}^{C}\operatorname{tr}\left(\Sigma_c^{-1}\sum_{i=1}^{H(s)}\mathrm{p}_\Lambda(c \mid \boldsymbol{x}_i)(\boldsymbol{x}_i-\boldsymbol{\mu}_c)(\boldsymbol{x}_i-\boldsymbol{\mu}_c)^\intercal \right.$$
$$\left. -\Sigma_c^{-1}\sum_{i=1}^{H(s)}2\,\mathrm{p}_\Lambda(c \mid \boldsymbol{x}_i)(\boldsymbol{x}_i-\boldsymbol{\mu}_c)(T_c\boldsymbol{y}(s))^\intercal+\Sigma_c^{-1}\sum_{i=1}^{H(s)}\mathrm{p}_\Lambda(c \mid \boldsymbol{x}_i)T_c\boldsymbol{y}(s)\boldsymbol{y}(s)^\intercal T_c^\intercal\right) \tag{A.17}$$

$$\log \mathrm{p}_T(X(s) \mid \boldsymbol{y}(s)) = \sum_{c=1}^{C}N_c(s)\log\left(\frac{1}{(2\pi)^{\frac{F}{2}}\left|\Sigma_c\right|^{\frac{1}{2}}}\right)$$
$$-\frac{1}{2}\sum_{c=1}^{C}\operatorname{tr}\left(\Sigma_c^{-1}\left[S_{X,X,c}(s)-2\boldsymbol{F}_{X,c}(s)(T_c\boldsymbol{y}(s))^\intercal+N_c(s)T_c\boldsymbol{y}(s)\boldsymbol{y}(s)^\intercal T_c^\intercal\right]\right) \tag{A.18}$$

$$=\sum_{c=1}^{C}N_c(s)\log\left(\frac{1}{(2\pi)^{\frac{F}{2}}\left|\Sigma_c\right|^{\frac{1}{2}}}\right)-\frac{1}{2}\sum_{c=1}^{C}\operatorname{tr}\left(\Sigma_c^{-1}S_{X,X,c}(s)\right)$$
$$+\sum_{c=1}^{C}\boldsymbol{y}(s)^\intercal T_c^\intercal\Sigma_c^{-1}\boldsymbol{F}_{X,c}(s)-\frac{1}{2}\sum_{c=1}^{C}\boldsymbol{y}(s)^\intercal T_c^\intercal\Sigma_c^{-1}N_c(s)T_c\boldsymbol{y}(s) \tag{A.19}$$

$$=\underbrace{\sum_{c=1}^{C}N_c(s)\log\left(\frac{1}{(2\pi)^{\frac{F}{2}}\left|\Sigma_c\right|^{\frac{1}{2}}}\right)-\frac{1}{2}\operatorname{tr}\left(\Sigma^{-1}S_{X,X}(s)\right)}_{G(s)}$$
$$\underbrace{+\boldsymbol{y}(s)^\intercal T^\intercal\Sigma^{-1}\boldsymbol{F}_X(s)-\frac{1}{2}\boldsymbol{y}(s)^\intercal T^\intercal\Sigma^{-1}N(s)T\boldsymbol{y}(s)}_{H_T(s,\boldsymbol{y}(s))} \tag{A.20}$$

$$=G(s)+H_T(s,\boldsymbol{y}(s)) \tag{A.21}$$

That concludes the derivation of the log-likelihood expression in Eq. 3.5, which is valid for both i-vector and s-vector models, as we haven't assumed any prior over the latent variable $\boldsymbol{y}(s)$ so far.

Using the Bayes' formula, the *aposteriori* density function of $\boldsymbol{y}(s)$ given an utterance $X(s)$

can be written as

$$\mathrm{p}_T(\boldsymbol{y}(s) \mid X(s)) = \frac{\mathrm{p}_T(X(s) \mid \boldsymbol{y}(s))\mathrm{p}(\boldsymbol{y}(s))}{\int\limits_{<\boldsymbol{y}>} \mathrm{p}_T(X(s) \mid \boldsymbol{y})\mathrm{p}(\boldsymbol{y})\mathrm{d}\boldsymbol{y}} \tag{A.22}$$

$$\propto \mathrm{p}_T(X(s) \mid \boldsymbol{y}(s))\mathrm{p}(\boldsymbol{y}(s)) \tag{A.23}$$

For the i-vector model, $\boldsymbol{y}(s)$ is assumed to have a standard normal distribution. Substituting for $\mathrm{p}(\boldsymbol{y}(s))$ and ignoring constant terms, we get:

$$\propto \exp\left(G(s) + H_T(s, \boldsymbol{y}(s)) - \frac{1}{2}\boldsymbol{y}(s)^\intercal \boldsymbol{y}(s)\right) \tag{A.24}$$

