BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

EXPLOITING UNCERTAINTY INFORMATION IN SPEAKER VERIFICATION AND DIARIZATION

VYUŽITÍ INFORMACÍ O NEJISTOTĚ V OVĚŘOVÁNÍ MLUVČÍHO A DIARIZACI MLUVČÍCH

PHD THESIS
DISERTAČNÍ PRÁCE

AUTHOR **ANNA SILNOVA**
AUTOR PRÁCE

SUPERVISOR **doc. Ing. LUKÁŠ BURGET, Ph.D.**
ŠKOLITEL

CO-SUPERVISOR **NIKO BRUMMER, Ph.D.**
ŠKOLITEL SPECIALISTA

BRNO 2022

Abstract

This thesis considers two models allowing to utilize uncertainty information in the tasks of Automatic Speaker Verification and Speaker Diarization.

The first model we consider is a modification of the widely-used Gaussian Probabilistic Linear Discriminant Analysis (G-PLDA) that models the distribution of the vector utterance representations called embeddings. In G-PLDA, the embeddings are assumed to be generated by adding a noise vector sampled from a Gaussian distribution to a speaker-dependent vector. We show that when assuming that the noise was instead sampled from a Student's T-distribution, the PLDA model (we call this version heavy-tailed PLDA) can use the uncertainty information when making the verification decisions. Our model is conceptually similar to the HT-PLDA model defined by Kenny et al. in 2010, but, as we show in this thesis, it allows for fast scoring, while the original HT-PLDA definition requires considerable time and computation resources for scoring. We present the algorithm to train our version of HT-PLDA as a generative model. Also, we consider various strategies for discriminatively training the parameters of the model. We test the performance of generatively and discriminatively trained HT-PLDA on the speaker verification task. The results indicate that HT-PLDA performs on par with the standard G-PLDA while having the advantage of being more robust against variations in the data pre-processing. Experiments on the speaker diarization demonstrate that the HT-PLDA model not only provides better performance than the G-PLDA baseline model but also has the advantage of producing better-calibrated Log-Likelihood Ratio (LLR) scores.

In the second model, unlike in HT-PLDA, we do not consider the embeddings as the observed data. Instead, in this model, the embeddings are normally distributed hidden variables. The embedding precision carries the information about the quality of the speech segment: for clean long segments, the precision should be high, and for short and noisy utterances, it should be low. We show how such probabilistic embeddings can be incorporated into the G-PLDA framework and how the parameters of the hidden embedding influence its impact when computing the likelihood with this model. In the experiments, we demonstrate how to utilize an existing neural network (NN) embedding extractor to provide not embeddings but parameters of probabilistic embedding distribution. We test the performance of the probabilistic embeddings model on the speaker diarization task. The results demonstrate that this model provides well-calibrated LLR scores allowing for better diarization when no development dataset is available to tune the clustering algorithm.

Keywords

Speaker Verification, Speaker Diarization, Probabilistic Linear Discriminant Analysis, Uncertainty Propagation, Discriminative Training

Reference

SILNOVA, Anna. *Exploiting uncertainty information in speaker verification and diarization.* Brno, 2022. PhD thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. Ing. Lukáš Burget, Ph.D.

Abstrakt

Tato práce se zabývá dvěma modely, které umožňují využít informace o nejistotě v úlohách automatického ověřování mluvčího a diarizace mluvčích.

První model, který zvažujeme, je modifikací široce používané gaussovské pravděpodobnostní lineární diskriminační analýzy (G-PLDA), modelující rozložení vektorových reprezentací promluv nazývaných embeddingy. V G-PLDA se předpokládá, že embeddingy jsou generovány přidáním šumového vektoru navzorkovaného z Gaussova rozložení k vektoru reprezentujícímu mluvčího. Ukazujeme, že za předpokladu, že šum byl místo toho vzorkován ze Studentova T-rozdělení, model PLDA (tuto verzi nazýváme PLDA s těžkým chvostem, heavy-tail, HT-PLDA) může při rozhodnutí o ověření mluvčího využít informace o nejistotě. Náš model je koncepčně podobný modelu HT-PLDA definovanému Kennym et al. v roce 2010, ale jak ukazujeme v této práci, umožňuje rychlé skórování, zatímco původní definice HT-PLDA je značně časové a výpočetně náročná. Představujeme algoritmus pro trénování naší verze HT-PLDA jako generativního modelu a zvažujeme rovněž různé strategie diskriminativního trénování parametrů tohoto modelu. Generativně a diskriminativně trénovanou HT-PLDA testujeme na úloze ověřování mluvčího. Výsledky naznačují, že HT-PLDA funguje podobně jako standardní G-PLDA, přičemž má výhodu v odolnosti vůči změnám v předzpracování dat. Experimenty s diarizací mluvčích ukazují, že HT-PLDA poskytuje nejen lepší výsledky než základní G-PLDA, ale skóre logaritmického poměru věrohodností (log-likelihood ratio, LLR) produkovaná tímto modelem jsou lépe kalibrována.

Ve druhém modelu nepovažujeme (na rozdíl od HT-PLDA) embeddingy za pozorovaná data. Místo toho jsou v tomto modelu embeddingy normálně rozložené skryté proměnné. Přesnost (precision) embeddingů nese informaci o kvalitě řečového segmentu: u čistých dlouhých segmentů by přesnost měla být vysoká a u krátkých a zašuměných promluv by měla být nízká. Ukazujeme, jak lze takové pravděpodobnostní embeddingy začlenit do skórování založeného na G-PLDA, a jak parametry skrytého embeddingu ovlivňují jeho vliv při výpočtu věrohodností s tímto modelem. V experimentech demonstrujeme, jak lze využít existující extraktor embeddingů založený na neuronové síti (NN) k produkci nikoli embeddingu, ale parametrů pravděpodobnostního rozložení embeddingu. Pravděpodobnostní embeddingy testujeme na úloze diarizace mluvčích. Výsledky ukazují, že tento model poskytuje dobře kalibrovaná skóre LLR umožňující lepší diarizaci, pokud není k dispozici vývojová datová sada pro ladění shlukovacího algoritmu.

Klíčová slova

Ověřováníí mluvčího, diarizace mluvčích, pravděpodobnostní lineární diskriminační analýza, šíření nejistoty, diskriminativní trénování

Citace

SILNOVA, Anna. *Využití informací o nejistotě v ověřování mluvčího a diarizaci mluvčích.* Brno, 2022. Disertační práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Školitel doc. Ing. Lukáš Burget, Ph.D.

Declaration

Hereby I declare that this thesis was prepared as my original work under the supervision of Dr. Lukáš Burget and Dr. Niko Brümmer. All the information derived from the published and unpublished work of others has been acknowledged in the text and included in the list of references.

....
Anna Silnova
August 2022

Acknowledgements

During my journey toward a Ph.D. degree, I have been surrounded by the encouragement and support of many people, and I would like to thank them here.

First, I would like to express my sincerest gratitude to my supervisor, Lukáš Burget, for being not only a supervisor but a great friend. His willingness to explain, discuss, and understand various complicated and not-so-complicated topics was a great support during all my years as a Ph.D. student. My second supervisor, Niko Brummer, is an infinite source of fascinating ideas. He is always open to a discussion. I want to thank Niko for making this thesis possible.

Second, I want to thank my colleagues from the Speech@FIT research group for our enjoyable time together in and out of the lab. Especially, I would like to thank past and present members of the Speaker and Language ID and Diarization sub-groups: Mireia Diez, Ondra Glembek, Federico Landini, Pavel Matějka, Lada Mosner, Ondra Novotný, Johan Rohdin, and Olda Plchot.

Special thanks belong to the head of our lab, Honza Černocký, for creating a supportive environment where I felt valued and looked after.

Then, I would like to thank the supervisor of my Master's thesis, Tomi Kinnunen, who introduced me to the world of speech technologies. Tomi provided excellent support at the start by recommending papers, explaining various complicated matters, and generally sharing his experience. Finally, he advised me to go to Brno to work on my thesis. If not for Tomi, the idea of going for a Ph.D. would never even visit my head.

Separate thanks belong to one of my long-term collaborators, Themos Stafylakis, for serving as an inspiring example of a devoted numbers-addicted researcher.

I would like to thank the members of STAR lab at SRI, especially Mitchell McLaren and Mahesh Nandwana, for the hospitable atmosphere they created during my stay at SRI.

A very special thanks belong to my family: my parents, Sergei and Oxana, and my sister Nadia. They encouraged me to start the journey towards a Ph.D. degree and never failed to invigorate me on the way. Knowing that there were people so unconditionally supporting me was always a great motivator not to give up.

Finally, I want to thank Sergei, my husband, for his infinite support. He helped me start a new life in another country, was by my side for all these years, and, finally, he seemed to believe that I could finish this thesis even at times when I did not.

Contents

List of abbreviations

AfV	Audio from Video
AHC	Agglomerative Hierarchical Clustering
ANN	Artificial Neural Network
BXE	Binary Cross-Entropy
CD	Contrastive Divergence
CRP	Chinese Restaurant Process
CTS	Conversational Telephone Speech
DCF	Detection Cost Function
DER	Diarization Error Rate
EM	Expectation Maximization
G-PLDA	Gaussian Probabilistic Linear Discriminant Analysis
GMM	Gaussian Mixture Model
GS	Gibbs Sampling
HT-PLDA	Heavy-tailed Probabilistic Linear Discriminant Analysis
IPM	Integral Probability Metric
KL	Kullback–Leibler (divergence)
L-BFGS	Limited Memory Broyden–Fletcher–Goldfarb–Shanno algorithm
LDA	Linear Discriminant Analysis
LLR	Log-Likelihood Ratio
LN	Length Normalization
LR	Logistic Regression
MAP	Maximum a Posteriori
MCL	Maximum Conditional Likelihood
MCMC	Markov Chain Monte Carlo
ML	Maximum Likelihood
MMD	Maximum Mean Discrepancy
MXE	Multi-Class Cross-Entropy
NIST	National Institute of Standards and Technology
NN	Neural Network
PCA	Principal Component Analysis
PCD	Persistent Contrastive Divergence
PDF	Probability Density Function
PLDA	Probabilistic Linear Discriminant Analysis
PSL	Pseudolikelihood
RKHS	Reproducing Kernel Hilbert Space
SD	Speaker Diarization
SDDS	Smart-Dumb/Dumb-Smart (sampling method)
SGD	Stochastic Gradient Descent

SM	Split-Merge (sampling method)
SNR	Signal to Noise Ratio
SR	Speaker Recognition
SRE	Speaker Recognition Evaluation
SV	Speaker Verification
SVM	Support Vector Machine
UBM	Universal Background Model
UPGMA	Unweighted Pair Group Method Using Arithmetic Averages
VAD	Voice Activity Detection
VB	Variational Bayes

Chapter 1

Introduction

This thesis is dedicated to the task of automatic *speaker recognition* (SR) and the closely related task of *speaker diarization* (SD). Let us consider both tasks.

Regarding SR, we concentrate on one of the forms of this problem - *text-independent speaker verification* (SV). That is, the SR system is supposed to compare two sets of audio recordings called *enrollment* and *test*. Each set is assumed to be generated by a single speaker. The task is to answer the question of whether the test set was generated by the same speaker as enrollment or whether enrollment and test were uttered by two different speakers. In this work, we discuss methods assuming that both sets, enrollment and test, are equivalent; exchanging them does not lead to a different result. A pair enrollment-test is referred to as a trial. If the speakers for both sets are the same, the trial is called *target trial*, while if the speakers are different, it is *non-target*. As follows from the name "text-independent" SV, the message conveyed by the speech should not be relevant for the final decision. On the contrary, in *text-dependent* SV, not only the identity but also the uttered phrase of the test recording has to match the identity and phrase of the enrollment recording for the trial to be considered target. We do not consider text-dependent SV in this thesis.

The task of Speaker Diarization is to annotate an audio recording that might contain speech of many speakers with speaker labels. In other words, the diarization system is supposed to break the recording into speaker-homogeneous regions and indicate which regions belong to the same speaker and which come from different speakers.

1.1 Classical approach to SV and SD

A typical modern SV system consists of three stages: frame-level acoustic feature extraction, utterance-level embedding extraction, and the back-end comparing the embeddings. The first stage converts an audio signal into a sequence of feature vectors, each of them extracted from a short excerpt from the speech where the acoustic properties of the signal are assumed to be constant. Usually, frame features are extracted every 10 ms, and the length of a single frame is 20 ms. The output of the feature-extraction stage for a single recording is a matrix where each row corresponds to one frame from the utterance. Thus, the number of rows in the feature matrices is different for recordings of different lengths. Direct usage of the acoustic feature matrices for speaker recognition is impractical because of the large amount of data and inconsistency in matrix sizes for the signals of different lengths. For these reasons, the set of acoustic features is further processed by a model transforming them into a fixed-length utterance-level vector representation, i.e., after this modeling stage, each

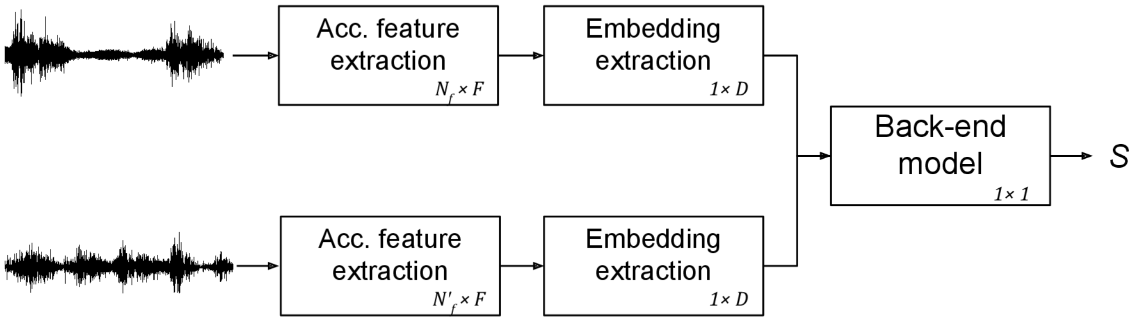

Figure 1.1: Typical speaker verification pipeline. The subscripts in the blocks correspond to the dimensionality of the output of the current stage.

audio recording is represented by a single vector of fixed size called embedding. A more detailed overview of the embedding extraction techniques is given in Section 2.4. Finally, embeddings of different utterances are compared using a back-end model. The general scheme of the speaker verification pipeline is shown in Figure 1.1. In this work, we adopt the same three-stage processing; however, we primarily concentrate on the last stage, the back-end model.

A similar approach is used in most of the diarization systems. As in SV, frame-level features are extracted for the recording of interest. Then, instead of extracting a single embedding for the whole recording, the audio is split into short segments, and an individual embedding is extracted for each of them. Finally, these embeddings are compared by the back-end model as in SV. The difference between speaker recognition and diarization is that instead of comparing a pair of (sets of) embeddings, one is interested in clustering or partitioning all of the embeddings from the same recording into speaker-homogeneous clusters. This clustering usually requires modeling the embeddings extracted from the recording; the same back-end models that are used in SV can be applied for the SD task. Hence, the back-end models we discuss in this thesis would be equally suitable for both SV and SD.

1.2 Motivation and outline of the thesis

Our primary motivation for this work is to utilize uncertainty information when making a speaker verification decision. That is, the confidence of the SR system should be affected by the quality of the input audio. Intuitively, short audio of poor quality should be trusted less than a long clean recording. To this end, we have worked in two main directions:

First, we start with the back-end model aimed to work with the audio recordings represented as embeddings. We present a variant of a heavy-tailed PLDA (HT-PLDA) model. We show how this model allows for better data-dependent uncertainty propagation compared to a widely used Gaussian PLDA (G-PLDA). Our model formulation allows for a relatively fast scoring, and we mitigate excessive computational requirements that were the main drawback of the original HT-PLDA [Kenny, 2010]. HT-PLDA is a generative model of the observed data, i.e., it represents our beliefs of how the data (embeddings) were generated. Here, we present two main strategies for training it. The first one is generative training, i.e., the parameters of HT-PLDA are estimated so that the model fits the training data well. That is, the parameters are chosen to maximize the likelihood of the observed training data. On the other hand, the discriminative approach does not aim at modeling the observed data. Instead, the parameters of the model are learned to optimize a pre-defined objective function which is believed to correlate well with the performance of the model on the task of interest. Usually, the model trained discriminatively does not reflect our beliefs on the underlying process generating the data. Instead, it is selected to model the separation boundary between the classes (e.g., target vs. non-target trials in SV). In our case, however, we keep the functional form of HT-PLDA even for discriminative training. We use various objective functions to learn model parameters.

HT-PLDA is a back-end model, and each audio recording is represented by an embedding vector (HT-PLDA treats the embeddings as the observed data). Even though it allows for better uncertainty propagation than a conventional G-PLDA, still, it has limited abilities to utilize the uncertainty since pre-extracted embeddings have already lost most of this information. The only source of uncertainty available to the HT-PLDA model is the distribution of the embeddings. Hence, we move to the second approach to utilize the uncertainty: we assume that the observed data are sets of acoustic features, while embeddings participate in the model as hidden variables. Then, instead of extracting a "point" vector embedding per recording, we extract the parameters of the embedding distribution. This distribution should be sharp for long high-quality audio recordings, i.e., the model should be certain where the true embedding is for such recordings; for short and noisy recordings, the embedding distribution should be flat - there is high uncertainty about where the embedding for such recording should be. In our approach, the parameters of the embedding distributions are estimated by a neural network (NN). We construct the network by utilizing the existing NN embedding extractor (the original embedding is used to infer the mean of the probabilistic embedding) and augmenting it with several additional blocks responsible for estimating the precision of the embedding distribution. We model the distribution of hidden embeddings with a Gaussian PLDA model. We train both G-PLDA and probabilistic embedding extractor jointly in a discriminative way.

1.3 Structure of the thesis

The rest of this thesis is organized as follows. Chapter 2 presents some general notes on SV and SD, including common data assumptions, standard approaches to solving SV and SD, and performance metrics to evaluate the quality of the model. Then, in Chapter 3, we discuss the PLDA model, its commonly-used variant Gaussian PLDA (G-PLDA), and introduce PLDA with heavy-tailed noise (HT-PLDA). This chapter is based on our previously published works [Brümmer et al., 2018a, Brümmer et al., 2018b]. Chapter 4 presents the HT-PLDA model trained as a generative model along with experiments and results achieved with it. This chapter expands on both theoretical and practical parts of our work [Silnova et al., 2018]. Then, Chapter 5 introduces a few options of estimating HT-PLDA parameters discriminatively. For each method, we pair the theoretical introduction with the section describing the corresponding experimental setup and results. Some of the discriminative approaches to training the HT-PLDA model were published before in [Brümmer et al., 2018b, Silnova et al., 2020], others are summarized here for the first time. In Chapter 6, we describe a model utilizing probabilistic embeddings initially presented in [Silnova et al., 2020][1], introduce the training strategy adopted in our experiments, and present experimental results of this model on the SD task. Finally, Chapter 7 concludes the thesis and provides some possible future work directions.

[1] [Silnova et al., 2020] received the inaugural Jack Godfrey Best Student Paper Award at Odyssey: The Speaker and Language Recognition Workshop, 2020, Tokyo, Japan.

1.4 Claims of the thesis

The main contributions of this thesis can be summarized in the following points:

- **PLDA with heavy-tailed noise**: We introduce a modification to a widely-used Gaussian PLDA model – HT-PLDA. Unlike the earlier version of HT-PLDA, our modification allows for fast scoring of speaker verification trials. Besides that, we demonstrate the mechanism of propagating the uncertainty in HT-PLDA.

- **Generative training recipe for HT-PLDA**: We derive a fast algorithm to learn the parameters of the HT-PLDA model. The experiments on the speaker verification task show that generatively trained HT-PLDA performs on par with the Gaussian PLDA baseline. Even though HT-PLDA does not always bring a performance gain compared to G-PLDA, it is more robust to variations in the input data.

- **Analysis of various discriminative training strategies for HT-PLDA**: We describe four different approaches to discriminative learning of the HT-PLDA parameters. The methods are tested on the speaker verification and speaker diarization tasks. The results of different training strategies are analyzed and compared.

- **Derivation of algorithms to sample from partition posterior**: One of the introduced discriminative training approaches requires sampling from the posterior distribution of possible partitions of training data into speaker clusters. We describe various sampling algorithms and derive the particulars for the HT-PLDA model.

- **Probabilistic speaker embeddings**: We introduce a model that considers the embeddings as hidden variables. We derive the formulae for computing the likelihood with the model and propose a discriminative training strategy to estimate the model parameters. Finally, we test the model performance on the speaker diarization task.

Chapter 2

Preliminaries of speaker verification and speaker diarization

2.1 Data assumptions

We start by introducing the assumptions we make about the data we are dealing with. Generally speaking, the data used for SR come from two categories: Training (Development) and Evaluation data. For both of them, the same assumptions are made:

- The data are in a form of a collection or set $\mathcal{R}$ of N audio recordings r_i: $\mathcal{R} = \{r_1, \ldots, r_N\}$. Recordings can be represented differently, for example, as raw audio, sequences of acoustic features, embeddings, etc. Each recording contains speech of a single person.

- Recordings of different speakers are independent of each other.

- Recordings of the same speaker are interchangeable (their order does not matter).

Additionally, we assume that for the training data, speaker labels are known and given as a vector $\mathcal{L} = \{l_1, \ldots, l_N\}$, where each l_i is an integer corresponding to r_i. $\mathcal{L}$ defines a partition of audio recordings from the dataset $\mathcal{R}$ into speaker clusters. Hence, we refer to it as a *speaker partition*. For the evaluation data, the task is to infer the labels.

Many probabilistic approaches and models in speaker recognition, including the PLDA model described later, make use of the above-mentioned assumptions. The assumptions are mathematically convenient, however, they might not be very realistic.

By De Finetti's theorem [Chow and Teicher, 1997], these assumptions are equivalent to the concept of the *hidden speaker identity variable*. For each speaker, we assume the existence of a hypothetical variable carrying the information about the speaker's identity. We assume that this variable is a real-valued d-dimensional vector $\mathbf{z} \in \mathbb{R}^d$. Such vectors are unobserved; hence, the name "hidden" speaker variables/vectors. Vectors $\mathbf{z}$ are uniquely assigned to speakers, so speaker identity can be perfectly deduced from its hidden vector if it is known. The observed data are assumed to be generated by some underlying process that uses a single speaker hidden variable to generate a single recording (the assumption for the recording to contain speech of a single speaker). Due to the exchangeability assumption, recordings of the same speaker (meaning they share the same $\bar{\mathbf{z}}$) are conditionally independent given $\mathbf{z}$.

$$P(\mathcal{R}_i \mid \bar{\mathbf{z}}) = \prod_{j=1:N_i} P(r_{ij} \mid \bar{\mathbf{z}}). \tag{2.1}$$

Above, $\mathcal{R}_i$ is a set of N_i recordings from $\mathcal{R}$ known to belong to speaker i.

As $\mathbf{z}$ is hidden, the exact value $\bar{\mathbf{z}}$ generating the data for a given speaker in the above equation is unknown. Hence, to compute the likelihood of the recordings from $\mathcal{R}_i$, one has to marginalize over all possible values of $\mathbf{z}$ generated from some prior distribution $\pi(\mathbf{z})$. In this document, we always assume standard normal prior $\pi(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid \mathbf{0}, \mathbf{I})$.

$$P(\mathcal{R}_i \mid H_s) = \Big\langle \prod_{j=1:N_i} P(r_{ij} \mid \mathbf{z}) \Big\rangle_{\pi(\mathbf{z})}. \tag{2.2}$$

Here and in the following, triangle brackets denote expectation. H_s denotes the hypothesis that all of the recordings r_{ij} from $\mathcal{R}_i$ belong to the same speaker i, i.e., they share the same speaker variable $\mathbf{z}$.

Finally, due to the between-speaker independence assumption:

$$P(\mathcal{R} \mid \mathcal{L}) = \prod_{i=1}^{m} P(\mathcal{R}_i|H_s) = \prod_{i=1}^{m} \left\langle \prod_{r \in \mathcal{R}_i} P(r \mid \mathbf{z}) \right\rangle_{\pi(\mathbf{z})}. \tag{2.3}$$

Expression (2.3) provides the likelihood for any specific partition $\mathcal{L}$ (defining m speaker clusters) of a set of recordings $\mathcal{R}$. However, this expression makes sense only in combination with some probabilistic model $\mathcal{M}$ (it can be either a true model generating the data or some approximation to it). For the real audio data, the true model is unknown (most likely, it even does not exist as our data assumptions are obviously not correct); so, we assume that $\mathcal{M}$ is some human-designed model (e.g., PLDA model as will be discussed in detail in Sections 3.1 and 3.2). Then, (2.3) evaluated for the training data would allow us to assess how likely the correct partition of the data $\mathcal{L}^*$ is, compared to other partitions $\mathcal{L}$ given the model $\mathcal{M}$.

However, to be able to evaluate (2.3), one has to make sure that two operations are tractable. First, there should be a tractable way to pool together recordings hypothesized to belong to the same speaker. Second, computing the expectation over the prior of the hidden variable should be feasible. Let us assume for now, that the model $\mathcal{M}$ allows for performing these operations.

2.2 Speaker verification

2.2.1 Obtaining a speaker verification score

Given the model $\mathcal{M}$, one can use (2.3) to evaluate the likelihood of the evaluation recordings given some hypothetical partition $\mathcal{L}$. And, what is more relevant for the SV task, two such partitions can be compared. Suppose, we are given some alternative partition $\mathcal{L}'$ of the data $\mathcal{R} = \{\mathcal{R}'_1, ..., \mathcal{R}'_{m'}\}$, then, we can form a likelihood ratio:

$$\frac{P(\mathcal{R} \mid \mathcal{L})}{P(\mathcal{R} \mid \mathcal{L}')} = \frac{\prod_{i=1}^{m} \left\langle \prod_{r \in \mathcal{R}_i} P(r \mid \mathbf{z}) \right\rangle_{\pi(\mathbf{z})}}{\prod_{i=1}^{m'} \left\langle \prod_{r \in \mathcal{R}'_i} P(r \mid \mathbf{z}) \right\rangle_{\pi(\mathbf{z})}}. \tag{2.4}$$

Logically, if the value of the likelihood ratio is higher than 1, the partition $\mathcal{L}$ is more likely to be a true partition of the data than $\mathcal{L}'$ (given model $\mathcal{M}$). If it is smaller than 1, the opposite is true.

For practical reasons, log-likelihood ratios are often preferred; the above formula in a logarithmic form is:

$$\log\left(\frac{P(\mathcal{R} \mid \mathcal{L})}{P(\mathcal{R} \mid \mathcal{L}')}\right) = \sum_{i=1}^{m} \log\left\langle \prod_{r\in\mathcal{R}_i} P(r \mid \mathbf{z})\right\rangle_{\pi(\mathbf{z})} - \sum_{i=1}^{m'} \log\left\langle \prod_{r\in\mathcal{R}'_i} P(r \mid \mathbf{z})\right\rangle_{\pi(\mathbf{z})}. \tag{2.5}$$

(2.4) or (2.5) allow comparing any pair of possible data partitions. However, in speaker verification, which is the main task of this work, the partitions of interest are very specific: either two sets of recordings $\mathcal{R}_1, \mathcal{R}_2$ (in the simplest case, these are individual recordings) come from the same speaker ($\mathcal{L}$ or H_s denoting same-speaker hypothesis) or from two different speakers ($\mathcal{L}'$ or H_d denoting different-speaker hypothesis). For this particular problem, (2.5) is:

$$\begin{aligned}\log\left(\frac{P(\mathcal{R}_1, \mathcal{R}_2 \mid \mathcal{L})}{P(\mathcal{R}_1, \mathcal{R}_2 \mid \mathcal{L}')}\right) &= \log\left\langle \prod_{r\in\mathcal{R}_1,\mathcal{R}_2} P(r \mid \mathbf{z})\right\rangle_{\pi(\mathbf{z})} \\ &\quad - \log\left\langle \prod_{r\in\mathcal{R}_1} P(r \mid \mathbf{z})\right\rangle_{\pi(\mathbf{z})} - \log\left\langle \prod_{r\in\mathcal{R}_2} P(r \mid \mathbf{z})\right\rangle_{\pi(\mathbf{z})}.\end{aligned} \tag{2.6}$$

(2.6) gives so-called verification score (*log-likelihood ratio score* (LLR)). The verification decision in SV depends on the score. The score reflects confidence in the same-speaker hypothesis. The higher the score, the more sure the system is that the trial is a target trial. The opposite is also true: lower scores correspond to the system being more sure in the trial to be non-target. Hard decisions are made by introducing a threshold: if the score is higher than its value, the trial is considered to be a target trial (system makes an *accept* decision); if it is lower than the threshold, the trial is non-target (*reject*).

LLR (2.6) can be used for both: to evaluate speaker verification performance of a trained model, or to train a model discriminatively (Section 5.2). More general (2.5) can also be used for some of the discriminative training strategies (Sections 5.3, 5.4, and 5.5).

2.2.2 Speaker verification performance metrics

At the end of the previous section, we mentioned how the LLR score could be used to decide whether the trial should be accepted or rejected. The question of this section is how to quantify the performance of the speaker verification system (model $\mathcal{M}$). To be able to do it, one has to have a labeled evaluation set $\mathcal{T}$ in the form of a collection of trials $t \in \mathcal{T}$ with the corresponding target/non-target labels. The goal of the system is to assign the correct labels to all trials in $\mathcal{T}$.

By fixing a threshold τ, one obtains a hard decision for each trial $t \in \mathcal{T}$ as described in Section 2.2.1. As an outcome, there are four categories of trials. The first two categories consist of correctly classified trials: correctly accepted target trials and correctly rejected non-target trials. The third category are the target trials which were rejected (so-called *false reject* or *miss errors*); suppose there is N_{miss} of them. And, finally, the fourth category are those trials which were incorrectly classified as targets (*false accept* or *false alarm*); N_{fa} of them.

Errors made by the system come from the trials of the last two categories. Then, for a given trial set, miss and false alarm rates are:

$$R_{\text{miss}} = \frac{N_{\text{miss}}}{N_{\text{tar}}} \tag{2.7}$$

$$R_{\text{fa}} = \frac{N_{\text{fa}}}{N_{\text{non}}}, \tag{2.8}$$

where N_{tar} and N_{non} are the total numbers of target and non-target trials in set $\mathcal{T}$. Note that the actual accept and reject decisions depend on the threshold value τ; by changing it, the number of trials in each category and, consequently, error rates change.

Equal Error Rate (EER) is a common metric used to summarize the performance of an SV system; it is defined as the false alarm or miss rate when the threshold $\tau = \hat{\tau}$ is set in such a way that these two error rates are equal:

$$\text{EER} = R_{\text{fa}}(\hat{\tau}) = R_{\text{miss}}(\hat{\tau}). \tag{2.9}$$

EER is one of the metrics we use to track the performance of the speaker verification systems in this thesis. However, when computing EER, both kinds of error are treated in the same way, while in some applications, one type of error might be more acceptable than the other. Hence, we also use another metric that takes this consideration into account.

Depending on the application, each type of error has a cost associated with it. Then, the goal is to minimize the total cost caused by the system's errors. Different systems can be compared with these total costs. To formalize this idea, let us introduce a *Detection Cost Function* (DCF) metric defined by American *National Institute of Standards and Technology* (NIST) [Martin and Greenberg, 2010] to evaluate the performance of a speaker verification system at a fixed operating point. By the operating point, we mean costs associated with errors of each type and prior probabilities of target/non-target trials.

DCF is defined as a weighted sum of two error probabilities:

$$C_{\text{Det}} = C_{\text{miss}} P(\text{error} \mid \text{tar}) P_{\text{tar}} + C_{\text{fa}} P(\text{error} \mid \text{non}) P_{\text{non}}. \tag{2.10}$$

To compute the value of the cost C_{Det} for trial set $\mathcal{T}$ and threshold τ, error probabilities $P(\text{error} \mid \text{tar})$ and $P(\text{error} \mid \text{non})$ are replaced with miss and false alarm rates R_{miss} and R_{fa} as they are the maximum likelihood estimates of the true probabilities. C_{miss} and C_{fa} define relative costs associated with each error type. $P_{\text{tar}} = P(H_s)$ is a prior probability of the same-speaker hypothesis (target trial), while $P_{\text{non}} = 1 - P_{\text{tar}} = P(H_d)$ is a prior to obtain a non-target trial. Together, a triplet $(C_{\text{miss}}, C_{\text{fa}}, P_{\text{tar}})$ defines an operating point of interest. These parameters are set by the operator of the SV system based on a target application.

To improve the interpretability, DCF is normalized with the best default value C_{default} that can be obtained without processing the data, i.e., always accept or reject every trial, whichever results in a lower cost:

$$\begin{aligned} C_{\text{default}} &= \min\{C_{\text{miss}} P_{\text{tar}}, C_{\text{fa}} P_{\text{non}}\}, \\ C_{\text{norm}} &= \frac{C_{\text{Det}}}{C_{\text{default}}}. \end{aligned} \tag{2.11}$$

Then, the value of $C_{\text{norm}} = 1$ corresponds to a "dumb" system which assigns the more likely (taking the costs into account) label to all of the trials; $C_{\text{norm}} = 0$ to a system that does not make any errors.

Similarly to error rates, DCF depends on a threshold value used to compute it. When the threshold is fixed by the system operator, the value of DCF is referred to as an *actual* DCF. An oracle metric called *minimum Detection Cost Function* (minDCF) is defined as a minimum possible value of DCF achieved by setting the threshold optimally:

$$\min \mathrm{DCF} = \min_{\tau} C_{\mathrm{norm}}(\tau). \tag{2.12}$$

Unlike EER, minDCF takes into account the relative importance of the two errors. However, it still does not provide a direct way to evaluate how the system would behave when applied to a real task (with an arbitrarily set threshold).

However, if the system is known to output the log-likelihood ratio scores, there is a way to analytically set the threshold for any operating point $(C_{\mathrm{miss}}, C_{\mathrm{fa}}, P_{\mathrm{tar}})$. The optimal threshold τ^* is selected to minimize the expected DCF. When the accept decision for a trial t is made, the probability of making an error is the probability of a false alarm, i.e., that the trial is non-target $P(\mathrm{non} \mid t)$. Conversely, when t is rejected, the system makes a miss error with probability $P(\mathrm{tar} \mid t)$. Consequently, the corresponding expected costs for trial t are:

$$\begin{aligned} \langle C \rangle_{\mathrm{accept}} &= C_{\mathrm{fa}} P(\mathrm{non} \mid t) = C_{\mathrm{fa}} P(t \mid \mathrm{non}) \frac{P_{\mathrm{non}}}{P(t)} \\ \langle C \rangle_{\mathrm{reject}} &= C_{\mathrm{miss}} P(\mathrm{tar} \mid t) = C_{\mathrm{miss}} P(t \mid \mathrm{tar}) \frac{P_{\mathrm{tar}}}{P(t)}. \end{aligned} \tag{2.13}$$

$P(t)$ in the expression above is the prior probability of a particular trial t. In practice, this quantity is impossible to evaluate, but as we will see shortly, that is not needed as $P(t)$ cancels out in the threshold expression.

To minimize the expected cost, we have to accept when $\langle C \rangle_{\mathrm{accept}} < \langle C \rangle_{\mathrm{reject}}$ and reject when $\langle C \rangle_{\mathrm{accept}} > \langle C \rangle_{\mathrm{reject}}$. Let us look at the case when the expected costs for both decisions are equal:

$$C_{\mathrm{fa}} P(t \mid \mathrm{non}) \frac{P_{\mathrm{non}}}{P(t)} = C_{\mathrm{miss}} P(t \mid \mathrm{tar}) \frac{P_{\mathrm{tar}}}{P(t)}. \tag{2.14}$$

By reorganizing the terms and taking a logarithm of both parts:

$$\log \frac{P(t \mid \mathrm{tar})}{P(t \mid \mathrm{non})} = \log \frac{C_{\mathrm{fa}}(1 - P_{\mathrm{tar}})}{C_{\mathrm{miss}} P_{\mathrm{tar}}} = \tau^*. \tag{2.15}$$

On the left side of (2.15), there is a log-likelihood ratio score, and, on the right side, there is a value of the score when both hypotheses are equally likely, i.e., the right side defines an optimal threshold τ^* if the system outputs true LLR scores. In this case, the values of DCF computed at τ^*, and minDCF are the same.

However, even with the probabilistic model that allows us to calculate LLR scores, the actual values of the scores are not well-calibrated LLRs because of the mismatch between model assumptions about the data and the actual data distribution. In such a case, fixing the threshold to τ^* would result in a higher value of DCF than minDCF. If so, the system is said to be miscalibrated, and the difference between actual and minimum DCFs is referred to as a calibration loss. DCF with the analytically set threshold (2.15) is the primary metric in NIST *speaker recognition evaluations* (SRE) [Lee et al., 2020]. This metric encourages the system to output the true log-likelihood ratio scores. Eventual adjustments in different

Table 2.1: DCF parameters used by NIST in speaker recognition evaluations of different years. Two DCFs were computed and averaged as a primary metric for cases when two sets of parameters were used.

Year	Task	C_{miss}	C_{fa}	P_{tar}
2016	-	1	1	0.01
		1	1	0.005
2018	Conversational Telephone Speech (CTS)	1	1	0.01
		1	1	0.005
	Audio from Video (AfV)	1	1	0.05
2019	Conversational Telephone Speech (CTS)	1	1	0.01
		1	1	0.005
	Audio from Video (AfV)	1	1	0.05

editions of the evaluations are made to the operating points of interest by setting parameters C_{miss}, C_{fa}, and P_{tar}.

The particular values of the parameters used by NIST in different editions of the SRE are given in Table 2.1. When two sets of parameters are given for a single task, the primary metric (C^{Prm}) was defined as an average of two DCFs computed at the two operating points. In this work, when evaluating the performance on the telephone condition, we adopt the parameters as they were defined by NIST in 2016, 2018 CTS, and 2019 CTS [Sadjadi et al., 2017, Sadjadi et al., 2019, Sadjadi et al., 2020]: for the telephone task, we report $C^{\text{Prm}}_{\text{min}}$ as the average minimum detection cost function for two operating points with the probability of target trials $P_{\text{tar}} = 0.01$ and $P_{\text{tar}} = 0.005$. When the performance is evaluated on Audio from Video data, $C^{\text{Prm}}_{\text{min}}$ is minDCF computed with the probability of target trial $P_{\text{tar}} = 0.05$.

Usually, the last step of the speaker verification system is the calibration. Its purpose is to minimize the calibration loss and to transform the scores output by the system to true LLRs. The common calibration strategy is a linear calibration:

$$s^* = as + b. \tag{2.16}$$

The new value of a score s^* is a linear transformation of the original score s. Parameters of the transformation a and b are shared across all scores. They are trained to optimize the Binary Cross-Entropy objective (as in Section 5.2) on a held-out set of development scores with known target/non-target labels. Such optimization results in a and b that make the scores closer to the true LLR for a wide range of operating points [Brümmer, 2010b].

As this calibration strategy is pretty simple and works well, the actual value of DCF is often not closely tracked at the time of system development. Instead, minDCF is watched and compared across systems. The hope is that the calibration step will mitigate the gap between actual and minimum DCF. In this work, we adopt the same strategy and track the performance of the speaker verification systems in terms of minDCF and EER.

2.3 Speaker diarization

The task of a diarization system is to split the audio recording containing the speech of one or several speakers into speaker-homogeneous regions and then cluster these regions into speaker clusters.

In general, speakers can overlap, leading to regions with more than one speaker talking simultaneously. However, here, we do not account for such a possibility. We assume that at any time, only one person is speaking or there is silence. We understand that this assumption is unrealistic and will lead to errors in the output of the diarization system due to overlapping speech. To reduce it, some separate approach to handle overlapped speech would be needed; we do not consider any of them in this thesis.

In the following Sections 2.3.1 and 2.3.2, we describe one general approach to diarization and a metric used to assess the performance of the diarization system.

2.3.1 Diarization pipeline

First, let us note that there are various strategies for diarization. Here, we concentrate on just one of them used in our work.

We perform diarization by the following procedure:

- Perform *voice activity detection* (VAD). For our experiments, we use ground truth VAD labels.

- Split the speech regions of the test utterance into short overlapping segments. We use 1.5 s long segments with 0.75 s overlap in this work. If the speech region is shorter than 1.5 s, the whole segment is used.

- For each of the segments, extract an embedding representation. For example, this might be one of the embeddings described in Section 2.4. In Chapter 6, we discuss an alternative to the commonly used vector embeddings (probabilistic embeddings utilizing the uncertainty information).

- Cluster the embeddings so that the segments of the same speaker are put to the same cluster. One of the clustering techniques widely used in diarization is a greedy algorithm called *Agglomerative Hierarchical Clustering* (AHC) [Jain and Dubes, 1988]. Below, we present it in more detail.

- Once the clustering of the embeddings is done, the performance of the diarization can be evaluated as described in Section 2.3.2.

Agglomerative Hierarchical clustering

AHC belongs to the family of hierarchical clustering methods. These methods transform a matrix of pairwise distances (or similarities) between the elements into a sequence of nested partitions. Hierarchical methods are usually split into two large groups: divisive and agglomerative. Divisive methods start with assigning all data points to a single cluster and then perform splitting until a stopping criterion is satisfied. On the contrary, agglomerative clustering starts from the finest partition of the data and then gradually merges clusters. Usually, in agglomerative methods, the stopping criterion is defined by setting a similarity threshold value σ: once all of the inter-cluster similarities are lower than the threshold, the clustering procedure stops. σ is a hyperparameter of the algorithm and has to be tuned to achieve good clustering.

AHC is formally presented in Algorithm 1, where we use the following notation: $\mathcal{R}_i$ is a cluster containing zero, one, or more datapoints r_j. At the initialization, all clusters $\mathcal{R}_i$ are

Algorithm 1: AHC

```
Initialize clusters and similarity matrix:
  $\mathcal{R}_i = r_i$, $\forall i$;
  $S_{ij} = s(r_i, r_j)$, $\forall i, j \neq i$;
  $S_{ii} = -\infty$;
while $S_{ij} > \sigma$, $\forall i, j$ do
    Find two closest clusters to merge:
      $i^*, j^* = \text{argmax}_{ij} S_{ij}$;
      Let $i^* < j^*$, then $\mathcal{R}_{i^*} = \mathcal{R}_{i^*} \cup \mathcal{R}_{j^*}$, $\mathcal{R}_{j^*} = \emptyset$;
    Update similarity matrix:
      $S_{i^*k} = S_{ki^*} = s(\mathcal{R}_{i^*}, \mathcal{R}_k), \forall k \neq i^*$;
      $S_{kj^*} = S_{j^*k} = -\infty$, $\forall k$
end
```

singletons. The similarity between clusters $\mathcal{R}_i$ and $\mathcal{R}_j$ is denoted as $s(\mathcal{R}_i, \mathcal{R}_j)$, similarity between any cluster and an empty cluster is assumed to be $-\infty$. By $s(r_i, r_j)$, we denote a similarity between individual data points. Matrix S is composed of similarity scores between all pairs of clusters currently obtained by the algorithm. It is initialized with the pairwise similarities of the individual data points and updated as the algorithm progresses.

A particular implementation of the AHC algorithm depends on the definition of pairwise point similarity and inter-cluster similarity. Regarding inter-cluster similarities, there are several approaches. For example, single-link AHC [Sneath et al., 1973] defines this similarity as the maximum of pairwise similarities of individual elements of two clusters:

$$s(\mathcal{R}_1, \mathcal{R}_2) = \max_{i,j} s(r_i, r_j) : r_i \in \mathcal{R}_1, r_j \in \mathcal{R}_2.$$

Complete-link AHC [King, 1967] defines it as the minimum of all pairwise similarities between two clusters:

$$s(\mathcal{R}_1, \mathcal{R}_2) = \min_{i,j} s(r_i, r_j) : r_i \in \mathcal{R}_1, r_j \in \mathcal{R}_2.$$

In diarization, *unweighted average linkage* AHC also known as *unweighted pair group method using arithmetic averages* (UPGMA) is often used [Sell et al., 2018]. In UPGMA, the similarity between two clusters is computed as an average pairwise similarity:

$$s(\mathcal{R}_1, \mathcal{R}_2) = \frac{1}{N_1 N_2} \sum_i \sum_j s(r_i, r_j), \forall r_i \in \mathcal{R}_1, r_j \in \mathcal{R}_2,$$

where N_1 and N_2 are the numbers of points in two clusters.

Here, we assume that the pairwise similarities between individual points are LLRs (2.6) evaluated given some back-end model $\mathcal{M}$ (we discuss different options for a model $\mathcal{M}$ in Sections 3.1, 3.2, and 6.1). Assuming such similarity scores, we are not limited to UPGMA or other generic algorithms. In fact, we can compute the proper LLR scores (2.6) for two sets of segments to belong to the same speaker class or two different ones. In terms of Speaker Verification, it would correspond to evaluating a score for a multi-enrollment/multi-test trial. From the theoretical point of view, this approach should be better as the scores at any stage of the algorithm have probabilistic interpretation, while we cannot say the same about intermediate scores used in UPGMA. Moreover, as discussed in [Silnova et al., 2020], such an approach corresponds to a greedy maximization of the log-likelihood of the correct segment clustering. At each stage of the algorithm, the current clustering is updated with one of the possible clusterings (those that differ from the current one only by having two clusters merged) such that the total log-likelihood of the data is maximized. We use this version of AHC in the experiments presented in this work.

2.3.2 Diarization performance metric

The result of the procedure described in the previous section is a sequence of speech segments with cluster labels assigned to them. Merging consecutive segments belonging to the same cluster, one gets the final diarization output. The question arises on how to evaluate the quality of the diarization, i.e., how to compare it with the reference ground truth label for the utterance. Here, we discuss a widely used (including this work) performance metric to evaluate the diarization quality.

As in the case of Speaker Verification, we follow the performance metric introduced by NIST for its Rich Transcription Meeting Recognition Evaluation [Fiscus et al., 2006]. The metric is called *Diarization Error Rate* (DER) and is computed for each test utterance individually.

First, all possible mappings between the system output labels and the reference labels are found. One mapping sets the correspondence between speaker clusters from the system output and the ground truth speaker labels. Then, for each mapping, DER is computed as described below; the lowest DER is used as the final performance metric.

For a fixed mapping, three components of the error made by the system are:

- The *miss* error (t_m) - total duration of the speech not assigned to any of the speakers.
- The *false alarm* error (t_{fa}) - total duration of the silent segments assigned some speaker label.
- The *speaker error* (t_s) - total duration of speech segments assigned to the wrong speaker.

The total error time is the sum of the three components. To compensate for the difference in length of individual utterances, the error time is normalized by the total speech duration T as defined by the reference labels.

$$\mathrm{DER} = \frac{t_m + t_{fa} + t_s}{T}. \tag{2.17}$$

Defined in this way, DER has the following properties. Perfect diarization output corresponds to DER = 0, and the empty hypothesis (not assigning any speaker labels for the whole utterance) results in DER = 1. However, one should notice that 1 is not an upper bound for DER. For example, if the system believes there are regions of overlapped speech when there are not, DER can be higher than 1.

Reference labels are the result of human annotation, and due to that, they might have certain problems. Precisely marking a border between two speakers or speech and silence is a difficult task, and there is no certainty that the annotator can perform it perfectly. Consequently, there will be errors impacting the total DER because a human annotator and a system disagree on the speech boundaries, which might be not the real errors. To compensate for the possible imprecision of annotators, NIST introduced a forgiveness interval ("collar") of 250 ms on each side of the reference speech boundary that is excluded from the DER computation. In this work, we do not use such intervals, i.e., the exact boundaries are used when computing DER. We refer interested readers to [Landini et al., 2022] for a thorough discussion of collars in DER evaluation.

2.4 Embedding extactors

As was mentioned in the introduction, embeddings are low-dimensional representations of audio recordings. Typically, they are derived from matrices of frame-by-frame acoustic features and used in their place to represent the audio. In this section, we briefly introduce two popular types of embeddings used in speaker recognition tasks and this thesis: i-vectors and x-vectors.

2.4.1 I-vectors

The i-vector paradigm emerged from the series of works on speaker recognition. The foundation for it was laid with the introduction of *Universal Background Model* (UBM) [Reynolds et al., 2000]. UBM represents a speaker-independent distribution of acoustic features, in other words, it intends to model the universe of all possible speakers. It is a *Gaussian Mixture Model* (GMM) trained on large amounts of data coming from many speakers. Individual Gaussians in the mixture represent some arbitrary phonetic or acoustic classes. The typical size of GMM used in speaker recognition applications is 1024-4096 components in the mixture. Parameters of the model θ (mixture weights, means, and covariance matrices of individual components) are optimized to maximize the likelihood of the training data. Typically, θ is learned by *Expectation-Maximization* (EM) algorithm. Then, UBM can be adapted using the maximum a posteriori (MAP) criterion to model any target speaker, the parameters of the speaker model are θ_s. Evaluating log-likelihoods of the test utterance with two models (UBM and an enrolled speaker model) allows to compute speaker verification score s as LLR:

$$s = \log P(\text{data} \mid \theta_s) - \log P(\text{data} \mid \theta). \tag{2.18}$$

Later, it was proposed to use the MAP-adapted speaker model not for direct scoring but rather for feature extraction [Campbell et al., 2006]. One can stack the means of the adapted GMM into a single vector, the so-called *supervector*, representing the whole utterance. Supervectors have an obvious advantage over acoustic feature representation: their size is fixed for all recordings. Supervectors can be used as input to some classifier, e.g., *support vector machine* (SVM). However, supervectors have a significant limitation in their applicability due to their high dimensionality. For example, for GMM of size 1024 and 39-dimensional acoustic features, the size of the supervector is $1024 \times 39 = 39936$. To overcome this disadvantage, the i-vector model was introduced [Dehak et al., 2011]. I-vector framework assumes that important speaker and channel variability in the supervector can be compressed into a low-dimensional representation:

$$\mathbf{M} = \mathbf{m} + \mathbf{Tr}. \tag{2.19}$$

Here, $\mathbf{M}$ is a GMM supervector, $\mathbf{m}$ is a speaker and channel-independent vector (in practice, UBM mean supervector). $\mathbf{T}$ is a so-called total variability matrix; its columns define the basis of low-dimensional subspace where the important variability of supervector space lies.

Usually, the size of this subspace is set to 400-600. $\mathbf{r}$ is a hidden vector having a standard normal prior; $\mathbf{M}$, in this model, is also normal:

$$\mathbf{M} \sim \mathcal{N}(\mathbf{m}, \mathbf{T}\mathbf{T}'). \tag{2.20}$$

Given the observation (frame-by-frame acoustic features), one can calculate the posterior for $\mathbf{r}$. Assuming Gaussian prior, the posterior is also Gaussian. I-vector is defined as a MAP point estimate of $\mathbf{r}$, i.e., it is the mean of the posterior distribution. Thus, the i-vector is a low-dimensional representation of the audio recording - embedding - that can be used as an input to some back-end model.

2.4.2 Neural Network Embeddings (X-vectors)

For a long time, i-vector embeddings were an integral component of state-of-the-art speaker verification systems. In parallel, researchers, inspired by the success of *Artificial Neural Networks* (ANN) in many machine learning tasks, pointed their attention to using ANNs in speaker verification. At first, ANNs were used to augment an existing i-vector approach, e.g., by replacing acoustic features with NN-derived feature matrices (bottleneck features) [McLaren et al., 2015, Lozano-Diez et al., 2016] or by using NN acoustic models in place of generative Gaussian mixture models for extraction of sufficient statistics [Lei et al., 2014, Kenny et al., 2014]. Later, however, the NN-derived embeddings started to appear. One of the first examples are text-dependent speaker embeddings called *d-vectors* [Heigold et al., 2016]. D-vector is an average activation of the last hidden layer of a neural network trained for speaker classification. Later, a similar approach was used for text-independent speaker verification: x-vector embeddings were introduced [Snyder et al., 2017, Snyder et al., 2018].

Afterward, many variants of the neural architectures and training objectives used for extracting speaker embeddings appeared [Zeinali et al., 2019, Okabe et al., 2018, Huang et al., 2018]. However, the majority of these models share the same global structure. The network is split into two parts: frame-level and segment-level. The first one is transforming an input speech signal frame-by-frame; then, there is a pooling mechanism aggregating the frame-level features into a single representation for the whole speech segment. The second part of the network operates on the segment level, typically passing the pooling output through several dense layers before the network output. The activation of one of the dense layers after the pooling is used as a speaker embedding. Typically, the network is trained to discriminate between training speakers, e.g., by classifying training examples into speaker classes. It is believed that the embeddings extracted from the network trained in such a way have to contain speaker information. Then, such audio representations will be useful for the final task of speaker verification, identification, or diarization.

Figure 2.1 displays an example of the embedding network architecture. This architecture follows the official Kaldi diarization recipe [Sell et al., 2018, Snyder et al., 2019]. Also, the same architecture is used to extract embeddings for the experiments described in this thesis. In this network, the frame-level layers are *Time Delay Neural Network* (TDNN) [Peddinti et al., 2015], the pooling is done by computing a mean and a standard deviation of the frame-level outputs of the TDNN layer before the pooling. There are two fully-connected layers after the pooling. The output of one of them or the concatenation of the two is considered as an embedding. The network is trained with a multi-class cross-entropy objective to classify training speakers.

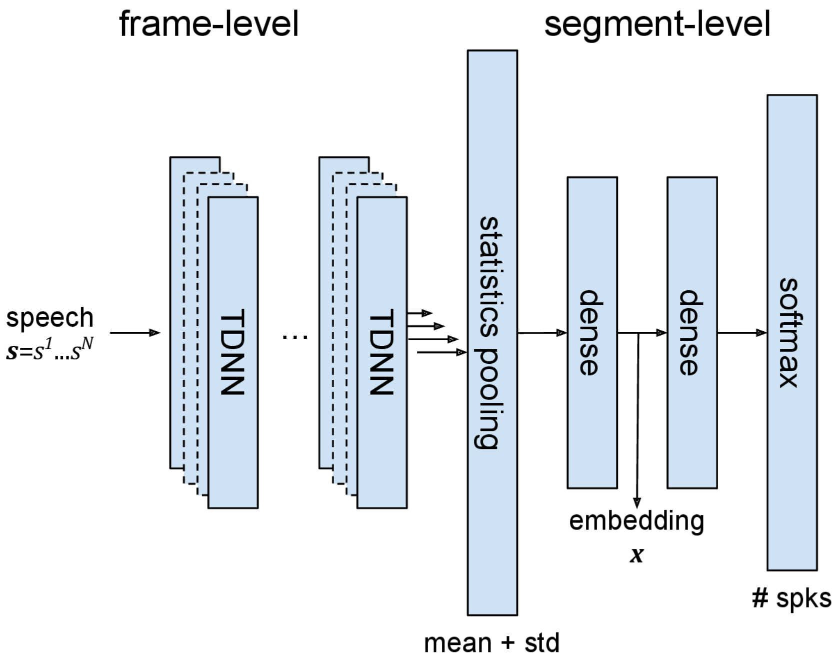

Figure 2.1: Neural network architecture used for speaker embedding extraction. The architecture follows the official Kaldi diarization recipe [Sell et al., 2018, Snyder et al., 2019]. This architecture is used for the experiments of this thesis. Also, it serves as a baseline and initialization for the probabilistic embedding model introduced in Chapter 6.2.2.

2.5 Data

This section gives a brief overview of the datasets used for training and evaluating the performance of the models described in this thesis. Further on, in the main part of this thesis, we will refer to the training and evaluation datasets by the names defined in this section.

2.5.1 Training data

For the experiments described in this thesis, we used the following data sources to train embedding extractors and back-end models:

VoxCeleb2 - the development part of a large-scale speaker verification dataset automatically collected from YouTube [Chung et al., 2018, Nagrani et al., 2019]. In total, this set contains over 1 million segments from more than 140k sessions belonging to 5994 speakers. The majority of the data are in English, however, some other languages are also present. In the original distribution, audio from a single recording session is split into several short segments. When VoxCeleb2 is used for training the embedding extractor network, its original form is used. When we use it to train a back-end model, we concatenate all the short audio chunks belonging to the same session before embedding extraction. We refer to this concatenated version of VoxCeleb2 as to **VoxCat**. When using VoxCeleb2 for discriminative training of the HT-PLDA model, for some of the experiments, we restrict the set to include the data from 2000 speakers only that results in approximately 48k concatenated recordings, we call this subset **VoxCat-S**. The experiments with VoxCat-S are aimed at choosing hyper-parameters of the training; once the parameters are selected with this small-scale dataset, we train one model on the whole VoxCat with the found parameters. A small dataset from 100 speakers is used as a cross-validation set for discriminative training and is called **VoxCat-CV** (it is disjoint from both VoxCat and VoxCat-S).

PRISM - training part of PRISM collection [Ferrer et al., 2012]. This set contains several datasets, including Fisher English parts 1 and 2, Switchboard 2 and 3, Switchboard cellular 1 and 2, and NIST SRE 2004-2010 (Mixer collection). Speakers overlapping with the evaluation set SRE10 c05 described below are excluded from this set. In total, this set contains approximately 100k utterances from 16k speakers. We use this set to train the i-vector extractor.

AMI - dataset consisting of 100 h of meeting recordings [Carletta et al., 2006]. It consists of 171 meetings, each containing speech of 3 to 5 speakers. We use this dataset for discriminative training of back-end models used in diarization systems.

2.5.2 Evaluation data

When evaluating the performance of the speaker verification models, we report the results on the following telephone evaluation conditions (datasets):

SRE10 c05,f - female part of condition 5 of NIST SRE 2010 [Martin and Greenberg, 2010]. These data are telephone recordings in English collected in North America. The female portion of SRE2010 is considered more difficult than the male one, so it is typical to report the result only on this subset to speed up the scoring. This set includes 236781 trials (3704 targets and 233077 non-targets).

SRE16 - evaluation dataset collected for NIST SRE 2016 [Sadjadi et al., 2017]. These are telephone recordings collected outside of North America. The evaluation set contains utterances in two languages: Tagalog and Cantonese. The evaluation protocol does not include cross-gender or cross-language trials. Consequently, two sub-conditions can be specified for this data: **SRE16, Cantonese** (19298 target trials and 946098 non-targets) and **SRE16, Tagalog** (17764 and 1003568 targets and non-targets, respectively) where the two languages are considered separately.

Alternatively, we report the performance on Audio from Video (AfV) data:

VoxCeleb1-O - original test set of VoxCeleb1 [Nagrani et al., 2017], it contains data from 40 speakers not overlapping with the rest of VoxCeleb 1 and 2. This set consists of 37720 trials (equal number of target and non-target trials).

VoxCeleb1-E - test set containing 579,818 trials (equal number of target and non-target trials) randomly sampled from the whole VoxCeleb1 dataset. It includes data from 1251 speakers.

VoxCeleb1-H - “hard” test set consisting of 550,894 (equal number of target and non-target trials) trials sampled from the whole VoxCeleb1 dataset so that enrollment and test segments belong to the same nationality-gender cluster. It includes data from 1251 speakers.

As both VoxCeleb1-E and VoxCeleb1-H contain data from the training and test set of VoxCeleb1, we use only VoxCeleb2 for training the models.

SITW core-core - core-core condition of Speakers in the Wild (SITW) evaluation dataset [McLaren et al., 2016] consisting of 721788 trials (3658 target and 718130 non-target). These are speech segments in English from open-source media. In nature, SITW is similar to VoxCeleb data.

Performance of the diarization systems described in this thesis is evaluated on

DIHARD 2019 dev and eval - diarization data released as the development and evaluation set for DIHARD 2019 [Ryant and et al., 2018, Ryant et al., 2019] diarization challenge. Both sets contain utterances from several different sources. Individual DIHARD domains differ by the number of speakers per utterance (from 1 to 10), level of background noise, amount of overlapped speech, etc. For example, LIBRIVOX are clean recordings of a single speaker, implying that there is no speech overlap. While CIR domain consists of restaurant recordings with up to 8 speakers per recording, noisy background, and on average 25% of the time, there is a speaker overlap. The full list of domains includes:

- Audiobooks
- Broadcast interview
- Child language
- Clinical
- Courtroom
- Map task
- Meeting
- Restaurant
- Sociolinguistic field recordings
- Sociolinguistic lab recordings
- Web video

A detailed description of each domain can be found in [Ryant et al., 2019]. The final performance is presented as the average performance on all domains.

Chapter 3

PLDA model

This chapter is dedicated to the *Probabilistic Linear Discriminant Analysis* (PLDA) model. PLDA is a probabilistic model initially introduced in computer vision [Ioffe, 2006,Prince and Elder, 2007] and subsequently adopted for speaker verification [Kenny, 2010]. In speaker recognition, PLDA models the embedding representations of audio recordings, i.e., the observed data are given in the form of D-dimensional real-valued vectors $\mathbf{r}_i \in \mathbb{R}^D$.

We start by presenting the most widely-used Gaussian PLDA model and then describe the modifications we introduce to it. We call our modified model heavy-tailed PLDA (HT-PLDA). This model was introduced in [Brümmer et al., 2018a], and [Brümmer et al., 2018b]. Another variant of HT-PLDA was proposed before in [Kenny, 2010]. In turn, we will mention how our model differs from the former one.

PLDA is based on the data assumptions made in Section 2.1 and utilizes the concept of hidden speaker identity variable. It assumes that in the observed data, the speaker component can be decoupled from the other factors generating the data. In practice, it means that the total variability of the observed data comes from two sources: inter-speaker variability, indicating the differences between individual speakers, and intra-speaker variability, showing the variations of different data points belonging to the same speaker.

There are several modifications to the PLDA model which differ slightly in the way inter- and intra-speaker variabilities are represented. Here, we restrict our attention to a model called Simple PLDA [Brümmer, 2010a]. Further, we will refer to it simply as PLDA. According to this model, the observed data can be expressed as:

$$\mathbf{r}_{ij} = \boldsymbol{\mu} + \mathbf{F}\mathbf{z}_i + \boldsymbol{\eta}_{ij}. \tag{3.1}$$

Here, $\mathbf{r}_{ij}$ is the j-th observed vector (embedding) of dimension D of the i-th speaker, $\boldsymbol{\eta}_{ij}$ is a vector of noise of dimension D, $\mathbf{z}_i$ is a hidden speaker variable of dimension $d \leq D$, $\boldsymbol{\mu}$ is a mean of the observed data. Usually, the data are centered before training or evaluating the PLDA model, so, in the future, we assume that $\boldsymbol{\mu}$ is just a vector of zeros and omit it from (3.1). In (3.1), $\mathbf{F}\mathbf{z}_i$ is responsible for inter-speaker variability representation, while $\boldsymbol{\eta}_{ij}$ corresponds to intra-speaker variability.

PLDA assumes that the data are generated by the following procedure. For each speaker i, a single hidden speaker variable is sampled from its prior distribution $\mathbf{z}_i \sim \pi(\mathbf{z})$. The sampled variable is projected into the observed data space by a transformation matrix $\mathbf{F}$. In the general case, $\mathbf{F}$ is assumed to be a D-by-d rectangular matrix, meaning that the speaker variable is forced to live in a low-dimensional subspace spanned by the columns of $\mathbf{F}$. Then, for each utterance of a given speaker, the vector of noise $\boldsymbol{\eta}_{ij}$ is sampled from its corresponding distribution. Noise is assumed to have the same dimensionality as the observed data. Then, factors depending on speaker and noise are added.

3.1 Gaussian PLDA (G-PLDA)

3.1.1 Likelihood evaluation

G-PLDA model assumes that the prior over speaker hidden variable is a multivariate standard normal distribution

$$\pi(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid \mathbf{0}, \mathbf{I}). \tag{3.2}$$

The noise is also Gaussian: each $\boldsymbol{\eta}_{ij}$ is sampled from a multivariate normal distribution with zero mean and a fixed covariance matrix $\mathbf{W}^{-1}$. The observed embedding is a sample from a Gaussian distribution as well:

$$\begin{aligned} P(\mathbf{r}_{ij} \mid \mathbf{z}_i) &= \mathcal{N}\big(\mathbf{r}_{ij} \mid \mathbf{F}\mathbf{z}_i, \mathbf{W}^{-1}\big) \\ &= \frac{|\mathbf{W}|^{\frac{1}{2}}}{(2\pi)^{\frac{D}{2}}} \exp\big[-\frac{1}{2}(\mathbf{r}_{ij} - \mathbf{F}\mathbf{z}_i)'\mathbf{W}(\mathbf{r}_{ij} - \mathbf{F}\mathbf{z}_i)\big] \\ &= \frac{|\mathbf{W}|^{\frac{1}{2}}}{(2\pi)^{\frac{D}{2}}} \exp\big[-\frac{1}{2}(\mathbf{z}_i'\mathbf{F}'\mathbf{W}\mathbf{F}\mathbf{z}_i - 2\mathbf{z}_i'\mathbf{F}'\mathbf{W}\mathbf{r}_{ij} + \mathbf{r}_{ij}'\mathbf{W}\mathbf{r}_{ij})\big]. \end{aligned} \tag{3.3}$$

Notice that the likelihood $P(\mathbf{r} \mid \mathbf{z})$ is a Gaussian distribution of the observed embedding $\mathbf{r}$, but, when considered as a function of the hidden variable $\mathbf{z}$, it is a Gaussian function. Considering the likelihood (3.3) as a function of $\mathbf{z}$ and ignoring the components not depending on $\mathbf{z}$, (3.3) can be rewritten as:

$$L(\mathbf{z}_i) \propto \exp\big[\mathbf{a}_{ij}'\mathbf{z}_i - \frac{1}{2}\mathbf{z}_i'\mathbf{B}\mathbf{z}_i\big], \tag{3.4}$$

where we have used the following definitions:

$$\mathbf{a}_{ij} = \mathbf{F}'\mathbf{W}\mathbf{r}_{ij}, \quad \mathbf{B} = \mathbf{F}'\mathbf{W}\mathbf{F}. \tag{3.5}$$

From (3.5) we see that vectors $\mathbf{a}_{ij}$ depend on the observed data $\mathbf{r}_{ij}$ and consequently differ for different data points, while matrix $\mathbf{B}$ depends only on the model parameters and is fixed for all $\mathbf{r}_{ij}$.

As was mentioned earlier, to compute the likelihood ratio (LR) (2.4) for two partitions $\mathcal{L}, \mathcal{L}'$, one has to be able to perform two operations: pooling the recordings hypothesized to belong to the same speaker and computing the expectation over the likelihood with respect to the prior $\pi(\mathbf{z})$.

The pooling operation is trivial:

$$\begin{aligned} \prod_{j=1}^{N_i} P(\mathbf{r}_{ij} \mid \mathbf{z}_i) &\propto \exp \sum_{j=1}^{N_i} (\mathbf{a}_{ij}'\mathbf{z}_i - \frac{1}{2}\mathbf{z}_i'\mathbf{B}\mathbf{z}_i) \\ &= \exp\Big[\sum_{j=1}^{N_i} \mathbf{a}_{ij}'\mathbf{z}_i - \frac{N_i}{2}\mathbf{z}_i'\mathbf{B}\mathbf{z}_i\Big]. \end{aligned} \tag{3.6}$$

As seen from (3.6), the likelihood of several recordings pooled together has the same functional form as the likelihood of a single recording with parameters equal to summed parameters (3.5) of individual likelihoods. This operation corresponds to collecting zero-order (N_i) and first-order ($\sum_{j=1}^{N_i} \mathbf{r}_{ij}$) statistics of the observed data.

The second required operation is evaluating the expectation with respect to the prior. Remember, G-PLDA assumes a standard normal prior $\mathcal{N}(\mathbf{z} \mid \mathbf{0}, \mathbf{I})$. For a (raw or pooled) likelihood, parameterized by $\mathbf{a}, \mathbf{B}$ (we have omitted the indices here), the expectation w.r.t. the prior is [Brümmer et al., 2018a]:

$$
\begin{aligned}
\big\langle P(\mathbf{r} \mid \mathbf{z}) \big\rangle_\pi &= \int_{\mathbb{R}^d} P(\mathbf{r} \mid \mathbf{z}) \pi(\mathbf{z}) \, \mathrm{d}\mathbf{z} \\
&\propto \int_{\mathbb{R}^d} \exp\big[\mathbf{a}'\mathbf{z} - \frac{1}{2}\mathbf{z}'\mathbf{B}\mathbf{z}\big] \frac{\exp\big(-\frac{1}{2}\mathbf{z}'\mathbf{I}\mathbf{z}\big)}{\sqrt{(2\pi)^d}} \, \mathrm{d}\mathbf{z} \\
&= \int_{\mathbb{R}^d} \frac{\exp\big[\mathbf{a}'\mathbf{z} - \frac{1}{2}\mathbf{z}'(\mathbf{B}+\mathbf{I})\mathbf{z}\big]}{\sqrt{(2\pi)^d}} \, \mathrm{d}\mathbf{z} \\
&= \frac{\exp\big[\frac{1}{2}\mathbf{a}'(\mathbf{B}+\mathbf{I})^{-1}\mathbf{a}\big]}{|\mathbf{B}+\mathbf{I}|^{\frac{1}{2}}} \int_{\mathbb{R}^d} \frac{\exp\big[\mathbf{a}'\mathbf{z} - \frac{1}{2}\mathbf{z}'(\mathbf{B}+\mathbf{I})\mathbf{z} - \frac{1}{2}\mathbf{a}'(\mathbf{B}+\mathbf{I})^{-1}\mathbf{a}\big]}{\sqrt{(2\pi)^d}|\mathbf{B}+\mathbf{I}|^{-\frac{1}{2}}} \, \mathrm{d}\mathbf{z} \\
&= \frac{\exp\big[\frac{1}{2}\mathbf{a}'(\mathbf{B}+\mathbf{I})^{-1}\mathbf{a}\big]}{|\mathbf{B}+\mathbf{I}|^{\frac{1}{2}}} \int_{\mathbb{R}^d} \mathcal{N}\big(\mathbf{z} \mid (\mathbf{B}+\mathbf{I})^{-1}\mathbf{a}, (\mathbf{B}+\mathbf{I})^{-1}\big) \, \mathrm{d}\mathbf{z} \\
&= \frac{\exp\big[\frac{1}{2}\mathbf{a}'(\mathbf{B}+\mathbf{I})^{-1}\mathbf{a}\big]}{|\mathbf{B}+\mathbf{I}|^{\frac{1}{2}}}.
\end{aligned}
\tag{3.7}
$$

The same formula in the log domain:

$$
\log\big\langle P(\mathbf{r} \mid \mathbf{z}) \big\rangle_\pi = \frac{1}{2}\mathbf{a}'(\mathbf{B}+\mathbf{I})^{-1}\mathbf{a} - \frac{1}{2}\log|\mathbf{B}+\mathbf{I}| + \text{const.} \tag{3.8}
$$

Joining (3.6) and (3.8), we get a general formula for log-likelihood of the data of speaker i having N_i recordings:

$$
\log P(\mathcal{R}_i \mid H_s) = \frac{1}{2}\sum_{j=1}^{N_i} \mathbf{a}'_{ij}(N_i\mathbf{B}+\mathbf{I})^{-1}\sum_{j=1}^{N_i} \mathbf{a}_{ij} - \frac{1}{2}\log|N_i\mathbf{B}+\mathbf{I}| + \text{const.} \tag{3.9}
$$

To find the total log-likelihood of the data, one has to sum expressions (3.9) over all speakers.

Computing log-likelihood (3.9) requires two expensive operations, namely matrix inversion $(N_i\mathbf{B}+\mathbf{I})^{-1}$ and finding a matrix determinant $|N_i\mathbf{B}+\mathbf{I}|$. But, since matrices $\mathbf{B}$ are fixed and equal for all of the recordings, these operations can be done only once for all of the speakers having the same number of recordings N_i. For example, when calculating LLR score (2.6) for single-enrollment/single-test verification, it is necessary to perform the matrix inversion and determinant only twice: for matrices $2\mathbf{B}+\mathbf{I}$ and $\mathbf{B}+\mathbf{I}$ used when computing log-likelihood of one- or two-speaker partitions respectively. Then, the results of these operations can be reused for all trials.

3.1.2 Model training

One of the most widely accepted ways to estimate the parameters of the G-PLDA model, $\mathbf{F}$ and $\mathbf{W}$, is an iterative Expectation-Maximization (EM) algorithm [Prince and Elder, 2007, Brümmer, 2010a]. EM training proceeds as a series of updates for $\mathbf{F}$ and $\mathbf{W}$ that lead to Maximum Likelihood (ML) estimates for these parameters. The complete derivation of the EM algorithm can be found in [Brümmer, 2010a]; here, we present only the update formulae for G-PLDA parameters. EM algorithm proceeds in the alternation of the following steps until convergence:

- **E-step** Compute matrices $\mathbf{K}$ and $\mathbf{L}$ using current model parameters:

$$\begin{aligned}\mathbf{K} &= \sum_{i=1}^{m}\big[(N_i\mathbf{B}+\mathbf{I})^{-1}\sum_{j=1}^{N_i}\mathbf{a}_{ij}\sum_{j=1}^{N_i}\mathbf{r}'_{ij}\big],\\ \mathbf{L} &= \sum_{i=1}^{m}N_i(N_i\mathbf{B}+\mathbf{I})^{-1}\big[\mathbf{I}+\sum_{j=1}^{N_i}\mathbf{a}_{ij}\sum_{j=1}^{N_i}\mathbf{a}'_{ij}(N_i\mathbf{B}+\mathbf{I})^{-1}\big].\end{aligned} \tag{3.10}$$

- **M-step** Update parameters $\mathbf{F}$ and $\mathbf{W}$:

$$\begin{aligned}\mathbf{F} &= \mathbf{K}'\mathbf{L}^{-1},\\ \mathbf{W}^{-1} &= \frac{1}{N}(\sum_{i=1}^{m}\sum_{j=1}^{N_i}\mathbf{r}_{ij}\mathbf{r}'_{ij}-\mathbf{F}\mathbf{K}).\end{aligned} \tag{3.11}$$

- **Minimum-divergence step** Rotate estimated $\mathbf{F}$:

$$\mathbf{F}\leftarrow\mathbf{F}\,\mathrm{chol}\Big(\frac{1}{m}\sum_{i=1}^{m}\big((N_i\mathbf{B}+\mathbf{I})^{-1}\big[\mathbf{I}+\sum_{j=1}^{N_i}\mathbf{a}_{ij}\sum_{j=1}^{N_i}\mathbf{a}'_{ij}(N_i\mathbf{B}+\mathbf{I})^{-1}\big]\big)\Big), \tag{3.12}$$

  where $\mathrm{chol}(\mathbf{A})\mathrm{chol}(\mathbf{A})'=\mathbf{A}$ denotes Cholesky decomposition.

The EM algorithm is a fast and simple way of training the G-PLDA model, however, that is not the only option. For example, there are various discriminative methods to estimate PLDA parameters [Burget et al., 2011, Borgström and McCree, 2013, Rohdin et al., 2014, Rohdin et al., 2016].

Due to its simplicity, G-PLDA is widely used. However, although the assumption of the fixed intra-class variability for all speakers makes G-PLDA computationally convenient, it limits its ability to utilize the uncertainty information from the embeddings. Hence, below, we present an alternative model that allows for uncertainty propagation.

3.2 Heavy-tailed PLDA (HT-PLDA)

In [Kenny, 2010], an alternative to the G-PLDA model, called heavy-tailed PLDA was shown to be a better model of i-vector embeddings [Dehak et al., 2011]. Instead of the standard normal prior for the speaker variable and Gaussian noise in G-PLDA, HT-PLDA of [Kenny, 2010] assumed that both speaker variables and noise are sampled from a Student's T-distribution. HT-PLDA was shown to outperform simpler G-PLDA, but the computational cost was considerable.

Subsequently, [Garcia-Romero and Espy-Wilson, 2011] showed that a simple length normalization (LN) procedure applied to the i-vector embeddings helps to "Gaussianize" them. This matched the performance of HT-PLDA, with negligible extra computational cost compared to G-PLDA. Thus, G-PLDA with length-normalization was established as the standard back-end for scoring in text-independent i-vector based speaker recognizers. Later, the same scoring recipe was adopted for embeddings extracted by a neural network, so-called x-vectors [Snyder et al., 2017, Snyder et al., 2018, McLaren et al., 2018].

Here, we simplify the model of [Kenny, 2010] by assuming that only the noise variable $\boldsymbol{\eta}$ is distributed according to Student's T-distribution. In contrast, the prior for the speaker variable $\mathbf{z}$ is a standard normal Gaussian as in G-PLDA. As we show below, this assumption allows us to develop a fast scoring recipe and thus mitigate the main drawback of the original HT-PLDA. As we will see in Section 3.2.3, the introduction of heavy-tailed noise results in better uncertainty propagation compared to G-PLDA.

Let us consider the generative process assumed by the model we propose in detail. The equation describing the observed data stays the same as in the general PLDA case ((3.1) with zero mean):

$$\mathbf{r}_{ij} = \mathbf{F}\mathbf{z}_i + \boldsymbol{\eta}_{ij}. \tag{3.13}$$

As in G-PLDA, hidden speaker variables are assumed to be sampled from standard normal distribution $\mathbf{z}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and $\mathbf{F}$ is a D-by-d *factor loading matrix*. However, now, the noise is generated differently than in the G-PLDA case. HT-PLDA assumes that noise variables are sampled from Gaussian distribution with variable precision. For every utterance, first, a positive scalar α_{ij} called *precision scaling factor* is sampled from a Gamma distribution with both shape and rate parameters set to $\frac{\nu}{2}$:

$$\alpha_{ij} \sim \mathcal{G}\left(\frac{\nu}{2}, \frac{\nu}{2}\right). \tag{3.14}$$

The parameter ν is known as the *degrees of freedom* [Kenny, 2010, Bishop, 2006]. Once the precision scaling factor is sampled, the noise variable is generated from a normal distribution:

$$\boldsymbol{\eta}_{ij} \sim \mathcal{N}(\mathbf{0}, (\alpha_{ij}\mathbf{W})^{-1}), \tag{3.15}$$

where $\mathbf{W}$ is a positive definite D-by-D matrix fixed for all recordings.

Similarly, as for G-PLDA (Section 3.1.1), the likelihood of an observed data point is:

$$P(\mathbf{r}_{ij} \mid \mathbf{z}_i, \alpha_{ij}) = \mathcal{N}(\mathbf{r}_{ij} \mid \mathbf{F}\mathbf{z}_i, (\alpha_{ij}\mathbf{W})^{-1}). \tag{3.16}$$

Marginalizing out the hidden precision scaling factors α_{ij}, we notice that the observed data $\mathbf{r}$ are distributed according to Student's T-distribution [Bishop, 2006]:

$$\begin{aligned} P(\mathbf{r}_{ij} \mid \mathbf{z}_i) &= \int_0^\infty \mathcal{N}(\mathbf{r}_{ij} \mid \mathbf{F}\mathbf{z}_i, (\alpha_{ij}\mathbf{W})^{-1})\mathcal{G}(\alpha_{ij} \mid \nu/2, \nu/2)\mathrm{d}\alpha \\ &= \mathcal{T}(\mathbf{r}_{ij} \mid \mathbf{F}\mathbf{z}_i, \mathbf{W}, \nu) \propto \left[1 + \frac{(\mathbf{r}_{ij} - \mathbf{F}\mathbf{z}_i)'\mathbf{W}(\mathbf{r}_{ij} - \mathbf{F}\mathbf{z}_i)}{\nu}\right]^{-\frac{\nu+D}{2}}. \end{aligned} \tag{3.17}$$

In (3.17), we have omitted a normalizing constant since it is irrelevant for our purposes.

Although we have a closed-form expression for the likelihood, it is not the case for the likelihood ratios of the form (2.4). But, given some additional assumptions, we can build an approximation such that the likelihood ratio can be written in a closed form.

3.2.1 Gaussian likelihood approximation

First, we show that likelihood (3.17) considered as a function of $\mathbf{z}$ is proportional to a probability density function (PDF) of a T-distribution. For that, let us notice that the quadratic form in (3.17) (with omitted indices) can be rewritten as follows:

$$\begin{aligned} (\mathbf{r} - \mathbf{F}\mathbf{z})'\mathbf{W}(\mathbf{r} - \mathbf{F}\mathbf{z}) &= \mathbf{z}'(\mathbf{F}'\mathbf{W}\mathbf{F})\mathbf{z} - 2\mathbf{z}'\mathbf{F}'\mathbf{W}\mathbf{r} + \mathbf{r}'\mathbf{W}\mathbf{r} \\ &= (\mathbf{z} - \mathbf{m})'(\mathbf{F}'\mathbf{W}\mathbf{F})(\mathbf{z} - \mathbf{m}) + \mathbf{r}'\mathbf{G}\mathbf{r} \\ &= Q_{\mathbf{z}|\mathbf{r}} + \mathbf{r}'\mathbf{G}\mathbf{r}, \end{aligned} \tag{3.18}$$

where we have defined:

$$\begin{aligned}\mathbf{m} &= (\mathbf{F}'\mathbf{W}\mathbf{F})^{-1}\mathbf{F}'\mathbf{W}\mathbf{r},\\ \mathbf{G} &= \mathbf{W} - \mathbf{W}\mathbf{F}(\mathbf{F}'\mathbf{W}\mathbf{F})^{-1}\mathbf{F}'\mathbf{W},\\ Q_{\mathbf{z}|\mathbf{r}} &= (\mathbf{z}-\mathbf{m})'\mathbf{F}'\mathbf{W}\mathbf{F}(\mathbf{z}-\mathbf{m}).\end{aligned} \tag{3.19}$$

Necessary assumption to perform the transformation of (3.18) is that the matrix $\mathbf{F}'\mathbf{W}\mathbf{F}$ is invertible.

Combining (3.18) and (3.17):

$$\begin{aligned}P(\mathbf{r}_{ij} \mid \mathbf{z}_i) &\propto \left[1 + \frac{(\mathbf{r}_{ij} - \mathbf{F}\mathbf{z}_i)'\mathbf{W}(\mathbf{r}_{ij} - \mathbf{F}\mathbf{z}_i)}{\nu}\right]^{-\frac{\nu+D}{2}}\\ &= \left[1 + \frac{Q_{\mathbf{z}|\mathbf{r}} + \mathbf{r}'_{ij}\mathbf{G}\mathbf{r}_{ij}}{\nu}\right]^{-\frac{\nu+D}{2}}\\ &= \left[\frac{\nu + \mathbf{r}'_{ij}\mathbf{G}\mathbf{r}_{ij} + Q_{\mathbf{z}|\mathbf{r}}}{\nu}\right]^{-\frac{\nu+D}{2}}\\ &= \left[\frac{\nu + \mathbf{r}'_{ij}\mathbf{G}\mathbf{r}_{ij}}{\nu} + \frac{Q_{\mathbf{z}|\mathbf{r}}}{\nu}\right]^{-\frac{\nu+D}{2}}.\end{aligned} \tag{3.20}$$

Now, we consider the rearranged likelihood (3.20) as a function of hidden vector $\mathbf{z}$. Omitting the terms not depending on $\mathbf{z}$:

$$\begin{aligned}L(\mathbf{z}_i) &\propto \left[1 + \frac{Q_{\mathbf{z}|\mathbf{r}}}{\nu} \cdot \frac{\nu}{\nu + \mathbf{r}'_{ij}\mathbf{G}\mathbf{r}_{ij}}\right]^{-\frac{\nu+D}{2}}\\ &= \left[1 + \frac{Q_{\mathbf{z}|\mathbf{r}}}{\nu+D-d} \cdot \frac{\nu+D-d}{\nu + \mathbf{r}'_{ij}\mathbf{G}\mathbf{r}_{ij}}\right]^{-\frac{\nu+D-d+d}{2}}\\ &= \left[1 + \frac{Q_{\mathbf{z}|\mathbf{r}}}{\nu'} \cdot \frac{\nu'}{\nu + \mathbf{r}'_{ij}\mathbf{G}\mathbf{r}_{ij}}\right]^{-\frac{\nu'+d}{2}}\\ &= \left[1 + \frac{(\mathbf{z}_i - \mathbf{m}_{ij})'\mathbf{F}'\mathbf{W}\mathbf{F}(\mathbf{z}_i - \mathbf{m}_{ij})\frac{\nu'}{\nu+\mathbf{r}'_{ij}\mathbf{G}\mathbf{r}_{ij}}}{\nu'}\right]^{-\frac{\nu'+d}{2}}\\ &\propto \mathcal{T}\big(\mathbf{z} \mid \mathbf{m}_{ij}, \frac{\nu'}{\nu + \mathbf{r}'_{ij}\mathbf{G}\mathbf{r}_{ij}}\mathbf{F}'\mathbf{W}\mathbf{F}, \nu'\big)\\ &= \mathcal{T}\big(\mathbf{z} \mid \mathbf{m}_{ij}, b_{ij}\mathbf{F}'\mathbf{W}\mathbf{F}, \nu'\big),\end{aligned} \tag{3.21}$$

where we have defined:

$$\nu' = \nu + D - d, \qquad b_{ij} = \frac{\nu'}{\nu + \mathbf{r}'_{ij}\mathbf{G}\mathbf{r}_{ij}}, \tag{3.22}$$

and, $\mathbf{m}$ and $\mathbf{G}$ are given by (3.19).