The term $G(s)$ is independent of y, and can be ignored. Substituting for $H_T(s, \boldsymbol{y}(s))$

$$\propto \exp\left(\boldsymbol{y}^\intercal T^\intercal \Sigma^{-1} \boldsymbol{F}_X(s) - \frac{1}{2}\boldsymbol{y}^\intercal T^\intercal \Sigma^{-1} N(s) T \boldsymbol{y} - \frac{1}{2}\boldsymbol{y}(s)^\intercal \boldsymbol{y}(s)\right) \tag{A.25}$$

$$\mathrm{p}_T(\boldsymbol{y}(s) \mid X(s)) = \exp\left(\boldsymbol{y}(s)^\intercal T^\intercal \Sigma^{-1} \boldsymbol{F}_X(s) - \frac{1}{2}\boldsymbol{y}(s)^\intercal \{\underbrace{I + T^\intercal \Sigma^{-1} N(s) T}_{\mathcal{L}(s)}\}\boldsymbol{y}(s)\right) \tag{A.26}$$

$$= \exp\left(\boldsymbol{y}(s)\,\mathcal{L}(s)\,\{\mathcal{L}(s)^{-1} T^\intercal \Sigma^{-1} \boldsymbol{F}_X(s)\} - \frac{1}{2}\boldsymbol{y}(s)^\intercal\,\mathcal{L}(s)\,\boldsymbol{y}(s)\right) \tag{A.27}$$

From this form, we can conclude that that the *aposteriori* distribution of $\boldsymbol{y}(s)$ given the feature frames $X(s)$ is Gaussian with covariance $\mathcal{L}(s)^{-1}$ and mean $\mathcal{L}(s)^{-1} T^\intercal \Sigma^{-1} \boldsymbol{F}_X(s)$.

For the s-vector model, we have assumed that:

$$\mathrm{p}(X(s)|\boldsymbol{y}(s), l(s)) = \mathrm{p}(X(s)|\boldsymbol{y}(s)) \tag{A.28}$$

It becomes tedious to derive the posterior by simply substituting the GMM prior of $\boldsymbol{y}(s)$ in Eq. A.23. However, for a particular language class l, Eq. A.23 can be modified as:

$$\mathrm{p}_T(\boldsymbol{y}(s) \mid X(s), l(s)) \propto \mathrm{p}_T(X(s) \mid \boldsymbol{y}(s))\, \mathrm{p}(\boldsymbol{y}(s) \mid l(s)) \tag{A.29}$$

Substituting the likelihood term and the class conditioned prior, and simplifying, we get:

$$\begin{aligned}
\mathrm{p}_T(\boldsymbol{y}(s) \mid X(s), l(s)) &\propto \exp\left(G(s) + H_T(s, \boldsymbol{y}(s)) - \frac{1}{2}\boldsymbol{y}(s)^\intercal \boldsymbol{y}(s)\right) && \text{(A.30)}\\
&\propto \exp\Bigg(\boldsymbol{y}^\intercal T^\intercal \Sigma^{-1} \boldsymbol{F}_X(s) - \frac{1}{2}\boldsymbol{y}^\intercal T^\intercal \Sigma^{-1} N(s) T \boldsymbol{y} \\
&\qquad - \frac{1}{2}(\boldsymbol{y}(s) - \boldsymbol{m}_{l(s)})^\intercal (\boldsymbol{y}(s) - \boldsymbol{m}_{l(s)})\Bigg) && \text{(A.31)}\\
&= \exp\left(\boldsymbol{y}(s)^\intercal \left(\boldsymbol{m}_{l(s)} + T^\intercal \Sigma^{-1} \boldsymbol{F}_X(s)\right) - \frac{1}{2}\boldsymbol{y}(s)^\intercal \{\underbrace{I + T^\intercal \Sigma^{-1} N(s) T}_{\mathcal{L}(s)}\} \boldsymbol{y}(s)\right) && \text{(A.32)}\\
&= \exp\left(\boldsymbol{y}(s)\, \mathcal{L}(s) \left\{\mathcal{L}(s)^{-1} \left(\boldsymbol{m}_{l(s)} + T^\intercal \Sigma^{-1} \boldsymbol{F}_X(s)\right)\right\} - \frac{1}{2}\boldsymbol{y}(s)^\intercal\, \mathcal{L}(s)\, \boldsymbol{y}(s)\right) && \text{(A.33)}
\end{aligned}$$

This is a Gaussian Probability Density Function (PDF) with covariance $\mathcal{L}(s)^{-1}$ and mean $\mathcal{L}(s)^{-1}\left(\boldsymbol{m}_{l(s)} + T^\intercal \Sigma^{-1} \boldsymbol{F}_X(s)\right)$.

The label conditioned data likelihood (Eq. 3.44 is rewritten here using the short hand notation $L_X(l)$ as:

$$L_X(l) = \log \mathrm{p}(X(s) \mid l) = \log \int\limits_{<\boldsymbol{y}(s)>} e^{(G(s)+H_T(s,\boldsymbol{y}(s))} \frac{1}{(2\pi)^{\frac{R}{2}}} e^{-\frac{1}{2}(\boldsymbol{y}(s)-\boldsymbol{m}_l)^\intercal(\boldsymbol{y}(s)-\boldsymbol{m}_l)} \mathrm{d}\boldsymbol{y} \tag{A.34}$$

Here, we present the intermediate simplification steps to get to Eq. 3.45. Note that, using Bayes' formula, and assuming uniform prior of $p(l) = \frac{1}{L}$, the language posteriors can be written in terms of the label conditioned log-likelihoods as :

$$\mathrm{p}(l|X(s)) = \frac{\mathrm{p}(X(s)|l)}{\sum_{l=1}^{L} \mathrm{p}(X(s)|l)} = \frac{e^{k+\log \mathrm{p}(X(s)|l)}}{\sum_{l=1}^{L} e^{k+\log \mathrm{p}(X(s)|l)}} \tag{A.35}$$