We have shown that the likelihood function of $\mathbf{z}$ is proportional to the PDF of a T-distribution for the hidden $\mathbf{z}$. Parameters for this distribution are ν', $\mathbf{m}_{ij}$, and $b_{ij}(\mathbf{F}'\mathbf{W}\mathbf{F})$. Notice that the degrees of freedom parameter ν' has to be strictly positive and for the mean to be defined, it must satisfy the condition $\nu' > 1$. So, the second assumption for the derivations (3.20), (3.21) to be valid is that $d \leq D$ is satisfied. That is not a strict assumption: in a typical G-PLDA speaker recognition system, it is often the case anyway. Moreover, often, the dimensionality of the speaker subspace is assumed to be much smaller than the one of the observed space $D >> d$, so that ν' is large. With the growth of the degrees of freedom parameter ν, the T-distribution approaches Gaussian, and in the limit, when ν reaches infinity, it is a Gaussian distribution. As in our case ν' has a high value, we assume that the T-distribution is already close enough to a Gaussian so that we can approximate the likelihood with a Gaussian function:

$$L(\mathbf{z}_i) \propto \mathcal{T}\big(\mathbf{z} \mid \mathbf{m}_{ij}, b_{ij}\mathbf{F}'\mathbf{W}\mathbf{F}, \nu'\big) \approx \mathcal{N}(\mathbf{z} \mid \mathbf{m}_{ij}, (b_{ij}\mathbf{F}'\mathbf{W}\mathbf{F})^{-1}). \tag{3.23}$$

Then, similar to G-PLDA, the likelihood can be written as:

$$L(\mathbf{z}_i) \propto \exp\big[\mathbf{a}'_{ij}\mathbf{z}_i - \frac{1}{2}\mathbf{z}'_i\mathbf{B}_{ij}\mathbf{z}_i\big], \tag{3.24}$$

where

$$\mathbf{B}_{ij} = b_{ij}\mathbf{F}'\mathbf{W}\mathbf{F}, \qquad\qquad \mathbf{a}_{ij} = \mathbf{B}_{ij}\mathbf{m}_{ij} = b_{ij}\mathbf{F}'\mathbf{W}\mathbf{r}_{ij}. \tag{3.25}$$

Comparing (3.5) and (3.25), notice that in HT-PLDA case both $\mathbf{a}_{ij}$ and $\mathbf{B}_{ij}$ depend on the observed data, while for G-PLDA, matrices $\mathbf{B}$ depend only on the model parameters.

3.2.2 Efficient evaluation of HT-PLDA likelihood

We have approximated the likelihood with a Gaussian function, so now it is possible to find the likelihood ratio scores (2.4) in a closed form. As mentioned above, the two operations necessary to compute the LR are pooling and evaluating the expectation with respect to the prior. The formulae stay almost the same as in the G-PLDA case (3.6),(3.7):

$$\begin{aligned} \prod_{j=1}^{N_i} P(\mathbf{r}_{ij} \mid \mathbf{z}_i) &\propto \exp\Big[\sum_{j=1}^{N_i} \mathbf{a}'_{ij}\mathbf{z}_i - \frac{1}{2}\mathbf{z}'_i \sum_{j=1}^{N_i} \mathbf{B}_{ij}\mathbf{z}_i\Big], \\ \big\langle P(\mathbf{r}_{ij} \mid \mathbf{z})\big\rangle_\pi &\propto \frac{\exp\big[\frac{1}{2}\mathbf{a}'_{ij}(\mathbf{B}_{ij}+\mathbf{I})^{-1}\mathbf{a}_{ij}\big]}{|\mathbf{B}_{ij}+\mathbf{I}|^{\frac{1}{2}}}. \end{aligned} \tag{3.26}$$

And, the log-likelihood of the data of speaker i having N_i recordings is:

$$\log P(\mathcal{R}_i \mid H_s) = \frac{1}{2}\sum_{j=1}^{N_i}\mathbf{a}'_{ij}\Big(\sum_{j=1}^{N_i}\mathbf{B}_{ij}+\mathbf{I}\Big)^{-1}\sum_{j=1}^{N_i}\mathbf{a}_{ij} - \frac{1}{2}\log|\sum_{j=1}^{N_i}\mathbf{B}_{ij}+\mathbf{I}| + \text{const.} \tag{3.27}$$

Then, the log-likelihood (2.3) of the whole dataset $\mathcal{R}$ partitioned into speaker clusters by $\mathcal{L}$ for the HT-PLDA model is:

$$\log P(\mathcal{R} \mid \mathcal{L}) = \frac{1}{2}\sum_{i=1}^{m}\Big[\sum_{j=1}^{N_i}\mathbf{a}'_{ij}\Big(\sum_{j=1}^{N_i}\mathbf{B}_{ij}+\mathbf{I}\Big)^{-1}\sum_{j=1}^{N_i}\mathbf{a}_{ij} - \log|\sum_{j=1}^{N_i}\mathbf{B}_{ij}+\mathbf{I}|\Big] + \text{const.} \tag{3.28}$$

As commented before, to evaluate (3.28), one has to perform matrix inversion and compute matrix determinant, which are expensive operations. For G-PLDA, it was not a problem since these operations had to be performed just a few times: as seen from (3.9), it is necessary to compute $(N_i\mathbf{B}+\mathbf{I})^{-1}$ and $|N_i\mathbf{B}+\mathbf{I}|$ only once for each N_i, and then the result could be reused for all speakers who have the same number of recordings. Now, however, the matrix $\mathbf{B}_{ij}$ is unique for every recording, which means that for every speaker i, matrix $\sum_{j=1}^{N_i} \mathbf{B}_{ij} + \mathbf{I}$ is also unique, and the expensive operations have to be done many more times than in G-PLDA case. Considering the example of computing the LLR score for the single-enrollment/single-test speaker verification trial, we saw that for the G-PLDA case, it was enough to compute determinant and matrix inversion only twice for all trials. At the same time, now, it has to be done for matrices $\mathbf{B}_{\text{enr}} + \mathbf{I}$, $\mathbf{B}_{\text{test}} + \mathbf{I}$ and $\mathbf{B}_{\text{enr}} + \mathbf{B}_{\text{test}} + \mathbf{I}$ separately for every trial. This leads to a significant slowdown of computing likelihood and LLR verification scores with the HT-PLDA model. However, we can overcome this problem:

First, notice that matrices $\mathbf{B}_{ij} = b_{ij}\mathbf{F}'\mathbf{W}\mathbf{F}$ differ only by scaling factors b_{ij}. The positive semi-definite matrix $\mathbf{F}'\mathbf{W}\mathbf{F}$ remains fixed for all recordings. This matrix can be factorized using eigenvalue decomposition as $\mathbf{V}\boldsymbol{\Lambda}\mathbf{V}'$. The matrix $\boldsymbol{\Lambda}$ is a diagonal matrix of non-negative eigenvalues $\lambda_k, k = 1 \dots d$, $\mathbf{V}$ is an orthonormal matrix having the eigenvectors as its columns. Then, the matrix $\mathbf{B} + \mathbf{I}$ (here, we drop the indices for brevity) is:

$$\mathbf{B} + \mathbf{I} = b\mathbf{V}\boldsymbol{\Lambda}\mathbf{V}' + \mathbf{I} = \mathbf{V}(b\boldsymbol{\Lambda} + \mathbf{I})\mathbf{V}'. \tag{3.29}$$

$\mathbf{B} + \mathbf{I}$ has the same orthonormal eigenvector bases as $\mathbf{F}'\mathbf{W}\mathbf{F}$, and the eigenvalues of this matrix are $b\lambda_k + 1$. From this, we can deduce the matrix inverse and determinant needed to compute the expectation from (3.26). The determinant is just a product of all eigenvalues of a matrix:

$$|\mathbf{B} + \mathbf{I}| = \prod_{k=1}^{d} (b\lambda_k + 1). \tag{3.30}$$

The inversion is also simple:

$$(\mathbf{B} + \mathbf{I})^{-1} = \mathbf{V}(b\boldsymbol{\Lambda} + \mathbf{I})^{-1}\mathbf{V}' \tag{3.31}$$

Now, putting (3.30) and (3.31) into the second equation from (3.26), we get:

$$\begin{aligned} \big\langle P(\mathbf{r} \mid \mathbf{z}) \big\rangle_{\pi} &\propto \frac{\exp\left[\frac{1}{2}\mathbf{a}'\mathbf{V}(b\boldsymbol{\Lambda} + \mathbf{I})^{-1}\mathbf{V}'\mathbf{a}\right]}{\prod_{k=1}^{d} (b\lambda_k + 1)^{\frac{1}{2}}} \\ &= \frac{\exp\left[\frac{1}{2}\sum_{k=1}^{d} \frac{\bar{a}_k^2}{b\lambda_k + 1}\right]}{\prod_{k=1}^{d} (b\lambda_k + 1)^{\frac{1}{2}}}. \end{aligned} \tag{3.32}$$

Here, $\bar{a}_k$ are the components of $\bar{\mathbf{a}} = \mathbf{V}'\mathbf{a}$. Then, (3.28) can be rewritten as:

$$\log P(\mathcal{R} \mid \mathcal{L}) = \frac{1}{2} \sum_{i=1}^{m} \sum_{k=1}^{d} \left[\frac{(\sum_{j=1}^{N_i} \bar{a}_{ijk})^2}{\sum_{j=1}^{N_i} b_{ij}\lambda_k + 1} - \log(\sum_{j=1}^{N_i} b_{ij}\lambda_k + 1) \right] + \text{const.} \tag{3.33}$$

To summarize, in our formulation of HT-PLDA, it is enough to compute a single eigenvalue decomposition for the fixed matrix $\mathbf{F}'\mathbf{W}\mathbf{F}$ and then, computation of the likelihood and the likelihood ratio can be performed with comparable speed to the G-PLDA model. To compute the likelihood, one needs to collect the following speaker statistics:

$$b_i = \sum_{j=1}^{N_i} b_{ij}, \qquad \bar{\mathbf{a}}_i = \sum_{j=1}^{N_i} \bar{\mathbf{a}}_{ij} = \mathbf{V}'\mathbf{F}'\mathbf{W} \sum_{j=1}^{N_i} b_{ij}\mathbf{r}_{ij}. \tag{3.34}$$

3.2.3 Notes on HT-PLDA

Let us make a few additional comments regarding the presented HT-PLDA model:

- Gaussian PLDA is a special case of the HT-PLDA model. One can realize it by noticing that when the degrees of freedom parameter of the noise distribution is set to infinity $\nu \to \infty$, the noise distribution is Gaussian as in the G-PLDA case. Also, looking at the scaling factors $b = \frac{\nu'}{\nu + \mathbf{r}'\mathbf{G}\mathbf{r}}$, it is evident that when ν is infinite, scalars b do not depend on the data anymore and are equal to 1. Then, the parameters of the likelihood function $L(\mathbf{z})$ are $\mathbf{a} = \mathbf{F}'\mathbf{W}\mathbf{r}$ and $\mathbf{B} = \mathbf{F}'\mathbf{W}\mathbf{F}$ for both G-PLDA and HT-PLDA.

- If ν is small, noise is heavy-tailed. In this situation, data starts affecting the estimated scalar b through the expression $\mathbf{r}'\mathbf{G}\mathbf{r}$. Let us look closer at it. Recall that $\mathbf{G}$ is defined as:

$$\mathbf{G} = \mathbf{W} - \mathbf{W}\mathbf{F}(\mathbf{F}'\mathbf{W}\mathbf{F})^{-1}\mathbf{F}'\mathbf{W}.$$

  Then,

$$\mathbf{F}'\mathbf{G} = \mathbf{F}'\mathbf{W} - \mathbf{F}'\mathbf{W}\mathbf{F}(\mathbf{F}'\mathbf{W}\mathbf{F})^{-1}\mathbf{F}'\mathbf{W} = \mathbf{0}.$$

  And, so:

$$\mathbf{r}'\mathbf{G}\mathbf{r} = (\mathbf{F}\mathbf{z} + \boldsymbol{\eta})'\mathbf{G}(\mathbf{F}\mathbf{z} + \boldsymbol{\eta}) = \mathbf{z}'\mathbf{F}'\mathbf{G}\mathbf{F}\mathbf{z} + 2\mathbf{z}'\mathbf{F}'\mathbf{G}\boldsymbol{\eta} + \boldsymbol{\eta}'\mathbf{G}\boldsymbol{\eta} = \boldsymbol{\eta}'\mathbf{G}\boldsymbol{\eta}.$$

  From the last equation, we see that speaker hidden variable $\mathbf{z}$ does not affect the estimated scalar b, and that noise $\boldsymbol{\eta}$ is the only factor impacting it. Intuitively, for the noisy recording, the noise component of the embedding should be large, making $\boldsymbol{\eta}'\mathbf{G}\boldsymbol{\eta}$ also large, which in turn, makes the scaling factor b low, and this recording has a lower impact when computing the log-likelihood (3.28). And in the opposite case, when the recording is of high quality, the noise is low while b is high, making the impact of this recording in the log-likelihood calculation larger. This is the mechanism for propagating uncertainty into the scoring. On the contrary, in G-PLDA, all scaling factors b are equal, and no uncertainty is propagated.

- As was mentioned above, when using G-PLDA, length normalization is a standard pre-processing step for the embeddings. However, in our case, the estimation of the uncertainty relies on the expression $\mathbf{r}'\mathbf{G}\mathbf{r}$. Our assumption is that length normalization interferes with the proper uncertainty estimation. Therefore, part of the experiments in the following chapters is dedicated to comparing G-PLDA with length-normalized embeddings against HT-PLDA with raw embeddings.

- Parameters of the model are $\theta = \{\mathbf{F}, \mathbf{W}, \nu\}$. These have to be known to use the model. As we have mentioned before, the parameter ν is responsible for turning the model into heavy-tailed PLDA if its value is low or into Gaussian PLDA in the limit case $\nu \to \infty$. Consequently, by fixing the value of ν, we essentially select what kind of model we intend to use. Hence, we will always consider ν to be fixed as a part of the experimental setup. In the following sections, we investigate various strategies to estimate parameters $\mathbf{F}$ and $\mathbf{W}$ with ν fixed. We refer to it as training of the HT-PLDA model, keeping in mind that G-PLDA is a special case of HT-PLDA and thus can be trained in the same way.

Chapter 4

HT-PLDA trained as a generative model

This chapter presents an approach to estimating the parameters of the generative HT-PLDA model defined in Section 3.2. Originally, it was presented in [Silnova et al., 2018].

The goal of the training is to find such model parameters $\theta = \{\mathbf{F}, \mathbf{W}\}$ that the marginal likelihood of the data $\mathcal{R}$ for the correct speaker partition $\mathcal{L}^*$: $P(\mathcal{R} \mid \mathcal{L}^*, \theta)$ (see (3.28)) is maximized. Recall the generative process assumed by the HT-PLDA model. It assumes that there are two sets of hidden variables involved in data generation. Each speaker i has a corresponding hidden speaker variable $\mathbf{z}_i$, and a set of precision scaling factors for Gaussian noise $\boldsymbol{\alpha}_i = \{\alpha_{ij}\}_{j=1}^{N_i}$ (3.13). These hidden variables are dependent in the true joint posterior $P(\mathbf{z}_i, \boldsymbol{\alpha}_i \mid \mathcal{R}_i)$, and this posterior cannot be written in a closed form. So, it is not possible to perform a direct optimization to find $\theta^* = \text{argmax}_\theta(P(\mathcal{R} \mid \ \mathcal{L}^*, \theta))$ and we have to rely on approximate inference methods.

4.1 Variational Bayes inference

Here, similar to [Kenny, 2010], we use a mean-field variational Bayes inference (VB) [Jaakkola, 2001, Bishop, 2006].

Generally, in variational inference, true posterior P is approximated by a distribution Q. Approximate distribution Q is from a restricted family of distributions. During the inference, we are looking for the member of this family that is the closest to the true posterior in terms of Kullback-Leibler (KL) divergence. The family of Q is selected so that the inference is feasible. Q, for example, can be restricted to belong to a specific parametric family, then only parameters of Q have to be estimated during inference.

Mean-field VB imposes another kind of restriction: the approximate distribution Q has to be a factorized distribution. That means the hidden variables have to be partitioned into disjoint groups assumed to be independent, and the total distribution is a product of several factors. However, no particular assumptions are made on the functional form of the factors. In the case of HT-PLDA, we approximate the posterior as the product of distributions over all hidden variables in the model:

$$P(\mathbf{z}, \boldsymbol{\alpha} \mid \mathcal{R}, \mathcal{L}) \approx Q(\mathbf{z}, \boldsymbol{\alpha}) = \prod_i Q_i(\mathbf{z}) Q_i(\boldsymbol{\alpha}). \tag{4.1}$$

Following the VB approach and taking into account factorization (4.1), we define the following decomposition of the total log-likelihood of data:

$$\begin{aligned}\log P(\mathcal{R} \mid \mathcal{L}) &= L + \mathrm{KL}(Q(\mathbf{z},\boldsymbol{\alpha}) \,||\, P(\mathbf{z},\boldsymbol{\alpha} \mid \mathcal{R},\mathcal{L})) = \\ &= L + \sum_i \mathrm{KL}(Q_i(\mathbf{z})Q_i(\boldsymbol{\alpha}) \,||\, P(\mathbf{z}_i,\boldsymbol{\alpha}_i \mid \mathcal{R}_i)),\end{aligned} \tag{4.2}$$

where $L = L(Q(\mathbf{z},\boldsymbol{\alpha}))$ is the *VB lower bound* given as (see Appendix A for the complete derivation):

$$\begin{aligned}L &= \int Q(\mathbf{z},\boldsymbol{\alpha}) \log \frac{P(\mathcal{R},\mathbf{z},\boldsymbol{\alpha} \mid \mathcal{L},\theta)}{Q(\mathbf{z},\boldsymbol{\alpha})} \,\mathrm{d}\mathbf{z}\,\mathrm{d}\boldsymbol{\alpha} \\ &= \sum_i \left\langle \log \frac{P(\mathcal{R}_i,\mathbf{z},\boldsymbol{\alpha} \mid \theta)}{Q_i(\mathbf{z})Q_i(\boldsymbol{\alpha})} \right\rangle_{Q_i(\mathbf{z})Q_i(\boldsymbol{\alpha})} \\ &= \sum_i \Big[\langle \log P(\mathcal{R}_i \mid \mathbf{z},\boldsymbol{\alpha},\theta)\rangle_{Q_i(\mathbf{z})Q_i(\boldsymbol{\alpha})} - \mathrm{KL}(Q_i(\mathbf{z}) \,||\, P(\mathbf{z})) - \mathrm{KL}(Q_i(\boldsymbol{\alpha}) \,||\, P(\boldsymbol{\alpha}))\Big] \\ &= \sum_i \Big[\frac{D}{2}\sum_j \big\langle \log(\alpha_{ij})\big\rangle + \frac{N_i}{2}\log|\mathbf{W}| - \frac{1}{2}\sum_j \langle\alpha_{ij}\rangle \mathbf{r}'_{ij}\mathbf{W}\mathbf{r}_{ij} + \langle\mathbf{z}_i\rangle'\mathbf{F}'\mathbf{W}\sum_j\langle\alpha_{ij}\rangle\mathbf{r}_{ij} \\ &\quad - \frac{1}{2}\sum_j \langle\alpha_{ij}\rangle \mathrm{tr}(\langle\mathbf{z}_i\mathbf{z}'_i\rangle\mathbf{F}'\mathbf{W}\mathbf{F}) - \mathrm{KL}(Q_i(\mathbf{z}) \,||\, P(\mathbf{z})) - \mathrm{KL}(Q_i(\boldsymbol{\alpha}) \,||\, P(\boldsymbol{\alpha}))\Big].\end{aligned} \tag{4.3}$$

The second term in (4.2) is KL divergence between approximate and true posterior. We want to find such distribution $Q(\mathbf{z},\boldsymbol{\alpha})$ that this divergence is minimized. As KL divergence is non-negative, it would be minimized by maximizing the lower bound L. Thus, training is done by maximizing L w.r.t. both θ and the Q-factors.

Mean-field VB recipe suggests to optimize L iteratively, doing partial maximizations w.r.t. $Q_i(\boldsymbol{\alpha})$, $Q_i(\mathbf{z})$, and θ in round-robin fashion. Optimization with respect to $Q(\boldsymbol{\alpha})$ and $Q(\mathbf{z})$ is variational, rather than parametric. Analytic solutions for Q factors can be found by evaluating expectations [Bishop, 2006]:

$$\begin{aligned}\log Q_i^*(\boldsymbol{\alpha}) &= \big\langle \log P(R_i,\boldsymbol{\alpha},\mathbf{z})\big\rangle_{Q_i(\mathbf{z})} + \mathrm{const}, \\ \log Q_i^*(\mathbf{z}) &= \big\langle \log P(R_i,\mathbf{z},\boldsymbol{\alpha})\big\rangle_{Q_i(\boldsymbol{\alpha})} + \mathrm{const}.\end{aligned} \tag{4.4}$$

Evaluation of expectations can be done in a closed form (we provide the derivations in Appendix A). The optimal $Q_i(\boldsymbol{\alpha})$ is a product of independent gamma distributions, and $Q_i(\mathbf{z})$ is a multivariate Gaussian:

$$\begin{aligned}Q_i^*(\boldsymbol{\alpha}) &\propto \prod_{j=1}^{N_i} \mathcal{G}\Big(\alpha_{ij} \mid \frac{\nu+D}{2}, \frac{\nu}{2} + \frac{1}{2}\mathbf{r}'_{ij}\mathbf{W}\mathbf{r}_{ij} - \mathbf{r}'_{ij}\mathbf{W}\mathbf{F}\langle\mathbf{z}_i\rangle + \frac{1}{2}\mathrm{tr}(\langle\mathbf{z}_i\mathbf{z}'_i\rangle\mathbf{F}'\mathbf{W}\mathbf{F})\Big), \\ Q_i^*(\mathbf{z}) &\propto \mathcal{N}\Big(\mathbf{z} \mid (\sum_{j=1}^{N_i}\langle\alpha_{ij}\rangle\mathbf{F}'\mathbf{W}\mathbf{F} + \mathbf{I})^{-1}\sum_{j=1}^{N_i}\langle\alpha_{ij}\rangle\mathbf{F}'\mathbf{W}\mathbf{r}_{ij}, (\sum_{j=1}^{N_i}\langle\alpha_{ij}\rangle\mathbf{F}'\mathbf{W}\mathbf{F} + \mathbf{I})^{-1}\Big).\end{aligned} \tag{4.5}$$

As seen from (4.5), the parameters of each Q-factor depend on the expectations taken with respect to the other factor. In practice, it implies that the parameters of $Q_i(\boldsymbol{\alpha})$ and $Q_i(\mathbf{z})$ can be estimated iteratively by fixing the parameters of one of them at the time and computing the expectations to evaluate the parameters of the other one. This procedure must be repeated for every speaker, which makes this recipe slow.

To get a fast alternative, we use an approximation to compute $Q_i(\boldsymbol{\alpha})$, which is different from the standard VB solution given by (4.5): $Q_i(\boldsymbol{\alpha})$ is selected so that $\langle \alpha_{ij} \rangle = b_{ij}$ (see (3.22)):

$$Q_i(\boldsymbol{\alpha}_i) = \prod_{j=1}^{N_i} \mathcal{G}\Big(\alpha_{ij} \mid \frac{\nu + D - d}{2}, \frac{\nu + \mathbf{r}'_{ij}\mathbf{G}\mathbf{r}_{ij}}{2}\Big). \tag{4.6}$$

By substituting b_{ij} for $\langle \alpha_{ij} \rangle$ in the mean-field solution (4.5), the second Q-factor is:

$$Q_i(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid \bar{\mathbf{z}}_i, \bar{\mathbf{B}}_i^{-1}) \propto P(\mathbf{z}) \prod_{j=1}^{N_i} \mathcal{N}(\mathbf{z} \mid \mathbf{m}_{ij}, \mathbf{B}_{ij}^{-1}), \tag{4.7}$$

where precision and mean are respectively given as:

$$\begin{aligned} \bar{\mathbf{B}}_i &= \sum_{j=1}^{N_i} b_{ij}\mathbf{F}'\mathbf{W}\mathbf{F} + \mathbf{I} = \sum_{j=1}^{N_i} \mathbf{B}_{ij} + \mathbf{I}, \\ \bar{\mathbf{z}}_i &= \bar{\mathbf{B}}_i^{-1} \sum_{j=1}^{N_i} b_{ij}\mathbf{F}'\mathbf{W}\mathbf{r}_{ij} = \bar{\mathbf{B}}_i^{-1} \sum_{j=1}^{N_i} \mathbf{a}_{ij}. \end{aligned} \tag{4.8}$$

Approximation (4.6) agrees with the likelihood approximation we made in Section 3.2.1 in the following sense: notice that the true posterior for $\mathbf{z}_i$ can be expressed as:

$$P(\mathbf{z} \mid \mathcal{R}_i) \propto P(\mathbf{z}) \prod_j P(\mathbf{r}_{ij} \mid \mathbf{z}). \tag{4.9}$$

True likelihoods, $P(\mathbf{r}_{ij} \mid \mathbf{z})$, are T-distributions. Comparing (4.7) and (4.9), one can notice that (4.7) is basically (4.9) with applied Gaussian likelihood approximation of (3.23).

Having the closed-form expressions for both Q-factors, training proceeds with maximizing VB lower bound L (4.3) w.r.t. parameters θ. By computing partial derivatives of L w.r.t. matrices $\mathbf{F}$ and $\mathbf{W}$ and setting them to zero, we get the update formulae:

$$\begin{aligned} \mathbf{F} &= \mathbf{K}'\mathbf{L}^{-1}, \\ \mathbf{W}^{-1} &= \frac{1}{N}\Big(\sum_{i=1}^{m}\sum_{j=1}^{N_i} b_{ij}\mathbf{r}_{ij}\mathbf{r}'_{ij} - \mathbf{F}\mathbf{K}\Big). \end{aligned} \tag{4.10}$$

Above, matrices $\mathbf{K}$ and $\mathbf{L}$ are computed as:

$$\begin{aligned} \mathbf{K} &= \sum_{i=1}^{m} \bar{\mathbf{z}}_i \sum_{j=1}^{N_i} b_{ij}\mathbf{r}'_{ij}, \\ \mathbf{L} &= \sum_{i=1}^{m}\sum_{j=1}^{N_i} b_{ij}\big(\bar{\mathbf{B}}_i^{-1} + \bar{\mathbf{z}}_i\bar{\mathbf{z}}'_i\big). \end{aligned} \tag{4.11}$$

To summarize, the training proceeds by alternating the following steps until convergence:

- Given current model parameters, estimate $Q(\boldsymbol{\alpha})$ and $Q(\mathbf{z})$ using (4.6) and (4.7) respectively.

- Estimate matrices $\mathbf{K}$ and $\mathbf{L}$ using (4.11).
- Update model parameters $\mathbf{F}$ and $\mathbf{W}$ using (4.10).
- Perform minimum-divergence steps on the hidden variables priors $P(\mathbf{z})$ and $P(\boldsymbol{\alpha})$:
  - Rescale estimated matrix $\mathbf{W}$ (see *Example 2.6: Multivariate t-distribution with known degrees of freedom* in [McClachlan and Krishnan, 2008]):

$$\mathbf{W} \leftarrow \frac{1}{N} \sum_{i=1}^{m} \sum_{j=1}^{N_i} b_{ij} \mathbf{W} \tag{4.12}$$

  - Rotate estimated matrix $\mathbf{F}$ [Brümmer, 2010c]:

$$\mathbf{F} \leftarrow \mathbf{F}\,\mathrm{chol}\left(\frac{1}{m} \sum_{i=1}^{m} (\bar{\mathbf{B}}_i^{-1} + \bar{\mathbf{z}}_i \bar{\mathbf{z}}_i')\right). \tag{4.13}$$

All of the above steps can be done in a closed form, and the resulting optimization algorithm proceeds very similarly to the EM-algorithm for training Gaussian PLDA [Brümmer, 2010a] (see Section 3.1.2). Comparing the two algorithms, one can see that the only difference between them is that for the new algorithm, the speaker statistics over observed data are weighted by scaling factors b_{ij}:

In the EM algorithm, the necessary statistics were:

$$N_i, \quad \sum_{j=1}^{N_i} \mathbf{r}_{ij}, \quad \sum_{j=1}^{N_i} \mathbf{r}_{ij}\mathbf{r}_{ij}', \tag{4.14}$$

while for VB training of HT-PLDA model, they are:

$$\sum_{j=1}^{N_i} b_{ij}, \quad \sum_{j=1}^{N_i} b_{ij}\mathbf{r}_{ij}, \quad \sum_{j=1}^{N_i} b_{ij}\mathbf{r}_{ij}\mathbf{r}_{ij}'. \tag{4.15}$$

As was noted in Section 3.2.3, when degrees of freedom parameter ν is low, b_{ij} scale down the impact of the noisy utterances and scale up the impact of the high quality utterances when computing the likelihood (3.33) and speaker statistics (4.15). When degrees of freedom parameter ν is set to infinity, all of the scalars b_{ij} become equal to 1, and VB training of HT-PLDA models becomes exactly equivalent to the EM algorithm for G-PLDA.

Finally, it is worth noting that the posterior precisions, $\bar{\mathbf{B}}_i$, can be simultaneously diagonalized, requiring but a single eigenvalue decomposition of $\mathbf{F}'\mathbf{W}\mathbf{F}$ per iteration (as in Section 3.2.2). This allows computing the inverse matrices $\bar{\mathbf{B}}_i^{-1}$ faster and consequently speeds up the training even more.

4.2 Experiments and results

To verify whether HT-PLDA is a competitive alternative to a standard G-PLDA for the task of automatic speaker verification, we performed a few sets of experiments.

4.2.1 HT-PLDA for i-vectors

The first set of experiments compares HT-PLDA and G-PLDA when models are combined with i-vector utterance embeddings (see Section 2.4.1). We assume that, unlike for the G-PLDA case, length normalization would harm the performance of HT-PLDA since it will interfere with its natural ability to estimate uncertainty. And on the contrary, lack of length normalization should harm the performance of G-PLDA while not affecting the HT-PLDA. We compare both PLDA models on two speaker verification conditions from Section 2.5: SRE10 c05,f and SRE16.

The same i-vector embeddings are used to train both PLDA models. They were extracted as described below. We used 60-dimensional spectral features: 20 MFCCs, including C_0, augmented with Δ and $\Delta\Delta$ coefficients. The features were short-term mean and variance normalized over a 3 second sliding window. With those features, we train a GMM UBM with 2048 diagonal components in a gender-independent fashion. Then, we collect sufficient statistics and train the i-vector extractor on PRISM dataset (see Section 2.5.1). The dimensionality of i-vector embeddings was set to $D = 600$. The same i-vector extractor is used for all i-vector experiments of this thesis; thus, we do not describe it in detail in the following sections.

We applied global mean normalization to the i-vectors (because our HT-PLDA model does not have a mean parameter). In some cases, length normalization was applied to i-vector embeddings. No other pre-processing was done. For both G-PLDA and HT-PLDA, the dimensionality of the speaker subspace was the same: $d = 200$.

The results are reported in terms of the equal error rate (EER, in %) as well as the average minimum detection cost function for two operating points of interest in the NIST SRE 2016 ($C_{\min}^{\text{Prm}}$) – see Section 2.2.2.

Table 4.2 compares traditional G-PLDA with HT-PLDA trained using VB approximation. The first two lines show the G-PLDA baseline, with and without length normalization. As expected, length normalization helps to shape the distribution of i-vectors to fit better Gaussian assumptions made by G-PLDA. Consequently, the performance of G-PLDA is significantly worse when no length normalization is applied.

The middle section (lines 3 and 4) of Table 4.2 presents the results for generative VB training of HT-PLDA as described in this chapter. No length normalization was applied to i-vectors before training or scoring the model. In the third line, training was done with $\nu \to \infty$ and testing with $\nu = 2$, meaning that at training time, the model was effectively G-PLDA, and the uncertainty could play its role only during testing. Comparing the results with the performance of G-PLDA trained on unnormalized i-vectors, we can notice that introducing heavy-tailed behavior only at test time can already considerably close the gap created by lack of length normalization.

In the fourth line, both training and testing had $\nu = 2$. That means that heavy-tailed mode was used both in training and testing. This setting also outperforms Gaussian PLDA without length normalization. However, both of these variants of HT-PLDA do not manage to improve the performance of G-PLDA with LN (line 1) for any of the test conditions.

To see how length normalization affects the performance of HT-PLDA, we train two more HT-PLDA models: again, one is fully heavy-tailed, and the other uses the heavy-tailed mode only at test time. This time, both models are trained on length-normalized i-vectors. The results are presented in the last two lines of Table 4.2. Although we expected that length normalization would harm the performance of the HT-PLDA model, the results suggest that it is not true. Comparing lines 3 to 5 and 4 to 6, we see that the performance

Table 4.1: x-vector topology proposed in [Snyder et al., 2019]. K in the first layer indicates different feature dimensionalities, T is the number of training segment frames and m in the last row is the number of speakers.

Layer	Layer context	(Input) × output
frame1	$[t-2, t-1, t, t+1, t+2]$	$(5 \times K) \times 1024$
frame2	$[t]$	1024×1024
frame3	$[t-4, t-2, t, t+2, t+4]$	$(5 \times 1024) \times 1024$
frame4	$[t]$	1024×1024
frame5	$[t-3, t, t+3]$	$(3 \times 1024) \times 1024$
frame6	$[t]$	1024×1024
frame7	$[t-4, t, t+4]$	$(3 \times 1024) \times 1024$
frame8	$[t]$	1024×1024
frame9	$[t]$	1024×2000
stats pooling (frame7,frame9)	$[0, T]$	2*1024+2*2000 × 512
segment1	$[0, T]$	512×512
softmax	$[0, T]$	$512 \times m$

did not degrade with the introduction of LN. Moreover, it seems to improve slightly for the SRE10 condition.

Even with LN applied, HT-PLDA is inferior to G-PLDA with LN. However, one has to notice that, unlike Gaussian PLDA, HT-PLDA is not as sensitive to length normalization: its presence or absence does not change the performance as dramatically as in the case of G-PLDA. Later, in Section 4.2.3, we will investigate more the robustness of HT-PLDA to the pre-processing done to the embeddings prior to model training.

4.2.2 HT-PLDA for x-vectors

We performed a similar experiment with x-vector embeddings instead of i-vectors. Now, we compare the performance of different back-ends on VoxCeleb1-O, VoxCeleb1-E, VoxCeleb1-H, and SITW core-core conditions described in 2.5.2. The performance metrics are EER and minimum detection cost function, with the prior probability of a target trial set to 0.05.

We extract the embeddings with x-vector neural network presented in Section 2.4.2. The particulars of training the embedding extractor are as follows. The acoustic features are 40 filterbank features with 16 kHz sampling frequency. The network is trained on 200 frames long segments randomly cropped from the original training recordings. We perform a four-fold augmentation of the training dataset: along with its original version, we add reverberation, noise, babble noise, and music. The network architecture is shown in Table 4.1. The same embedding extractor is used in all subsequent x-vector experiments.

We used VoxCeleb2 and VoxCat sets described in Section 2.5.1 to train the embedding extractor and back-end PLDA models, respectively[1].

[1] Here, as in the majority of the experiments in this thesis, the data sets used for training the embedding extractor and the back-end model are overlapping in terms of speakers. This is a sub-optimal scenario for the following reason. When training the extractor, it is supposed to decrease the within-speaker and increase the across-speaker spread in the embedding space. This effect is more pronounced for the speakers used for training than for the new speakers. Consequently, when the back-end is trained on the same speakers, the estimates of within-class and between-class distributions will not fit well the test data from a set of new speakers. Thus, a better strategy to train the back-end might be to have a separate set of speakers not used for training the extractor. However, at the time of performing the experiments of this thesis, we did not consider this issue.

Table 4.2: Comparison of error rates on SRE10 and SRE16 of Gaussian PLDA versus generatively trained heavy-tailed PLDA using i-vectors. The performance metrics are $C_{\mathrm{min}}^{\mathrm{Prm}}$=0.5 minDCF$_{0.01}$ + 0.5 minDCF$_{0.005}$, and EER(%).

#	System	LN	SRE10 c05,f		SRE16, all		SRE16, Cantonese		SRE16, Tagalog	
			$C_{\mathrm{min}}^{\mathrm{Prm}}$	EER	$C_{\mathrm{min}}^{\mathrm{Prm}}$	EER	$C_{\mathrm{min}}^{\mathrm{Prm}}$	EER	$C_{\mathrm{min}}^{\mathrm{Prm}}$	EER
1	G-PLDA, EM	yes	**0.26**	**2.5**	0.97	**16.5**	0.68	**9.7**	0.99	**21.0**
2	G-PLDA, EM	no	0.33	4.0	0.97	17.8	0.69	11.5	**0.98**	21.3
3	HT-PLDA, trn $\nu = \infty$, tst $\nu = 2$	no	0.30	2.7	0.97	16.7	0.68	10.0	**0.98**	21.2
4	HT-PLDA, trn $\nu = 2$, tst $\nu = 2$	no	0.31	3.2	0.97	16.9	0.69	10.4	0.99	21.3
5	HT-PLDA, trn $\nu = \infty$, tst $\nu = 2$	yes	0.27	2.7	**0.96**	16.6	**0.67**	10.1	**0.98**	21.2
6	HT-PLDA, trn $\nu = 2$, tst $\nu = 2$	yes	0.28	3.0	0.97	16.8	0.69	10.3	0.99	21.3

Table 4.3: Comparison of error rates on VoxCeleb and SITW of Gaussian PLDA versus generatively trained heavy-tailed PLDA using x-vectors. The performance metrics are $\mathrm{C}_{\mathrm{min}}^{\mathrm{Prm}}$=minDCF$_{0.05}$, and EER(%).

#	System	LN	VoxCeleb1-O		VoxCeleb1-E		VoxCeleb1-H		SITW core-core	
			$C_{\mathrm{min}}^{\mathrm{Prm}}$	EER	$C_{\mathrm{min}}^{\mathrm{Prm}}$	EER	$C_{\mathrm{min}}^{\mathrm{Prm}}$	EER	$C_{\mathrm{min}}^{\mathrm{Prm}}$	EER
1	G-PLDA, EM	yes	**0.12**	**1.6**	0.13	**1.8**	0.23	3.7	0.21	**3.0**
2	G-PLDA, EM	no	0.18	3.0	0.17	3.0	0.26	5.1	0.30	8.1
3	HT-PLDA, trn $\nu = \infty$, tst $\nu = 2$	no	**0.12**	1.9	**0.12**	2.0	**0.20**	**3.4**	**0.19**	3.5
4	HT-PLDA, trn $\nu = 2$, tst $\nu = 2$	no	**0.12**	2.0	**0.12**	2.1	0.21	3.6	**0.19**	3.6
5	HT-PLDA, trn $\nu = \infty$, tst $\nu = 2$	yes	**0.12**	1.9	**0.12**	2.0	**0.20**	**3.4**	**0.19**	4.4
6	HT-PLDA, trn $\nu = 2$, tst $\nu = 2$	yes	**0.12**	2.0	0.13	2.1	0.21	3.6	**0.19**	4.6

For all of the models, 512-dimensional embeddings were centered, then their dimensionality was reduced to 300 by linear discriminant analysis (LDA), then, optionally, the LN step was applied. We set the size of the hidden speaker subspace d to 200 for all models.

Similarly to the i-vector experiment, we compare the performance of the baseline G-PLDA model to the HT-PLDA model initialized from G-PLDA (when training the model, the degrees of freedom parameter ν was set to infinity, while when testing, it was set to 2) and HT-PLDA model that was trained generatively as described in Section 4.1. All three models were trained on x-vectors with and without LN. The results are shown in Table 4.3. Looking at the results, we confirm the findings of the i-vector experiments: The presence of length normalization is critical for a good performance of the baseline G-PLDA model (compare lines 1 and 2 of the table), while for both HT-PLDAs, we do not observe the same behavior (lines 3 and 5, and 4 and 6). In most cases, the performance of HT-PLDA with and without LN is similar. The only exception is SITW core-core condition, where not doing LN provides better performance in terms of EER. HT-PLDA with and without LN outperforms G-PLDA without LN on all of the test conditions, while when comparing HT-PLDA to G-PLDA trained on length-normalized x-vectors, we see that it results in the same or better $C_{\min}^{\mathrm{Prm}}$ for all conditions, while EER for VoxCeleb1-O, VoxCeleb1-E, and SITW degrades when switching from G-PLDA with LN to any variant of HT-PLDA.

4.2.3 Robustness of HT-PLDA to embedding pre-processing

As was seen in the example of i-vectors and x-vectors (Sections 4.2.1 and 4.2.2), HT-PLDA is less sensitive to the presence of LN compared to G-PLDA. LN is a simple operation that is cheap to perform, so the robustness of HT-PLDA in the absence of LN is not such a great advantage over G-PLDA, especially taking into account that G-PLDA provides better results. Often, however, many pre-processing steps are applied to embeddings before training the PLDA model. If HT-PLDA is robust to their presence, absence, or permutation, it will save time and resources in tuning the model.

We performed the following experiment with the x-vector based system. We use different combinations and order of length normalization and LDA dimensionality reduction for 512-dimensional neural network embeddings (x-vectors of Section 2.4.2) prior to training Gaussian and heavy-tailed PLDA models for testing on VoxCeleb1-E set. The only pre-processing step, always present for all of the systems, is centering.

The results are shown in Table 4.4. Each line of the table corresponds to a different pre-processing procedure applied to x-vectors before training the models. For example, the line designated as LN+LDA corresponds to a system where both length normalization and LDA were done, and LN was done before LDA, while LDA+LN means that both of them were performed but in the opposite order. The first two columns correspond to the results of the G-PLDA model, while the last two are showing HT-PLDA performance.

As can be seen, the performance of Gaussian PLDA varies greatly depending on what kind of pre-processing was applied to x-vectors. Especially striking is the difference in performance induced by the order of LN and LDA (lines 4 and 5). So, one may conclude that when selecting and tuning the back-end for some particular task, several options of embedding pre-processing have to be tried before applying G-PLDA. On the other hand, HT-PLDA does not seem to be affected by these factors. Even without any pre-processing, it provides competitive results, which can be slightly improved by including LN or LDA, but the difference would not be that prominent.

HT-PLDA could serve as a robust model that quickly provides a reasonable baseline for

Table 4.4: Comparison of performance on VoxCeleb1-E of Gaussian and Heavy-tailed PLDA with different pre-processing steps applied on input x-vectors. The performance metrics are $C_{\min}^{\text{Prm}}$=minDCF$_{0.05}$, and EER(%).

#	Pre-processing steps	G-PLDA		HT-PLDA	
		$C_{\min}^{\text{Prm}}$	EER	$C_{\min}^{\text{Prm}}$	EER
1	-	0.18	3.0	0.12	1.9
2	LN	0.22	3.6	0.12	1.8
3	LDA	0.17	3.0	0.12	2.1
4	LN + LDA	0.22	3.6	0.13	2.0
5	LDA + LN	0.13	1.8	0.13	2.1
6	LN + LDA +LN	0.14	1.9	0.13	2.1

an embedding system. While this baseline performance might still be improved by carefully tuning G-PLDA, the performance gains are moderate.

4.2.4 Scaling factors b depending on the audio quality and duration

According to the theory, different recordings $\mathbf{r}_{ij}$ lead to different scaling factors b_{ij} for the hidden speaker variable as defined by (3.22). We expect that for clean recordings, these scalars should be higher, and for noisy ones, they should be lower. That is the way the uncertainty is propagated in the model. The same should be true for the recordings of different lengths: short recordings should have lower b_{ij} than long ones. To verify that it is indeed the case, we perform the following experiments.

We use the x-vector system trained on the VoxCeleb2 database. To train the x-vector extractor, we use the original training data along with four augmented versions of them. Once the network is trained, we extract x-vector embeddings for the concatenated version of VoxCeleb2 development utterances, i.e., we use VoxCat set as defined in Section 2.5. Also, we extract x-vectors for the augmented versions of the utterances.

Then, we keep x-vectors of 100 speakers aside and use the rest to train the HT-PLDA model with the VB generative training approach. The trained model is then used to compute the scaling factors b for each set-aside recording. Here, we treat each recording independently, i.e., no information about the speaker identity is used, and recordings of the same speaker are not pooled together.

For the first experiment, we filter out of the pool of set-aside recordings all utterances shorter than 60 seconds in order to eliminate the possible effect of the recording duration. Then, we plot the histograms of the scalars for the original version of the data along with the augmented versions of the same utterances (different types of augmentations are pooled together for better clarity of the graph). The normalized histograms are shown in Figure 4.1.

As expected, the histogram corresponding to the original data is shifted to the right compared to the histogram plotted for the augmented utterances. In other words, the scaling factors for the original data are generally higher than those for augmented data. However, the histograms are highly overlapped: many original utterances have lower scaling factors than many of the augmented utterances. There can be several reasons for this phenomenon: one possible explanation is that we have no evidence of the original data being clean. So, some of the original recordings may actually be of a lower quality than

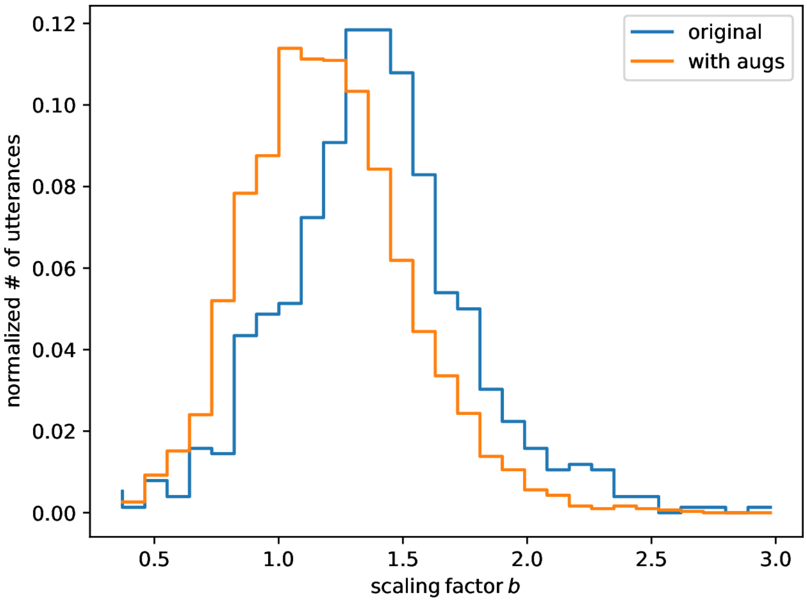

Figure 4.1: Normalized histogram of the scaling factors b for the original and augmented versions of the VoxCat utterances longer than 60 seconds.

some other recordings with augmentations. Another interpretation for the precision factors to be similar for audio of different quality is that the x-vector extractor network saw the same data we used for the plots at training time. Also, augmentations of the same types were used when training the extractor. Consequently, the extractor should have learned to suppress these augmentations and extract similar embeddings for differently augmented versions of the data.

Apart from the quality of the audio, the duration of the recording should affect the estimated precision. To verify this hypothesis, we use the same embeddings as for the previous experiment (this time without eliminating the embeddings for short utterances). Now, however, we look only at the original recordings. For each of them, we plot its scalar b against its duration. The result is shown in Figure 4.2. As expected, low scaling factors correspond to recordings of short duration. As the duration increases, the precision grows. However, for one-minute or longer utterances, duration stops being the primary driver affecting the value of b.

4.2.5 Summary

As can be seen from the experiments of Sections 4.2.1 to 4.2.3, HT-PLDA trained on i-vector and x-vector embeddings do not necessarily provide a performance gain compared to a simpler G-PLDA model. At the same time, HT-PLDA has the advantage of being more robust to the variations in the input data and to the configuration of embedding pre-processing steps.

Also, we have seen that HT-PLDA model is indeed able to utilize uncertainty information (Section 4.2.4). Especially well it was pronounced in the experiment considering utterances of various durations.

In the following chapter, we consider alternative methods of training the HT-PLDA

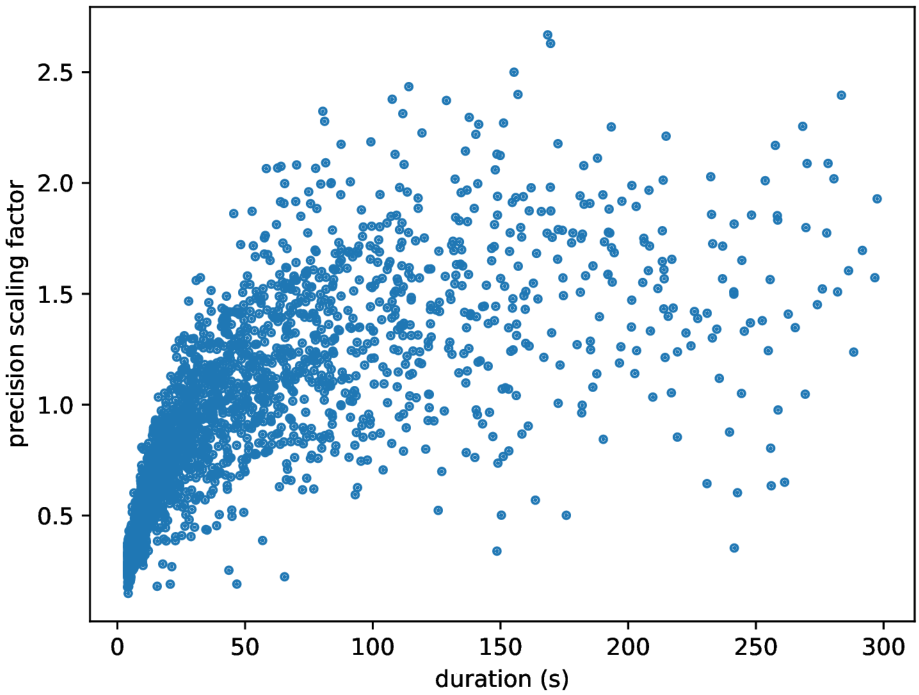

Figure 4.2: Dependence of the precision scaling factors b on the utterance duration. Each dot on the graph corresponds to one utterance from the VoxCat set. No augmentations were applied.

model to see whether its properties are kept and whether HT-PLDA is still a competitive model compared to G-PLDA when the training strategy is changed.

Chapter 5

Discriminative training of HT-PLDA model

This chapter concentrates on discriminative approaches to estimate the parameters of the HT-PLDA model of Section 3.2. The approach of Section 5.2 was originally discussed in [Brümmer et al., 2018b], similar objective to the one used in Section 5.5 was presented by us in [Silnova et al., 2020]. Other discriminative approaches to training the HT-PLDA model described in this thesis have not been published.

5.1 Maximum conditional likelihood

Maximum likelihood (ML) is a natural criterion for supervised training of generative models. That is, the model parameters θ^* are chosen as:

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, P(\text{data} \mid \text{labels}, \theta). \tag{5.1}$$

We used similar training criterion in Chapter 4. There, $\mathcal{R}$ and $\mathcal{L}$ were used as *data* and *labels* and θ were the parameters of HT-PLDA model.

Similarly, the natural criterion for discriminative training is *Maximum Conditional Likelihood* (MCL):

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, P(\text{labels} \mid \text{data}, \theta). \tag{5.2}$$

If all data points are independent of each other, the MCL objective can be factorized as:

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \prod_i P(\text{label}_i \mid \text{datum}_i, \theta) = \underset{\theta}{\operatorname{argmin}} \sum_i (-\log P(\text{label}_i \mid \text{datum}_i, \theta)). \tag{5.3}$$

The last expression is the *cross-entropy* objective. One example of such a criterion is described in Section 5.2. More details are given there, but in short: to use the cross-entropy objective of the form (5.3), we organize the original dataset $\mathcal{R}$ with labels $\mathcal{L}$ into a set of training examples – speaker verification trials, where each trial consists of a pair of recordings and a binary label indicating whether both recordings belong to the same speaker or not. We assume that the trials are independent so that we can use the cross-entropy objective. As it is a two-class classification problem, we call the objective *Binary cross-entropy* (BXE). Similarly, Section 5.5 describes a more general training strategy where the training examples are not limited to pairs of recordings but can be tuples of several recordings. For each such tuple, we want to be able to partition it into speaker clusters. As there is a finite number of possible partitions, we can enumerate them. Then, the partitioning problem can be considered a classification problem: each tuple belongs to one of the possible partition classes, and the label corresponding to a such tuple is the index of the correct partition. Thus, we can train a model to properly partition tuples of recordings into speaker clusters with *Multi-class cross entropy* (MXE).

Alternatively, we may consider the set of recordings $\mathcal{R}$ and the corresponding set of speaker identity labels (correct partition) $\mathcal{L}^*$ directly as *data* and *labels* in (5.2):

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, P(\mathcal{L}^* \mid \mathcal{R}, \theta). \tag{5.4}$$

According to the data assumptions (see Section 2.1), individual recordings may or may not depend on each other, depending on whether they come from the same speaker or different speakers. Hence, it is not possible to factorize MCL into components corresponding to individual recordings. Rather, we have to consider the set $\mathcal{R}$ as a single example where $\mathcal{L}^*$ serves as a single “label” for the whole dataset. In other words, in this case, training with the MCL objective means maximizing the posterior of the correct partition of the entire data set into speaker clusters.

Let us look closer at the posterior probability of the correct partition:

$$P(\mathcal{L}^*|\mathcal{R}, \theta) = \frac{P(\mathcal{R}|\mathcal{L}^*)P(\mathcal{L}^*)}{\sum_{\mathcal{L}} P(\mathcal{R}|\mathcal{L})P(\mathcal{L})}. \tag{5.5}$$

Computing the posterior exactly is possible only for rather small data sets. The reason is the normalization term in the denominator where the sum is computed over all possible partitions $\mathcal{L}$ of the set $\mathcal{R}$. The number of possible partitions of the set of N recordings is the N-th Bell number B_N. The growth of Bell numbers with an increase of N is prohibitively fast. For example, for $N = 75$, the corresponding Bell number exceeds 10^{80}. And the typical data set for which we would like to use training objective (5.5) contains thousands of recordings. Moreover, the denominator depends on the values of parameters θ and cannot be ignored in the optimization process. So, even though there is an explicit formula for the partition posterior, in practice, it is impossible to use it for training and even to evaluate it.

Hence, we need to consider some alternatives to directly optimizing (5.5). We consider two such alternatives: pseudolikelihood in Section 5.3 and approximate posterior of the correct partition with *Markov chain Monte-Carlo* (MCMC) sampling in Section 5.4. Also, notice that the objective of the interest operates on the posterior distribution, and the to-be-trained PLDA model supplies only the likelihood term $P(\mathcal{R} \mid \mathcal{L})$. Hence, some prior distribution over different partitions $P(\mathcal{L})$ has to be defined. We use a two-parameter version of *Chinese restaurant process* (CRP) as a prior in this work. The details of CRP are provided in Appendix B.

5.2 Discriminative training with Binary Cross-Entropy

The underlying task behind *Binary Cross-Entropy* (BXE) objective function applied to pairs of recordings is the speaker verification problem. Speaker verification is the task of answering whether the speaker in some given recording, called test, is the same as in the fixed reference recording – enrollment. Keeping in mind the task, it is natural to consider the data to be organized as a set of pairs of recordings – trials. Given a dataset $\mathcal{R}$ of size

N, one can produce $\frac{N(N-1)}{2}$ trials where we have excluded the trials of the same recording compared to itself. Also, for many methods, including HT-PLDA, the trial is considered symmetric, so there is no difference in which of the two recordings is called "enrollment" and which "test". Hence, there is a division by 2 in the possible number of trials. The goal is to separate target trials where both enrollment and test recordings contain speech of the same speaker from non-target trials for which the speakers in the enrollment and test are different. The assumption we make is that all of the trials are independent of each other.

First, let us consider a single trial. One can assign an LLR score of the form (2.6) to it. Then, as was discussed in Section 2.2.1, the decision on whether the trial is target or non-target can be based on this score. If the score is higher than some threshold τ, the trial is assumed to be a target trial; inversely, if the score is lower than the threshold, we assume the trial is non-target. The score depends on the recordings in the trial as well as on the parameters θ of the assumed underlying model (HT-PLDA in our case). The log-probability of correctly classifying a single trial is:

$$\begin{aligned}\log P(l_{ij} \mid \mathbf{r}_i, \mathbf{r}_j) &= \log \frac{P(\mathbf{r}_i, \mathbf{r}_j \mid l_{ij}) P(l_{ij})}{P(\mathbf{r}_i, \mathbf{r}_j \mid l = 1) P(l = 1) + P(\mathbf{r}_i, \mathbf{r}_j \mid l = -1) P(l = -1)} \\ &= -\log\bigl(1 + \alpha_{ij} e^{(-l_{ij} s_{ij})}\bigr).\end{aligned} \tag{5.6}$$

Here, $l_{ij} \in \{1, -1\}$ is a true label assigned to a trial $(\mathbf{r}_i, \mathbf{r}_j)$ (1 corresponds to target and -1 to non-target trials). Notice that in the other sections, labels were speaker labels associated with recordings $\mathbf{r} \in \mathcal{R}$, while, here, each label has a binary value assigned to a pair of recordings. $s_{ij} = s_{ij}(\theta)$ denotes the LLR score. α_{ij} is a scalar representing the ratio between prior probabilities of the incorrect and correct classes.

By noticing that

$$\log \alpha_{ij} = l_{ij} \log \frac{P(l = -1)}{P(l = 1)} = l_{ij}\tau,$$

(5.6) can be written as :

$$\log P(l_{ij} \mid \mathbf{r}_i, \mathbf{r}_j) = -\log\bigl(1 + e^{-l_{ij}(s_{ij} - \tau)}\bigr), \tag{5.7}$$

where τ depends on the prior probabilities of the classes (or, rather, on the prior probability of the target trial $P(l = 1)$, as the prior of non-target is simply $P(l = -1) = 1 - P(l = 1)$). Notice that τ is the optimal threshold if the relative costs of the miss and false alarm errors are equal (see Section 2.2.2) (and also if the score s_{ij} can be trusted as a good LLR score).

Now, we expand our attention to the whole training data set and consider the MCL objective (5.2) which, in this case, is maximizing a (log-)probability of correct labeling of the training trials. As the trials are assumed to be independent, the log-probability is a sum of expressions (5.7) where the sum is calculated over all trials in the training set. Just by inverting the signs, one can minimize the BXE error function:

$$E(\theta) = \sum_{\{i,j\}} \log\bigl(1 + e^{-l_{ij}(s_{ij} - \tau)}\bigr), \tag{5.8}$$

where θ denotes model parameters. Optimal parameters are obtained by minimizing the cross-entropy objective: $\theta^* = \operatorname{argmin}_\theta(E(\theta))$. $E(\theta)$ can be further modified by introducing scalar weights for individual components of the sum:

$$E(\theta) = \sum_{\{i,j\}} \beta_{i,j} \log\bigl(1 + e^{-l_{ij}(s_{ij} - \tau)}\bigr). \tag{5.9}$$

This can be useful, for example, to weight target and non-target trials as their proportion in the training set may differ from the one assumed by the target application.

(5.8) (or (5.9)) is a general definition of the BXE objective, that was repeatedly used for speaker recognition in the past [Brümmer et al., 2007, Burget et al., 2011, Brümmer et al., 2014, Heigold et al., 2016, Snyder et al., 2016, Rohdin et al., 2018, Rohdin et al., 2020]. To apply the BXE objective to estimation of parameters of the HT-PLDA model, we plug in the correct expressions for the LLR score s_{ij} derived from the model.

Closed-form expression for the LLR score with HT-PLDA model can be obtained by combining (2.6) and (3.28) (see Appendix C for the complete derivation):

$$s_{ij}^{HT-PLDA} = \frac{1}{2}\mathbf{r}_i'\mathbf{\Gamma}_i\mathbf{r}_i + \frac{1}{2}\mathbf{r}_j'\mathbf{\Gamma}_j\mathbf{r}_j + \mathbf{r}_i'\mathbf{\Lambda}_{ij}\mathbf{r}_j + k_{ij}, \tag{5.10}$$

where we have used the following definitions:

$$\begin{aligned}
\mathbf{\Gamma}_i &= b_i^2\mathbf{WF}\big[((b_i+b_j)\mathbf{F}'\mathbf{WF}+\mathbf{I})^{-1} - (b_i\mathbf{F}'\mathbf{WF}+\mathbf{I})^{-1}\big]\mathbf{F}'\mathbf{W},\\
\mathbf{\Gamma}_j &= b_j^2\mathbf{WF}\big[((b_i+b_j)\mathbf{F}'\mathbf{WF}+\mathbf{I})^{-1} - (b_j\mathbf{F}'\mathbf{WF}+\mathbf{I})^{-1}\big]\mathbf{F}'\mathbf{W},\\
\mathbf{\Lambda}_{ij} &= b_ib_j\mathbf{WF}\big((b_i+b_j)\mathbf{F}'\mathbf{WF}+\mathbf{I}\big)^{-1}\mathbf{F}'\mathbf{W},\\
k_{ij} &= -\frac{1}{2}\log|(b_i+b_j)\mathbf{F}'\mathbf{WF}+\mathbf{I}| + \frac{1}{2}\log|b_i\mathbf{F}'\mathbf{WF}+\mathbf{I}| + \frac{1}{2}\log|b_j\mathbf{F}'\mathbf{WF}+\mathbf{I}|,\\
b &= \frac{\nu'}{\nu + \mathbf{r}'(\mathbf{W}-\mathbf{WF}(\mathbf{F}'\mathbf{WF})^{-1}\mathbf{F}'\mathbf{W})\mathbf{r}}.
\end{aligned} \tag{5.11}$$

HT-PLDA parameters: factor loading matrix $\mathbf{F}$, noise precision matrix $\mathbf{W}$, and degrees of freedom ν participate in calculating the LLR score and, consequently, BXE objective (5.8). As before, we fix ν and do not attempt to re-estimate it. As we have mentioned in Section 3.2.3, ν is responsible for the choice of the model (G-PLDA vs. HT-PLDA); hence we select the model we want to use before training it. The two other parameter matrices are learned by minimizing the objective.

There are various algorithms to conduct minimization of the objective function. For example, for the discriminative training of G-PLDA in [Burget et al., 2011] authors use *Limited memory Broyden–Fletcher–Goldfarb–Shanno* (LBFGS) algorithm to do full-batch optimization. We decided to use the *Stochastic Gradient Descent* (SGD) method similarly to [Rohdin et al., 2018].

We would like to emphasize a few practical aspects of the learning process:

- As the training is stochastic, the question of how to form the batches arises. In [Rohdin et al., 2018], this question was addressed for a similar problem of joint training of embedding extractor and G-PLDA with BXE objective. There, two strategies are discussed: forming mini-batches with trials between as many speakers as possible (maximizing the number of non-target trials) or forming them to have as many utterances per speaker as possible (maximizing the number of target trials). However, in [Rohdin et al., 2018], the training was performed for an end-to-end system, i.e., the embedding extractor and back-end model were trained jointly, and the size of one batch had to be rather small because of the memory limitations. In our case, we train only the back-end model, meaning that we can afford to have larger mini-batches. Thus, we always select a set of speakers to form the batch. Then, all of the target trials of these speakers and a subset of all possible non-target trials between the selected and all the other speakers are used in the mini-batch. Non-target trials are selected in such a way that by the end of the iteration, all of them were seen.

- Training would be very slow if we directly optimize BXE with LLRs given by (5.10) and (5.11) due to a large number of expensive operations such as matrix inversion and computation of matrix determinant. Section 3.2.2 discussed a way to speed up these calculations by noticing that by doing a single eigenvalue decomposition, the two operations significantly simplify.

- $\mathbf{W}$ is a precision matrix. So, it has to be symmetric and positive semi-definite. However, there is no guarantee that after the SGD update, this will be respected. To overcome this problem, we represent $\mathbf{W} = \hat{\mathbf{W}}'\hat{\mathbf{W}}$ and learn components of $\hat{\mathbf{W}}$ rather than learning $\mathbf{W}$.

- Parameters of the model can be initialized randomly. However, one can use the parameters of the generatively trained HT-PLDA or G-PLDA as an initialization for the discriminative training. In this case, one can include the regularization term to the objective preventing the trained parameters from deviating severely from the initial values. In our experiments, we use L_2 regularization, i.e., the final objective to optimize is:

$$\hat{E}(\theta) = E(\theta) + c(||\text{vec}(\mathbf{F} - \mathbf{F}_0)||_2^2 + ||\text{vec}(\hat{\mathbf{W}}'\hat{\mathbf{W}} - \mathbf{W}_0)||_2^2), \tag{5.12}$$

  where c is a coefficient regulating the strength of regularization, $\mathbf{F}$, $\hat{\mathbf{W}}$ and $\mathbf{F}_0$, $\mathbf{W}_0$ are, respectively, current and initial values of the parameters to learn, $||\cdot||_2$ is L_2 vector norm and "vec" denotes the vectorization operation.

  When parameters are initialized randomly, we rather use early stopping of the training once we observe performance degradation on the cross-validation set.

- The above considerations are not specific to the BXE objective. Later, when training the back-end model with other discriminative objectives, we also use the efficient approach to compute likelihoods of Section 3.2.2, the factorization of the precision matrix (to make sure that it is positive semi-definite), and L_2 regularization.

5.2.1 Experiments and results

We performed the experiments to assess the SV performance of the HT-PLDA model trained with binary cross-entropy objective with i-vector and x-vector embeddings.

BXE-trained HT-PLDA for i-vectors

First, let us discuss the i-vector based system. Here, we use the same i-vectors (see Section 2.4.1) as the experiment of Section 4.2. However, unlike before, for the discriminative training via SGD, we split the PLDA training dataset (PRISM set of Section 2.5) into training and cross-validation subsets. Cross-validation was done on a randomly selected subset of 10% of the speakers, leaving the other 90% for training. This gave 8740 utterances for cross-validation and 90309 for training. We report the performance of the i-vector based system on SRE10 c05,f and SRE16 evaluation sets in terms of EER(%) and $C_{\text{min}}^{\text{Prm}}$=0.5 minDCF$_{0.01}$ + 0.5 minDCF$_{0.005}$.

When training HT-PLDA discriminatively with objective (5.8), we initialize the training with parameters of the baseline G-PLDA model, set degrees of freedom ν to 2, and retrain other parameters of HT-PLDA (matrices $\mathbf{F}$ and $\mathbf{W}$) regularizing them towards the initial parameter values.

The results are shown in Table 5.1. The first part of the table repeats the results from Table 4.2 for two baseline systems. These are Gaussian PLDA applied to length-normalized i-vectors and i-vectors without length normalization. In the middle part of the table, we present the results of the generatively trained HT-PLDA models corresponding to lines 4 and 6 from Table 4.2. Finally, last two lines of Table 5.1 show the performance of HT-PLDA models trained discriminatively with BXE objective (5.8). Thus, comparing lines 1-2 to lines 3-4 or 5-6 allows us to see the effect of the back-end model, and lines 3-4 and 5-6 compare different approaches to training the HT-PLDA model.

For discriminatively trained HT-PLDA, we see the same trend as in Section 4.2: HT-PLDA seems to be more robust to the presence or absence of the LN step; its performance does not vary as much as the performance of G-PLDA. Comparing lines 3-4 to 5-6, one can notice that for some of the conditions (SRE10 c05,f), the discriminative training of the HT-PLDA provides better results than the generative and for others ($C_{\text{min}}^{\text{Prm}}$ for SRE16, Cantonese) the trend is opposite. One of the possible reasons for the discriminative training not working that well for SRE16 is the training data that we are using: there is no data similar to SRE16 in our training set. Indeed, generative training works better when limited training data is available, while discriminative training strategies benefit from utilizing the in-domain training data.

BXE-trained HT-PLDA for x-vectors

For x-vector embeddings, we perform a similar experiment. The embedding extractor is trained on the VoxCeleb2 dataset. VoxCat set is used to train the baseline Gaussian PLDA, generatively trained HT-PLDA, and HT-PLDA trained discriminatively with BXE. VoxCat-CV is the cross-validation set for stochastic training (see Section 2.5 for more detail). We center the data and apply LDA transformation reducing the dimensionality of embeddings from 512 to 300. We report the performance in terms of EER(%) and $C_{\text{min}}^{\text{Prm}}$=minDCF$_{0.05}$ on the VoxCeleb1-O, VoxCeleb1-E, VoxCeleb1-H, and SITW core-core.

As the baseline, we use G-PLDA trained on unnormalized x-vectors and x-vectors with length normalization applied. Also, we use two generatively trained HT-PLDA models. Finally, we train two HT-PLDA models discriminatively with the BXE criterion. Both models are initialized with the baseline G-PLDA model (with corresponding embedding pre-processing LN vs. no LN). We do L_2 regularization of parameters towards their initial values. The regularization strength (coefficient c from (5.12)) was tuned on VoxCeleb1-O: c was set so that EER on this condition was not degrading with training.

The results of these experiments are presented in Table 5.2. Lines 1 – 4 correspond to GPLDA baselines and generatively trained HT-PLDAs from Table 4.3. Lines 5 and 6 of the table correspond to two HT-PLDA models trained discriminatively with the BXE criterion. As one can see from the results, training HT-PLDA discriminatively for unnormalized x-vectors provides a significant improvement compared to the G-PLDA model from which it was initialized. However, it is worth mentioning that most of the gain comes from switching to the heavy-tailed variant of PLDA even before the model is retrained (compare lines 2 and 3 of Table 4.3). We also see that the results of the BXE-trained HT-PLDA model are slightly better than those of the generatively trained model; however, the performance difference is marginal – the most significant improvements are in EER for SITW and VoxCeleb1-H conditions.

For x-vectors with LN, training HT-PLDA generatively or discriminatively is beneficial only for VoxCeleb1-H condition, where we see the performance improvement in terms of

Table 5.1: Comparison of error rates on SRE10 and SRE16 of Gaussian PLDA versus HT-PLDA trained generatively or discriminatively with BXE criterion (5.8). The performance is reported in terms of $C_{\min}^{\mathrm{Prm}}$=$0.5\,\mathrm{minDCF}_{0.01} + 0.5\,\mathrm{minDCF}_{0.005}$, and EER(%).

#	System	LN	SRE10 c05,f		SRE16, all		SRE16, Cantonese		SRE16, Tagalog	
			$C_{\min}^{\mathrm{Prm}}$	EER	$C_{\min}^{\mathrm{Prm}}$	EER	$C_{\min}^{\mathrm{Prm}}$	EER	$C_{\min}^{\mathrm{Prm}}$	EER
1	G-PLDA, EM	yes	0.26	2.5	0.96	16.5	**0.68**	9.7	0.99	21.0
2	G-PLDA, EM	no	0.33	4.0	0.97	17.8	0.69	11.5	0.98	21.3
3	HT-PLDA, VB	yes	0.28	3.0	0.97	16.8	0.69	10.3	0.99	21.3
4	HT-PLDA, VB	no	0.31	3.2	0.97	16.9	0.69	10.4	0.99	21.3
5	HT-PLDA, BXE	yes	**0.19**	**2.0**	**0.87**	16.1	0.78	10.3	**0.95**	21.4
6	HT-PLDA, BXE	no	0.21	2.1	0.90	**15.1**	0.73	**9.3**	0.97	**20.2**

Table 5.2: Comparison of error rates on VoxCeleb1 and SITW test sets of Gaussian PLDA versus HT-PLDA trained generatively or discriminatively with BXE criterion (5.8). The performance is reported in terms of $C_{\min}^{\mathrm{Prm}}$=$\mathrm{minDCF}_{0.05}$, and EER(%).

#	System	LN	VoxCeleb1-O		VoxCeleb1-E		VoxCeleb1-H		SITW core-core	
			$C_{\min}^{\mathrm{Prm}}$	EER	$C_{\min}^{\mathrm{Prm}}$	EER	$C_{\min}^{\mathrm{Prm}}$	EER	$C_{\min}^{\mathrm{Prm}}$	EER
1	G-PLDA, EM	yes	**0.12**	**1.6**	0.13	**1.8**	0.23	3.7	0.21	**3.0**
2	G-PLDA, EM	no	0.18	3.0	0.17	3.0	0.26	5.1	0.30	8.1
3	HT-PLDA, VB	yes	**0.12**	2.0	0.13	2.1	0.21	3.6	0.19	4.6
4	HT-PLDA, VB	no	**0.12**	2.0	0.12	2.1	0.21	3.6	0.19	3.6
5	HT-PLDA, BXE	yes	**0.12**	1.9	0.12	2.0	**0.20**	3.4	0.19	4.3
6	HT-PLDA, BXE	no	**0.12**	1.8	**0.11**	1.9	**0.20**	**3.3**	**0.18**	3.1

both $C_{\min}^{\mathrm{Prm}}$ and EER. There is an improvement in $C_{\min}^{\mathrm{Prm}}$ for SITW condition; however, we see significant degradation in EER when switching from G-PLDA to HT-PLDA. This results from the performance drop when switching G-PLDA to heavy-tailed mode without retraining it (lines 1 and 5 of Table 4.3).

Comparing lines 3-4 to 5-6, we notice that discriminatively trained models perform slightly better or the same as generative ones for unnormalized and length-normalized x-vectors. As noted in the i-vector experiment description, this is expected as we train the models on an in-domain training set. If there was a shift between training and evaluation data, we expect the trend to be the opposite.

Overall, we confirm our previous observation – the performance of the HT-PLDA model trained on embeddings with or without length normalization is similar, i.e., HT-PLDA is robust to embedding pre-processing.

5.3 Pseudolikelihood

As discussed before, discriminative training with objective (5.4) and (5.5) is intractable in a realistic scenario due to the high number of computations required to evaluate the objective.

Here, we look at the alternative objective function called *pseudolikelihood* (PSL), which was introduced as an approximation to the true intractable likelihood [Besag, 1975, Dawid and Musio, 2015]. PSL is a composition of likelihoods computed for a single observed variable at a time while others are fixed. Below, we present how this objective looks in our case.

We start by fixing one data partition $\mathcal{L}$ for all of the points in a dataset except for one $\mathbf{r}_i$ with a label l_i according to $\mathcal{L}$. The probability of assigning a label l_i to $\mathbf{r}_i$ is:

$$\begin{aligned} P(l_i \mid \mathbf{r}_i, \mathcal{R}_{\setminus i}, \mathcal{L}_{\setminus i}) &= \frac{P(\mathcal{R} \mid \mathcal{L}_{\setminus i}, l_i) P(l_i \mid \mathcal{L}_{\setminus i})}{P(\mathbf{r}_i, \mathcal{R}_{\setminus i} \mid \mathcal{L}_{\setminus i})} \\ &= \frac{P(\mathcal{R} \mid \mathcal{L}) P(l_i \mid \mathcal{L}_{\setminus i})}{\sum_{l'_i} P(\mathcal{R} \mid \mathcal{L}_{\setminus i}, l'_i) P(l'_i \mid \mathcal{L}_{\setminus i})} \\ &= \frac{P(\mathcal{R} \mid \mathcal{L}) P(l_i \mid \mathcal{L}_{\setminus i})}{\sum_{l'_i} P(\mathcal{R} \mid \mathcal{L}'_i) P(l'_i \mid \mathcal{L}_{\setminus i})}. \end{aligned} \tag{5.13}$$

Here, $\mathcal{L}_{\setminus i}$ denotes the whole partition (set of labels) with a single label l_i excluded, $\mathcal{L}'_i$ is a partition which is the same as $\mathcal{L}$ but with label l_i changed to l'_i. Unlike the posterior for the partition $\mathcal{L}$ of the set $\mathcal{R}$, the posterior of assigning a label for a single point is tractable as the number of components in the denominator can not be higher than $m+1$, where m is a number of speakers assumed by $\mathcal{L}_{\setminus i}$. $P(l'_i \mid \mathcal{L}_{\setminus i})$ is defined by the CRP prior and can be computed with (B.2). $P(\mathcal{R} \mid \mathcal{L})$ is a likelihood for the whole partition. As shown in Section 3.2, there is a closed-form formula (3.28) to approximate the (log-)likelihood of HT-PLDA up to a normalization constant, but this normalization constant is the same for both numerator and denominator in (5.13) so it can be safely ignored.

Pseudolikelihood then is defined as a product of the posteriors of individual points to be correctly assigned, given that the assignment of the other points is fixed. That is:

$$\psi(\mathcal{R}, \mathcal{L}^*, \theta) = \prod_i P(l^*_i \mid \mathcal{L}^*_{\setminus i}, \mathcal{R}, \theta). \tag{5.14}$$

Above, we explicitly include the model parameters θ into the objective function.

In some sense, PSL follows the intention of the objective (5.4) - maximizing posterior of the correct partition. In (5.4), the correct partition of a dataset is compared to all possible partitions; each component of PSL compares very similar partitions and looks at arguably the most difficult cases when a single point has to be correctly assigned a label.

Similarly to other objectives, instead of maximizing PSL, we can minimize its negative logarithm:

$$\underset{\theta}{\operatorname{argmax}}\, \psi(\mathcal{R}, \mathcal{L}^*, \theta) = \underset{\theta}{\operatorname{argmin}}\big[-\log \psi(\mathcal{R}, \mathcal{L}^*, \theta)\big] = \underset{\theta}{\operatorname{argmin}}\big[-\sum_i \log P(l_i^* \mid \mathcal{L}^*_{\setminus i}, \mathcal{R}, \theta)\big]. \tag{5.15}$$

Notice that the PSL objective function cannot be decomposed into a sum of independent error functions: each component in the sum (5.15) depends on the whole training dataset. Thus, we can not use stochastic training; instead, we have to use full-batch optimization techniques to learn the model parameters.

5.3.1 Experiments and results

For the PSL training, we experiment with i-vector and x-vector embeddings. The general comment for both sets of experiments is that the training with PSL objective takes a long time. This is because no stochastic training is possible for the PSL objective – to make a single update, one has to compute the objective and the gradients for the whole training data. Thus, it was not feasible to tune the training hyper-parameters; the presented results are achieved with a single run of the training without tuning the learning rate, the strength of regularization, etc.

HT-PLDA trained with PSL on i-vector embeddings

The i-vectors used in this experiment are the same as used in Sections 4.2.1 and 5.2.1: the i-vector extractor was trained on PRISM dataset, 600-dimensional i-vectors were extracted and centered. For one of the baseline models, the embeddings were length normalized. All back-end models were trained on 90% of the speakers from PRISM dataset (same as when training the model with BXE). Parameters of the CRP prior are estimated on the training set so that the expected number of speakers for the given number of recordings (B.5) was the same as the true number of speakers in the training set (see Appendix B for more details). We test the performance on SRE10 c05,f and SRE16. The results are shown in Table 5.3.

The first two lines of the table repeat the baseline results of Section 4.2.1. These results are achieved with the Gaussian PLDA model applied to normalized and unnormalized i-vectors. The third line corresponds to the results of the HT-PLDA model initialized from the second baseline and further trained with the PSL objective computed for the training set. HT-PLDA model trained with PSL objective outperforms the baseline G-PLDA trained on unnormalized i-vectors on SRE10 c05,f and SRE16, all conditions. In contrast, on Cantonese and Tagalog sub-conditions of SRE16, it only improves one of the performance metrics. When HT-PLDA is compared to the first baseline (G-PLDA with LN), we notice that for the majority of test conditions, there is performance degradation when training HT-PLDA with PSL on unnormalized i-vectors.

HT-PLDA trained with PSL on x-vector embeddings

For x-vector embeddings, the experimental setup is the same as in Section 4.2.2: x-vector extractor is trained on VoxCeleb2, back-end models are trained on VoxCat set, this set is also used to estimate the parameters of CRP similarly to i-vector experiment.

The embeddings are centered and their dimensionality is reduced by LDA from 512 to 300. Then, the embeddings could be length-normalized or not. For all discriminatively and generatively trained models, the size of the speaker subspace (hidden vector $\mathbf{z}$) was fixed to 200.

We report the performance in terms of EER(%) and $C_{\text{min}}^{\text{Prm}}$=minDCF$_{0.05}$ on SITW core-core, VoxCeleb1-O, VoxCeleb1-E, and VoxCeleb1-H. The results are shown in Table 5.4.

The first four lines of the table correspond to G-PLDA baseline models and generatively trained HT-PLDAs from Table 4.3: both are trained on raw or length normalized x-vectors. Lines 5 and 6 show the results of two HT-PLDA models trained discriminatively to maximize the pseudolikelihood of the training data. Both models are initialized with the baseline G-PLDA model (with corresponding embedding pre-processing) with L_2 regularization of parameters towards their initial values.

Analyzing the results, we make several observations: discriminatively trained HT-PLDA models perform similarly or slightly better than generatively trained ones; for most of the test conditions, discriminative training of HT-PLDA on raw embeddings results in better performance than training on length-normalized embeddings; G-PLDA with LN provides the best results in terms of EER on all test conditions except VoxCeleb1-H, however, $C_{\text{min}}^{\text{Prm}}$ is usually better for HT-PLDA models.

5.4 Approximate partition posterior

In this section, we return to objective (5.4). Let us repeat it here for convenient referencing:

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \frac{P(\mathcal{R}|\mathcal{L}^*, \theta)P(\mathcal{L}^*)}{\sum_{\mathcal{L}} P(\mathcal{R}|\mathcal{L}, \theta)P(\mathcal{L})} \tag{5.16}$$

$\mathcal{L}^*$ is the correct partition of the data $\mathcal{R}$ into speaker clusters; the sum in the denominator is over all possible partitions. Optimizing this objective would force the model to maximize the posterior probability of the correct partition.

However, as mentioned before, computing this posterior exactly is possible only for small-scale problems where the number of points in the set $\mathcal{R}$ does not exceed 10-12. For practical tasks, $\mathcal{R}$ would have several thousand elements, and computation of the normalizer in (5.16) would be impossible due to a large number of components in the sum (Bell number).

Hence, instead of computing the objective exactly, we propose to use its approximation and also an approximation of its gradient. We rely on numerical sampling, often referred to as the Monte Carlo method. The general idea here is based on the assumption that even though the correct distribution cannot be evaluated, it is still possible to sample from it. Then, we can draw a set of n independent samples $\mathcal{L}_1, \ldots, \mathcal{L}_n$ from the posterior and approximate the sum in the denominator by a sum over unique sampled partitions.

Then, the posterior becomes:

$$P(\mathcal{L}^* \mid \mathcal{R}, \theta) = \frac{P(\mathcal{R}|\mathcal{L}^*, \theta)P(\mathcal{L}^*)}{\sum_{\mathcal{L}} P(\mathcal{R}|\mathcal{L}, \theta)P(\mathcal{L})} \approx \frac{P(\mathcal{R}|\mathcal{L}^*, \theta)P(\mathcal{L}^*)}{\sum_{i=1}^{n} P(\mathcal{R}|\mathcal{L}_i, \theta)P(\mathcal{L}_i)}. \tag{5.17}$$

With such an approximation, the denominator is underestimated. But, by sampling new unique partitions, the estimation approaches the actual value of the normalizer. In the

Table 5.3: Comparison of error rates on SRE 2010 and 2016 of Gaussian PLDA with and without LN, versus discriminatively trained heavy-tailed PLDA using i-vectors. Discriminative training is performed with the PSL objective. The performance metrics are $C_{\text{min}}^{\text{Prm}}$=$0.5\,\text{minDCF}_{0.01} + 0.5\,\text{minDCF}_{0.005}$, and EER(%)

#	System	LN	SRE10 c05,f		SRE16, all		SRE16, Cantonese		SRE16, Tagalog	
			$C_{\text{min}}^{\text{Prm}}$	EER	$C_{\text{min}}^{\text{Prm}}$	EER	$C_{\text{min}}^{\text{Prm}}$	EER	$C_{\text{min}}^{\text{Prm}}$	EER
1	G-PLDA, EM	yes	**0.26**	**2.5**	0.96	**16.5**	**0.68**	**9.7**	0.99	**21.0**
2	G-PLDA, EM	no	0.33	4.0	0.97	17.8	0.69	11.5	0.98	21.3
3	HT-PLDA, PSL	no	0.28	2.6	**0.93**	17.3	0.75	10.9	**0.97**	22.2

Table 5.4: Comparison of error rates of Gaussian PLDA with and without LN, versus discriminatively trained heavy-tailed PLDA using x-vectors. Discriminative training is performed with the PSL objective. The performance is reported on VoxCeleb1-O, VoxCeleb1-E, VoxCeleb1-H, and SITW core-core condition in terms of $C_{\text{min}}^{\text{Prm}}$=$\text{minDCF}_{0.05}$, and EER(%).

#	System	LN	VoxCeleb1-O		VoxCeleb1-E		VoxCeleb1-H		SITW core-core	
			$C_{\text{min}}^{\text{Prm}}$	EER	$C_{\text{min}}^{\text{Prm}}$	EER	$C_{\text{min}}^{\text{Prm}}$	EER	$C_{\text{min}}^{\text{Prm}}$	EER
1	G-PLDA, EM	yes	**0.12**	**1.6**	0.13	**1.8**	0.23	3.7	0.21	**3.0**
2	G-PLDA, EM	no	0.18	3.0	0.17	3.0	0.26	5.1	0.30	8.1
3	HT-PLDA, VB	yes	**0.12**	2.0	0.13	2.1	0.21	3.6	0.19	4.6
4	HT-PLDA, VB	no	**0.12**	2.0	**0.12**	2.1	0.21	3.6	0.19	3.6
5	HT-PLDA, PSL	yes	**0.12**	2.0	**0.12**	2.1	0.21	3.5	0.19	4.0
6	HT-PLDA, PSL	no	**0.12**	1.8	**0.12**	1.9	**0.20**	**3.3**	**0.18**	3.2

limit, when the number of drawn unique samples reaches the number of possible partitions $n \rightarrow B_N$, the estimation of the denominator becomes exact. However, the vast majority of the possible partitions would have a very low probability and would not significantly contribute to the sum in the denominator. So, an accurate approximation can be achieved with a relatively low number of the most probable samples. Our assumption (perhaps naive) is that all relevant partitions with significant contributions to the denominator were sampled. With high probability, samples drawn from the posterior would come from the regions of its high probability. In other words, they would be partitions assigned a high posterior probability. And so, the sum over these samples would closely approximate the true normalizer.