Hence, while simplifying Eq. A.34, we can ignore the additive constant terms independent of l, as it doesn't affect the label posterior computation. Separating the terms independent of l and $\boldsymbol{y}(s)$ to simplify Eq. A.34, we get:

$$\begin{aligned} L_X(l) &= \log \frac{1}{(2\pi)^{\frac{R}{2}}} e^{G(s)} \int\limits_{<\boldsymbol{y}(s)>} e^{H_T(s,\boldsymbol{y}(s)) - \frac{1}{2}\boldsymbol{y}(s)^\intercal \boldsymbol{y}(s) - \frac{1}{2}\boldsymbol{m}_l{}^\intercal \boldsymbol{m}_l + \boldsymbol{y}(s)^\intercal \boldsymbol{m}_l} \mathrm{d}\boldsymbol{y} \\ &= k - \frac{1}{2}\boldsymbol{m}_l{}^\intercal \boldsymbol{m}_l + \log \int\limits_{<\boldsymbol{y}(s)>} e^{\boldsymbol{y}(s)^\intercal T^\intercal \Sigma^{-1} \boldsymbol{F}_X(s) - \frac{1}{2}\boldsymbol{y}(s)^\intercal T^\intercal \Sigma^{-1} N(s) T \boldsymbol{y}(s) - \frac{1}{2}\boldsymbol{y}(s)^\intercal \boldsymbol{y}(s) + \boldsymbol{y}(s)^\intercal \boldsymbol{m}_l} \mathrm{d}\boldsymbol{y} \\ &= k - \frac{1}{2}\boldsymbol{m}_l{}^\intercal \boldsymbol{m}_l + \log \int\limits_{<\boldsymbol{y}(s)>} e^{\boldsymbol{y}(s)^\intercal (\boldsymbol{m}_l + T^\intercal \Sigma^{-1} \boldsymbol{F}_X(s)) - \frac{1}{2}\boldsymbol{y}(s)^\intercal \mathcal{L}(s) \boldsymbol{y}(s)} \mathrm{d}\boldsymbol{y} \end{aligned} \tag{A.36}$$

As the argument of the integral is a scaled Gaussian density function in $\boldsymbol{y}(s)$, it can be simplified as follows, which gives us Eq. 3.45.

$$\begin{aligned} L_X(l) &= k - \frac{1}{2}\boldsymbol{m}_l{}^\intercal \boldsymbol{m}_l + \log \left(|\mathcal{L}(s)|^{-\frac{1}{2}} e^{\frac{1}{2}(\boldsymbol{m}_l + T^\intercal \Sigma^{-1} \boldsymbol{F}_X(s))^\intercal \mathcal{L}(s)^{-1} (\boldsymbol{m}_l + T^\intercal \Sigma^{-1} \boldsymbol{F}_X(s))} \right) \\ &= k - \frac{1}{2}\boldsymbol{m}_l{}^\intercal \boldsymbol{m}_l + \frac{1}{2}(\boldsymbol{m}_l + T^\intercal \Sigma^{-1} \boldsymbol{F}_X(s))^\intercal \mathcal{L}(s)^{-1} (\boldsymbol{m}_l + T^\intercal \Sigma^{-1} \boldsymbol{F}_X(s)) \\ &= k - \frac{1}{2}\boldsymbol{m}_l{}^\intercal (I -^2 \mathcal{L}(s)^{-1}) \boldsymbol{m}_l + \boldsymbol{m}_l{}^\intercal \mathcal{L}(s)^{-1} T^\intercal \Sigma^{-1} \boldsymbol{F}_X(s) \end{aligned} \tag{A.37}$$

Appendix B

In this appendix, we give a brief discussion on proper scoring rules [115, 105]. We follow the notation used in [105], as it is concise and simple. Binary *proper scoring rules* are a family of special cost functions of the form $C^*(q, h)$, which evaluates the goodness of the recognizer output q, for a trial where hypothesis h is true. Here, $h \in \{\texttt{tar}, \texttt{non}\}$, where `tar` and `non` represent the target and non-target hypotheses respectively. Examples of Binary PSRs include:

- Logarithmic PSR: Considering q to be the posterior probability $\mathrm{p}(h = \texttt{tar}|X)$, given an observation X, the logarithmic PSR is defined as

$$C^*(q, h = \texttt{tar}) = -log(q) \quad ; \quad C^*(q, h = \texttt{non}) = -log(1-q) \tag{B.1}$$

- The zero-one score: If a decision rule is given by thresholding the recognizer output q, and the Bayesian costs assigned to the costs are 0 for a correct decision and 1 for an error (false alarm/miss), the zero-one score is defined as

$$C^*(q, h = \texttt{tar}) = 1 - u(q - \theta) \quad ; \quad C^*(q, h = \texttt{non}) = u(q - \theta) \tag{B.2}$$

- Brier score: It is defined as:

$$C^*(q, h = \texttt{tar}) = (1-q)^2 \quad ; \quad C^*(q, h = \texttt{non}) = q^2 \tag{B.3}$$

The equation (3) of [105] gives a template to build an objective function for binary classifiers:

$$\bar{C}_w^{\pi} = \frac{\pi}{T} \sum_{i \in \mathcal{T}} C_w^*(q_i, \texttt{tar}) + \frac{1-\pi}{N} \sum_{i \in \mathcal{N}} C_w^*(q_i, \texttt{non}) \tag{B.4}$$

Here, q_i is the recognizer's posterior for trial i, $\mathcal{T}$ is a set of T target trials, and $\mathcal{N}$ is a set of N non-target trial indices. $C_w^*()$ is a binary proper scoring rule.