Generally, estimating the denominator $P(\mathcal{R})$ is a challenging problem; various alternatives to the approximation described above can be found in the literature on Bayesian model evidence (see, for example, [Friel and Wyse, 2012]). However, most of the other options are more complex than the one described here. Thus, we opted for this conceptually simpler approach, understanding that it provides a rather crude approximation to the model evidence.

To train the model, we need to be able to compute the gradient of the objective. One way would be to calculate the gradient of the approximate objective (5.17). However, here, we use a different approach and directly derive an approximation to the gradient of the true posterior. For convenience, below, we consider the log-posterior and its gradient rather than the posterior itself:

$$\begin{aligned}
\nabla_\theta \log P(\mathcal{L}^* \mid \mathcal{R}, \theta) &= \nabla_\theta \log(P(\mathcal{R} \mid \mathcal{L}^*, \theta)P(\mathcal{L}^*)) - \nabla_\theta \log\big(\sum_{\mathcal{L}} P(\mathcal{R} \mid \mathcal{L}, \theta)P(\mathcal{L})\big) \\
&= \nabla_\theta \log(P(\mathcal{R} \mid \mathcal{L}^*, \theta)) - \frac{\sum_{\mathcal{L}} \nabla_\theta (P(\mathcal{R} \mid \mathcal{L}, \theta)P(\mathcal{L}))}{\sum_{\mathcal{L}} P(\mathcal{R}|\mathcal{L}, \theta)P(\mathcal{L})} \\
&= \nabla_\theta \log(P(\mathcal{R} \mid \mathcal{L}^*, \theta)) - \sum_{\mathcal{L}} \nabla_\theta \log P(\mathcal{R}|\mathcal{L}, \theta) \frac{P(\mathcal{R} \mid \mathcal{L}, \theta)P(\mathcal{L})}{\sum_{\mathcal{L}} P(\mathcal{R} \mid \mathcal{L}, \theta)P(\mathcal{L})} \\
&= \nabla_\theta \log(P(\mathcal{R} \mid \mathcal{L}^*, \theta)) - \langle \nabla_\theta \log P(\mathcal{R} \mid \mathcal{L}, \theta) \rangle_{P(\mathcal{L}|\mathcal{R})} \\
&\approx \nabla_\theta \log(P(\mathcal{R} \mid \mathcal{L}^*, \theta)) - \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \log P(\mathcal{R} \mid \mathcal{L}_i, \theta).
\end{aligned} \tag{5.18}$$

The last line of (5.18) shows that the exact gradient of the log-posterior can be approximated by a positive gradient of the log-likelihood for the correct partition and a negative average of the gradients of the log-likelihoods for sampled partitions. Note that here, the samples $\mathcal{L}_1, \ldots, \mathcal{L}_n$ are no longer assumed to be unique. This method of estimating a gradient is known as *Contrastive Divergence* (CD) [Hinton, 2002, Tieleman, 2008].

Intuitively, such an approximation to a gradient makes sense, as the first component pushes the log-likelihood (and, consequently, log-posterior) up, while the second suppresses the log-posteriors of the sampled partitions (which are probable given the current model parameters). When the posterior of the correct partition increases, it will be sampled more often, so the gradient will decrease. In the limit case, when all of the sampled partitions are the correct one, the gradient vanishes.

To summarize, if one is able to retrieve samples from the posterior distribution over partitions, it is possible to compute both an approximate objective with (5.17) and an approximate gradient with (5.18). Hence, a sampling method is needed, which we address below in Sections 5.4.1 to 5.4.5.

Experiment on the approximate training

Before we move to the particular sampling algorithm, let us first address the following questions. How good are the approximations (5.17) and (5.18), and can they really be used for model training? To answer them, we run a small-scale experiment on a synthetic dataset. The set is very small and consists of only eight 10-dimensional embeddings; we assume a 2-dimensional latent subspace ($D = 10, d = 2$). The embeddings are generated from the HT-PLDA model with known parameters. There are just eight data points for us to be able to compute the objective and the gradient exactly and compare these true values with their approximations.

Figure 5.1 shows the progress of the objective (approximate or exact) with the progress of the training for this simple experiment. The horizontal axis of the graph corresponds to the number of parameter updates, and the vertical one shows the value of the objective at each iteration. We run two trainings with 50 iterations of gradient ascent in each. The first computes the exact posterior and the exact gradient (we can do it since there are only 4140 possible partitions for a set of eight points) - this is an ideal scenario for the training (the progress of the training is shown by a solid red line on the plot). We also report the approximate objective computed with the sampling procedure at each training iteration (dashed red line). The second system uses an approximate posterior estimate and approximate gradient to make an update - this is how we are going to train the model on real data; there is no other choice for a large-scale dataset. The progress of the approximate posterior with the approximate gradient training strategy is shown as a solid blue line on the plot. Finally, we compute the exact objective for the parameters updated with approximate gradients (dashed blue line). This way, we can see whether the approximate training would improve the true objective.

In all cases, to get the approximation, we run a single chain of Smart-Dumb/Dumb-Smart (SDDS) sampler (introduced below in Section 5.4.4) and use 50 samples to compute the approximate objective and gradient. Notice that all 50 samples are used for the gradient estimation, but fewer are used for the objective function if some of the 50 samples are repeating.

As seen from the plot, when one has the access to the exact gradients, the training proceeds faster than with the approximate approach. After the same number of updates, the model parameters trained with the exact gradients provide a better true objective. At the same time, we see that even the approximate way of computing gradients improves the exact objective. It suggests that such training is still useful and can be used to estimate the parameters of the HT-PLDA model. Unfortunately, the approximate objective does not really correlate with the exactly computed one. Consequently, using it to track the performance should be done with caution.

5.4.1 General notes on sampling

First, we start with a general introduction to sampling strategies used in this work.

Many sampling methods are based on the idea that the samples from the distribution of interest $p(z)$ can be obtained by the following process: first, fix some distribution $q(z)$ called a *proposal distribution*. Proposal distribution should be selected so that one can sample from it. Then, draw samples from the proposal distribution and evaluate how well they fit into the distribution of interest. Drawn samples are filtered so that only those evaluated to be

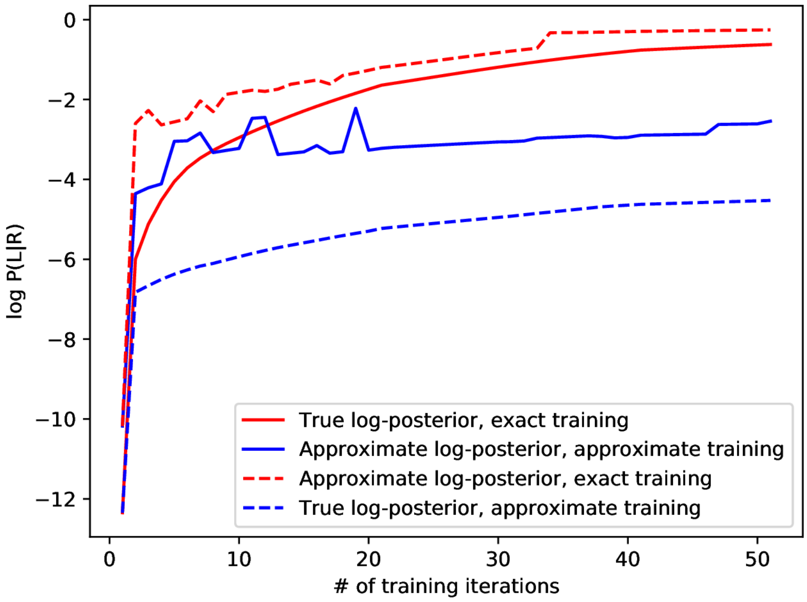

Figure 5.1: Comparison of approximate and exact computation of training objective and gradients for training HT-PLDA model with objective (5.4). The experiment is done on a small synthetic dataset of 8 10-dimensional points for which the exact objective could be evaluated.

a good fit to the distribution of interest are kept. Particular sampling methods differ in the way they define the proposal distribution. Also, the evaluation of the sample to fit the desired distribution differs for different methods.

In scope of this work, we use *Markov Chain Monte Carlo* (MCMC) sampling with *Metropolis-Hastings* acceptance test [Metropolis and Ulam, 1949, Hastings, 1970, Bishop, 2006].

In this method, the samples are drawn sequentially. The proposal distribution is defined so that the current sample drawn from it depends on the value of the previous sample $\hat{z}_{t+1} \sim q(z \mid z_t)$. The value sampled from the proposal distribution is a candidate to be included in the sampling chain. The candidate is accepted with the probability $A_t(\hat{z}_{t+1}, z_t)$ depending on the last accepted sample and the current candidate. If the candidate is accepted, it is added to the sequence of the accepted samples $z_{t+1} = \hat{z}_{t+1}$. If it is not accepted, the candidate is discarded, and the last accepted sample is used in its place, i.e., $z_{t+1} = z_t$. It can be shown that the sequence of accepted samples $z_1, \ldots, z_t, \ldots$ forms a Markov chain (hence, the name of the method) with the equilibrium distribution $p(z)$.

Let us look closer at the probability of accepting the candidate. In the Metropolis-Hastings algorithm, it is calculated as:

$$A_t(z_{t+1}, z_t) = \min\left[1, \frac{p(z_{t+1})q(z_t \mid z_{t+1})}{p(z_t)q(z_{t+1} \mid z_t)}\right]. \tag{5.19}$$

Note that the expression for the acceptance probability includes the desired distribution $p(z)$ evaluated at z_t and z_{t+1}. The assumption made here is that even if it is impossible to evaluate the distribution exactly, at least it should be possible to represent it as $p(z) = \ Z \cdot \ \hat{p}(z)$ where $\hat{p}(z)$ can be evaluated and the normalizing term Z does not depend on z. In the case of sampling from the posterior of the correct partition (5.5), $\hat{p}(z)$ is the numerator that can be evaluated, and $Z = Z(\theta)$ is the inverse denominator, which does not depend on a particular partition of interest but just on the model parameters. Then, in the acceptance test, the normalizing factors cancel and can be ignored:

$$A_t(z_{t+1}, z_t) = \min\left[1, \frac{\hat{p}(z_{t+1})q(z_t \mid z_{t+1})}{\hat{p}(z_t)q(z_{t+1} \mid z_t)}\right]. \tag{5.20}$$

The elements of the acceptance test balance each other: the ratio $\frac{\hat{p}(z_{t+1})}{\hat{p}(z_t)}$ takes into account the relative probability of a next sample compared to the previous one. And, the ratio $\frac{q(z_t|z_{t+1})}{q(z_{t+1}|z_t)}$ shows how probable it is to "reverse" the sampling if the current candidate is accepted. In other words, how probable would it be to draw a current sample from a proposal distribution $q(z \mid z_{t+1})$. So, for the candidate to be accepted, at least one of the two should be true: the candidate has a significantly higher probability than the last accepted sample according to the desired distribution, or the acceptance of this candidate would not lead to a stagnate chain, i.e., there is a high probability to return to the last accepted sample again.

After running a sampling procedure for some time, one obtains a sequence of accepted samples. However, this sequence cannot be directly treated as independent samples from the desired distribution. First, the beginning of the sequence corresponds to a burn-in period, when the Markov chain was still far from its equilibrium. So, one has to discard some fixed number of first accepted samples. Second, neighboring samples in a chain are highly correlated. A solution to overcome this problem is to discard relatively many samples in a chain between those to keep.

One of the best-known MCMC sampling algorithms is *Gibbs sampling* (GS) [Geman and Geman, 1984]. It assumes that the distribution of interest is defined for high-dimensional variable $\mathbf{z} = \{z_1, \ldots, z_n\}$; and, that even though the joint distribution $p(\mathbf{z})$ is intractable, the conditional distribution over the individual elements $p(z_i \mid \mathbf{z}_{\setminus i})$ can be evaluated. Then, the proposal distribution $q(\mathbf{z}_{t+1} \mid \mathbf{z}_t)$ is constructed so that all of the elements except for z_i of the proposed sample keep their value with probability 1, and the probability of the changed element is given by the conditional $p(z_i \mid \mathbf{z}_{\setminus i})$:

$$\begin{aligned} &\mathbf{z}_t = \{z_1^t, \ldots, z_i^t, \ldots, z_n^t\} \quad \mathbf{z}_{t+1} = \{z_1^t, \ldots, z_i^{t+1}, \ldots, z_n^t\}, \\ &q(\mathbf{z}_{t+1} \mid \mathbf{z}_t) = p(z_i^{t+1} \mid z_1^t, \ldots, z_{i-1}^t, z_{i+1}^t, \ldots, z_n^t). \end{aligned} \tag{5.21}$$

In the case of Gibbs sampling, it can be shown that the Metropolis-Hastings acceptance test is always equal to 1, and, consequently, every candidate from a Markov chain is accepted.

Now, we return from the general discussion of the sampling algorithms to our particular problem. We need to be able to sample from the posterior over the partitions of a set $P(\mathcal{L} \mid \mathcal{R})$ assuming that the underlying model generating the observed data is HT-PLDA with fixed parameters $\hat{\theta}$. In the following sections, we consider several algorithms to perform this task including *Gibbs Sampling* (GS) [Geman and Geman, 1984], *Split-Merge* (SM) [Jain and Neal, 2004] and *Smart-Dumb/Dumb-Smart* (SDDS) [Wang and Russell, 2015]. We start by building a Gibbs sampler for the partition sampling problem.

5.4.2 Sampling from the partition posterior: Gibbs sampling

Suppose we are at a time point t of the sampling process, and the current state of the Markov chain is $\mathcal{L}^{(t)}$. $\mathcal{L}^{(t)}$ defines a partition of a set of N observations into m clusters. By the definition of GS, we fix all of the labels $\mathcal{L}^{(t)}$ except for one $l_i^{(t)}$:

$$l_i^{(t+1)} \sim P(l_i^{(t)} \mid \mathbf{r}_i, \mathcal{R}_{\setminus i}, \mathcal{L}_{\setminus i}^{(t)}). \tag{5.22}$$

Then, $\mathcal{L}^{(t+1)}$ is repeating $\mathcal{L}^{(t)}$ except for $l_i^{(t+1)}$. Sampled label $l_i^{(t+1)}$ can take values from 1 to $m+1$ as the point can either join an existing cluster or start a new cluster. The same procedure is repeated for other labels one at a time. The labels to sample can be selected randomly by sampling an index i from a uniform distribution. Another option is to visit all of the indices in a round-robin fashion. Note that the proposal distribution in GS has the same form as individual components of the PSL (5.13):

$$\begin{aligned} q(\mathcal{L}^{(t+1)} \mid \mathcal{L}^{(t)}) &= P(l_i^{(t+1)} \mid \mathbf{r}_i, \mathcal{R}_{\setminus i}, \mathcal{L}_{\setminus i}^{(t)}) \\ &= \frac{P(\mathcal{R} \mid \mathcal{L}_{\setminus i}^{(t)}, l_i^{(t+1)}) P(l_i^{(t+1)} \mid \mathcal{L}_{\setminus i}^{(t)})}{\sum_{j=1}^{m+1} P(\mathcal{R} \mid \mathcal{L}_{\setminus i}^{(t)}, l_i^{(t+1)} = j) P(l_i^{(t+1)} = j \mid \mathcal{L}_{\setminus i}^{(t)})}. \end{aligned} \tag{5.23}$$

Here, as in PSL, we assume Chinese Restaurant Process prior: $P(l_i^{(t+1)} \mid \mathcal{L}_{\setminus i}^{(t)})$ is computed with (B.2). Likelihood terms $P(\mathcal{R} \mid \mathcal{L})$ can be computed with (2.3) and (3.26) up to some normalization constant which cancels in the ratio.

Thus, Gibbs sampling provides a relatively easy way to sample from a posterior over partitions. However, this approach has a major drawback: as Gibbs sampling changes a single label at a time, it has a low mixing capability, i.e., to move from one probable partition to another, it might be necessary to pass through a sequence of very improbable partitions. That means that Gibbs sampling can get stuck in a single mode of the posterior distribution (i.e., all generated samples are from that single mode), and it might take "infinitely" long time before exploring other plausible partitions [1].

Hence, a more general algorithm able to take more dramatic steps in a partition state space is needed.

5.4.3 Sampling from the partition posterior: Split-Merge algorithm

The Split-Merge algorithm proposed in [Jain and Neal, 2004] has the desired property that GS lacks – it can perform rapid steps in a partition state space. In this algorithm, instead of sampling a label for a single data point at a time, the proposal distribution allows for transitions from a current partition by splitting or merging its clusters.

Let us consider in detail a single iteration of the Split-Merge algorithm.

Suppose, that the current state of the sampler at time t is $\mathcal{L}^{(t)}$, that partitions N points into m clusters. Partition labels are $l_i^{(t)} \in \{1 \dots m\}$ and each cluster $i = 1, \dots, m$ has $N_i^{(t)}$ elements. To construct the next candidate $\mathcal{L}^{(t+1)}$, perform the following steps:

- sample two indices i and j from a uniform distribution, $l_i^{(t)}$ and $l_j^{(t)}$ are the corresponding labels defined by $\mathcal{L}^{(t)}$.

[1] Here, by the local mode of the partition posterior distribution, we understand a set of partitions that are different only in a few labels and are similarly likely. For example, we can describe a mode as some "mode representative" partition and the partitions obtained by moving several points from one cluster to another in the "representative". While, if it is necessary to change many labels (e.g., split a large cluster), then we say that the new partition belongs to another mode of the posterior distribution.

- If $l_i^{(t)} = l_j^{(t)}$, i.e., both of the sampled points belong to the same cluster, a split is performed. Without loss of generality, assume $l_i^{(t)} = l_j^{(t)} = m$.
    - Introduce a new cluster label $m+1$.
    - Assign label m to point r_i, and $m+1$ to point r_j.
    - For each remaining points $r_{k \notin \{i,j\}}$ such that $l_k^{(t)} = m$, assign independently and with equal probability label m or $m+1$.
    - Compute the acceptance probability and sample from it. If the proposed candidate is accepted, $\mathcal{L}^{(t+1)}$ has $m+1$ clusters. The new numbers of elements in clusters are then: $N_i^{(t+1)} = N_i^{(t)} \ \forall i < m$ and, $N_m^{(t+1)} + N_{m+1}^{(t+1)} = N_m^{(t)}$.
- If $l_i^{(t)} \neq l_j^{(t)}$, i.e., two sampled points belong to two different clusters, a merge is performed. Without loss of generality, assume, $l_i^{(t)} = m-1$, $l_j^{(t)} = m$.
    - For each point r_k such that $l_k^{(t)} = m$ assign label $m-1$.
    - Compute the acceptance probability and sample from it. If the sample is accepted, $\mathcal{L}^{(t+1)}$ has $m-1$ clusters. The new numbers of elements in clusters are then: $N_i^{(t+1)} = N_i^{(t)} \ \forall i < m-1$ and, $N_{m-1}^{(t)} + N_m^{(t)} = N_{m-1}^{(t+1)}$.

The remaining question is how to perform the acceptance test. According to Metropolis-Hastings algorithm, acceptance probability is calculated by (5.20). In our notation, it is:

$$A_t(\mathcal{L}^{(t+1)}, \mathcal{L}^{(t)}) = \min\left[1, \frac{P(\mathcal{R} \mid \mathcal{L}^{(t+1)})P(\mathcal{L}^{(t+1)})q(\mathcal{L}^{(t)} \mid \mathcal{L}^{(t+1)})}{P(\mathcal{R} \mid \mathcal{L}^{(t)})P(\mathcal{L}^{(t)})q(\mathcal{L}^{(t+1)} \mid \mathcal{L}^{(t)})}\right].$$

$P(\mathcal{R} \mid \mathcal{L})$ (3.28) and $P(\mathcal{L})$ (B.6) components are given by the HT-PLDA model and CRP prior, respectively. Note that for HT-PLDA, when computing the likelihood ratio between two partitions, both numerator and denominator are the products over terms corresponding to clusters defined by these partitions (see (2.3)). As two partitions of interest $\mathcal{L}^{(t+1)}$ and $\mathcal{L}^{(t)}$ are the same for almost all of the clusters, most of the product components cancel, and they do not have to be evaluated to perform a single sampling iteration.

Let us look at how to compute $q(\mathcal{L}^{(t+1)} \mid \mathcal{L}^{(t)})$ and $q(\mathcal{L}^{(t)} \mid \mathcal{L}^{(t+1)})$. For the split case, they are:

$$\begin{aligned}
q(\mathcal{L}^{(t+1)} \mid \mathcal{L}^{(t)}) &= \frac{1}{N}\frac{1}{N-1}\left(\frac{N_m^{(t)}}{N}\right)^2\left(\frac{1}{2}\right)^{N_m^{(t)}-2} \\
q(\mathcal{L}^{(t)} \mid \mathcal{L}^{(t+1)}) &= \frac{1}{N}\frac{1}{N-1}\frac{N_m^{(t+1)}}{N}\frac{N_{m+1}^{(t+1)}}{N} \\
\frac{q(\mathcal{L}^{(t)} \mid \mathcal{L}^{(t+1)})}{q(\mathcal{L}^{(t+1)} \mid \mathcal{L}^{(t)})} &= \frac{N_m^{(t+1)}N_{m+1}^{(t+1)}}{N_m^{(t)}N_m^{(t)}}2^{N_m^{(t)}-2}.
\end{aligned} \tag{5.24}$$

The first equation of (5.24) shows a probability of a cluster being split. Terms $1/N$ and $1/(N-1)$ correspond to probabilities of selecting two points of interest uniformly, $(N_m^{(t)}/N)^2$ is the probability of two selected points to come from the cluster m. Finally, $(1/2)^{N_m^{(t)}-2}$ corresponds to the probability of independently assigning new cluster labels to the remaining $N_m^{(t)}-2$ points from the original cluster. The second equation of (5.24) gives the probability of an inverse proposal which, in fact, is a merge of two clusters in state $\mathcal{L}^{(t+1)}$. There, term $1/N(N-1)$ is still the probability to select two points uniformly, $(N_m^{(t+1)}/N)$ and $(N_{m+1}^{(t+1)}/N)$ are the probabilities of the selected points to belong to the clusters m and $m+1$ in state $\mathcal{L}^{(t+1)}$. Finally, the third equation of (5.24) gives a ratio needed to compute the acceptance test.

Similarly, for the merge step:

$$\begin{aligned}
q(\mathcal{L}^{(t+1)} \mid \mathcal{L}^{(t)}) &= \frac{1}{N}\frac{1}{N-1}\frac{N_{m-1}^{(t)}}{N}\frac{N_m^{(t)}}{N} \\
q(\mathcal{L}^{(t)} \mid \mathcal{L}^{(t+1)}) &= \frac{1}{N}\frac{1}{N-1}\left(\frac{N_{m-1}^{(t+1)}}{N}\right)^2\left(\frac{1}{2}\right)^{N_{m-1}^{(t+1)}-2} \\
\frac{q(\mathcal{L}^{(t)} \mid \mathcal{L}^{(t+1)})}{q(\mathcal{L}^{(t+1)} \mid \mathcal{L}^{(t)})} &= \frac{N_{m-1}^{(t+1)}N_{m-1}^{(t+1)}}{N_{m-1}^{(t)}N_m^{(t)}}\left(\frac{1}{2}\right)^{N_{m-1}^{(t+1)}-2}.
\end{aligned} \tag{5.25}$$

Indeed, the SM algorithm provides a way to perform drastic moves in the partition space. It can explore the state space better than Gibbs sampling due to its ability to overpass low-probability regions on the transition from one probable area to another. On the other hand, Split-Merge introduces a somewhat complicated scheme of building the proposal distribution and requires performing an acceptance test, increasing computational complexity. Moreover, the acceptance rate of Split-Merge is likely to be very low. As the decisions to split and merge are entirely random, it is quite likely that most of the proposed partitions will not be accepted because of the extremely low likelihood. For example, there are 256 ways to split a cluster of 10 points into two: most of the splits will lead to a highly unlikely clustering, and only a few would be sensible. So, it might take a large number of computations and a long time before an accepted sample appears.

5.4.4 Sampling from the partition posterior: Smart-Dumb/Dumb-Smart algorithm

Smart-Dumb/Dumb-Smart [Wang and Russell, 2015] algorithm tries to address the problem of a low acceptance rate of the Split-Merge while keeping its advantage of high mobility in the state space.

The main idea of the SM is kept: the proposal distribution is designed to perform a split of a cluster into two or a merge of two different clusters. What is different in SDDS is how these splits and merges are done. It is proposed to replace random proposals of SM by *smart* split and merge. Smart proposal distribution makes likely partitions to be sampled with a higher probability. In other words, it tries to increase $\frac{P(\mathcal{R}|\mathcal{L}^{(t+1)})P(\mathcal{L}^{(t+1)})}{P(\mathcal{R}|\mathcal{L}^{(t)})P(\mathcal{L}^{(t)})}$ part of the acceptance test.

However, introducing smart splits and merges does not necessarily lead to a higher acceptance rate compared to a standard SM. The reason is the second part of the acceptance test $\frac{q(\mathcal{L}^{(t)}|\mathcal{L}^{(t+1)})}{q(\mathcal{L}^{(t+1)}|\mathcal{L}^{(t)})}$. When a smart proposal is given, it is highly unlikely that at the next iteration, it will be reversed by an opposite smart proposal, i.e., $q(\mathcal{L}^{(t)} \mid \mathcal{L}^{(t+1)})$ is extremely low. For example, suppose the proposal distribution suggests splitting a cluster into two well-separated parts. In that case, the probability of merging those parts back again is low, and a smart merge strategy would assign a low probability to sample it from the proposal distribution. That leads to the total acceptance probability being low and many samples not passing the acceptance test.

To overcome this problem, the authors of the SDDS propose to pair each smart proposal with its *dumb* version. And, when computing the acceptance test, consider the dumb reverse for the smart proposal and the other way around. Even as dumb proposals might be quite unlikely, the idea is that still, their probability would be higher than a smart reverse of a smart proposal.

Now, we consider the particulars of SDDS. The notation is the same as before: $\mathcal{L}^{(t)}$ is the current partition of N points at time t, m is the number of the clusters defined by $\mathcal{L}^{(t)}$, $N_i^{(t)}, \quad \forall i \in \{1 \dots m\}$ are the numbers of elements in the clusters according to the current partition. Additionally, the SDDS algorithm assumes that all points in the dataset are arbitrarily ordered prior to sampling, and their order remains fixed during the sampling. Also, we introduce an *action* variable $a_t \in \{$'SS-DM', 'SM-DS', 'DS-SM', 'DM-SS'$\}$, where the first letter stands for Smart or Dumb and the second for Split or Merge. Then, a single iteration of the SDDS algorithm is:

- Sample a_t uniformly.
- If $a_t =$ 'SS-DM', perform smart split
  - Sample cluster index i based on the inverse likelihood of each cluster: $i \sim \text{Cat}(p_1, \dots, p_m)$, $p_k \propto 1/P(\mathcal{R}_k \mid H_s)$, H_s is a single speaker hypothesis. Without loss of generality, assume $i = m$. Such choice of the distribution to sample the index of the cluster to split was proposed by the authors of the SDDS algorithm. However, we saw that this choice might be suboptimal. Below, when describing the Smart Split step in detail, we comment on what modifications we make to the original algorithm.
  - Introduce a new label $m + 1$.
  - Retrieve the indices for points in a cluster m, without loss of generality, assume that the points have indices $1, \dots, N_m^{(t)}$.
  - Assign the first point from $\mathcal{R}_m$ label m.
  - For the second point, compute the probabilities of it joining the first point or staying separate. Sample label m or $m + 1$ based on these probabilities.
  - Repeat the same procedure for the remaining points, i.e., sample one label at a time based on the probabilities of this point joining one of the two emerging clusters.
  - Compute a value of an acceptance probability taking into account that the inverse proposal is a dumb merge. If the proposal is accepted, $\mathcal{L}^{(t+1)}$ has $m + 1$ clusters. The new numbers of cluster elements: $N_i^{(t+1)} = N_i^{(t)} \ \forall i < m$ and, $N_m^{(t+1)} + N_{m+1}^{(t+1)} = N_m^{(t)}$.
- If $a_t =$ 'SM-DS', perform smart merge
  - Randomly sample cluster index i. Assume, $i = m$.
  - The second cluster j is sampled based on the posterior to be merged with the cluster i: $j \sim \text{Cat}(p_1, \dots, p_{m-1})$, $p_k \propto P(\mathcal{R} \mid \mathcal{L}_k^{(t+1)})P(\mathcal{L}_k^{(t+1)})$, where $\mathcal{L}_k^{(t+1)}$ is a partition where clusters i and k have the same speaker label. Assume, $j = m - 1$.
  - For each point r_k such that $l_k^{(t)} = m$, assign label $m - 1$.

  - Compute the acceptance probability taking into account that the inverse proposal is a dumb split. If the proposal is accepted, $\mathcal{L}^{(t+1)}$ has $m-1$ clusters. The new numbers of clusters elements: $N_i^{(t+1)} = N_i^{(t)} \;\; \forall i < m-1$ and, $N_{m-1}^{(t)} + N_m^{(t)} = N_{m-1}^{(t+1)}$.

- If $a_t =$ 'DS-SM', perform dumb split
  - Randomly sample cluster index i. Assume, $i = m$.
  - Introduce a new label $m+1$.
  - For each point r_k such that $l_k^{(t)} = m$, assign independently and with equal probability label m or $m+1$.
  - Compute the acceptance probability taking into account that the inverse proposal is a smart merge. If the proposal is accepted, $\mathcal{L}^{(t+1)}$ has $m+1$ clusters. The new numbers of clusters elements: $N_i^{(t+1)} = N_i^{(t)} \;\; \forall i < m$ and, $N_m^{(t+1)} + N_{m+1}^{(t+1)} = N_m^{(t)}$.

- If $a_t =$ 'DM-SS', perform dumb merge
  - Randomly and independently sample two cluster indices i, j. Assume, $i = m, j = m-1$.
  - For each point r_k such that $l_k^{(t)} = m$ assign a label $m-1$.
  - Compute the acceptance probability taking into account that the inverse proposal is a smart split. If the proposal is accepted, $\mathcal{L}^{(t+1)}$ has $m-1$ clusters. The new numbers of clusters elements: $N_i^{(t+1)} = N_i^{(t)} \;\; \forall i < m-1$ and, $N_{m-1}^{(t)} + N_m^{(t)} = N_{m-1}^{(t+1)}$.

This is a general procedure for performing a single step in SDDS. Let us consider possible proposal strategies separately and see how to compute the acceptance probability for each of them.

Smart Split - Dumb Merge, Dumb Merge - Smart Split

When performing a smart split, the first step is to select a cluster for splitting. According to the paper proposing SDDS [Wang and Russell, 2015], it should be sampled from a distribution inversely proportional to the likelihoods of individual clusters, i.e., cluster m is chosen with probability proportional to $\frac{1}{P(\mathcal{R}_m|H_s)}$. In our experiments, however, we saw that this approach results in a very high probability for small clusters (sometimes consisting of a single point) to be selected for a split. Hence, we modified the probability of selecting cluster m for a split as follows:

$$p_m \propto \frac{\prod_{j=1}^{N_m^{(t)}} P(r_j)}{P(\mathcal{R}_m \mid H_s)}. \tag{5.26}$$

The intuition behind using (5.26) is to promote splitting those clusters which would "like" to be split, i.e., those for which the likelihood of splitting the cluster into individual points is high relative to the likelihood of that cluster as a whole.

Once the cluster to split is selected, for the first point of the cluster r_1, the label m is assigned with the probability 1. Suppose, the assignment of the points $r_1, \ldots, r_{k-1}$ is already done: they were assigned labels $\mathcal{L}_{k-1} = \{l_1, \ldots, l_{k-1}\}$. We denote the subset of points from

$r_1, \ldots, r_{k-1}$, which were assigned label m, as $\mathcal{R}'_m$, and similarly, points from $\mathcal{R}'_{m+1}$ were assigned label $m+1$ ($\mathcal{R}'_m \cap \mathcal{R}'_{m+1} = \emptyset$). Now, we are assigning label $l_k \in \{m, m+1\}$ to r_k. It is sampled with probability

$$P(l_k = i \mid \mathcal{R}'_m, \mathcal{R}'_{m+1}, r_k, \mathcal{L}_{k-1}) = \frac{P(\mathcal{R}'_i, r_k \mid \mathcal{L}_{k-1}, l_k = i) P(l_k = i \mid \mathcal{L}_{k-1})}{\sum_{j \in \{m, m+1\}} P(\mathcal{R}'_j, r_k \mid \mathcal{L}_{k-1}, l_k = j) P(l_k = j \mid \mathcal{L}_{k-1})}. \tag{5.27}$$

Then, the probability of the full split is computed iteratively:

$$q(\mathcal{L}^{(t+1)} \mid \mathcal{L}^{(t)}) \propto \frac{\prod_{j=1}^{N_m^{(t)}} P(r_j)}{P(\mathcal{R}_m \mid H_s)} \prod_{k=2}^{N_m^{(t)}} P(l_k \mid \mathcal{R}'_m, \mathcal{R}'_{m+1}, r_k, \mathcal{L}_{k-1}). \tag{5.28}$$

The inverse proposal is a dumb merge. Its proposal probability is:

$$q(\mathcal{L}^{(t)} \mid \mathcal{L}^{(t+1)}) = 2 \frac{1}{m+1} \frac{1}{m}, \tag{5.29}$$

where components $1/(m+1)$ and $1/m$ correspond to selecting two clusters one after the other uniformly from $m+1$ clusters defined by $\mathcal{L}^{(t+1)}$. The scalar 2 corresponds to the fact that two clusters could be selected in a different order but resulting in the same partition $\mathcal{L}^{(t)}$.

So, we have all components in place to be able to compute the acceptance probability. Proposal probabilities are given by (5.28) and (5.29), and the likelihood and the prior are defined by the model.

Proposal probabilities for a symmetrical move dumb merge – smart split are computed analogously to (5.28) and (5.29). Proposal probability for the dumb merge is given by:

$$q(\mathcal{L}^{(t+1)} \mid \mathcal{L}^{(t)}) = \frac{1}{m} \frac{1}{m-1}, \tag{5.30}$$

notice that the values in denominator are different from (5.29). That is because now the merging happens for the partition $\mathcal{L}^{(t)}$ with m clusters.

The inverse proposal now is the smart split; the split follows the same procedure as described above for the direct smart split proposal, i.e., point-by-point assignment of new labels for the cluster to split. Then, its proposal probability is given as follows:

$$q(\mathcal{L}^{(t)} \mid \mathcal{L}^{(t+1)}) \propto \frac{\prod_{j=1}^{N_{m-1}^{(t+1)}} P(r_j)}{P(\mathcal{R}_{m-1} \mid H_s)} \prod_{k=2}^{N_{m-1}^{(t+1)}} P(l_k \mid \mathcal{R}'_{m-1}, \mathcal{R}'_m, r_k, \mathcal{L}_{k-1}). \tag{5.31}$$

Above, we took into account the fact that $\mathcal{L}^{(t+1)}$ after dumb merge defines only $m-1$ clusters, $N_{m-1}^{(t+1)}$ is the total number of elements in the newly merged cluster $m-1$.

Notice that smart split processes the data points in a specific order. For the same cluster, if the points are reordered, (5.28) would result in a different proposal probability and possibly a different split. That is why SDDS has a requirement to define and fix the order of the data points before running the sampling procedure. Otherwise, one would have to account for all possible permutations of the points in the cluster $m-1$ when computing the probability of a smart split inverse proposal. However, as we order the points in advance, there is only one way of performing a smart split step and, consequently, only one way of computing the inverse proposal probability for dumb merge.

Smart Merge - Dumb Split, Dumb Split - Smart Merge

For the smart merge, first, a random cluster is selected with probability $\frac{1}{m}$. Then, the second one is sampled with a probability proportional to the likelihood of a merged cluster: $P(\mathcal{R}_m, \mathcal{R}_{m-1} \mid H_s)$. So, the proposal probability is:

$$q(\mathcal{L}^{(t+1)} \mid \mathcal{L}^{(t)}) = \frac{1}{m} \frac{P(\mathcal{R}_m, \mathcal{R}_{m-1} \mid H_s) P(\mathcal{L}^{(t+1)}_{m-1})}{\sum_{j=1}^{m-1} P(\mathcal{R}_m, \mathcal{R}_j \mid H_s) P(\mathcal{L}^{(t+1)}_j)} = \frac{c_m}{m} P(\mathcal{R}_m, \mathcal{R}_{m-1} \mid H_s) P(\mathcal{L}^{(t+1)}_{m-1}). \tag{5.32}$$

Above, we denoted the normalizer $1/\sum_{j=1}^{m-1} P(\mathcal{R}_m, \mathcal{R}_j \mid H_s) P(\mathcal{L}^{(t+1)}_j)$ as c_m. The probability of an inverse dumb split proposal is calculated as:

$$q(\mathcal{L}^{(t)} \mid \mathcal{L}^{(t+1)}) = \frac{1}{m-1} \left(\frac{1}{2}\right)^{N^{(t+1)}_{m-1}}, \tag{5.33}$$

where the first ratio accounts for the probability of a cluster to be selected for a split (uniformly out of $m-1$ clusters defined by $\mathcal{L}^{(t+1)}$). The second term is a probability of a particular split for the selected cluster. Again, we have all of the elements to compute the acceptance test.

The symmetrical dumb split step uses similar equations. First, the proposal probability is calculated as:

$$q(\mathcal{L}^{(t+1)} \mid \mathcal{L}^{(t)}) = \frac{1}{m} \left(\frac{1}{2}\right)^{N^{(t)}_m}. \tag{5.34}$$

And, the probability of the inverse smart merge is

$$q(\mathcal{L}^{(t)} \mid \mathcal{L}^{(t+1)}) = (c_m + c_{m+1}) \frac{1}{m+1} P(\mathcal{R}_m, \mathcal{R}_{m+1} \mid H_s) P(\mathcal{L}^{(t)}_{m+1}). \tag{5.35}$$

Notice that the inverse smart merge now has a scalar $(c_m + c_{m+1})$. This accounts for the fact that for a merge, one cluster is selected randomly, and the second is joined with it. So, we have to consider both cases: when the second cluster was merged with the first and the other way around. c_m denotes the normalizing coefficient for selecting cluster m first and merging $m+1$ with it, and c_{m+1} corresponds to the second case.

5.4.5 Combination of SDDS and GS

The proposal strategy of SDDS allows for large steps in a sample space while preventing the acceptance rate from being extremely low. The authors of the method propose to combine it with Gibbs Sampling. They suggest interleaving SDDS steps with GS. The motivation is to combine the advantages of both strategies: on the one hand, SDDS accounts for the dramatic movements in the state space, and on the other hand, GS can refine the samples by changing the labels of the individual observations. For example, GS would not have any problems when a single point has to be moved from one cluster to another. In contrast, SDDS would have to perform a split isolating this single point into a separate cluster first, and only after that do a merge of this one-point cluster with the second one. The intermediate sample might be very unlikely, so it is almost impossible to accept it. In that case, SDDS would not be able to perform this simple adjustment of the sampler state.

Notice that the authors of SDDS propose to do a full iteration of GS between each SDDS step. That means that GS has to be run for all of the points. However, this slows the sampling process significantly as each label has to be sampled individually. Instead, we propose to sample just a few labels after each SDDS step. For example, in most of our experiments, we sample as many labels after each SDDS proposal as needed to pass through all data points three times during sampling. Moreover, we propose using the approximate version of GS, where several labels are sampled simultaneously. It is not theoretically correct as the probabilities for the next label to sample depend on the sampled values from the previous ones. However, assuming that the number of labels to sample is much smaller than the total number of points, we assume that the GS points are independent of each other, and their sampled labels will not be significantly affected by the approximate sampling procedure.

To summarize, SDDS in combination with GS provides a way of sampling the partitions from their posterior distribution. Using it, we can employ the contrastive divergence strategy to compute the approximate gradient of the objective (5.4). Hence, we can use, e.g., gradient ascent to train the parameters of the HT-PLDA model.

An important question to be answered when using such complicated sampling methods as described above is how to verify that the sampler is implemented correctly and indeed generates samples from the intended distribution. Also, it is important to be able to track when the sampler is warmed-up and to compare different samplers (e.g., comparing SM against SDDS). As these questions deviate from the main storyline of this thesis, we do not include this discussion here, but we comment on them in Appendix D.

5.4.6 Speaker Verification experiments and results

For the experiments on training the HT-PLDA model by maximizing the approximate posterior of the correct partition with contrastive divergence, we use only x-vector embeddings. The experiments are split into two main parts. We start with the experiments aimed at answering the questions of how different settings of the SDDS sampler affect the model training. In particular, we are interested in what impact the initialization of the Markov chain has on the convergence of the algorithm, how many Gibbs sampling steps have to be done between each iteration of the SDDS algorithm, what is the difference between using samples from the same Markov chain compared to running several chains in parallel, and what is the necessary number of samples to use for the gradient approximation. These experiments are done by training the model on a smaller VoxCat-S set (as described in Section 2.5). The experiments and results are presented in Appendix E.

After that, we train new models with those sampler settings that we found to be optimal and see how the trained models perform on speaker verification and diarization tasks. The speaker verification results are presented in this section, while diarization experiments are shown in the next Section 5.4.7.

Summarizing the findings of the experiments presented in Appendix E, we are performing the training with the approximate partition posterior objective with the following settings: we use a single sample from the posterior; to get this sample, we run SDDS for 200 iterations and interleave them with Gibbs Sampling so that we pass through all the training data at least once in the course of sampling. Markov chain at each iteration is initialized with the sample used at the previous iteration, i.e., we perform Persistent Contrastive Divergence [Tieleman, 2008] rather than a standard CD. In this section, we are interested to see how the improvement of the true partition posterior probability translates into speaker verification performance.

We train the HT-PLDA model on the whole VoxCat set, unlike in the experiments on the sampler settings (Appendix E) where only a smaller VoxCat-S was used. The embeddings we use are x-vectors from Section 2.4.2 trained on VoxCeleb2 set. Prior to training the HT-PLDA model, the embeddings are centered, and their dimensionality is reduced to 300 by LDA. Then, the embeddings are used as they are, or we perform the length normalization step. We test the performance on VoxCeleb1-O, VoxCeleb1-E, VoxCeleb1-H, and SITW core-core conditions.

Table 5.5 summarizes the speaker verification results for the experiments we performed. As before, as baseline models, we use generatively trained G-PLDA and HT-PLDA models trained on embeddings with and without LN (lines 1-4 of Table 5.5). Then, the last two lines of the table correspond to HT-PLDA models trained with Persistent Contrastive Divergence to maximize the approximate posterior of the correct data partition. As can be seen, for both normalized and unnormalized embeddings, discriminatively trained HT-PLDA models outperform corresponding HT-PLDAs trained generatively. However, the relative improvement is larger in the case of unnormalized embeddings. Also, discriminative training of HT-PLDA on embeddings without LN results in similar or better performance than both G-PLDA baselines. While when LN is applied to the embeddings, G-PLDA outperforms HT-PLDA models on most of the test conditions (all except VoxCeleb1-H and $C_{\text{min}}^{\text{Prm}}$ for SITW core-core). Thus, we can conclude that discriminative training brings some performance gain compared to generatively trained models. Interestingly, the best results are achieved when the HT-PLDA model is trained on the embeddings without LN. That supports our intuition that length normalization should be detrimental to the performance of HT-PLDA due to its effect on estimated uncertainty b.

5.4.7 Speaker Diarization experiments and results

As seen from the results of the Speaker Verification experiments, the approximate partition posterior is an effective objective for training the back-end HT-PLDA model for verification purposes. However, this objective is very different from the target objective of the speaker verification model. Thus, we believe that we might not observe all of the benefits of training the model to maximize the posterior of the correct partition when looking at speaker verification performance only.

Based on these considerations, we believe it makes sense to test the HT-PLDA trained with this objective on the diarization task. Maximizing the posterior of the correct partition is more consistent with the goal of diarization than with the verification problem. Hence, we expect it to be more beneficial to use a model trained with this objective in diarization.

We perform the diarization following the standard pipeline described in Section 2.3.1 (for all models except for the first baseline). All of the elements of the diarization system are fixed except for the PLDA model used to provide the LLR scores for the AHC clustering algorithm. Then, different PLDA models are compared in how they perform in the diarization task. We test the performance on both development and evaluation sets of DIHARD 2019. In this case, we use the development set for selecting the hyper-parameters for the training and the optimal threshold to stop merging clusters in AHC.

Baselines

In this experiment, we use four baselines (lines 1-4 of Table 5.6). All of them use the same set of x-vector embeddings as described in 2.4.2. The extractor and the baseline G-PLDA were

Table 5.5: Speaker recognition performance for the HT-PLDA models trained to maximize the approximate posterior of the correct partition of the training data. The performance is reported on VoxCeleb1-O, VoxCeleb1-E, VoxCeleb1-H, and SITW core-core condition in terms of $C_{\text{min}}^{\text{Prm}}$=minDCF$_{0.05}$, and EER(%).

#	System	LN	VoxCeleb1-O		VoxCeleb1-E		VoxCeleb1-H		SITW core-core	
			$C_{\text{min}}^{\text{Prm}}$	EER	$C_{\text{min}}^{\text{Prm}}$	EER	$C_{\text{min}}^{\text{Prm}}$	EER	$C_{\text{min}}^{\text{Prm}}$	EER
1	G-PLDA, EM	yes	**0.12**	**1.6**	0.13	**1.8**	0.23	3.7	0.21	3.0
2	G-PLDA, EM	no	0.18	3.0	0.17	3.0	0.26	5.1	0.30	8.1
3	HT-PLDA, VB	yes	**0.12**	2.0	0.13	2.1	0.21	3.6	0.19	4.6
4	HT-PLDA, VB	no	**0.12**	2.0	**0.12**	2.1	0.21	3.6	0.19	3.6
5	HT-PLDA,Part. Post. full	yes	**0.12**	1.9	**0.12**	2.0	**0.20**	3.4	0.19	4.3
6	HT-PLDA,Part. Post. full	no	**0.12**	1.7	**0.12**	**1.8**	0.21	**3.3**	**0.18**	**2.9**

trained on VoxCeleb2 and VoxCat sets, respectively. In all cases, embeddings were length-normalized before PLDA training. The first baseline system is the Kaldi [Povey et al., 2011] diarization recipe [Sell et al., 2018] with the same G-PLDA model as used in the second baseline. The difference between the two systems is the pre-processing that is done to the embeddings prior to scoring them with a PLDA: in addition to LN, Kaldi recipe performs per-utterance principal component analysis (PCA) dimensionality reduction to retain 30% of variability in the embeddings and, after that, additional length normalization. Also, the first two baselines use different versions of the AHC algorithm: the first baseline uses the UPGMA AHC, while in the other systems, we use the AHC algorithm with correct LLR scores for the intermediate steps. The first baseline is presented here to show the performance that can be achieved by using additional pre-processing of the embeddings. We could not use the same pre-processing of the embeddings for the trained HT-PLDA model because of its requirement to have the hidden speaker space of significantly lower dimensionality than the observed space. Hence, the second baseline is presented for the comparison of G-PLDA and HT-PLDA with fixed pre-processing of the embeddings.

The third line shows the performance of the diarization system with a G-PLDA model given by interpolation of two G-PLDAs. One of them is the same model as used in baseline 2 (trained on VoxCat), and the second one is the G-PLDA model trained on AMI data. The interpolation weights for both models are equal to 0.5. Finally, the last baseline (line 4) is the same as baseline 3, just this time, we transform the G-PLDA to HT-PLDA by setting the degrees of freedom parameter to 2.

Discriminative training strategy

For training with the objective of maximizing the posterior of the correct partition, we adopt a different strategy than we used in speaker verification experiments. Now, we do not use all training data as a single training example. Rather, each utterance from the training set is used separately. AMI dataset consists of meeting recordings, consequently, each utterance contains speech of several speakers. Each utterance is processed the same way as the test diarization data, i.e., split into short overlapping speech segments. In this case, each training utterance represents a separate partitioning problem. Then, we can define the objective for each utterance separately: to maximize the posterior of the correct partition (the correct diarization solution) for this particular utterance. One can compute an approximate gradient by using a sampling algorithm for each training utterance separately. We use the following strategy for training the model: we compute the approximate gradients for ten utterances, average them, and make a single update to the model parameters. Then, the next ten utterances are used to make the next update, so we proceed until there is no more data left. Then, the utterances are shuffled so that new batches are formed, and the process is repeated.

To compute the gradient for a single utterance, we use one sample from the SDDS algorithm. SDDS iterations are interleaved with Gibbs sampling steps so that each point is visited three times during the sampling process. Unlike for speaker verification, we do not use persistent contrastive divergence as by the time we visit the same utterance again, the model parameters have already changed substantially. Instead, SDDS is initialized with the finest partition (each data-point forms a separate cluster) every time. To compensate for the initialization that is far from the high probability region of the posterior, we run more SDDS steps than in the verification experiments; in all diarization experiments, the number of SDDS steps was fixed to 900. The prior for each training utterance is set so that the expected number of speakers in the utterance is the true one (see Appendix B).

Training data

To train the model in the way described above, we need a training set where each utterance contains speech of several speakers and is annotated with diarization labels. That is why we introduced an additional dataset (AMI) for training compared to the baselines trained on VoxCat only. In our experiments, we explore two strategies for discriminative training of the model on the additional data (AMI dataset in addition to VoxCat). The first approach we follow is to initialize the HT-PLDA model on the G-PLDA model trained on the VoxCat set (baseline 2), then retrain the model on AMI with regularization towards initial parameters' values. The second approach is to assume that we have two PLDA models, one is the HT-PLDA model based on baseline 2 (that is fixed), and the other is the HT-PLDA we train on AMI (this one is initialized from the baseline PLDA). Then, the two models are interpolated, and the SDDS sampling is done with the parameters of the interpolated model. However, the approximate gradient is computed with respect to the parameters of the second HT-PLDA only. In other words, we train the second model to interpolate well with the initial one.

When just a single training dataset is used to train the model (lines 1-2), we center the evaluation data with the mean of this dataset. When the model uses both VoxCat and AMI training data (i.e., lines 3-6), the evaluation data are centered with the vector that is the average of the means of two training sets, while each training set is centered with its own mean. In our experiments, we have noticed that the proper centering of the evaluation data is critical: the results significantly degrade when only one of the training means (i.e., VoxCat or AMI) is used for centering or, even when the evaluation data are centered with their own mean. We do not understand the reasons for this behavior and would like to investigate this in the future.

Results and discussion

The results of four baseline systems, along with two discriminatively trained models, are shown in Table 5.6. For each system, the results are shown for two values of threshold σ used to stop merging clusters in AHC. One is $\sigma = 0$ – optimal threshold if the PLDA model outputs proper LLR scores; the other is the optimal value of σ tuned on the development set, i.e., optimal σ is set in the oracle way for "dev" set.

Comparing lines 1 and 2 of Table 5.6, we see the effect of the additional pre-processing done by the first approach as well as the alternative AHC algorithm (UPGMA for the first approach). When comparing lines 2 and 3, we should see what improvement comes from using the additional data for training. Lines 3 and 4 demonstrate the effect of having the HT-PLDA model instead of the G-PLDA. Finally, comparing the discriminatively trained HT-PLDA models (lines 5 and 6) with baseline 4 shows the effect of discriminative training.

First, we want to point out that all four baselines show similarly good performance if one is allowed to tune the threshold of the clustering algorithm. In particular, we do not see any significant improvement in the optimal performance when using the Kaldi recipe compared to our baselines without additional PCA and length-normalization done before scoring the embeddings. However, taking the default zero threshold results in significant performance degradation for our Gaussian PLDA models (lines 2 and 3), while the first baseline shows a smaller performance drop. That indicates that the G-PLDA scores are not correct LLRs; in other words, the scores provided by the model are miscalibrated, although, for the Kaldi

Table 5.6: Comparison of diarization performance of G-PLDA and HT-PLDA on DIHARD 2019 development and evaluation sets. HT-PLDA is trained discriminatively to maximize the approximate posterior of the correct diarization of the training utterances. The optimal AHC stopping threshold is selected to minimize DER on DIHARD 2019 development set. The performance is reported in terms of DER (%).

#	System	σ=0		σ optimal	
		dev	eval	dev	eval
1	Baseline Kaldi	27.12	27.74	20.45	21.35
2	G-PLDA Vox	50.78	49.15	21.81	22.38
3	G-PLDA Vox+AMI	61.12	59.82	22.28	22.24
4	HT-PLDA Vox+AMI	20.41	21.80	20.20	21.66
5	HT-PLDA Vox →AMI, Part. Post. full	20.29	21.18	**19.83**	**20.86**
6	HT-PLDA Vox + AMI, Part. Post. full	**20.28**	**21.16**	19.89	20.92

recipe (line 1), miscalibration is less severe. An interesting observation is that when we interpolate two G-PLDA models (line 3), the performance does not improve compared to the single model even with the optimally set threshold, i.e., we cannot conclude that the additional training data definitely improves the performance (at least in our experiments).

Second, when heavy-tailed behavior is turned on for the interpolation of the models, we see an improvement in the performance of the resulting model for both optimal and zero threshold scenarios. HT-PLDA model outperforms or results in a similar performance to other baselines, and, most importantly, we do not observe large differences in the optimal and zero threshold setting for this model.

Comparing the discriminatively trained HT-PLDA models (lines 5 and 6) to comparable G-PLDA baseline systems (lines 2 and 3), we see that the former have better performance in both scenarios: the default threshold $\sigma = 0$ and the optimally set threshold. However, the improvement from discriminative training of HT-PLDA is relatively small (compare lines 5-6 to line 4). It shows that interpolation of two HT-PLDA models is a good back-end model for diarization and indicates that how the model was trained is less important.

We notice that both discriminatively trained models, the one trained to combine well with the baseline PLDA (line 6) and the one initialized from the baseline and retrained on AMI (line 5), perform similarly well in both scenarios: when the AHC stopping threshold is set in an oracle way or when it is set analytically. Moreover, discriminative training of HT-PLDA results in a well-calibrated system (the performance differences between the default and optimal thresholds are rather small).

Finally, the discriminatively trained HT-PLDA outperforms the first baseline, especially in the case of the analytically set threshold. This shows that the additional PCA and LN done by the Kaldi baseline are not essential for good diarization performance.

5.5 Partitioning tuples of recordings

Training the HT-PLDA model to maximize the approximate posterior of the correct partition was discussed in detail in the previous Section 5.4. The sampling-based method was shown moderately effective for training HT-PLDA models, but it has a significant drawback: its high computational and time complexity. To cope with this problem, we propose to use a stochastic version of the same objective. We follow the same principle as in the case of training the model with the BXE objective. There, training data were organized as a set of binary trials considered independent. Then, the objective is a sum of objective functions computed for each pair. Here, we repeat the same procedure and reorganize the data into tuples (sets of several individual data points). The number of elements in a tuple (in the following, the size of a tuple) is selected to be low enough to allow for exactly computing the posterior of the correct partition. And the objective function we consider is a multi-class cross-entropy.

$$E(\theta) = \sum_{(\mathcal{R}_t, \mathcal{L}_t^*) \in \mathcal{T}} - \log P(\mathcal{L}_t^* \mid \mathcal{R}_t, \theta) \tag{5.36}$$

$$= \sum_{(\mathcal{R}_t, \mathcal{L}_t^*) \in \mathcal{T}} - \log \frac{P(\mathcal{R}_t \mid \mathcal{L}_t^*, \theta) P(\mathcal{L}_t^*)}{\sum_{\mathcal{L}_t'} P(\mathcal{R}_t \mid \mathcal{L}_t', \theta) P(\mathcal{L}_t')}. \tag{5.37}$$

Above, we denote as $\mathcal{T}$ a set of all possible (tuple $\mathcal{R}_t$, partition $\mathcal{L}_t^*$) pairs of a feasible size that can be created out of the set $\mathcal{R}$ with labels $\mathcal{L}^*$. The number of the components in the denominator of each term is the N_t-th Bell number, where N_t is the size of the corresponding tuple.

Notice that when $\mathcal{T}$ is limited to include pairs of points, this objective is just BXE. However, when tuples can be larger, the objective becomes more general than the binary cross-entropy. And still, it is much simpler compared to partitioning a large dataset.

In the case of BXE, we discussed two approaches to training the model. One is to use some full-batch optimization method, e.g., L-BFGS as used in [Burget et al., 2011] to train the G-PLDA model. The other is to rely on a stochastic optimization method such as SGD. However, the prerequisite for SGD is that the objective function can be broken up into many terms, such that each of those terms (when scaled up) is an unbiased estimate of the full criterion. That is exactly the case for BXE, and we use stochastic optimization when training HT-PLDA with this objective.

In the case of the objective (5.36), most probably, full-batch optimization will be infeasible due to a large number of tuples in $\mathcal{T}$. But luckily, we can use the stochastic method as the objective can be broken up into separate batches similar to BXE. Moreover, we propose to generate the batches dynamically and define one iteration as a certain number of parameter updates (batches seen) rather than the number of batches needed to see all of the training examples. As we do not plan to use all possible training examples, we propose the following scheme to generate them as the training progresses:

- Fix the prior distribution over partitions for the expected size of the training examples. If the size is fixed, there will be only one distribution. If the size varies, one prior distribution for each possible size of the training tuple has to be generated. Here, we use a CRP prior with fixed parameters. Such training examples generation ensures that the training data agrees with the prior. As discussed in Appendix B, this is important to assure that the trained model provides proper likelihoods.

- Fix some order of the speakers in the training data set.

- For each individual example $t \in \mathcal{T}$:
  - Sample the size N_t of the example (optional).
  - Sample one partition from the prior over B_{N_t} options. Suppose the partition has m_t clusters.

- For each cluster in the partition, sample the needed number of utterances from the data of the next speaker on the list. If the current speaker does not have enough utterances, skip to the next speaker.

Once we pass over all the training speakers, we shuffle the speakers and start from the beginning of the new speaker list. Constructing training examples in this way, we ensure that all training speakers are used.

In Appendix F, we describe the procedure allowing us to evaluate the objective (5.36) efficiently by splitting each training minibatch into sets of examples of the same size N_t. Then, for each of these sets, one can pre-compute several sparse matrices allowing for fast summation to calculate speaker statistics (3.34) and likelihoods (3.33).

5.5.1 Speaker Verification experiments and results

Size of the tuples

Before, we mentioned that the tuples used as training examples must have a reasonable size, but we did not elaborate further. As mentioned, the main computational challenge when computing partition posterior for a single tuple is computing its normalizing term. For a tuple of size 8, there are $B_8 = 4140$ components in the normalizing term, while for size 9, there are $B_9 = 21147$. That is already prohibitively high taking into account that for a single update, we would have to compute the posterior for a large number of tuples. So, we limit the size of tuples to 8 or less in our experiments.

Also, it is unclear what size of the examples will be beneficial in terms of performance and whether it should be the same for all of the examples. To address this question, we run the following experiments. The experimental setup here is the same as in Sections 5.2.1, 5.3.1, 5.4.6, i.e., we use x-vector embeddings extracted for VoxCat set. The embeddings are centered, and their dimensionality is reduced to 300 with LDA. We did not use length normalization in this experiment. The results of the baseline models as well as of the models trained to partition tuples of recordings are shown in Table 5.7.

As two baselines, we use the generatively trained G-PLDA and the HT-PLDA model discriminatively trained with the BXE objective (line 4 of Table 5.2). Baselines are trained using the same data as we use for the discriminative training to maximize the posterior of the correct partition for training examples. All of the trained models are initialized with the G-PLDA (baseline model, line 1 of Table 5.7) and, as before, we use L_2 regularization of the model parameters towards the initial ones.

To begin with, we train the HT-PLDA model with objective (5.36) and the tuple size is fixed to 2; the results of this model are presented on line 3 of Table 5.7. The model trained in this way should be equivalent to the baseline from line 2. The only difference is how we form batches and define iterations in the new setting. Before, we always made sure that all of the examples were used for training, while now, some of the examples might be used several times while some are not used at all.

Then, we increase the size of training examples to 4 and 8; the results of these models are shown in lines 4 and 5, respectively. Finally, for the training, we sample tuples of sizes varying from 2 to 8, and tuples of each size are sampled with equal probability (line 6 of Table 5.7).

Analyzing the results, one notices that training on the pairs of recordings (line 3) results in slightly worse performance than training with the BXE objective on all possible pairs (line 2). It is understandable as the training objective in these two cases is the same, but

Table 5.7: Comparison of different sizes of the training examples for training HT-PLDA model to partition tuples of recordings. The experiments are done with x-vector embeddings without length normalization. The results are shown for VoxCeleb1-O and SITW core-core conditions in terms of $C_{\text{min}}^{\text{Prm}}$=minDCF$_{0.05}$, and EER(%).

#	System	Size	VoxCeleb1-O		SITW core-core	
			$C_{\text{min}}^{\text{Prm}}$	EER	$C_{\text{min}}^{\text{Prm}}$	EER
1	G-PLDA EM	-	0.18	3.0	0.30	8.1
2	HT-PLDA BXE	2	0.12	1.8	0.18	3.1
3	HT-PLDA Part.Post tuple	2	0.12	1.9	0.19	3.4
4	HT-PLDA Part.Post tuple	4	0.12	1.8	0.18	3.3
5	HT-PLDA Part.Post tuple	8	0.11	1.7	0.18	3.0
6	HT-PLDA Part.Post tuple	2-8	0.12	1.7	0.17	2.9

the strategy of presenting the examples to a model is different. In the former case, not all of the possible training pairs are used.

Training on examples of larger size leads to moderate performance improvements, with the best performance achieved when examples of 8 embeddings are used for training. However, it has to be said that for smaller training examples (pairs of recordings), we observed that the model quickly learns to correctly classify most of the examples. Consequently, the error on these examples is low; thus, the gradients are close to zero. In other words, training on such examples is ineffective, and the actual training happens only when the model is presented with some hard example. In the case of larger examples, many more of them are challenging for the model, and thus, the training is more effective.

Finally, training on examples of various lengths leads to the same or better performance than training on the largest tuples only. However, such training takes longer to complete as for training examples of each size, the objective and the gradient have to be computed separately (see Appendix F), while for the batches consisting of examples of the same size, the gradient can be computed more effectively. Thus, we will not proceed with training on examples of varying sizes.

Approach to construct the training examples

In the previous experiment, we noticed that many of the training examples are very easy for the model. Then, most of the training examples quickly become useless for the training as the error for them is low and the gradient is close to zero. That slows down the training, and, as we expect, it leads to poorer performance of the trained model. To overcome this problem, we generated the training examples so that they are more difficult for the model and, consequently, the training is more effective. That is done by modifying the method of example generation described above. As follows from the process, a single example consists of data from speakers nearby in the speaker list. We make use of this fact and place the most similar speakers next to each other on the list so that when the example is created, it will be composed of utterances of similar speakers. Consequently, it would be harder for the model to separate their clusters compared to completely different speakers. To place similar speakers nearby in the list, we follow the procedure below.

Every time we need to reset the speaker list, we use the current model parameters to

Table 5.8: Comparison of two strategies for generating training examples for training HT-PLDA model to partition tuples of recordings. The experiments are done with x-vector embeddings without length normalization. The results are shown for VoxCeleb1-O and SITW core-core conditions in terms of $C_{\text{min}}^{\text{Prm}}$=minDCF$_{0.05}$, and EER(%).

#	System	Size	Examples	VoxCeleb1-O		SITW core-core	
				$C_{\text{min}}^{\text{Prm}}$	EER	$C_{\text{min}}^{\text{Prm}}$	EER
1	HT-PLDA Part.Post tuple	2	random	0.12	1.9	0.19	3.4
2	HT-PLDA Part.Post tuple	8	random	0.11	1.7	0.18	3.0
3	HT-PLDA Part.Post tuple	2	hard	0.11	1.6	0.16	2.7
4	HT-PLDA Part.Post tuple	8	hard	0.11	1.7	0.18	2.9
5	HT-PLDA Part.Post tuple	2	hard+random	0.11	1.7	0.17	2.7
6	HT-PLDA Part.Post tuple	8	hard+random	0.12	1.6	0.16	2.6

build a matrix of pairwise similarity scores (LLR speaker verification scores). For enrollment and test, we use all data available for the speakers: embeddings for each speaker are averaged so that there is a single embedding per speaker. Then, the first speaker is selected randomly. The second one is the one that has the highest similarity score to the first; the third is the most similar to the second among the speakers that are left, and so on. We proceed in this way till the end of the list. As a result, we have a list of speakers, where each next one is the most similar available speaker to the previous one.

We compare this approach with the one where the speakers are placed randomly. Table 5.8 shows the results. Here, we use the HT-PLDA models trained on randomly generated examples of sizes 2 and 8 as two baselines. The results are presented in lines 1 and 2 of Table 5.8 and repeat the results from Table 5.7. Then, two models are trained with the hard example generation procedure described above; their results can be found in lines 3 and 4. Finally, we test the scenario when half of the examples are generated randomly, and the other half is supposed to be more difficult for the model. Our motivation for including these results is to prevent the model from over-training on hard training examples by sacrificing the performance on easier ones. The results of the models trained in this way are presented in lines 5 and 6 of Table 5.8.

Results indicate that for training with pairs of recordings, our scheme of generating harder training examples was effective: there is a consistent performance improvement of the model trained only on hard examples compared to the random example generation strategy. However, we do not see a similar effect for big training examples. Training only on hard tuples of size eight does not bring any performance gain compared to random examples. It can be explained by the fact that larger examples of eight recordings are already challenging enough for the model to train effectively even on randomly generated examples. When the model is trained on the mixture of randomly generated and hard examples, we notice that for pairs of recordings (tuples of size two), there is no performance gain in adding easier examples compared to the model trained only on difficult pairs. In contrast, for tuples of size 8, there is a small improvement when random examples are added to the hard ones.

Finally, we present the results on all test conditions VoxCeleb1-O, VoxCeleb1-E, VoxCeleb1-H, and SITW core-core for embeddings with and without LN. These results are achieved with the models trained with the strategy that we found the most beneficial in the previous experiments: we train HT-PLDA models discriminatively with objective (5.36) on the mixture of hard and randomly generated examples of size 8. The results of these models, along with those of baseline G-PLDA and HT-PLDA models, are shown in Table 5.9.

Similar to other discriminative training approaches, we notice that HT-PLDA trained on unnormalized embeddings performs better than the one trained on x-vectors with LN. Moreover, this model outperforms all generatively trained baseline models: G-PLDA and HT-PLDA trained on embeddings with and without LN.

5.5.2 Speaker Diarization experiments and results

Similarly to training HT-PLDA to maximize the posterior probability of the correct partition for the whole dataset, we test the models trained with objective (5.36) on the speaker diarization task. Our experimental setup repeats the experiments of Section 5.4.7: we use a pre-trained baseline G-PLDA model trained on VoxCat set and then discriminatively retrain it using AMI dataset. In all cases, we use x-vector embeddings trained on VoxCeleb2; the embeddings are always length-normalized before model training. At test time, we score the embeddings of the test utterances with a trained PLDA model; the matrix of pairwise LLR scores is used as an input to the AHC algorithm that is supposed to cluster the embeddings into speaker clusters. Similarly to Section 5.4.7, we center each training dataset (VoxCat or AMI) with its own mean vector. When two datasets are used for training the model, the evaluation data are centered with the average of the means of two sets.

In the diarization experiment, we deviate from the training examples preparation scheme presented above. Instead, the training examples for the objective (5.36) are generated by the following procedure:

- Sample a training utterance from AMI dataset.
- For the given utterance, sample 8 speech segments randomly. Fix some order of the segments and use their corresponding speaker labels to define a correct partition for the example.

As a prior, in this case, we use CRP described in Appendix B with parameters set in such a way that for the total number of segments of the whole training set (sum of segments for all training utterances), the expected number of speakers (see (B.5)) is equal to the true number of speakers in the training set. That is done by setting one of the parameters to zero and finding the other one by a numerical optimization method.

The results of the baseline and discriminatively trained HT-PLDA models are presented in Table 5.10. The first part of the table repeats the baseline results presented in Section 5.4.7. The first baseline is Kaldi recipe [Sell et al., 2018] that, in addition to centering and LN done by all models, performs per-utterance PCA dimensionality reduction and additional LN. Then, there are two G-PLDA models: one is trained on VoxCat only; the other one is the interpolation of two G-PLDAs trained on VoxCat and AMI sets. Finally, the last baseline is the interpolated G-PLDA that was turned into HT-PLDA by setting the degrees of freedom parameter to 2.

The second part of the table corresponds to the results of HT-PLDA models trained with objective (5.36) on AMI dataset. Similar to Section 5.4.7, we explore two ways of training the model on the additional training data. The first one is initializing the parameters with the parameters of the baseline VoxCat PLDA model and retraining them with regularization towards their initial values (line 5 of the table). The second is training a separate HT-PLDA model that is interpolated with the baseline model (line 6).

Table 5.9: Speaker recognition performance of generatively trained PLDA models and HT-PLDA models trained to maximize the posterior of the correct partition for training tuples of 8 recordings. The performance is reported on VoxCeleb1-O, VoxCeleb1-E, VoxCeleb1-H, and SITW core-core condition in terms of $C_{\text{min}}^{\text{Prm}}$=minDCF$_{0.05}$, and EER(%).

#	System	LN	VoxCeleb1-O		VoxCeleb1-E		VoxCeleb1-H		SITW core-core	
			$C_{\text{min}}^{\text{Prm}}$	EER	$C_{\text{min}}^{\text{Prm}}$	EER	$C_{\text{min}}^{\text{Prm}}$	EER	$C_{\text{min}}^{\text{Prm}}$	EER
1	G-PLDA, EM	yes	0.12	**1.6**	0.13	**1.8**	0.23	3.7	0.21	3.0
2	G-PLDA, EM	no	0.18	3.0	0.17	3.0	0.26	5.1	0.30	8.1
3	HT-PLDA, VB	yes	0.12	2.0	0.13	2.1	**0.21**	3.6	0.19	4.6
4	HT-PLDA, VB	no	0.12	2.0	**0.12**	2.1	**0.21**	3.6	0.19	3.6
5	HT-PLDA,Part. Post. tuples	yes	0.13	1.9	0.13	2.0	0.22	3.7	0.19	3.9
6	HT-PLDA,Part. Post. tuples	no	**0.11**	**1.6**	**0.12**	**1.8**	**0.21**	**3.2**	**0.16**	**2.6**

Table 5.10: Comparison of diarization performance of G-PLDA and HT-PLDA on DIHARD 2019 development and evaluation sets. HT-PLDA is trained discriminatively to maximize the posterior probability of the correct partition of tuples of 8 segments. The optimal AHC stopping threshold is selected to minimize DER on DIHARD 2019 development set. The performance is reported in terms of DER (%).

#	System	σ=0		σ optimal	
		dev	eval	dev	eval
1	Baseline Kaldi	27.12	27.74	20.45	21.35
2	G-PLDA Vox	50.78	49.15	21.81	22.38
3	G-PLDA Vox+AMI	61.12	59.82	22.28	22.24
4	HT-PLDA Vox+AMI	20.41	21.80	20.20	21.66
5	HT-PLDA Vox → AMI, Part.Post. tuples	20.85	21.45	20.27	21.13
6	HT-PLDA Vox+AMI, Part.Post. tuples	**19.95**	**20.77**	**19.71**	**20.58**

Regarding the results, we confirm the conclusions of Section 5.4.7: all HT-PLDA models provide better calibrated LLR scores than any of the G-PLDA baselines resulting in a smaller gap between optimal clustering (oracle stopping threshold) and arbitrary threshold. In this experiment, we observe an advantage of training the AMI HT-PLDA model to interpolate well with the baseline over simple initialization on the baseline. This was not the case for HT-PLDAs trained discriminatively to maximize the approximate posteriors of the whole training utterances (compare lines 5 and 6 of Table 5.6). Also, we observe a larger improvement from discriminative training compared to the HT-PLDA baseline (lines 4 and 6 of Table 5.10).

5.6 Summary of discriminative training strategies

Here, for more convenient comparison we present the results of all discriminative training approaches from Sections 5.2 to 5.5 side-by-side. Table 5.11 shows the results for the baseline G-PLDA models (lines 1 and 2), G-PLDA models turned into heavy-tailed ones by switching degrees of freedom parameter ν from infinity to 2 (lines 3 and 4), generatively trained HT-PLDA models with VB algorithm of Chapter 4 (lines 5 and 6), and discriminatively trained HT-PLDAs with various training strategies (lines 7-14). All models were trained on x-vector embeddings with and without length normalization. In all cases, embeddings were centered, and their dimensionality was reduced from 512 to 300 by LDA prior to model training. Discriminative training of HT-PLDA models was always initialized from the corresponding G-PLDA (meaning that the performance of the model before training was as shown on lines 3 and 4 for normalized and unnormalized embeddings, respectively), and, during the training, we regularized the parameters towards their initial values.

Looking at the results of HT-PLDAs with various discriminative training objectives, we notice that the performance of the models trained on embeddings without LN is generally better than the model trained on length-normalized embeddings. We do not observe the same trend in generatively trained models: for G-PLDA, length normalization is essential for good performance, and for HT-PLDA, there is no significant difference between embeddings with or without LN (everywhere except for EER of SITW core-core condition).

Also, we notice that discriminative training is usually helpful in terms of speaker verification performance: in most cases, it results in better performance than the performance of the model from which discriminative training was initialized. When comparing discriminatively trained HT-PLDA models to G-PLDA with LN (the best performing baseline), we notice that most of them have comparable performance, while some training strategies result in considerable improvements on VoxCeleb1-H and SITW core-core conditions. However, it is worth noting that the discriminative training was done on data very similar in nature to the evaluation data. As was mentioned before, this is the scenario when discriminative approaches perform the best. On the other hand, we would expect the generative approach to outperform the discriminative ones for a dataset considerably different from the evaluation data.

Comparing different discriminative approaches, our observation is that training the model to partition relatively small tuples of embeddings results in the best performance. Moreover, for VoxCeleb1-O and SITW conditions, the performance of the model trained in this way is the best across all generative and discriminative training strategies. One possible explanation is that such an objective provides a good compromise between the complexity and feasibility of the training. On the one hand, training examples are more challenging than, for example, in BXE, so training is more effective, and the model has a chance of learning more complex dependencies in the training data. On the other hand, the training is much faster than for PSL and partitioning the whole training dataset; thus, we can afford better tuning of the training hyper-parameters.

Finally, we want to note that different discriminative approaches require different computational and time resources. To make a single parameters update when training with PSL or maximizing the posterior of the correct partition requires considerable time and computations to evaluate the objective and the gradients. This makes these objectives much harder to use in practice as the time needed to tune the training parameters can be prohibitively high. At the same time, the results are not necessarily better than those of the generatively trained model. On the other hand, the objectives allowing for stochastic optimization (BXE and partitioning tuples of embeddings) can be incorporated into the SV or SD system development much easier. Even though they are considerably slower than any of the generative approaches, the performance gains can be worth the investment of the resources.

Thus, we believe that generative training should be used in most use cases. When in-domain training data are available, it could be worth considering stochastic training with BXE or partitioning tuples of recordings. And, only if there is a strong reason to believe that computationally demanding objectives like, e.g., PSL are worth investing into, they should be tried.

Table 5.11: Speaker recognition performance for x-vector based G-PLDA and HT-PLDA models trained generatively or discriminatively. The performance is reported on VoxCeleb1-O, VoxCeleb1-E, VoxCeleb1-H, and SITW core-core conditions in terms of $C_{\text{min}}^{\text{Prm}}$=minDCF$_{0.05}$, and EER(%).

#	System	LN	VoxCeleb1-O		VoxCeleb1-E		VoxCeleb1-H		SITW core-core	
			$C_{\text{min}}^{\text{Prm}}$	EER	$C_{\text{min}}^{\text{Prm}}$	EER	$C_{\text{min}}^{\text{Prm}}$	EER	$C_{\text{min}}^{\text{Prm}}$	EER
1	G-PLDA, EM	yes	0.12	**1.6**	0.13	**1.8**	0.23	3.7	0.21	3.0
2	G-PLDA, EM	no	0.18	3.0	0.17	3.0	0.26	5.1	0.30	8.1
3	G-PLDA, EM→ HT-PLDA	yes	0.12	1.9	0.12	2.0	**0.20**	3.4	0.19	4.4
4	G-PLDA, EM→ HT-PLDA	no	0.12	1.9	0.12	2.0	**0.20**	3.4	0.19	3.5
5	HT-PLDA, VB	yes	0.12	2.0	0.13	2.1	0.21	3.6	0.19	4.6
6	HT-PLDA, VB	no	0.12	2.0	0.12	2.1	0.21	3.6	0.19	3.6
7	HT-PLDA, BXE	yes	0.12	1.9	0.12	2.0	**0.20**	3.4	0.19	4.3
8	HT-PLDA, BXE	no	0.12	1.8	**0.11**	1.9	**0.20**	3.3	0.18	3.1
9	HT-PLDA, PSL	yes	0.12	2.0	0.12	2.1	0.21	3.5	0.19	4.0
10	HT-PLDA, PSL	no	0.12	1.8	0.12	1.9	**0.20**	3.3	0.18	3.2
11	HT-PLDA, Part. Post. full	yes	0.12	1.9	0.12	2.0	**0.20**	3.4	0.19	4.3
12	HT-PLDA, Part. Post. full	no	0.12	1.7	0.12	**1.8**	0.21	3.3	0.18	2.9
13	HT-PLDA, Part. Post. tuples	yes	0.13	1.9	0.13	2.0	0.22	3.7	0.19	3.9
14	HT-PLDA, Part. Post. tuples	no	**0.11**	**1.6**	0.12	**1.8**	0.21	**3.2**	**0.16**	**2.6**

Chapter 6

Probabilistic embeddings

In this chapter, we discover a different approach to incorporating uncertainty information into speaker verification or diarization pipeline. The approach discussed here can be equally used in both SV and SD tasks. Here, we target a diarization problem; hence, the choices of the model definition and a training objective are tailored toward this task. This chapter is based on our publication [Silnova et al., 2020], where we have initially described the idea; here, we present it in greater detail.

It is worth noting that similar approaches were proposed for face recognition problem [Shi and Jain, 2019,Chen et al., 2021], image and cross-modal retrieval [Oh et al., 2018,Karpukhin et al., 2022,Chun et al., 2021], and speaker verification [Kuzmin et al., 2022]. In most of the mentioned papers, similar to our approach, the embeddings are treated as hidden variables. However, these papers propose to use posterior probability for embeddings given observed data, while in our approach, we use likelihoods for hidden embeddings instead. We will discuss our motivation to use likelihoods later in this chapter. Besides, in i-vector-based speaker verification, the embedding posterior uncertainty is explicitly given by the generative i-vector model. [Cumani et al., 2013,Cumani et al., 2014,Kenny et al., 2013,Stafylakis et al., 2013] provide the details on integrating this uncertainty information into PLDA back-end model. In our case, however, we deal with discriminative embedding extractors and thus cannot build upon these works.

The rest of the chapter proceeds as follows. We start by introducing the model where the embeddings are represented by hidden variables, unlike in the previous chapters where the embeddings were the observed data. Then, we describe a procedure for computing the likelihood of the observed data (e.g., matrices of acoustic features). We proceed with the scheme for estimating the parameters of embedding distributions. Finally, we describe some diarization experiments with the proposed model.

6.1 Model description

In the standard approach to SV, each speech segment is assumed to be represented as a single vector – embedding. Then, the back-end model works with the embeddings as with the observed data. This approach has its benefits: it represents both training and test data in a compact way, allows for relatively simple back-end models to be used, and, as we have shown in the case of HT-PLDA, there is a possibility for uncertainty propagation in this setting. In the previous chapters, by “uncertainty propagation” we understood the uncertainty about the speaker identity variable. The assumption was that the embeddings

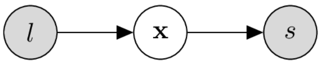

Figure 6.1: Graphical model of the generative process for a single speech segment assumed by the probabilistic embedding model.

themselves are well estimated but their distribution exhibits uncertainty about speaker identities: if distributions of embeddings for two speakers overlap, one cannot be certain which speaker a particular embedding belongs to. The approach we adopt now is different: we design the embeddings to explicitly represent the uncertainty in the form of a probability distribution. We represent the embedding as a hidden variable, i.e., we cannot directly observe it. For a high-quality long speech recording, the desirable property of this hidden variable would be to have a very sharp distribution and, for a noisy recording, to have a flat distribution. In other words, we assume the existence of some ideal true embedding; one should be certain about it for high-quality audio and be uncertain about the embedding value for low-quality recordings. Figure 6.1 shows a high-level graphical model formalizing the previous general notes. It displays the generative process assumed for a single speech segment.

Here, we define two observed variables: s and l are a speech segment and its corresponding speaker label, respectively. $\mathbf{x}$ is a hidden embedding variable. To work with the model, we have to specify the form of the embedding distribution and the model of dependencies between the variables in the graph. We propose the following choices:

- The relation between a speaker label and a hidden embedding is modeled through the introduction of a hidden speaker variable $\mathbf{z}$. Here, as before, we assume a single speaker variable per speaker, so there is a direct correspondence between $\mathbf{z}$ and l. We also assume that the embedding and the speaker variable are related by a Gaussian PLDA model (3.3) with parameters $\mathbf{F}$ and $\mathbf{W}$ - a factor loading matrix and a noise precision, respectively.

- The form of the relation between $\mathbf{x}$ and s emerges from the considerations on the feasibility of the inference in the complete model and the PLDA model assumed for $\mathbf{x}$. The particulars are shown below; here, we only mention that the posterior distribution of the hidden embedding variable for a single speech segment s is modeled by a normal distribution with parameters depending on the observed speech: $P(\mathbf{x} \mid s) = \mathcal{N}(\mathbf{x} \mid \bar{\mathbf{x}}, \bar{\mathbf{B}}^{-1})$. The expressions for the mean $\bar{\mathbf{x}}$ and the precision $\bar{\mathbf{B}}$ are given by (6.5). However, as described later, in our recipe, it is not necessary to know them explicitly.

Before moving forward, we would like to comment on our motivation for using the graphical model of Figure 6.1. We were inspired by a generative PLDA and wanted to include it in the model. For this reason, our model had to contain the dependency of the embeddings on the labels (the generative process of PLDA assumes this). Keeping this part of the model fixed, we had two options for connecting hidden embedding $\mathbf{x}$ and observed speech s. One is to assume that the embedding is generated given the speech segment, i.e., the arrow connecting $\mathbf{x}$ and s points in the opposite direction to the one in Figure 6.1. Such a choice would better suit a discriminatively trained embedding extractor. However, in this case, speaker labels and observed speech become independent if the embedding is not observed, and it is impossible to recognize the labels from the speech. Thus, we opted for the second possibility depicted in Figure 6.1.

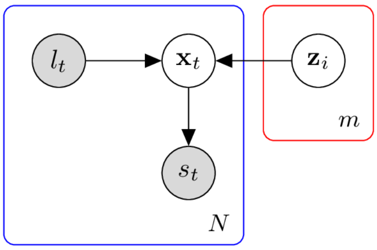

Figure 6.2: Graphical model of the generative process for N speech segments.

Considering the whole dataset of N speech segments and taking into account the assumption that the embeddings are generated by a PLDA model, we can get a more detailed graphical model in Figure 6.2.

The meaning of variables l, s, and $\mathbf{x}$ is the same as before; these are a speaker label, an observed speech segment, and a hidden embedding, only this time, there are N of them. m vectors $\mathbf{z}_i$ are hidden speaker variables. There is a single speaker variable connected to a hidden embedding per utterance. Speaker label l makes the selection of which one it is.

Let us describe the assumed generative process depicted in Figure 6.2. First, N speaker labels $\mathcal{L} = l_1 \ldots l_N$ are sampled from the assumed prior (Chinese Restaurant Process, in our case). Assuming that labels $\mathcal{L}$ partition the data into m speaker clusters, sample m hidden speaker variables $\mathbf{z}_i$ from standard normal prior: $\mathbf{z}_i \sim \pi(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$. For each utterance of speaker i, sample hidden embedding $\mathbf{x}_{ij}$ from the Gaussian PLDA model with parameters $\mathbf{F}, \mathbf{W}$ and using hidden vector $\mathbf{z}_i$ (see (6.2)). Finally, having a hidden embedding, sample the observation $s_{ij} \sim P(s_{ij} \mid \mathbf{x}_{ij})$. In practice, it is not possible to define $P(s_{ij} \mid \mathbf{x}_{ij})$ exactly due to its high complexity. Thus, even though we assume the described generative process, it is impossible to use this model as a generative model. We cannot use it to generate synthetic data unless we define $P(s_{ij} \mid \mathbf{x}_{ij})$. However, as will be shown later, it is not necessary to know $P(s_{ij} \mid \mathbf{x}_{ij})$ to use this model to compute likelihood ratio scores for different partitions of the data or to be able to train the model discriminatively. It is enough to assume that $P(s_{ij} \mid \mathbf{x}_{ij})$ belongs to a rather large family of distributions (see (6.3)).