The softDCF function was defined in (Eq. 4.16) as:

$$softDCF = \frac{1}{|\mathcal{T}|} \sum_{i \in \mathcal{T}} [1 - \sigma(\alpha(s_i - \theta))] + \frac{\beta}{|\mathcal{N}|} \sum_{i \in \mathcal{N}} \sigma(\alpha(s_i - \theta)) \tag{B.5}$$

By considering $q_i = \sigma(\alpha(s_i - \theta))$ as the posterior probability, we can compare it to Eq. B.4 to identify what the term $C_w^*()$ is. We get

$$C^*(q, h = \texttt{tar}) = (1 - q) \quad ; \quad C^*(q, h = \texttt{non}) = q \tag{B.6}$$

As this is a proper scoring rule, the softDCF loss function is, in fact, a prior weighted proper scoring rule.

Bibliography

[1] Thomas F Quatieri. *Discrete-time speech signal processing: principles and practice*. Pearson Education India, 2002. 3

[2] John Makhoul. Linear prediction: A tutorial review. *Proceedings of the IEEE*, 63(4):561–580, 1975. 3

[3] Sriram Ganapathy. *Signal analysis using autoregressive models of amplitude modulation*. PhD thesis, Johns Hopkins University, 2012. 3

[4] Purvi Agrawal. *Neural representation learning for speech and audio signals*. PhD thesis, Indian Institute of Science, 2021. 3

[5] Arthur S House and Edward P Neuburg. Toward automatic identification of the language of an utterance. i. preliminary methodological considerations. *The Journal of the Acoustical Society of America*, 62(3):708–713, 1977. 6

[6] Deidre Cimarusti and R Ives. Development of an automatic identification system of spoken languages: Phase i. In *ICASSP'82. IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 7, pages 1661–1663. IEEE, 1982. 6

[7] Bishnu S Atal. Automatic recognition of speakers from their voices. *Proceedings of the IEEE*, 64(4):460–475, 1976. 6

[8] Aaron E Rosenberg. Automatic speaker verification: A review. *Proceedings of the IEEE*, 64(4):475–487, 1976. 6

[9] M. A. Zissman. Comparison of four approaches to automatic language identification of telephone speech. *IEEE Transactions on Speech and Audio Processing*, 4(1):31–44, 1996. 6

[10] Jiri Navratil. Spoken language recognition-a step toward multilinguality in speech processing. *IEEE Transactions on Speech and Audio Processing*, 9(6):678–685, 2001. 6

[11] Najim Dehak, Patrick J Kenny, Réda Dehak, Pierre Dumouchel, and Pierre Ouellet. Front-end factor analysis for speaker verification. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(4):788–798, 2010. 8, 12, 14, 15, 62, 105

[12] George R Doddington. Speaker recognition—Identifying people by their voices. *Proceedings of the IEEE*, 73(11):1651–1664, 1985. 11

[13] Douglas OShaughnessy. Speaker recognition. *IEEE Assp Magazine*, 3:4–17, 1986. 11

[14] Douglas A Reynolds and Richard C Rose. Robust text-independent speaker identification using Gaussian mixture speaker models. *IEEE Transactions on Speech and Audio Processing*, 3(1):72–83, 1995. 11, 14, 32, 105

[15] Douglas A Reynolds, Thomas F Quatieri, and Robert B Dunn. Speaker Verification using Adapted Gaussian Mixture Models. *Digital signal processing*, 10(1-3):19–41, 2000. 11, 32, 105

[16] Patrick Kenny. Joint factor analysis of speaker and session variability: Theory and algorithms. *(Report) CRIM*, 14:1–17, 2006. 12, 14, 32, 35, 39, 44, 62

[17] Patrick Kenny, Gilles Boulianne, Pierre Ouellet, and Pierre Dumouchel. Joint Factor

Analysis Versus Eigenchannels in Speaker Recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(4):1435–1447, 2007. 12

[18] Sergey Ioffe. Probabilistic linear discriminant analysis. In *European Conference on Computer Vision*, pages 531–542. Springer, 2006. 12, 17, 62

[19] Seiichi Nakagawa, Yoshio Ueda, and Takashi Seino. Speaker-independent, text-independent language identification by HMM. In *Second International Conference on Spoken Language Processing*, 1992. 12

[20] TJ HAZEN. Automatic language identification using an segment-based approach. In *Proc. EUROSPEECH'93*, pages 1303–1306, 1993. 13

[21] Marc A Zissman and Elliot Singer. Automatic language identification of telephone speech messages using phoneme recognition and n-gram modeling. In *Proceedings of ICASSP'94. IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 1, pages I–305. IEEE, 1994. 13

[22] Marc A Zissman. Comparison of four approaches to automatic language identification of telephone speech. *IEEE Transactions on speech and audio processing*, 4(1):31, 1996. 13

[23] Pedro A Torres-Carrasquillo, Elliot Singer, Mary A Kohler, Richard J Greene, Douglas A Reynolds, and John R Deller Jr. Approaches to language identification using Gaussian mixture models and shifted delta cepstral features. In *Interspeech*, 2002. 13

[24] William M Campbell, Elliot Singer, Pedro A Torres-Carrasquillo, and Douglas A Reynolds. Language recognition with support vector machines. In *Odyssey: The Speaker and Language Recognition Workshop (2004)*, pages 285–288, 2004. 13, 32, 51

[25] David A Van Leeuwen and Niko Brummer. Channel-dependent gmm and multi-class logistic regression models for language recognition. In *2006 IEEE Odyssey-The Speaker and Language Recognition Workshop*, pages 1–8. IEEE, 2006. 13

[26] William M Campbell, Douglas E Sturim, and Douglas A Reynolds. Support vector machines using GMM supervectors for speaker verification. *IEEE signal processing letters*, 13(5):308–311, 2006. 13

[27] Najim Dehak, Patrick J Kenny, Réda Dehak, Pierre Dumouchel, and Pierre Ouellet. Front-End Factor Analysis For Speaker Verification. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(4):788–798, 2011. 14, 15, 32, 35, 36, 70, 105

[28] Ehsan Variani, Xin Lei, Erik McDermott, Ignacio Lopez Moreno, and Javier Gonzalez-Dominguez. Deep neural networks for small footprint text-dependent speaker verification. In *ICASSP*, pages 4052–4056. IEEE, 2014. 16

[29] Amirsina Torfi, Jeremy Dawson, and Nasser M Nasrabadi. Text-independent speaker verification using 3d convolutional neural networks. In *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2018. 16

[30] Maxim Tkachenko, Alexander Yamshinin, Nikolay Lyubimov, Mikhail Kotov, and Marina Nastasenko. Language identification using time delay neural network d-vector on short utterances. In *International conference on speech and computer*, pages 443–449. Springer, 2016. 16

[31] David Snyder, Pegah Ghahremani, Daniel Povey, Daniel Garcia-Romero, Yishay Carmiel, and Sanjeev Khudanpur. Deep neural network-based speaker embeddings for end-to-end speaker verification. In *Spoken Language Technology Workshop (SLT)*, pages 165–170. IEEE, 2016. 16, 72, 78

[32] David Snyder, Daniel Garcia-Romero, Gregory Sell, Daniel Povey, and Sanjeev Khudanpur. X-vectors: Robust DNN Embeddings for Speaker Recognition. In *ICASSP*, pages 5329–5333, 2018. 16, 70, 71, 73, 106

[33] Daniel Povey, Gaofeng Cheng, Yiming Wang, Ke Li, Hainan Xu, Mahsa Yarmohammadi, and Sanjeev Khudanpur. Semi-orthogonal low-rank matrix factorization for deep neural networks. In *Proc. Interspeech*, pages 3743–3747, 2018. 16, 84

[34] Yingke Zhu, Tom Ko, David Snyder, Brian Mak, and Daniel Povey. Self-Attentive Speaker Embeddings for Text-Independent Speaker Verification. In *Proc. Interspeech*, pages 3573–3577, 2018. 16

[35] Joon Son Chung, Arsha Nagrani, and Andrew Zisserman. VoxCeleb2: Deep Speaker Recognition. In *Proc. Interspeech*, pages 1086–1090, 2018. 16, 83, 101

[36] Javier Gonzalez-Dominguez, Ignacio Lopez-Moreno, Haşim Sak, Joaquin Gonzalez-Rodriguez, and Pedro J. Moreno. Automatic language identification using long short-term memory recurrent neural networks. In *Proc. Interspeech*, pages 2155–2159, 2014. 16

[37] Ruben Zazo, Alicia Lozano-Diez, and Joaquin Gonzalez-Rodriguez. Evaluation of an lstm-rnn system in different nist language recognition frameworks. In *Odyssey*, pages 231–236, 2016. 16

[38] Bharat Padi, Anand Mohan, and Sriram Ganapathy. Towards relevance and sequence modeling in language recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:1223–1232, 2020. 17

[39] Chia-Ping Chen, Su-Yu Zhang, Chih-Ting Yeh, Jia-Ching Wang, Tenghui Wang, and Chien-Lin Huang. Speaker characterization using tdnn-lstm based speaker embedding. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6211–6215. IEEE, 2019. 17

[40] Chien-Lin Huang. Exploring effective data augmentation with tdnn-lstm neural network embedding for speaker recognition. In *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 291–295. IEEE, 2019. 17

[41] Simon JD Prince and James H Elder. Probabilistic linear discriminant analysis for inferences about identity. In *International Conference on Computer Vision (ICCV) (2007)*, pages 1–8, 2007. 17, 32, 106

[42] Patrick Kenny. Bayesian Speaker Verification with Heavy-Tailed Priors. In *Proc. Odyssey*, pages 14–21, 2010. 18, 70, 74

[43] David Martinez, Oldřich Plchot, Lukáš Burget, Ondřej Glembek, and Pavel Matějka. Language recognition in ivectors space. In *INTERSPEECH (2011)*, pages 861–864, 2011. 18, 33, 41, 51

[44] Sandro Cumani, Niko Brümmer, Lukáš Burget, Pietro Laface, Oldřich Plchot, and Vasileios Vasilakakis. Pairwise Discriminative Speaker Verification in the i-Vector Space. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(6):1217–1227, 2013. 19, 72, 74