As in the previous chapters, we are interested in comparing likelihoods for partitions of the data; in order to do so, we have to be able to compute the likelihood ratio (2.4). To be able to compute it, one has to know how to compute the likelihood for the partition labels $P(\mathcal{S} \mid \mathcal{L})$. With the current model, we still assume the same properties of the data described in Section 2.1: independence of recordings of different speakers and interchangeability of recordings of the same speaker. Hence, we can use the same factorization of the likelihood (2.3) as used for the standard PLDA model. The only difference now is that instead of a set of observed embeddings $\mathcal{R}$, we consider a set of observed speech segments $\mathcal{S}$. As before, to compute the likelihood, one has to be able to perform three operations:

- Compute a likelihood of a single segment for a speaker variable $P(s \mid \mathbf{z})$.
- Pool segments assumed to belong to the same speaker i, i.e., compute

$$P(s_{i1}, \ldots, s_{iN_i} \mid \mathbf{z}) = \prod_{j=1}^{N_i} P(s_{ij} \mid \mathbf{z}).$$

- Compute the expected value with respect to the speaker variable prior

$$\left\langle \prod_{j=1}^{N_i} P(s_{ij} \mid \mathbf{z}) \right\rangle_{\mathbf{z} \sim \pi(\mathbf{z})} .$$

We will consider all three operations in turn. We start with computing the likelihood of a single speech segment. As a speech segment and its corresponding speaker hidden variable are not connected directly in our model but through a hidden embedding (see the graphical model in Figure 6.2), when computing $P(s \mid \mathbf{z})$, we have to marginalize over all possible values of the embedding $\mathbf{x}$:

$$P(s \mid \mathbf{z}) = \int P(s \mid \mathbf{x}) P(\mathbf{x} \mid \mathbf{z}) \, \mathrm{d}\mathbf{x}. \tag{6.1}$$

As we have mentioned before, we model the relation between a hidden embedding and a speaker variable by a Gaussian PLDA model (see (3.3)):

$$P(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}\big(\mathbf{x} \mid \mathbf{F}\mathbf{z}, \mathbf{W}^{-1}\big) = \frac{|\mathbf{W}|^{\frac{1}{2}}}{(2\pi)^{\frac{D}{2}}} \exp\left[-\frac{1}{2}(\mathbf{x} - \mathbf{F}\mathbf{z})'\mathbf{W}(\mathbf{x} - \mathbf{F}\mathbf{z})\right], \tag{6.2}$$

where $\mathbf{F}$ and $\mathbf{W}$ are the parameters of the G-PLDA model - the factor loading matrix and the noise precision, respectively. Then, the second factor in the integral of (6.1) is Gaussian. If we assume that the first factor (likelihood for hidden embedding $\mathbf{x}$) $P(s \mid \mathbf{x})$ is also Gaussian function of $\mathbf{x}$, then the whole integral (6.1) can be computed in a closed form. We use exactly this assumption and obtain:

$$P(s \mid \mathbf{x}) = h(s) \exp\left[-\frac{1}{2}\mathbf{x}'\mathbf{B}(s)\mathbf{x} + \mathbf{x}'\mathbf{B}(s)\hat{\mathbf{x}}(s)\right]. \tag{6.3}$$

Here, $h(s)$ is some strictly-positive real-valued function that is assumed to depend only on the observed speech and not on the hidden embedding. This function carries all the complexities of the distribution over speech. If we are after computing $P(s \mid \mathbf{x})$ exactly (or generating data from the model), $h(s)$ has to be known. But, as we are interested in expressions of the form (2.4), there is no need to evaluate $h(s)$ as it cancels in the ratio[1]. $\hat{\mathbf{x}}(s)$ and $\mathbf{B}(s)$ are the parameters of the likelihood function, which is a Gaussian function of the embedding. Both $\hat{\mathbf{x}}(s)$ and $\mathbf{B}(s)$ depend on the speech segment s. Below, for brevity, we will omit writing dependency on s explicitly. Instead, we will use $\mathbf{B}$ and $\hat{\mathbf{x}}$, keeping in mind that both depend on the observed speech s.

From now on, we refer to computing parameters $\mathbf{B}$ and $\hat{\mathbf{x}}$ of the likelihood function as to extracting probabilistic embedding. To show that these parameters indeed carry the uncertainty information about the embedding, let us consider the posterior for a single speech segment s. Combining (6.3), (6.2) and taking into account the standard normal

[1] It has to be mentioned that there is no guarantee that $h(s)$ will always exist for a discriminatively trained model. We discuss this issue in more detail in Appendix A of [Silnova et al., 2020]. There, we argue that there are two possible ways of dealing with this: we can either develop a generative training scheme that would make sure that $h(s)$ always exists or we can follow the common practice of designing discriminative models where this issue is ignored. Here, we follow the second approach keeping the generative training recipe for future work.

prior for hidden speaker variable $\mathbf{z}$, the posterior can be shown to be Gaussian[2]:

$$\begin{aligned}
P(\mathbf{x} \mid s) &\propto P(s \mid \mathbf{x})P(\mathbf{x}) \\
&= P(s \mid \mathbf{x}) \int P(\mathbf{x} \mid \mathbf{z})\pi(\mathbf{z})\,\mathrm{d}\mathbf{z} \\
&\propto h(s)\exp\left[-\frac{1}{2}\mathbf{x}'\mathbf{B}\mathbf{x} + \mathbf{x}'\mathbf{B}\hat{\mathbf{x}}\right]\exp\left[-\frac{1}{2}\mathbf{x}'(\mathbf{W} - \mathbf{W}\mathbf{F}(\mathbf{F}'\mathbf{W}\mathbf{F}+\mathbf{I})^{-1}\mathbf{F}'\mathbf{W})\mathbf{x}\right] \\
&\propto h(s)\exp\left[-\frac{1}{2}\mathbf{x}'\bar{\mathbf{B}}\mathbf{x} + \mathbf{x}'\bar{\mathbf{B}}\bar{\mathbf{x}}\right],
\end{aligned} \tag{6.4}$$

where we used the following notation:

$$\begin{aligned}
\bar{\mathbf{B}} &= \mathbf{B} + \mathbf{W} - \mathbf{W}\mathbf{F}(\mathbf{F}'\mathbf{W}\mathbf{F}+\mathbf{I})^{-1}\mathbf{F}'\mathbf{W}, \\
\bar{\mathbf{x}} &= \bar{\mathbf{B}}^{-1}\mathbf{B}\hat{\mathbf{x}}.
\end{aligned} \tag{6.5}$$

Thus, we have shown that the posterior of $\mathbf{x}$ for a single s is a Gaussian distribution with mean $\bar{\mathbf{x}}$ and precision $\bar{\mathbf{B}}$ which depend on PLDA parameters, $\hat{\mathbf{x}}$, and $\mathbf{B}$. As PLDA parameters do not depend on the observed data s, we see that the only factors affecting the posterior distribution $P(\mathbf{x} \mid s)$ are the parameters of the likelihood function $\hat{\mathbf{x}}, \mathbf{B}$. In other words, by varying values of the parameters of the likelihood function, the probabilistic embedding extractor can control the uncertainty assigned to each embedding. For example, as seen from the first equation of (6.5), assigning higher "precision" for the likelihood function results in higher precision for the embedding posterior distribution.

Let us stress that we provided the equations (6.4), (6.5) for the posterior only to demonstrate that our model allows for uncertainty propagation. Knowing the posterior (6.4) is not needed to use the proposed model: we do not use it for training or scoring.

Finally, analyzing (6.4), we can motivate for why we decided to use likelihoods $P(s \mid \mathbf{x})$ instead of posteriors $P(\mathbf{x} \mid s)$ unlike other probabilistic embeddings papers (see, for example, [Oh et al., 2018]). As seen from the first line of (6.4), the posterior incorporates the prior over hidden embeddings $P(\mathbf{x})$. Thus, the embedding extractor that outputs the posterior parameters implicitly incorporates some fixed prior $P(\mathbf{x})$. This fact restricts the use of the model: one cannot change $P(\mathbf{x})$ depending on the task at hand. For example, in our model, when the dataset contains a single speech segment, $P(\mathbf{x})$ is given by PLDA, and it is a Gaussian as seen in (6.4). On the other hand, if there are several data points and the labels are not given, $P(\mathbf{x})$ incorporates CRP for speaker labels and the PLDA back-end. Ideally, the extractor computing $P(\mathbf{x} \mid s)$ for our model has to differentiate between these two cases. Besides, even if the prior $P(\mathbf{x})$ in a model is fixed (this is the approach taken in [Oh et al., 2018]), in many cases, it will be a complex distribution. Consequently, the posterior has to incorporate this complexity. This would complicate both training and inference. For these reasons, we opted for extracting the parameters of the likelihood $P(s \mid \mathbf{x})$ allowing us to plug in any appropriate prior when needed.

We propose to model the dependence between speech recordings and likelihood parameters $\hat{\mathbf{x}}$ and $\mathbf{B}$ by a neural network. We will discuss its architecture in more detail later in Section 6.2.1, for now we assume that we have a way to compute these parameters given a speech segment, i.e., there exists a function $g(s)$ that implements mapping $s \rightarrow (\hat{\mathbf{x}}(s), \mathbf{B}(s))$.

[2] Notice that the posterior is a single Gaussian only for the case when the labels are observed (or if the dataset contains a single datapoint as is the case in (6.4)). If the labels are not observed, we would need to integrate out not only the speaker variable $\mathbf{z}$ but also sum out hidden labels. The posterior becomes a mixture of Gaussians in this case.

Substituting $P(\mathbf{x} \mid \mathbf{z})$ and $P(s \mid \mathbf{x})$ given by (6.2) and (6.3) in (6.1), we get the following expression for the likelihood for the speaker variable:

$$P(s \mid \mathbf{z}) = f(s) \exp \left[-\frac{1}{2} \mathbf{z}'\mathbf{F}'(\mathbf{W} - \mathbf{W}'(\mathbf{B}+\mathbf{W})^{-1}\mathbf{W})\mathbf{F}\mathbf{z} + \mathbf{z}'\mathbf{F}'\mathbf{W}(\mathbf{B}+\mathbf{W})^{-1}\mathbf{B}\hat{\mathbf{x}} \right], \quad (6.6)$$

where we have collected all of the likelihood's components not depending on the speaker variable into a single factor $f(s)$. Again, as we are not interested in computing likelihood itself, but LR scores (2.4), $f(s)$ can be ignored as it will cancel in the ratio.

Therefore, if we obtain $\mathbf{B}$ and $\hat{\mathbf{x}}$, (6.6) provides a way to compute the likelihood for any speech segment and speaker variable. However, in practice, it would be very computationally expensive because of the matrix inverse operation. Moreover, the matrix to invert, $\mathbf{B}+\mathbf{W}$, depends on a speech segment meaning that this expensive operation has to be done for each segment.

To cope with this difficulty, we restrict our attention to the special case of the G-PLDA model – the two-covariance model. This model assumes that both speaker and embedding variables live in the same D-dimensional space $\mathbf{z}, \mathbf{x} \in \mathbb{R}^D$. It means that the factor loading matrix $\mathbf{F}$ is a square full-rank matrix. Notice that this assumption is opposite to the one we made for the HT-PLDA model.

For any PLDA model, there exists a linear transformation such that in the transformed space, both within-class and across-class covariances are diagonal. Even more, one of them can be the identity matrix. We assume that this linear transformation was applied; within-class covariance (and precision) matrix is diagonal $\mathbf{W} = \text{diag}(w_1 \dots w_D)$, $w_d > 0$ and across-class matrix is identity $\mathbf{F}'\mathbf{F} = \mathbf{I}$. The two-covariance model then implies that the factor loading matrix is itself identity $\mathbf{F} = \mathbf{I}$.

Finally, as the specific form of the embedding parameters is a part of the model design, we can select them for our own convenience: we arbitrarily set matrix $\mathbf{B}$ to be diagonal $\mathbf{B} = \text{diag}(b_1 \dots b_D)$, $b_d >= 0$ understanding that this assumption might limit expressive power of the model. Then, (6.6) becomes:

$$P(s \mid \mathbf{z}) \propto \prod_{d=1}^{D} \exp \left[\frac{w_d b_d}{w_d + b_d} (\hat{x}_d z_d - \frac{1}{2} z_d^2) \right], \quad (6.7)$$

where x_d and z_d are the individual elements of the vectors $\mathbf{x}$ and $\mathbf{z}$; b_d and w_d are the diagonal elements of the embedding parameter $\mathbf{B}$ and the PLDA within-class precision matrix.

Then, pooling together several segments $\mathcal{S}_i = \{s_1, \dots s_{N_i}\}$ of the same speaker i (for the same value of $\mathbf{z}$) is trivial:

$$\begin{aligned} P(\mathcal{S}_i \mid \mathbf{z}) &= \prod_{j=1}^{N_i} P(s_j \mid \mathbf{z}) \\ &\propto \prod_{j=1}^{N_i} \prod_{d=1}^{D} \exp \left[\frac{w_d b_{jd}}{w_d + b_{jd}} (\hat{x}_{jd} z_d - \frac{1}{2} z_d^2) \right] \\ &= \prod_{d=1}^{D} \exp \left[\sum_{j=1}^{N_i} \frac{w_d b_{jd}}{w_d + b_{jd}} \hat{x}_{jd} z_d - \frac{1}{2} \sum_{j=1}^{N_i} \frac{w_d b_{jd}}{w_d + b_{jd}} z_d^2 \right] \\ &= \prod_{d=1}^{D} \exp \left[\bar{a}_d z_d - \frac{1}{2} \bar{b}_d z_d^2 \right], \end{aligned} \quad (6.8)$$

where

$$\bar{a}_d = \sum_{j=1}^{N_i} \frac{w_d b_{jd}}{w_d + b_{jd}} \hat{x}_{jd}, \qquad \bar{b}_d = \sum_{j=1}^{N_i} \frac{w_d b_{jd}}{w_d + b_{jd}}. \tag{6.9}$$

$b_{jd}, \hat{x}_{jd}$ are the d-th elements of embedding likelihood function parameters of the j-th speech segment s_j. $\bar{a}_d$ and $\bar{b}_d$ are the individual elements of cumulative statistics for speaker i.

Finally, let us consider the last operation needed for computing the likelihood $P(\mathcal{S} \mid \mathcal{L})$ – the expectation of $P(\mathcal{S}_i \mid \mathbf{z})$ with respect to the prior $\pi(\mathbf{z})$, i.e., computing the likelihood for the same-speaker hypothesis of the segments from set $\mathcal{S}_i$: $P(\mathcal{S}_i \mid H_s)$. In the G-PLDA model that we adopted here, the prior for $\mathbf{z}$ is a standard normal distribution. Then, the expectation is just a convolution of two Gaussians and can be found in a closed form:

$$\begin{aligned} P(\mathcal{S}_i \mid H_s) &= \langle P(\mathcal{S}_i \mid \mathbf{z}) \rangle_{\mathbf{z} \sim \pi(\mathbf{z})} \\ &\propto \int \prod_{d=1}^{D} \exp\left[\bar{a}_d z_d - \frac{1}{2}\bar{b}_d z_d^2\right] \pi(\mathbf{z})\, \mathrm{d}\mathbf{z} \\ &\propto \int \prod_{d=1}^{D} \exp\left[\bar{a}_d z_d - \frac{1}{2}\bar{b}_d z_d^2\right] \exp(-\frac{1}{2}\mathbf{z}'\mathbf{z})\, \mathrm{d}\mathbf{z} \\ &= \int \prod_{d=1}^{D} \exp\left[\bar{a}_d z_d - \frac{1}{2}(\bar{b}_d + 1) z_d^2\right] \mathrm{d}\mathbf{z} \\ &\propto \prod_{d=1}^{D} \frac{1}{(\bar{b}_d + 1)^{\frac{1}{2}}} \exp \frac{\bar{a}_d^2}{2(\bar{b}_d + 1)}. \end{aligned} \tag{6.10}$$

The derivations above are done analogously to (3.7) where more detailed derivations were given.

Given (6.10), log-likelihood is:

$$\log P(\mathcal{S}_i \mid H_s) = \frac{1}{2} \sum_{d=1}^{D} \left[\frac{\bar{a}_d^2}{(\bar{b}_d + 1)} - \log(\bar{b}_d + 1) \right] + \text{const}. \tag{6.11}$$

Looking closer at (6.9), we see the interaction between PLDA within-class precision parameters and embedding parameters. If, for some recording s_j, diagonal element of $\mathbf{B}$ at dimension d is very small $b_{jd} \approx 0$ (the uncertainty about embedding along this dimension is high), then the whole ratio $\frac{w_d b_{jd}}{w_d + b_{jd}}$ becomes close to zero and this dimension is effectively ignored when computing the statistics for a given speaker (and consequently also the likelihood). If, on the other hand $b_{jd} \gg w_d$, then the weight $\frac{w_d b_{jd}}{w_d + b_{jd}}$ becomes practically w_d. If it is true for all recordings, the model degrades to the standard Gaussian PLDA likelihood estimation formulae. If the parameters of the embedding are estimated correctly, we have a way to weigh the dimensions in the embedding space when computing the likelihood so that the reliable dimensions have a higher weight and affect the likelihood more than those that one cannot trust.

To compute the log-likelihood for any partition of the speech recordings into speaker clusters (2.3), one has to sum terms (6.11) computed for individual clusters defined by such partition:

$$\log P(\mathcal{S} \mid \mathcal{L}) = \frac{1}{2} \sum_{i=1}^{m} \sum_{d=1}^{D} \left[\frac{\bar{a}_{id}^2}{(\bar{b}_{id} + 1)} - \log(\bar{b}_{id} + 1) \right] + \text{const}, \tag{6.12}$$

where we assume that $\mathcal{L}$ partitions $\mathcal{S}$ into m speaker clusters, $\bar{a}_{id}$ and $\bar{b}_{id}$ are (d-th dimension of) statistics (6.9) computed for i-th speaker defined by $\mathcal{L}$.

Log-likelihood (6.12) can be computed up to a normalizing term that depends on the observations $\mathcal{S}$ but not on the partition $\mathcal{L}$. As we are interested not in log-likelihood itself but LLR (2.5), we can ignore the normalizers as they will cancel in the ratio.

To summarize, we derived a closed-form solution for evaluating the LLR score between two partitions for a Gaussian PLDA model with embeddings considered as hidden variables. To compute the likelihood for a partition of N recordings one needs: N vectors $\hat{\mathbf{x}}_j$, N vectors $\mathbf{b}_j$ (diagonals of $\mathbf{B}_j$) - parameters of the likelihood function (6.3), and a diagonal PLDA within-class precision matrix $\mathbf{W}$. As was mentioned before, the relation between a speech segment and parameters controlling embedding distribution is modeled by a neural network, so ultimately, to get $\mathbf{b}_j$ and $\hat{\mathbf{x}}_j$, we need to know the network parameters.

Also, if one defines a prior over partitions $P(\mathcal{L})$ and uses (6.12) to compute the likelihood, then, for small sets $\mathcal{S}$, it is possible to compute the posterior for the particular partition $\mathcal{L}^*$:

$$P(\mathcal{L}^* \mid \mathcal{S}) = \frac{P(\mathcal{L}^*)P(\mathcal{S} \mid \mathcal{L}^*)}{\sum_{\mathcal{L}} P(\mathcal{L})P(\mathcal{S} \mid \mathcal{L})}. \tag{6.13}$$

6.2 Experimental design

In this section, we describe the design choices for the embedding extractor and the training objective for the model described above. The suggestions here are only a subset of many existing options. Both the extractor architecture and the training objective can be tailored to a particular task of interest. Here, we describe a setting that was used in our experiments.

6.2.1 Estimating embedding parameters

The parameters $\hat{\mathbf{x}}$ and $\mathbf{b}$ for each speech recording have to be estimated by a neural network. We propose to use a network trained to extract speaker embeddings (not probabilistic) as a base network and augment it to estimate both vectors of parameters. Using this approach, we impose an assumption that the observed data come in a form that the original embedding extractor used as an input, i.e., if the original embedding network used raw audio signal or a matrix of acoustic features as an input, then our model also assumes s to be represented in the same way. For simplicity, we assume that s is a matrix of acoustic features.

As an architecture we will build upon, we use the x-vector extraction network as described in Section 2.4.2. Figure 2.1 (and Table 4.1) displays the x-vector extractor architecture used in our experiments. This network was pre-trained and then modified to extract probabilistic embedding parameters. First, we strip from the network all layers after the embedding layer. Then, the network is augmented with additional layers and blocks. The first of them is a single linear layer transforming the original embedding. The output of the linear layer serves as the parameter $\hat{\mathbf{x}}$ of the probabilistic embedding. This linear layer is assumed to perform the diagonalizing transformation that is assumed by our model (it makes sure that the across-class covariance of PLDA is the identity matrix and the within-class covariance is diagonal). This layer is initialized with a transformation matrix set to diagonalize G-PLDA trained on the original embeddings. Then, an additional sub-network of two dense layers is added to the original architecture: as input, it takes the output of the statistics pooling layer and additionally, the length of the speech segment in frames. This network is supposed to output parameter $\mathbf{b}$ of the embedding likelihood function. The activation function for both of the layers is set to softplus. That way, the output vector

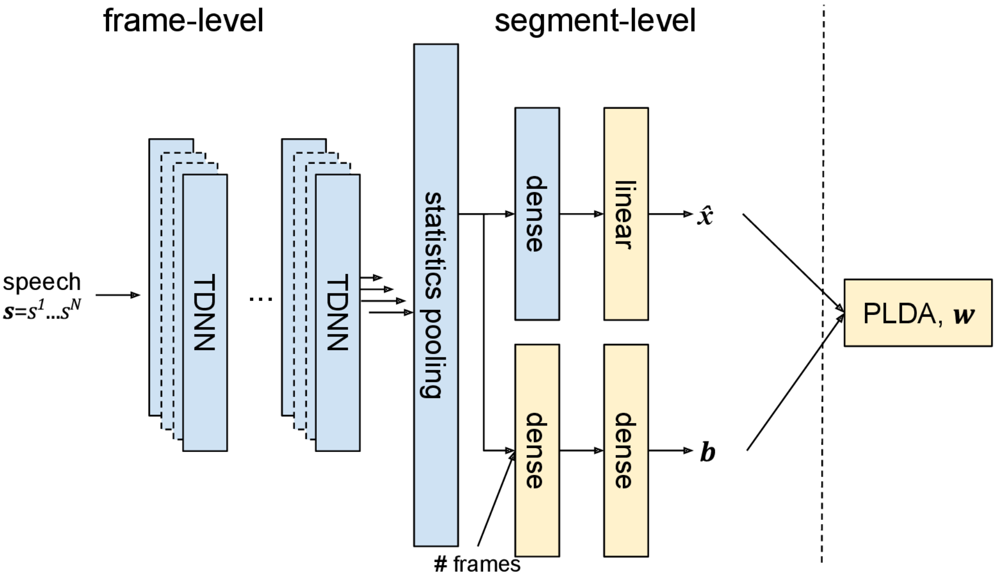

Figure 6.3: The architecture for speaker probabilistic embedding extraction based on the pre-trained non-probabilistic embedding network (Figure 2.1). The gray blocks correspond to the parts of the original network and are fixed, the yellow blocks are the additional layers to train.

is guaranteed to be non-negative. The initialization for the parameters of these layers is done randomly in such a way that the condition $b_{jd} \gg w_d$ is satisfied (i.e., we approximate infinite precision for the embedding). It is done by sampling weights from a standard normal distribution and setting a large constant bias for each dimension. The resulting network is depicted in Figure 6.3. Grey blocks on the scheme correspond to the parts of the original x-vector extractor, while yellow ones are the new layers described above.

Figure 6.3 also includes a PLDA block even though it is not a part of the embedding extractor. This is because we want to train both probabilistic embedding extractor and PLDA jointly. At the beginning of training, PLDA within-class precision is set to a diagonalized within-class precision of the PLDA trained with baseline embeddings. Initialized in this way, PLDA and the embedding extractor very closely resemble the baseline system (it is not exact due to non-infinite embedding precisions).

6.2.2 Training criterion

Following the design of the embedding extractor described above, to use the model, we need to have the following parameters $\theta = \{\theta_{\hat{\mathbf{x}}}, \theta_{\mathbf{b}}, \mathbf{w}\}$, where $\theta_{\hat{\mathbf{x}}}$ is the matrix to transform the embedding into $\hat{\mathbf{x}}$, $\theta_{\mathbf{b}}$- the parameters of the neural network extracting $\mathbf{b}$ and $\mathbf{w}$ is the diagonal of PLDA within-class precision $\mathbf{W}$. To train these parameters, we propose to use the same objective as described in Section 5.5: maximizing the posterior probability of the correct partition of training examples consisting of several utterances (5.36). Then, as in the case of HT-PLDA, we can train the model parameters using stochastic gradient ascent. To speed up the training, we evaluate the objective with the procedure described in Appendix F. In short, one can pre-compute several sparse matrices allowing to efficiently perform summations when computing statistics (6.9) and likelihoods (6.12) for all possible partitions of the training examples.

In the case of probabilistic embeddings, we restrict the set of training examples to those containing 8 points only. This is done considering that we plan to test this model primarily on the diarization task and it seems logical to stimulate the model to learn how to partition sets with a large number of speech segments. Also, 8 is the largest feasible size of the example for which one can compute the partition posterior exactly (see Section 5.5).

6.3 Experiment

We test the performance of the probabilistic embedding model on the diarization task repeating the experimental setup of Section 5.5.2. The first baseline model is the Kaldi diarization recipe. The embeddings and G-PLDA model we use in this approach are the same as for the second baseline. However, compared to the approach adopted here, the Kaldi recipe uses additional pre-processing stages as per-utterance PCA projection to retain 30% of the variability in the embeddings, additional length normalization, and projection of the PLDA parameters into the same low-dimensional space as well as another version of clustering algorithm – UPGMA AHC [Sell et al., 2018]. The results of the recipe on the DIHARD 2019 development and evaluation sets are shown in line 1 of Table 6.1.

The rest of the models described here follow the approach described in Section 2.3.1 and differ by a model to compute LLR scores for the clustering algorithm. The second baseline uses the G-PLDA model trained on 512-dimensional length-normalized neural network embeddings to provide LLR scores for AHC. The embedding extractor and PLDA were trained on the VoxCeleb2 and VoxCat sets, respectively (see Section 2.5). Prior to scoring, the embeddings are multiplied with the LDA transformation matrix that not only diagonalizes the PLDA covariance matrices but also reduces the dimensionality of the data to 500, keeping the dimensions of the highest within-class precision (it is the same as keeping the highest across-class variability dimensions). The results of this model are in line 2 of Table 6.1. Notice that the baseline G-PLDA model from Table 5.10 had different performance because of different data pre-processing and G-PLDA settings: before, we did not use LDA, but we had to use low-dimensional speaker subspace, while now, we need to be consistent with probabilistic embedding approach and use LDA and two-covariance model. The embedding extractor and PLDA from the baseline (line 2 of Table 6.1) are used to initialize the probabilistic embedding model as described in Section 6.2.1. The probabilistic embedding model now replaces the G-PLDA of the baseline when generating LLR scores for the AHC algorithm. Before training, the diarization performance of the baseline and the probabilistic embeddings is the same.

Then, we train the parameters of the probabilistic embedding extractor and PLDA on the AMI dataset. The training examples for the objective (5.36) are generated by the same procedure as in experiments of Section 5.5.2:

- Sample a training utterance.
- For the given utterance, sample 8 speech segments (for diarization purposes, each utterance is split into short overlapping segments) randomly. Fix some order of the segments and use their corresponding speaker labels to define a correct partition for the example.

As a prior distribution $P(\mathcal{L})$, we use the Chinese Restaurant Process (see Appendix B) [Pitman, 1995, Fox et al., 2011, Zhang et al., 2019]. We assigned the parameters of the CRP such that the expected number of clusters (see (B.5)) for N segments was equal to the true number of speakers in the training set. N here is the total number of short segments in all

Table 6.1: Comparison of diarization performance of generatively and discriminatively trained G-PLDA, with and without embedding uncertainty on DIHARD 2019 development and evaluation sets. The performance is reported in terms of DER (%).

#	System	σ=0		σ optimal	
		dev	eval	dev	eval
1	Baseline Kaldi	27.12	27.74	**20.45**	**21.35**
2	Baseline G-PLDA	44.28	41.09	23.82	23.98
3	G-PLDA, Part.Post. tuples	26.95	26.34	22.38	22.68
4	PLDA+uncertainty, Part.Post. tuples	**22.95**	**23.38**	21.61	21.84

utterances in the training set. We always set one of the parameters to zero (concentration or discount), and the second one is found by a numerical optimization method. For the fixed number of points N, parameters of CRP affect the distribution of number of clusters (see Figure B.1). When deciding which out of two CRP parameters to set to zero, we select it so that the variance of the distribution of the number of clusters is maximized. CRP gives an exchangeable distribution that is invariant to the order of the speaker labels. This prior is appropriate for our procedure of randomly selecting the segments to form the training examples. However, if we generated the examples out of consecutive segments, we would want the prior to incorporate the information that the neighboring segments are likely to share the same speaker identity. Then, some other prior should be considered.

To isolate the impact of the additional data and the discriminative training from the introduction of the uncertainty used, we train two models. For the first one, we fix the parameters of the embedding precision network and train only the LDA transformation of the embedding and PLDA precision; this corresponds to retraining the baseline PLDA on additional data with objective (5.36). The uncertainty information about the embedding is not used in this case. The result is shown in line 3 of Table 6.1. Then, we train a model where all of the parameters are learned, including the embedding parameters. The sub-network extracting the 500-dimensional vectors $\mathbf{b}$ has two hidden layers, each having 1000 nodes. The results of the model using embedding uncertainty information are shown in line 4 of Table 6.1. Comparing lines 2 and 3, we can see the effect of the discriminative retraining of the PLDA model on AMI data. Finally, comparing lines 3 and 4, we can see the impact of the uncertainty information on the diarization performance.

6.3.1 Discussion

As can be seen from the results, both of the baseline models are miscalibrated. The performance when the AHC stopping threshold is set to 0 is worse than the optimal one. Threshold 0 is the optimal maximum-likelihood threshold; the fact that it leads to inferior performance tells that the G-PLDA models of the baselines do not output proper LLR scores for AHC. However, the Kaldi recipe baseline does not suffer from the miscalibration as much as the other one. Also, the optimal performance tuned on the development set for the Kaldi recipe is better than for the second baseline. By retraining the baseline from line 2 with the discriminative objective, we greatly mitigate the calibration problem and also improve the optimal performance. This model outperforms the first baseline for the 0 threshold case, meaning that in the case when there is no development set to tune the threshold, it would result in a better performance. Finally, allowing the model to use the uncertainty information together with retraining PLDA brings even further improvements in the performance and improves the calibration. Although the optimal performance of this model is still worse than the Kaldi baseline, it is already reasonably close to it. And, in the case when no development set is available to tune the threshold (using zero threshold), this method results in a significant improvement over both baselines.

In summary, one can conclude that even though introducing uncertainty information in the form of probabilistic embeddings to the diarization approach does not necessarily result in a better performance than can be achieved without it, it allows for the model to be more robust and better calibrated.

Chapter 7

Conclusion

7.1 Summary

This thesis presented two models allowing for utilizing uncertainty in speaker verification and diarization.

First, we introduced a modification to the Heavy-tailed PLDA model. The original HT-PLDA was shown effective for speaker verification task but had a significant drawback of high time and computational complexity. Our variant of HT-PLDA mitigates the limitations of the former approach and allows for fast scoring of verification trials. Also, we have shown that this model allows for the uncertainty information to propagate, unlike in more widely used Gaussian PLDA.

We presented one generative and various discriminative approaches to train HT-PLDA. The generative method is faster and more reliable when only a limited amount of in-domain training data is available. Discriminative methods, on the other hand, are more demanding in terms of computations and time. However, when enough in-domain data is available for training, the discriminative methods can outperform the generative one. Among various training objectives that we have considered, we found that training the HT-PLDA model to partition tuples of several recordings into speaker clusters provided the best performance on average. Besides, training with this objective requires considerably less computational and time resources than when training with some other objectives presented in this thesis, making it more applicable in practice.

We have presented speaker verification results for the HT-PLDA models trained on i-vector and x-vector embeddings with and without length normalization. We have shown that the performance of the HT-PLDA model (either generatively or discriminatively trained) does not depend on the data pre-processing as much as in the case of G-PLDA. We have observed that length-normalization is crucial for the good performance of G-PLDA, while its presence has almost no effect on the results of HT-PLDA. Moreover, we have shown that other pre-processing steps, like dimensionality reduction, can significantly impact G-PLDA performance. We do not observe the same trend for HT-PLDA.

We have used HT-PLDA for speaker diarization: in the experiments, we have shown that the scores provided by HT-PLDA are better calibrated LLRs than scores estimated with G-PLDA. This allows HT-PLDA to provide better diarization performance, especially in the case when no development data are available for tuning the stopping threshold for the clustering algorithm.

Second, this thesis presented an approach to utilize uncertainty information by considering speaker embeddings as normally distributed hidden variables rather than observed points in D-dimensional space. We have presented a scheme for augmenting an existing x-vector neural network to provide not a point embedding but parameters of the embedding distribution. Then, these embedding distributions were modeled with the G-PLDA model. We have shown how the uncertainty estimate affects the computation of speaker statistics and, consequently, the likelihood. We used one of the discriminative approaches used for training HT-PLDA – maximizing the posterior of the correct partition for training examples consisting of several speech recordings – to train both PLDA and embedding extractor jointly. Finally, we have tested the performance of the resulting model on the diarization problem, where the probabilistic embedding model was used to compute similarity scores used by the clustering algorithm. Here, similarly to the HT-PLDA case, we have observed that the model allowing for uncertainty propagation provides better calibrated LLRs resulting in a better diarization performance than the baseline G-PLDA model.

The common conclusion for both HT-PLDA and probabilistic embeddings is that these models do not necessarily result in better performance than the baseline G-PLDA model. On the other hand, uncertainty information allows them to be more robust and reliable (e.g., see data pre-processing experiment of Section 4.2.2 or diarization with AHC clustering in Section 6.3).

7.2 Future work

In the future, we would like to improve the uncertainty estimation in both presented methods. This can be approached from different perspectives:

- Training the embedding extractor jointly with the back-end model. Typically, the embedding extractors are trained to discriminate between the speakers in the training dataset. Such training does not encourage the uncertainty information to get propagated to the embeddings.

  However, in our experiments, we fixed the embeddings in the case of HT-PLDA. For probabilistic embeddings, we trained only the new layers that were added to a pre-trained and fixed embedding network.

  We believe that if the embedding extractor is trained together with the back-end that utilizes the uncertainty, it can learn to preserve this information.

  Training the extractor together with the back-end is feasible only with some discriminative training approach. Thus, we can consider some of the training objectives presented here.

- For the extractor to be able to adequately estimate the uncertainty, input segments of a great variety of durations, signal to noise ratio (SNR) levels and other quality factors should be provided during training. It was not the case in our experiments: e.g., for training the probabilistic embeddings, all training segments were of the same duration. We believe that the models were not utilized to their full potential because of the limited variability in the training data.

- The only quality measures in the current implementation of the probabilistic embedding approach were segment duration and the output of the statistic pooling layer. The model is supposed to estimate the embedding precision based on these inputs. Adding additional quality measures such as, e.g., SNR estimate, the average energy of the signal, or even the input spectrogram could help the model to estimate the embedding uncertainty better. Consequently, we can expect performance benefits from adding such inputs to the model.

Bibliography

[Besag, 1975] Besag, J. (1975). Statistical analysis of non-lattice data. *Journal of the Royal Statistical Society, series D (The Statistician)*, 24:179–195.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

[Borgström and McCree, 2013] Borgström, B. J. and McCree, A. (2013). Discriminatively Trained Bayesian Speaker Comparison of i-vectors. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7659–7662.

[Brooks et al., 2011] Brooks, S., Gelman, A., Jones, G., and Meng, X.-L. (2011). *Handbook of Markov Chain Monte Carlo*. CRC press.

[Brümmer, 2010a] Brümmer, N. (2010a). EM for Simple PLDA. Technical report, Agnitio Research, South Africa.

[Brümmer, 2010b] Brümmer, N. (2010b). *Measuring, refining and calibrating speaker and language information extracted from speech*. PhD thesis, Stellenbosch: University of Stellenbosch.

[Brümmer, 2010c] Brümmer, N. (2010c). A minimum divergence recipe for VBEM. Technical report, Agnitio Research, South Africa.

[Brümmer et al., 2018a] Brümmer, N., Burget, L., Garcia, P., Plchot, O., Rohdin, J., Romero, D., Snyder, D., Stafylakis, T., Swart, A., and Villalba, J. (2017-2018a). Meta-embeddings: a probabilistic generalization of embeddings in machine learning. In progress. Draft available: github.com/bsxfan/meta-embeddings.

[Brümmer et al., 2007] Brümmer, N., Burget, L., Černocky, J., Glembek, O., Grézl, F., Karafíat, M., van Leeuwen, D. A., Matějka, P., Schwarz, P., and Strasheim, A. (2007). Fusion of heterogeneous speaker recognition systems in the STBU submission for the NIST Speaker Recognition Evaluation 2006. *IEEE TASLP*, 15(7).

[Brümmer et al., 2018b] Brümmer, N., Silnova, A., Burget, L., and Stafylakis, T. (2018b). Gaussian meta-embeddings for efficient scoring of a heavy-tailed PLDA model. In *Odyssey: The Speaker and Language Recognition Workshop*, pages 349–356, Les Sables d'Olonne, France.

[Brümmer et al., 2014] Brümmer, N., Swart, A., and van Leeuwen, D. (2014). A comparison of linear and non-linear calibrations for speaker recognition. In *Odyssey: The Speaker and Language Recognition Workshop*, Joensuu, Finland.

[Burget et al., 2011] Burget, L., Plchot, O., Cumani, S., Glembek, O., Matějka, P., and Brümmer, N. (2011). Discriminatively trained Probabilistic Linear Discriminant Analysis for Speaker Verification. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, Prague, Czechia.

[Campbell et al., 2006] Campbell, W. M., Sturim, D. E., and Reynolds, D. A. (2006). Support Vector Machines using GMM Supervectors for Speaker Verifcation. *IEEE Signal Processing Letters*, 13:308–311.

[Carletta et al., 2006] Carletta, J., Ashby, S., Bourban, S., Flynn, M., Guillemot, M., Hain, T., Kadlec, J., Karaiskos, V., Kraaij, W., Kronenthal, M., Lathoud, G., Lincoln, M., Lisowska, A., McCowan, I., Post, W., Reidsma, D., and Wellner, P. (2006). The AMI Meeting Corpus: A Pre-announcement. In *International Conference on Machine Learning for Multimodal Interaction*, MLMI'05, pages 28–39, Berlin, Heidelberg. Springer-Verlag.

[Chen et al., 2021] Chen, K., Lv, Q., and Yi, T. (2021). Fast and reliable probabilistic face embeddings in the wild. *arXiv preprint arXiv:2102.04075*.

[Chow and Teicher, 1997] Chow, Y. S. and Teicher, H. (1997). *Probability theory: Independence, interchangeability, martingales.* Springer Texts in Statistics. Springer, New York, 3 edition.

[Chun et al., 2021] Chun, S., Oh, S. J., De Rezende, R. S., Kalantidis, Y., and Larlus, D. (2021). Probabilistic embeddings for cross-modal retrieval. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8415–8424.

[Chung et al., 2018] Chung, J. S., Arsha Nagrani, A., and Zisserman, A. (2018). VoxCeleb2: Deep Speaker Recognition. In *Interspeech*.

[Cumani et al., 2013] Cumani, S., Plchot, O., and Laface, P. (2013). Probabilistic linear discriminant analysis of i-vector posterior distributions. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7644–7648.

[Cumani et al., 2014] Cumani, S., Plchot, O., and Laface, P. (2014). On the use of i–vector posterior distributions in probabilistic linear discriminant analysis. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(4):846–857.

[Dawid and Musio, 2015] Dawid, A. and Musio, M. (2015). Bayesian model selection based on proper scoring rules. *Bayesian Analysis*, 10(2).

[Dehak et al., 2011] Dehak, N., Kenny, P., Dehak, R., Dumouchel, P., and Ouellet, P. (2011). Front-End Factor Analysis for Speaker Verification. *IEEE Transactions on Audio, Speech and Language Processing*, 19(4):788–798.

[Ferrer et al., 2012] Ferrer, L., Bratt, H., Burget, L., Cernocky, H., Glembek, O., Graciarena, M., Lawson, A., Lei, Y., Matejka, P., Plchot, O., et al. (2012). Promoting robustness for speaker modeling in the community: the PRISM evaluation set. In *Proceedings of NIST 2011 workshop*, pages 1–7.

[Fiscus et al., 2006] Fiscus, J. G., Ajot, J., Michel, M., and Garofolo, J. S. (2006). The rich transcription 2006 spring meeting recognition evaluation. In Renals, S., Bengio, S., and Fiscus, J. G., editors, *International Conference on Machine Learning for Multimodal Interaction*, pages 309–322, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Fox et al., 2011] Fox, E. B., Sudderth, E. B., Jordan, M. I., and Willsky, A. S. (2011). A sticky HDP-HMM with application to speaker diarization. *The Annals of Applied Statistics*, pages 1020–1056.

[Friel and Wyse, 2012] Friel, N. and Wyse, J. (2012). Estimating the evidence–a review. *Statistica Neerlandica*, 66(3):288–308.

[Garcia-Romero and Espy-Wilson, 2011] Garcia-Romero, D. and Espy-Wilson, C. Y. (2011). Analysis of i-vector length normalization in speaker recognition systems. In *Interspeech*, Florence, Italy.

[Geman and Geman, 1984] Geman, S. and Geman, D. (1984). Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741.

[Ghahramani and Griffiths, 2005] Ghahramani, Z. and Griffiths, T. (2005). Infinite latent feature models and the Indian buffet process. *Advances in neural information processing systems*, 18.

[Goldwater et al., 2011] Goldwater, S., Griffiths, T. L., and Johnson, M. (2011). Producing power-law distributions and damping word frequencies with two-stage language models. *Journal of machine Learning Research*.

[Hastings, 1970] Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109.

[Heigold et al., 2016] Heigold, G., Moreno, I., Bengio, S., and Shazeer, N. (2016). End-to-end text-dependent speaker verification. In *International Conference on Acoustics, Speech and Signal Processing*, Shanghai, China.

[Hinton, 2002] Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Comput.*, 14(8):1771–1800.

[Huang et al., 2018] Huang, Z., Wang, S., and Yu, K. (2018). Angular Softmax for Short-Duration Text-independent Speaker Verification. In *Interspeech*, Hyderabad, India.

[Ioffe, 2006] Ioffe, S. (2006). Probabilistic Linear Discriminant Analysis. In *9th European Conference on Computer Vision*, Graz, Austria.

[Jaakkola, 2001] Jaakkola, T. S. (2001). Approximation Methods. *Advanced mean field methods: theory and practice*, page 129.

[Jain and Dubes, 1988] Jain, A. K. and Dubes, R. C. (1988). *Algorithms for clustering data.* Prentice-Hall, Inc.

[Jain and Neal, 2004] Jain, S. and Neal, R. M. (2004). A Split-Merge Markov Chain Monte Carlo Procedure for the Dirichlet Process Mixture Model. *Journal of Computational and Graphical Statistics*, 13(1):158–182.