[45] Alvin F. Martin, Craig S. Greenberg, John M. Howard, George R. Doddington, and John J. Godfrey. NIST Language Recognition Evaluation - Past and Future. In *Proc. The Speaker and Language Recognition Workshop (Odyssey 2014)*, pages 145–151, 2014. 21

[46] Mozilla Common Voice Dataset. `https://voice.mozilla.org/en/datasets`, 2018. [Online; accessed 19-July-2018]. 23, 34, 50

[47] Kevin Walker and Stephanie Strassel. The rats radio traffic collection system. In *Odyssey 2012-The Speaker and Language Recognition Workshop*, 2012. 24, 34

[48] K. J. Han, S. Ganapathy, M. Li, M. Omar, and S. Narayanan. TRAP Language identification system for RATS phase II evaluation. In *Interspeech*. ISCA, 2013. 25

[49] Seyed Omid Sadjadi, Craig Greenberg, Elliot Singer, Douglas Reynolds, Lisa Mason, and Jaime Hernandez-Cordero. The 2018 NIST Speaker Recognition Evaluation. In *Proc. Interspeech*, pages 1483–1487, 2019. 25, 83

[50] Omid Sadjadi. NIST 2019 Speaker Recognition Evaluation: CTS Challenge - Evaluation Plan. `https://www.nist.gov/system/files/documents/2019/07/22/2019_nist_speaker_recognition_challenge_v8.pdf`, 2019. 25

[51] Hossein Zeinali, Kong Aik Lee, Jahangir Alam, and Lukas Burget. Short-duration Speaker Verification (SdSV) Challenge 2020: the Challenge Evaluation Plan. *arXiv preprint arXiv:1912.06311*, 2019. 25

[52] Mitchell McLaren, Luciana Ferrer, Diego Castan, and Aaron Lawson. The Speakers in the Wild (SITW) speaker recognition database. In *Proc. Interspeech*, pages 818–822, 2016. 25, 83

[53] Mahesh Kumar Nandwana, Julien van Hout, Colleen Richey, Mitchell McLaren, Maria A. Barrios, and Aaron Lawson. The VOiCES from a Distance Challenge 2019. In *Proc. Interspeech*, pages 2438–2442, 2019. 25, 29, 83

[54] Niko Brümmer and Johan Du Preez. Application-independent evaluation of speaker detection. *Computer Speech & Language*, 20(2-3):230–275, 2006. 28, 78, 90, 93, 104

[55] Lukáš Burget, Oldřich Plchot, Sandro Cumani, Ondřej Glembek, Pavel Matějka, and Niko Brümmer. Discriminatively trained probabilistic linear discriminant analysis for speaker verification. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP) (2011)*, pages 4832–4835. IEEE, 2011. 32, 72, 74, 75

[56] Najim Dehak, Pedro A Torres-Carrasquillo, Douglas Reynolds, and Reda Dehak. Language recognition via i-vectors and dimensionality reduction. In *INTERSPEECH (2011)*, pages 857–860, 2011. 32, 33, 51, 105

[57] Ruchir Travadi, Maarten Van Segbroeck, and Shrikanth S Narayanan. Modified-prior i-vector estimation for language identification of short duration utterances. In *INTERSPEECH (2014)*, pages 3037–3041, 2014. 32, 63

[58] Seyed Omid Sadjadi et al. The 2017 NIST language recognition evaluation. In *Odyssey Speaker and Language Recognition Workshop (2018)*, pages 82–89, 2018. 33, 34, 50, 53, 54, 58, 106

[59] Daniel Garcia-Romero and Carol Y Espy-Wilson. Analysis of i-vector length normalization in speaker recognition systems. In *INTERSPEECH (2011)*, pages 249–252, 2011. 33, 70, 74, 77, 85

[60] George Saon, Hagen Soltau, David Nahamoo, and Michael Picheny. Speaker adaptation of neural network acoustic models using i-vectors. In *ASRU (2013)*, pages 55–59, 2013. 33

[61] Yun Lei, Nicolas Scheffer, Luciana Ferrer, and Mitchell McLaren. A novel scheme for speaker recognition using a phonetically-aware deep neural network. In *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 1695–1699. IEEE, 2014. 33, 111

[62] Fred Richardson, Douglas Reynolds, and Najim Dehak. Deep neural network approaches to speaker and language recognition. *IEEE Signal Processing Letters*, 22(10):1671–1675, 2015. 33

[63] Shreyas Ramoji and Sriram Ganapathy. Supervised i-vector modeling-theory and applications. In *INTERSPEECH (2018)*, pages 1091–1095, 2018. 33, 53, 57, 58, 105

[64] Shreyas Ramoji and Sriram Ganapathy. Supervised I-vector modeling for language and accent recognition. *Computer Speech & Language*, 60:101030, 2020. 33, 105

[65] Patrick Kenny, Gilles Boulianne, and Pierre Dumouchel. Eigenvoice modeling with sparse training data. *IEEE transactions on speech and audio processing*, 13(3):345–354, 2005. 36, 113

[66] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977. 36

[67] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933. 46

[68] Jongseo Sohn, Nam Soo Kim, and Wonyong Sung. A statistical model-based voice activity detection. *IEEE signal processing letters*, 6(1):1–3, 1999. 51

[69] Bharat Padi, Shreyas Ramoji, Vaishnavi Yeruva, Satish Kumar, and Sriram Ganapathy. The LEAP Language Recognition System for LRE 2017 Challenge-Improvements and Error Analysis. In *Proc. Odyssey 2018 The Speaker and Language Recognition Workshop*, pages 31–38, 2018. 57

[70] Ming Li and Shrikanth Narayanan. Simplified supervised i-vector modeling with application to robust and efficient language identification and speaker verification. *Computer Speech & Language*, 28(4):940–958, 2014. 58, 59, 63

[71] Ruben Zazo, Alicia Lozano-Diez, and Joaquin Gonzalez-Rodriguez. Evaluation of an lstm-rnn system in different nist language recognition frameworks. In *Proc. of Odyssey 2016 Speaker and Language Recognition Workshop*. ATVS-UAM, June 2016. 58, 59

[72] Bharat Padi, Anand Mohan, and Sriram Ganapathy. End-to-end language recognition using attention based hierarchical gated recurrent unit models. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5966–5970. IEEE, 2019. 58, 59

[73] Ruben Zazo, Alicia Lozano-Diez, Javier Gonzalez-Dominguez, Doroteo T Toledano, and Joaquin Gonzalez-Rodriguez. Language identification in short utterances using long short-term memory (lstm) recurrent neural networks. *PloS one*, 11(1):e0146917, 2016. 59

[74] Laurens Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(Nov):2579–2605, 2008. 59, 98

[75] Jianbo Ma, Vidhyasaharan Sethu, Eliathamby Ambikairajah, and Kong Aik Lee. Speaker-phonetic vector estimation for short duration speaker verification. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5264–5268. IEEE, 2018. 63, 64

[76] Jianbo Ma, Vidhyasaharan Sethu, Eliathamby Ambikairajah, and Kong Aik Lee. Generalized variability model for speaker verification. *IEEE Signal Processing Letters*, 25(12):1775–1779, 2018. 63, 64

[77] Chengzhu Yu, Gang Liu, Seongjun Hahm, and John HL Hansen. Uncertainty propagation in front end factor analysis for noise robust speaker recognition. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4017–4021. IEEE, 2014. 64

[78] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210. IEEE, 2015. 65

[79] Shreyas Ramoji, Prashant Krishnan, and Sriram Ganapathy. NPLDA: A Deep Neural PLDA Model for Speaker Verification. In *Proc. Odyssey*, pages 202–209, 2020. 69, 70, 87, 89

[80] Shreyas Ramoji, Prashant Krishnan, and Sriram Ganapathy. Neural PLDA Modeling for End-to-End Speaker Verification. In *Interspeech*, pages 4333–4337, 2020. 69, 70

[81] Shreyas Ramoji, Prashant Krishnan, and Sriram Ganapathy. Plda inspired siamese networks for speaker verification. *Computer Speech & Language*, 76:101383, 2022. 69

[82] David Snyder, Daniel Garcia-Romero, Gregory Sell, Alan McCree, Daniel Povey, and Sanjeev Khudanpur. Speaker recognition for multi-speaker conversations using x-vectors. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5796–5800, 2019. 70, 84

[83] Andrew O Hatch, Sachin Kajarekar, and Andreas Stolcke. Within-class covariance normalization for svm-based speaker recognition. In *Ninth international conference on spoken language processing*, 2006. 70

[84] David A Van Leeuwen and Niko Brümmer. An introduction to application-independent evaluation of speaker recognition systems. In *Speaker classification I*, pages 330–353. Springer, 2007. 71, 79

[85] Sandro Cumani and Pietro Laface. Large-Scale Training of Pairwise Support Vector Machines for Speaker Recognition. *IEEE Transactions on Audio, Speech and Language Processing*, 22(11):1590–1600, 2014. 72

[86] Sandro Cumani and Pietro Laface. Generative pairwise models for speaker recognition. In *Proc. Odyssey*, pages 273–279, 2014. 72, 76

[87] Jesús Villalba, Nanxin Chen, David Snyder, Daniel Garcia-Romero, Alan McCree, Gregory Sell, Jonas Borgstrom, Leibny Paola García-Perera, Fred Richardson, Réda Dehak, et al. State-of-the-art speaker recognition with neural network embeddings in NIST SRE18 and Speakers in the Wild evaluations. *Computer Speech & Language*, 60:101026, 2020. 72, 84

[88] Luciana Ferrer and Mitchell McLaren. Optimizing a Speaker Embedding Extractor

Through Backend-Driven Regularization. In *Proc. Interspeech*, pages 4350–4354, 2019. 72

[89] Victoria Mingote, Antonio Miguel, Dayana Ribas, Alfonso Ortega, and Eduardo Lleida. Optimization of False Acceptance/Rejection Rates and Decision Threshold for End-to-End Text-Dependent Speaker Verification Systems. In *Proc. Interspeech*, pages 2903–2907, 2019. 72

[90] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a Siamese time delay neural network. In *Advances in Neural Information Processing Systems*, pages 737–744, 1994. 72

[91] G. Heigold, I. Moreno, S. Bengio, and N. Shazeer. End-to-end text-dependent speaker verification. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5115–5119, 2016. 72, 78

[92] Yichi Zhang, Meng Yu, Na Li, Chengzhu Yu, Jia Cui, and Dong Yu. Seq2seq attentional siamese neural networks for text-dependent speaker verification. In *ICASSP*, pages 6131–6135, 2019. 72

[93] Li Wan, Quan Wang, Alan Papir, and Ignacio Lopez Moreno. Generalized end-to-end loss for speaker verification. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4879–4883, 2018. 72, 78, 90

[94] Johan Rohdin, Anna Silnova, Mireia Diez, Oldřch Plchot, Pavel Matějka, and Lukáš Burget. End-to-end DNN based speaker recognition inspired by i-vector and PLDA. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4874–4878, 2018. 72

[95] Chunlei Zhang and Kazuhito Koishida. End-to-end text-independent speaker verification with triplet loss on short utterances. In *Proc. Interspeech*, pages 1487–1491, 2017. 72

[96] Phani Sankar Nidadavolu, Saurabh Kataria, Jesús Villalba, Paola Garcia-Perera, and Najim Dehak. Unsupervised feature enhancement for speaker verification. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7599–7603. IEEE, 2020. 72

[97] Joon Son Chung, Jaesung Huh, Seongkyu Mun, Minjae Lee, Hee-Soo Heo, Soyeon Choe, Chiheon Ham, Sunghwan Jung, Bong-Jin Lee, and Icksang Han. In Defence of Metric Learning for Speaker Recognition. In *Proc. Interspeech*, pages 2977–2981, 2020. 73, 76, 100, 101

[98] Luciana Ferrer and Mitchell McLaren. A discriminative condition-aware backend for speaker verification. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6604–6608, 2020. 73, 78, 90, 93

[99] Luciana Ferrer and Mitchell Mclaren. A Speaker Verification Backend for Improved Calibration Performance across Varying Conditions. In *Proc. Odyssey*, pages 372–379, 2020. 73, 78

[100] Luciana Ferrer, Mitchell McLaren, and Niko Brümmer. A speaker verification backend with robust performance across conditions. *Computer Speech & Language*, 71:101258, 2022. 73

[101] Aleksandr Sizov, Kong Aik Lee, and Tomi Kinnunen. Unifying Probabilistic Linear Discriminant Analysis Variants in Biometric Authentication. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 464–475. Springer, 2014. 74

[102] Shreyas Ramoji, Anand Mohan, Bhargavram Mysore, Anmol Bhatia, Prachi Singh, Harsha Vardhan, and Sriram Ganapathy. The LEAP Speaker Recognition System for NIST

SRE 2018 Challenge. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5771–5775. IEEE, 2019. 76, 85

[103] Brecht Desplanques, Jenthe Thienpondt, and Kris Demuynck. ECAPA-TDNN: Emphasized Channel Attention, Propagation and Aggregation in TDNN Based Speaker Verification. In *Interspeech*, pages 3830–3834, 2020. 76

[104] Ke Ding. A note on kaldi's plda implementation. *arXiv preprint arXiv:1804.00403*, 2018. 77

[105] Niko Brümmer and George Doddington. Likelihood-ratio calibration using prior-weighted proper scoring rules. In *Proc. Interspeech*, pages 1976–1980, 2013. 78, 121, 122

[106] Arsha Nagrani, Joon Son Chung, and Andrew Zisserman. VoxCeleb: A Large-Scale Speaker Identification Dataset. In *Proc. Interspeech*, pages 2616–2620, 2017. 83

[107] Daniel Povey, Arnab Ghoshal, and Gilles Boulianne. The Kaldi Speech Recognition Toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, 2011. 84, 85

[108] Shreyas Ramoji, V Krishnan, Prachi Singh, Sriram Ganapathy, et al. Pairwise Discriminative Neural PLDA for Speaker Verification. *arXiv preprint arXiv:2001.07034*, 2020. 85

[109] Arsha Nagrani, Joon Son Chung, Weidi Xie, and Andrew Zisserman. Voxceleb: Large-scale speaker verification in the wild. *Computer Speech & Language*, 60:101027, 2020. 100

[110] Yooyoung Lee, Craig Greenberg, Eliot Godard, Asad A Butt, Elliot Singer, Trang Nguyen, Lisa Mason, and Douglas Reynolds. The 2022 NIST Language Recognition Evaluation. *arXiv preprint arXiv:2302.14624*, 2023. 110

[111] Anna Silnova, Josef Slavicek, Ladislav Mošner, Michal Klco, Oldrich Plchot, Pavel Matejka, Junyi Peng, and Themos Stafylakis3 Lukáš Burget. ABC System Description for NIST LRE 2022. 110

[112] Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. Hubert: Self-supervised speech representation learning by masked prediction of hidden units. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:3451–3460, 2021. 111

[113] Steffen Schneider, Alexei Baevski, Ronan Collobert, and Michael Auli. wav2vec: Unsupervised Pre-Training for Speech Recognition. In *Proc. Interspeech 2019*, pages 3465–3469, 2019. 111

[114] Sanyuan Chen, Chengyi Wang, Zhengyang Chen, Yu Wu, Shujie Liu, Zhuo Chen, Jinyu Li, Naoyuki Kanda, Takuya Yoshioka, Xiong Xiao, et al. Wavlm: Large-scale self-supervised pre-training for full stack speech processing. *IEEE Journal of Selected Topics in Signal Processing*, 16(6):1505–1518, 2022. 112

[115] Niko Brummer. *Measuring, refining and calibrating speaker and language information extracted from speech.* PhD thesis, Stellenbosch: University of Stellenbosch, 2010. 121