[Karpukhin et al., 2022] Karpukhin, I., Dereka, S., and Kolesnikov, S. (2022). Probabilistic embeddings revisited. *arXiv preprint arXiv:2202.06768.*

[Kenny, 2010] Kenny, P. (2010). Bayesian Speaker Verification with Heavy-Tailed Priors. In *Odyssey: The Speaker and Language Recognition Workshop*, Brno, Czech Republic. keynote presentation.

[Kenny et al., 2014] Kenny, P., Gupta, V., Stafylakis, T., Ouellet, P., and Alam, J. (2014). Deep Neural Networks for extracting Baum-Welch statistics for Speaker Recognition. In *Odyssey: The Speaker and Language Recognition Workshop*, Joensuu, Finland.

[Kenny et al., 2013] Kenny, P., Stafylakis, T., Ouellet, P., Alam, M. J., and Dumouchel, P. (2013). PLDA for speaker verification with utterances of arbitrary duration. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7649–7653. IEEE.

[King, 1967] King, B. (1967). Step-wise clustering procedures. *Journal of the American Statistical Association*, 62(317):86–101.

[Kuzmin et al., 2022] Kuzmin, N., Fedorov, I., and Sholokhov, A. (2022). Magnitude-aware probabilistic speaker embeddings. In *Odyssey: The Speaker and Language Recognition Workshop*, Beijing, China.

[Landini et al., 2022] Landini, F., Profant, J., Diez, M., and Burget, L. (2022). Bayesian HMM clustering of x-vector sequences (VBx) in speaker diarization: theory, implementation and analysis on standard tasks. *Computer Speech & Language*, 71:101254.

[Lee et al., 2020] Lee, K. A., Sadjadi, S. O., Li, H., and Reynolds, D. A. (2020). Two decades into Speaker Recognition Evaluation-are we there yet? *Computer Speech and Language*, 61.

[Lei et al., 2014] Lei, Y., Scheffer, N., Ferrer, L., and McLaren, M. (2014). A novel scheme for speaker recognition using a phonetically-aware deep neural network. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1695–1699.

[Lozano-Diez et al., 2016] Lozano-Diez, A., Silnova, A., Matejka, P., Glembek, O., Plchot, O., Pešán, J., Burget, L., and Gonzalez-Rodriguez, J. (2016). Analysis and optimization of bottleneck features for speaker recognition. In *Odyssey: The Speaker and Language Recognition Workshop*, pages 21–24.

[Martin and Greenberg, 2010] Martin, A. F. and Greenberg, C. S. (2010). The NIST 2010 speaker recognition evaluation. In *Eleventh Annual Conference of the International Speech Communication Association*.

[McClachlan and Krishnan, 2008] McClachlan, G. J. and Krishnan, T. (2008). *The EM Algorithm and Extensions*. John Wiley & Sons, 2 edition.

[McLaren et al., 2018] McLaren, M., Castan, D., Nandwana, M. K., Ferrer, L., and Yılmaz, E. (2018). How to train your speaker embeddings extractor. In *Odyssey: The Speaker and Language Recognition Workshop*, Les Sables d'Olonne, France.

[McLaren et al., 2016] McLaren, M., Ferrer, L., Castan, D., and Lawson, A. (2016). The 2016 Speakers in the Wild Speaker Recognition Evaluation. In *Interspeech*, pages 823–827.

[McLaren et al., 2015] McLaren, M., Lei, Y., and Ferrer, L. (2015). Advances in deep neural network approaches to speaker recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4814–4818.

[Metropolis and Ulam, 1949] Metropolis, N. and Ulam, S. (1949). The Monte Carlo method. *Journal of the American statistical association*, 44(247):335–341.

[Nagrani et al., 2019] Nagrani, A., Chung, J. S., Xie, W., and Zisserman, A. (2019). Voxceleb: Large-scale speaker verification in the wild. *Computer Speech and Language.*

[Nagrani et al., 2017] Nagrani, A., Chung, J. S., and Zisserman, A. (2017). Voxceleb: a large-scale speaker identification dataset. In *Interspeech*, Stockholm, Sweden.

[Oh et al., 2018] Oh, S. J., Murphy, K. P., Pan, J., Roth, J., Schroff, F., and Gallagher, A. C. (2018). Modeling uncertainty with hedged instance embeddings. In *International Conference on Learning Representations*.

[Okabe et al., 2018] Okabe, K., Koshinaka, T., and Shinoda, K. (2018). Attentive statistics pooling for deep speaker embedding. *arXiv preprint arXiv:1803.10963*.

[Peddinti et al., 2015] Peddinti, V., Povey, D., and Khudanpur, S. (2015). A time delay neural network architecture for efficient modeling of long temporal contexts. In *Interspeech*, Dresden, Germany.

[Pitman, 1995] Pitman, J. (1995). Exchangeable and partially exchangeable random partitions. *Probability Theory and Related Fields*, 102:145–158.

[Povey et al., 2011] Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., et al. (2011). The Kaldi speech recognition toolkit. In *IEEE 2011 workshop on automatic speech recognition and understanding*. IEEE Signal Processing Society.

[Prince and Elder, 2007] Prince, S. J. and Elder, J. H. (2007). Probabilistic linear discriminant analysis for inferences about identity. In *IEEE 11th International Conference on Computer Vision*.

[Reynolds et al., 2000] Reynolds, D. A., Quatieri, T. F., and Dunn, R. B. (2000). Speaker verification using adapted gaussian mixture models. *Digital Signal Processing*, 10:19–41.

[Rohdin et al., 2014] Rohdin, J., Biswas, S., and Shinoda, K. (2014). Constrained discriminative PLDA training for speaker verification. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1670–1674.

[Rohdin et al., 2016] Rohdin, J., Biswas, S., and Shinoda, K. (2016). Robust discriminative training against data insufficiency in PLDA-based speaker verification. *Computer Speech and Language*, 35:32–57.

[Rohdin et al., 2020] Rohdin, J., Silnova, A., Diez, M., Plchot, O., Matějka, P., Burget, L., and Glembek, O. (2020). End-to-end DNN based text-independent speaker recognition for long and short utterances. *Computer Speech and Language*, 59:22–35.

[Rohdin et al., 2018] Rohdin, J., Silnova, A., Diez, M., Plchot, O., Matějka, P., and Burget, L. (2018). End-to-End DNN Based Speaker Recognition Inspired by I-Vector and PLDA. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4874–4878.

[Rossi, 2015] Rossi, G. (2015). Hamming distance between partitions, clustering comparison and information. In *2015 International Conference on Pure Mathematics, Applied Mathematics and Computational Methods (PMAMCM 2015). Zakynthos Island, Greece. Mathematics and Computers in Science and Engineering Series*, volume 48, pages 101–107.

[Ryant et al., 2019] Ryant, N., Church, K., Cieri, C., Cristia, A., Du, J., Ganapathy, S., and Liberman, M. (2019). The second DIHARD diarization challenge: Dataset, task, and baselines. In *Interspeech*.

[Ryant and et al., 2018] Ryant, N. and et al. (2018). DIHARD Corpus. Linguistic Data Consortium.

[Sadjadi et al., 2019] Sadjadi, S. O., Greenberg, C., Singer, E., Reynolds, D., Mason, L., and Hernandez-Cordero, J. (2019). The 2018 NIST speaker recognition evaluation. In *Interspeech*, Graz, Austria.

[Sadjadi et al., 2020] Sadjadi, S. O., Greenberg, C., Singer, E., Reynolds, D., Mason, L., and Hernandez-Cordero, J. (2020). The 2019 NIST speaker recognition evaluation CTS challenge. In *Odyssey: The Speaker and Language Recognition Workshop*, pages 266–272.

[Sadjadi et al., 2017] Sadjadi, S. O., Kheyrkhah, T., Tong, A., Greenberg, C., Reynolds, D., Singer, E., Mason, L., and Hernandez-Cordero, J. (2017). The 2016 NIST speaker recognition evaluation. In *Interspeech*, Stockholm, Sweden.

[Sell et al., 2018] Sell, G., Snyder, D., McCree, A., Garcia-Romero, D., Villalba, J., Maciejewski, M., Manohar, V., Dehak, N., Povey, D., Watanabe, S., et al. (2018). Diarization is hard: Some experiences and lessons learned for the JHU team in the inaugural DIHARD challenge. In *Interspeech*, pages 2808–2812, Hyderabad, India.

[Shi and Jain, 2019] Shi, Y. and Jain, A. K. (2019). Probabilistic face embeddings. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6902–6911.

[Silnova et al., 2018] Silnova, A., Brümmer, N., Garcia-Romero, D., Snyder, D., and Burget, L. (2018). Fast Variational Bayes for Heavy-tailed PLDA Applied to i-vectors and x-vectors. In *Interspeech*, pages 72–76, Hyderabad, India.

[Silnova et al., 2020] Silnova, A., Brümmer, N., Rohdin, J., Stafylakis, T., and Burget, L. (2020). Probabilistic embeddings for speaker diarization. In *Odyssey: The Speaker and Language Recognition Workshop*, pages 24–31, Tokyo, Japan.

[Sneath et al., 1973] Sneath, P. H., Sokal, R. R., et al. (1973). *Numerical taxonomy. The principles and practice of numerical classification.* W. H. Freeman and Co.

[Snyder et al., 2017] Snyder, D., Garcia-Romero, D., Povey, D., and Khudanpur, S. (2017). Deep neural network embeddings for text-independent speaker verification. In *Interspeech*, Stockholm, Sweden.

[Snyder et al., 2019] Snyder, D., Garcia-Romero, D., Sell, G., McCree, A., Povey, D., and Khudanpur, S. (2019). Speaker recognition for multi-speaker conversations using x-vectors. In *ICASSP*.

[Snyder et al., 2018] Snyder, D., Garcia-Romero, D., Sell, G., Povey, D., and Khudanpur, S. (2018). X-vectors: Robust DNN embeddings for speaker recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, Calgary, Canada.

[Snyder et al., 2016] Snyder, D., Ghahremani, P., Povey, D., Garcia-Romero, D., Carmiel, Y., and Khudanpur, S. (2016). Deep neural network-based speaker embeddings for end-to-end speaker verification. In *IEEE Workshop on Spoken Language Technology*.

[Sriperumbudur et al., 2009] Sriperumbudur, B. K., Fukumizu, K., Gretton, A., Schölkopf, B., and Lanckriet, G. R. (2009). On integral probability metrics, ϕ-divergences and binary classification. *arXiv preprint arXiv:0901.2698*.

[Stafylakis et al., 2013] Stafylakis, T., Kenny, P., Ouellet, P., Perez, J., Kockmann, M., and Dumouchel, P. (2013). Text-dependent speaker recognition using PLDA with uncertainty propagation. In *Interspeech*, Lyon, France.

[Tieleman, 2008] Tieleman, T. (2008). Training Restricted Boltzmann Machines Using Approximations to the Likelihood Gradient. In *International Conference on Machine Learning*, ICML '08, page 1064–1071, Helsinki, Finland.

[Wang and Russell, 2015] Wang, W. and Russell, S. (2015). A Smart-Dumb/Dumb-Smart Algorithm for Efficient Split-Merge MCMC. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, UAI'15, page 902–911, Arlington, Virginia, USA. AUAI Press.

[Yang et al., 2018] Yang, J., Liu, Q., Rao, V., and Neville, J. (2018). Goodness-of-fit testing for discrete distributions via Stein discrepancy. In *International Conference on Machine Learning*, pages 5561–5570. PMLR.

[Zeinali et al., 2019] Zeinali, H., Wang, S., Silnova, A., Matejka, P., and Plchot, O. (2019). BUT System Description to VoxCeleb Speaker Recognition Challenge 2019. In *Proceedings of The VoxCeleb Challange Workshop*, Graz, Austria.

[Zhang et al., 2019] Zhang, A., Wang, Q., Zhu, Z., Paisley, J., and Wang, C. (2019). Fully supervised speaker diarization. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6301–6305.

[Zhang, 2008] Zhang, X. (2008). A very gentle note on the construction of Dirichlet process. *The Australian National University, Canberra*.

Appendix A

Derivation of VB lower bound and optimal Q-factors for HT-PLDA

Here, we show the complete derivations of the results presented in Section 4.1. Namely, we derive the variational lower bound L and the expressions for optimal factors $Q(\boldsymbol{\alpha})$ and $Q(\mathbf{z})$.

A.1 VB lower bound

As described in Chapter 4, VB lower bound is computed as:

$$\begin{aligned} L &= \int Q(\mathbf{z}, \boldsymbol{\alpha}) \log \frac{P(\mathcal{R}, \mathbf{z}, \boldsymbol{\alpha} \mid \mathcal{L}, \theta)}{Q(\mathbf{z}, \boldsymbol{\alpha})} \, \mathrm{d}\mathbf{z} \, \mathrm{d}\boldsymbol{\alpha} \\ &= \sum_i \left\langle \log \frac{P(\mathcal{R}_i, \mathbf{z}, \boldsymbol{\alpha} \mid H_s, \theta)}{Q_i(\mathbf{z}) Q_i(\boldsymbol{\alpha})} \right\rangle_{Q_i(\mathbf{z})Q_i(\boldsymbol{\alpha})} \\ &= \sum_i \left\langle \log P(\mathcal{R}_i \mid H_s, \mathbf{z}, \boldsymbol{\alpha}, \theta) + \log \frac{P(\mathbf{z})}{Q_i(\mathbf{z})} + \log \frac{P(\boldsymbol{\alpha})}{Q_i(\boldsymbol{\alpha})} \right\rangle_{Q_i(\mathbf{z})Q_i(\boldsymbol{\alpha})} \\ &= \sum_i \left[\langle \log P(\mathcal{R}_i \mid H_s, \mathbf{z}, \boldsymbol{\alpha}, \theta) \rangle_{Q_i(\mathbf{z})Q_i(\boldsymbol{\alpha})} + \left\langle \log \frac{P(\mathbf{z})}{Q_i(\mathbf{z})} \right\rangle_{Q_i(\mathbf{z})} + \left\langle \log \frac{P(\boldsymbol{\alpha})}{Q_i(\boldsymbol{\alpha})} \right\rangle_{Q_i(\boldsymbol{\alpha})} \right], \end{aligned} \tag{A.1}$$

where the last two expressions are negative KL divergences between variational factors $Q(\mathbf{z})$ and $Q(\boldsymbol{\alpha})$ and priors for hidden speaker variable $P(\mathbf{z})$ and precision scaling factors $P(\boldsymbol{\alpha})$ respectively.

Expression $\langle \log P(\mathcal{R}_i \mid H_s, \mathbf{z}, \boldsymbol{\alpha}, \theta) \rangle_{Q_i(\mathbf{z})Q_i(\boldsymbol{\alpha})}$ can be evaluated taking into account that, in HT-PLDA model, likelihood is a Gaussian function with parameters depending on both $\mathbf{z}$ and $\boldsymbol{\alpha}$ (see (3.16)). (3.16) gives the likelihood $P(\mathbf{r}_{ij} \mid \mathbf{z}, \boldsymbol{\alpha}, \theta)$ for a single datapoint $\mathbf{r}_{ij}$. The likelihood of $\mathcal{R}_i$ consisting of N_i vectors belonging to the same speaker is:

$$P(\mathcal{R}_i \mid H_s, \mathbf{z}, \boldsymbol{\alpha}, \theta) = \prod_{j=1}^{N_i} P(\mathbf{r}_{ij} \mid \mathbf{z}, \boldsymbol{\alpha}, \theta) = \prod_{j=1}^{N_i} \mathcal{N}(\mathbf{r}_{ij} \mid \mathbf{F}\mathbf{z}_i, (\alpha_{ij}\mathbf{W})^{-1}). \tag{A.2}$$

Taking into account (A.2), the expectation becomes:

$$
\begin{aligned}
&\langle \log P(\mathcal{R}_i \mid H_s, \mathbf{z}, \boldsymbol{\alpha}, \theta)\rangle_{Q_i(\mathbf{z})Q_i(\boldsymbol{\alpha})} = \left\langle \sum_{j=1}^{N_i} \log \mathcal{N}(\mathbf{r}_{ij} \mid \mathbf{F}\mathbf{z}_i, (\alpha_{ij}\mathbf{W})^{-1}) \right\rangle_{Q_i(\mathbf{z})Q_i(\boldsymbol{\alpha})} \\
&= \left\langle \sum_{j=1}^{N_i} \left[\frac{1}{2}\log|\alpha_{ij}\mathbf{W}| - \frac{D}{2}\log(2\pi) - \frac{1}{2}(\mathbf{r}_{ij}-\mathbf{F}\mathbf{z}_i)'(\alpha_{ij}\mathbf{W})(\mathbf{r}_{ij}-\mathbf{F}\mathbf{z}_i)\right]\right\rangle_{Q_i(\mathbf{z})Q_i(\boldsymbol{\alpha})} \\
&= \left\langle \sum_{j=1}^{N_i}\left[\frac{D}{2}\log(\alpha_{ij}) + \frac{1}{2}\log|\mathbf{W}| - \frac{1}{2}\alpha_{ij}\mathbf{r}'_{ij}\mathbf{W}\mathbf{r}_{ij} + \alpha_{ij}\mathbf{z}'_i\mathbf{F}'\mathbf{W}\mathbf{r}_{ij} - \frac{1}{2}\alpha_{ij}\mathbf{z}'_i\mathbf{F}'\mathbf{W}\mathbf{F}\mathbf{z}_i\right]\right\rangle_{Q_i(\mathbf{z})Q_i(\boldsymbol{\alpha})} + \text{const} \\
&= \frac{D}{2}\sum_j\langle\log(\alpha_{ij})\rangle + \frac{N_i}{2}\log|\mathbf{W}| - \frac{1}{2}\sum_j\langle\alpha_{ij}\rangle\mathbf{r}'_{ij}\mathbf{W}\mathbf{r}_{ij} + \langle\mathbf{z}_i\rangle'\mathbf{F}'\mathbf{W}\sum_j\langle\alpha_{ij}\rangle\mathbf{r}_{ij} \\
&- \frac{1}{2}\sum_j\langle\alpha_{ij}\rangle\mathrm{tr}(\langle\mathbf{z}_i\mathbf{z}'_i\rangle\mathbf{F}'\mathbf{W}\mathbf{F}) + \text{const.}
\end{aligned}
\tag{A.3}
$$

Substituting (A.3) in (A.1), we get the final expression for VB lower bound:

$$
\begin{aligned}
L = \sum_i\Bigg[&\frac{D}{2}\sum_j\langle\log(\alpha_{ij})\rangle + \frac{N_i}{2}\log|\mathbf{W}| - \frac{1}{2}\sum_j\langle\alpha_{ij}\rangle\mathbf{r}'_{ij}\mathbf{W}\mathbf{r}_{ij} + \langle\mathbf{z}_i\rangle'\mathbf{F}'\mathbf{W}\sum_j\langle\alpha_{ij}\rangle\mathbf{r}_{ij} \\
&- \frac{1}{2}\sum_j\langle\alpha_{ij}\rangle\mathrm{tr}(\langle\mathbf{z}_i\mathbf{z}'_i\rangle\mathbf{F}'\mathbf{W}\mathbf{F}) - \mathrm{KL}(Q_i(\mathbf{z}) \mid\mid P(\mathbf{z})) - \mathrm{KL}(Q_i(\boldsymbol{\alpha}) \mid\mid P(\boldsymbol{\alpha}))\Bigg] + \text{const.}
\end{aligned}
\tag{A.4}
$$

A.2 Estimating optimal $Q(\boldsymbol{\alpha})$

Analytic solutions for $Q(\boldsymbol{\alpha})$ can be found by evaluating expectation over $Q(\mathbf{z})$ [Bishop, 2006]:

$$
\begin{aligned}
\log Q_i(\boldsymbol{\alpha}) &= \left\langle \log P(R_i, \boldsymbol{\alpha}, \mathbf{z})\right\rangle_{Q_i(\mathbf{z})} + \text{const} \\
&= \left\langle \log P(R_i \mid \boldsymbol{\alpha}, \mathbf{z}) + \log P(\boldsymbol{\alpha}) + \log P(\mathbf{z})\right\rangle_{Q_i(\mathbf{z})} + \text{const} \\
&= \left\langle \log P(R_i \mid \boldsymbol{\alpha}, \mathbf{z})\right\rangle_{Q_i(\mathbf{z})} + \log P(\boldsymbol{\alpha}) + \text{const,}
\end{aligned}
\tag{A.5}
$$

where $P(R_i \mid \boldsymbol{\alpha}, \mathbf{z})$ is given by (A.2) and the prior $P(\boldsymbol{\alpha})$ is Gamma distribution with shape and rate parameters set to $\frac{\nu}{2}$ (see (3.14)). Then, substituting (3.16) and (3.14) in (A.5):

$$
\begin{aligned}
\log Q_i(\boldsymbol{\alpha}) = & \left\langle \sum_{j=1}^{N_i} \left[\frac{D}{2}\log(\alpha_{ij}) + \frac{1}{2}\log|\mathbf{W}| - \frac{1}{2}\alpha_{ij}\mathbf{r}'_{ij}\mathbf{W}\mathbf{r}_{ij} + \alpha_{ij}\mathbf{z}'_i\mathbf{F}'\mathbf{W}\mathbf{r}_{ij} - \frac{1}{2}\alpha_{ij}\mathbf{z}'_i\mathbf{F}'\mathbf{W}\mathbf{F}\mathbf{z}_i \right] \right\rangle_{Q_i(\mathbf{z})} \\
& + \sum_{j=1}^{N_i} \left[\frac{\nu}{2}\log\left(\frac{\nu}{2}\right) - \log\Gamma\left(\frac{\nu}{2}\right) + \left(\frac{\nu}{2} - 1\right)\log(\alpha_{ij}) - \frac{\nu}{2}\alpha_{ij} \right] + \text{const} \\
& = \sum_{j=1}^{N_i} \left[\frac{D}{2}\log(\alpha_{ij}) - \frac{1}{2}\alpha_{ij}\mathbf{r}'_{ij}\mathbf{W}\mathbf{r}_{ij} + \langle\mathbf{z}_i\rangle'\mathbf{F}'\mathbf{W}\alpha_{ij}\mathbf{r}_{ij} - \frac{1}{2}\alpha_{ij}\mathrm{tr}(\langle\mathbf{z}_i\mathbf{z}'_i\rangle\mathbf{F}'\mathbf{W}\mathbf{F}) \right. \\
& \left. + \left(\frac{\nu}{2} - 1\right)\log(\alpha_{ij}) - \frac{\nu}{2}\alpha_{ij} \right] + \text{const} \\
& = \sum_{j=1}^{N_i} \left[\left(\frac{D}{2} + \frac{\nu}{2} - 1\right)\log(\alpha_{ij}) \right. \\
& \left. + \alpha_{ij}\left(-\frac{\nu}{2} - \frac{1}{2}\mathbf{r}'_{ij}\mathbf{W}\mathbf{r}_{ij} + \langle\mathbf{z}_i\rangle'\mathbf{F}'\mathbf{W}\alpha_{ij}\mathbf{r}_{ij} - \frac{1}{2}\mathrm{tr}(\langle\mathbf{z}_i\mathbf{z}'_i\rangle\mathbf{F}'\mathbf{W}\mathbf{F}) \right) \right] + \text{const} \\
& = \sum_{j=1}^{N_i} \log\mathcal{G}\left(\alpha_{ij} \mid \frac{\nu + D}{2}, \frac{\nu}{2} + \frac{1}{2}\mathbf{r}'_{ij}\mathbf{W}\mathbf{r}_{ij} - \mathbf{r}'_{ij}\mathbf{W}\mathbf{F}\langle\mathbf{z}_i\rangle + \frac{1}{2}\mathrm{tr}(\langle\mathbf{z}_i\mathbf{z}'_i\rangle\mathbf{F}'\mathbf{W}\mathbf{F}) \right) + \text{const}.
\end{aligned}
\tag{A.6}
$$

Thus, the optimal $Q_i(\boldsymbol{\alpha})$ is a product of independent gamma distributions:

$$
Q_i(\boldsymbol{\alpha}) \propto \prod_{j=1}^{N_i} \mathcal{G}\left(\alpha_{ij} \mid \frac{\nu + D}{2}, \frac{\nu}{2} + \frac{1}{2}\mathbf{r}'_{ij}\mathbf{W}\mathbf{r}_{ij} - \mathbf{r}'_{ij}\mathbf{W}\mathbf{F}\langle\mathbf{z}_i\rangle + \frac{1}{2}\mathrm{tr}(\langle\mathbf{z}_i\mathbf{z}'_i\rangle\mathbf{F}'\mathbf{W}\mathbf{F}) \right). \tag{A.7}
$$

A.3 Estimating optimal $Q(\mathbf{z})$

Similar to estimating the optimal $Q_i(\boldsymbol{\alpha})$, one can find the optimal Q-factor for hidden speaker variable $\mathbf{z}$:

$$
\begin{aligned}
\log Q_i(\mathbf{z}) &= \langle \log P(R_i, \boldsymbol{\alpha}, \mathbf{z}) \rangle_{Q_i(\boldsymbol{\alpha})} + \text{const} \\
&= \langle \log P(R_i \mid \boldsymbol{\alpha}, \mathbf{z}) + \log P(\boldsymbol{\alpha}) + \log P(\mathbf{z}) \rangle_{Q_i(\boldsymbol{\alpha})} + \text{const} \\
&= \langle \log P(R_i \mid \boldsymbol{\alpha}, \mathbf{z}) \rangle_{Q_i(\boldsymbol{\alpha})} + \log P(\mathbf{z}) + \text{const},
\end{aligned}
\tag{A.8}
$$

where the first component can be computed using (A.2); the prior $P(\mathbf{z})$ is a standard normal distribution as assumed by the PLDA model. Then (A.8) becomes:

$$
\begin{aligned}
\log Q_i(\mathbf{z}) = &\Big\langle \sum_{j=1}^{N_i} \Big[\frac{D}{2}\log(\alpha_{ij}) + \frac{1}{2}\log|\mathbf{W}| - \frac{1}{2}\alpha_{ij}\mathbf{r}'_{ij}\mathbf{W}\mathbf{r}_{ij} + \alpha_{ij}\mathbf{z}'_i\mathbf{F}'\mathbf{W}\mathbf{r}_{ij} - \frac{1}{2}\alpha_{ij}\mathbf{z}'_i\mathbf{F}'\mathbf{W}\mathbf{F}\mathbf{z}_i \Big] \Big\rangle_{Q_i(\boldsymbol{\alpha})} \\
&- \frac{d}{2}\log(2\pi) - \frac{1}{2}\mathbf{z}'_i\mathbf{z}_i + \text{const} \\
&= \mathbf{z}'_i\mathbf{F}'\mathbf{W}\sum_{j=1}^{N_i}\langle\alpha_{ij}\rangle\mathbf{r}_{ij} - \frac{1}{2}\mathbf{z}'_i(\sum_{j=1}^{N_i}\langle\alpha_{ij}\rangle\mathbf{F}'\mathbf{W}\mathbf{F})\mathbf{z}_i - \frac{1}{2}\mathbf{z}'_i\mathbf{z}_i + \text{const} \\
&= \log\mathcal{N}\Big(\mathbf{z} \mid (\sum_{j=1}^{N_i}\langle\alpha_{ij}\rangle\mathbf{F}'\mathbf{W}\mathbf{F} + \mathbf{I})^{-1}\sum_{j=1}^{N_i}\langle\alpha_{ij}\rangle\mathbf{F}'\mathbf{W}\mathbf{r}_{ij}, (\sum_{j=1}^{N_i}\langle\alpha_{ij}\rangle\mathbf{F}'\mathbf{W}\mathbf{F} + \mathbf{I})^{-1}\Big) + \text{const}.
\end{aligned} \tag{A.9}
$$

The optimal $Q_i(\mathbf{z})$ is a multivariate Gaussian:

$$
Q_i(\mathbf{z}) \propto \mathcal{N}\Big(\mathbf{z} \mid (\sum_{j=1}^{N_i}\langle\alpha_{ij}\rangle\mathbf{F}'\mathbf{W}\mathbf{F} + \mathbf{I})^{-1}\sum_{j=1}^{N_i}\langle\alpha_{ij}\rangle\mathbf{F}'\mathbf{W}\mathbf{r}_{ij}, (\sum_{j=1}^{N_i}\langle\alpha_{ij}\rangle\mathbf{F}'\mathbf{W}\mathbf{F} + \mathbf{I})^{-1}\Big). \tag{A.10}
$$

Appendix B

Chinese Restaurant Process

Here, we describe Chinese Restaurant Process prior used when defining discriminative training objectives for HT-PLDA and probabilistic embedding models in Sections 5.3, 5.4, 5.5, and 6.2.2.

We are interested in defining the distribution over all possible partitions of the dataset of size N into speaker clusters. *Chinese Restaurant process* [Goldwater et al., 2011,Pitman, 1995,Fox et al., 2011,Zhang et al., 2019,Ghahramani and Griffiths, 2005] provides a recursive procedure to construct such distribution. It is defined as follows.

Let us consider a case when the partition of $N-1$ out of N points is fixed, and we are interested in computing the probability of the label of the remaining point. Assume the remaining point has index N; otherwise, we can just re-index them so that it is true.

$$\mathcal{L}_{\setminus N} = \{l_1, l_2, ..., l_{N-1}\}, l_i \in \{1, ..., m\}. \tag{B.1}$$

Here, $\mathcal{L}_{\setminus N}$ denotes the partition of the whole dataset with the last point removed. $\mathcal{L}_{\setminus N}$ distributes $N-1$ data points (recordings) to m speaker clusters. The label of the last point can belong to $\{1, ..., m\}$, but this point can also form a separate cluster and can be assigned a new label $l_N = m+1$. According to CRP, the probability for these alternatives is computed as:

$$P(l_N \mid \mathcal{L}_{\setminus N}) = \begin{cases} \frac{\sum_{i=1}^{N-1} \mathbb{I}(l_i=j)-\beta}{N-1+\alpha}, & \text{for } l_N = j,\ j = 1...m \\ \frac{m\beta+\alpha}{N-1+\alpha}, & \text{for } l_N = m+1. \end{cases} \tag{B.2}$$

Here, $\alpha \geq 0$ and $0 \leq \beta < 1$ are the CRP parameters called *concentration* and *discount*, respectively. $\mathbb{I}$ is an indicator function and $\sum_i \mathbb{I}(l_i = j)$ simply tells how many points belong to the cluster j according to $\mathcal{L}_{\setminus N}$.

Partition probability for the whole set is:

$$P(\mathcal{L}) = P(l_N \mid \mathcal{L}_{\setminus N})P(\mathcal{L}_{\setminus N}). \tag{B.3}$$

The first term is given by (B.2), while the second one can be computed by recursive application of (B.2).

Thus, the probability of the whole partition $\mathcal{L}$ can be computed iteratively. At each step of the recursion, we assume that the labels are consecutive integers. If that is not the case, simple reassignment of the labels can fix it.

Finally, in the recursion, we arrive at the situation where the first point has to be assigned its label. Logically, with probability 1, it is assigned the first available label 1:

$$P(l_1 = 1) = P(l_1 = 1 \mid \emptyset) = 1. \tag{B.4}$$

The same procedure applied in the opposite direction (from a single point to the partition of N points) describes a generative process assumed by CRP. The generative process illustrates the restaurant analogy (hence, the name CRP), where customers come one by one to a restaurant and decide on whether to join other customers at an already occupied table or sit at an empty table.

For CRP with fixed parameters, the expected number of clusters for N data points is given by [Zhang, 2008]:

$$\langle m \rangle = \begin{cases} 1, & \alpha = 0, \beta = 0, \\ \frac{\Gamma(\alpha+\beta+N)\Gamma(\alpha+1)}{\beta\Gamma(\alpha+N)\Gamma(\alpha+\beta)} - \frac{\alpha}{\beta}, & \alpha \geq 0, \beta > 0, \\ \alpha(\Psi(\alpha+N) - \Psi(\alpha)), & \alpha > 0, \beta = 0, \end{cases} \tag{B.5}$$

where $\Gamma(\cdot)$ and $\Psi(\cdot)$ are gamma and digamma functions, respectively.

CRP defines an exchangeable distribution, meaning that the value of $P(\mathcal{L})$ does not depend on the particular order the points are selected during the computation process described above. There is a closed-form expression to compute this probability for a given partition, i.e., running the iterative procedure is unnecessary [Ghahramani and Griffiths, 2005]:

$$P(\mathcal{L}) = \frac{\Gamma(\alpha)}{\Gamma(\alpha+N)} \frac{\beta^m \Gamma(\frac{\alpha}{\beta}+m)}{\Gamma(\frac{\alpha}{\beta})} \prod_{j=1}^{m} \frac{\Gamma(\sum_{i=1}^{N} \mathbb{I}(l_i = j) - \beta)}{\Gamma(1-\beta)}, \tag{B.6}$$

where $\Gamma(\cdot)$ is gamma function, and $\mathcal{L}$ is a partition of N datapoints into m clusters.

To summarize, CRP defines a suitable prior for our purpose of discriminative training to maximize the (approximate) posterior of the correct partition. The prior probability of any partition can be conveniently computed with (B.6). Let us add a few notes:

- Properties of the CRP are defined by choice of its parameters α, β. These parameters control the expected number of clusters and the variance of the number and sizes of clusters in different realizations of CRP. Figure B.1 illustrates this fact. It was created as follows. We select two sets of parameters that result in the same expected number of clusters (200 clusters for 1000 points) and run 1000 realizations of CRP with each set. Then, we plot the histograms of the actual number of clusters in the obtained realizations. As we see, even though the expected number of clusters for both sets of parameters is the same, the variance of that number differs significantly.

- To use CRP as a prior for discriminative training, one has to set the parameters beforehand so that they reflect the properties of the training data. If that is not done, we expect that the training would skew the model likelihoods to compensate for the mismatching prior. Ideally, we would want to estimate the parameters of the prior on n datasets as:

$$\alpha^*, \beta^* = \underset{(\alpha,\beta)}{\operatorname{argmax}} \prod_{i=1}^{n} P(\mathcal{L}_i^* \mid \alpha, \beta), \tag{B.7}$$

  where, $\mathcal{L}_i^*$ is a true partition of training set i. For example, this can be done in the case of diarization, where each utterance represents a separate partitioning problem. The parameters set in this way would not be overtrained on one particular training set and can be expected to fit the test data well. However, in practice, in all our experiments,

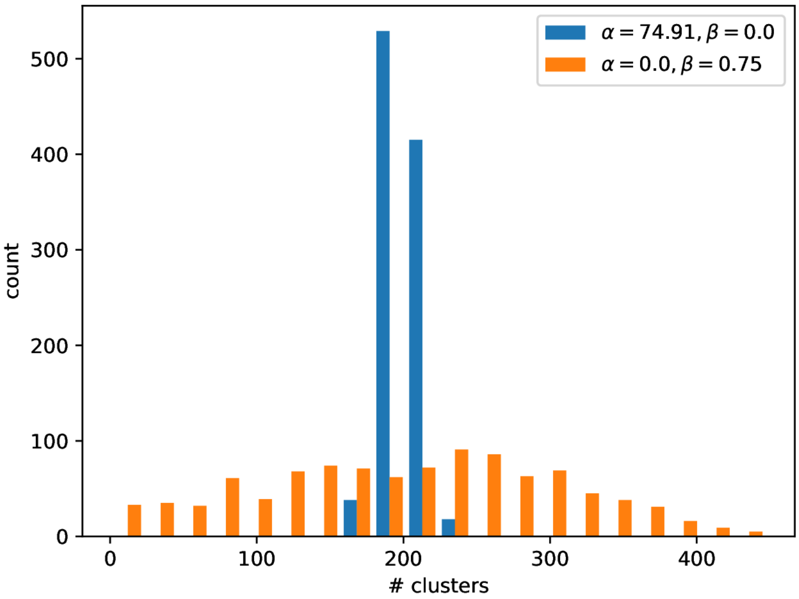

Figure B.1: Histogram of the actual number of clusters in 1000 realizations of CRP with parameters $\alpha = 74.91, \beta = 0$ (blue) and $\alpha = 0, \beta = 0.75$ (orange). Both sets of parameters result in the expected number of clusters $\langle m \rangle = 200$ for 1000 data points.

we use just a single dataset to estimate the parameters of CRP. By fixing one of the parameters α or β, we can set the second one so that the expected number of clusters $\langle m \rangle$ (see (B.5)) equals the true number of speakers defined by $\mathcal{L}^*$. In practice, we typically set $\beta = 0$ and numerically solve the equation $\langle m \rangle = m^*$ to find α.

Appendix C

LLR score for HT-PLDA model

Here, we derive the expression for the LLR score computed with the HT-PLDA model introduced in Section 3.2.

The general definition of a log-likelihood ratio score s_{ij} for two recordings $\mathbf{r}_i$ and $\mathbf{r}_j$ (HT-PLDA assumes that they are embeddings) is given by (2.6):

$$\begin{aligned} s_{ij} =& \log P(\mathbf{r}_i, \mathbf{r}_j \mid H_s) - \log P(\mathbf{r}_i, \mathbf{r}_j \mid H_d) \\ =& \log \big\langle P(\mathbf{r}_i \mid \mathbf{z}) P(\mathbf{r}_j \mid \mathbf{z}) \big\rangle_{\pi(\mathbf{z})} - \log \big\langle P(\mathbf{r}_i \mid \mathbf{z}) \big\rangle_{\pi(\mathbf{z})} - \log \big\langle P(\mathbf{r}_j \mid \mathbf{z}) \big\rangle_{\pi(\mathbf{z})}. \end{aligned} \tag{C.1}$$

By using (3.28) to compute the log-expectations over prior of the hidden $\mathbf{z}$, LLR score becomes:

$$\begin{aligned} s_{ij} =& \frac{1}{2}\left[(\mathbf{a}_i + \mathbf{a}_j)' \big[(\mathbf{B}_i + \mathbf{B}_j) + \mathbf{I}\big]^{-1} (\mathbf{a}_i + \mathbf{a}_j) - \log |(\mathbf{B}_i + \mathbf{B}_j) + \mathbf{I}|\right] \\ & - \frac{1}{2}\left[\mathbf{a}_i' \big[\mathbf{B}_i + \mathbf{I}\big]^{-1} \mathbf{a}_i - \log |\mathbf{B}_i + \mathbf{I}|\right] - \frac{1}{2}\left[\mathbf{a}_j' \big[\mathbf{B}_j + \mathbf{I}\big]^{-1} \mathbf{a}_j - \log |\mathbf{B}_j + \mathbf{I}|\right]. \end{aligned} \tag{C.2}$$

Notice that (3.28) also includes an additive constant, but these cancel when LLR is computed, hence we do not include them in (C.2). $\mathbf{a}$ and $\mathbf{B}$ are defined by (3.25). Substituting them in (C.2), we get s_{ij} as a function of HT-PLDA parameters $\mathbf{W}$,$\mathbf{F}$, ν, and embeddings $\mathbf{r}_i$ and $\mathbf{r}_j$:

$$\begin{aligned} s_{ij} =& \frac{1}{2}\left[(b_i \mathbf{F}'\mathbf{W}\mathbf{r}_i + b_j \mathbf{F}'\mathbf{W}\mathbf{r}_j)' \big[(b_i + b_j)\mathbf{F}'\mathbf{W}\mathbf{F} + \mathbf{I}\big]^{-1} (b_i \mathbf{F}'\mathbf{W}\mathbf{r}_i + b_j \mathbf{F}'\mathbf{W}\mathbf{r}_j) - \log |(b_i + b_j)\mathbf{F}'\mathbf{W}\mathbf{F} + \mathbf{I}|\right] \\ & - \frac{1}{2}\left[(b_i \mathbf{F}'\mathbf{W}\mathbf{r}_i)' \big[b_i \mathbf{F}'\mathbf{W}\mathbf{F} + \mathbf{I}\big]^{-1} b_i \mathbf{F}'\mathbf{W}\mathbf{r}_i - \log |b_i \mathbf{F}'\mathbf{W}\mathbf{F} + \mathbf{I}|\right] \\ & - \frac{1}{2}\left[(b_j \mathbf{F}'\mathbf{W}\mathbf{r}_j)' \big[(b_j \mathbf{F}'\mathbf{W}\mathbf{F} + \mathbf{I}\big]^{-1} b_j \mathbf{F}'\mathbf{W}\mathbf{r}_j - \log |b_j \mathbf{F}'\mathbf{W}\mathbf{F} + \mathbf{I}|\right], \end{aligned} \tag{C.3}$$

where scalars b are given by (3.22):

$$b = \frac{\nu + D - d}{\nu + \mathbf{r}'(\mathbf{W} - \mathbf{W}\mathbf{F}(\mathbf{F}'\mathbf{W}\mathbf{F})^{-1}\mathbf{F}'\mathbf{W})\mathbf{r}}. \tag{C.4}$$

Reorganizing the terms in (C.3), we get the final expression for HT-PLDA LLR score:

$$\begin{aligned}
s_{ij} = &\frac{1}{2}b_i^2\mathbf{r}_i'\mathbf{W}\mathbf{F}\Big[\big[(b_i+b_j)\mathbf{F}'\mathbf{W}\mathbf{F}+\mathbf{I}\big]^{-1} + \big[b_i\mathbf{F}'\mathbf{W}\mathbf{F}+\mathbf{I}\big]^{-1}\Big]\mathbf{F}'\mathbf{W}\mathbf{r}_i+ \\
&+\frac{1}{2}b_j^2\mathbf{r}_j'\mathbf{W}\mathbf{F}\Big[\big[(b_i+b_j)\mathbf{F}'\mathbf{W}\mathbf{F}+\mathbf{I}\big]^{-1} + \big[b_j\mathbf{F}'\mathbf{W}\mathbf{F}+\mathbf{I}\big]^{-1}\Big]\mathbf{F}'\mathbf{W}\mathbf{r}_j+ \\
&+b_ib_j\mathbf{r}_i'\mathbf{W}\mathbf{F}\big[(b_i+b_j)\mathbf{F}'\mathbf{W}\mathbf{F}+\mathbf{I}\big]^{-1}\mathbf{F}'\mathbf{W}\mathbf{r}_j- \\
&-\frac{1}{2}\log|(b_i+b_j)\mathbf{F}'\mathbf{W}\mathbf{F}+\mathbf{I}| + \frac{1}{2}\log|b_i\mathbf{F}'\mathbf{W}\mathbf{F}+\mathbf{I}| + \frac{1}{2}\log|b_j\mathbf{F}'\mathbf{W}\mathbf{F}| = \\
&=\frac{1}{2}\mathbf{r}_i'\mathbf{\Gamma}_i\mathbf{r}_i + \frac{1}{2}\mathbf{r}_j'\mathbf{\Gamma}_j\mathbf{r}_j + \mathbf{r}_i'\mathbf{\Lambda}_{ij}\mathbf{r}_j + k_{ij},
\end{aligned} \tag{C.5}$$

where we have defined:

$$\begin{aligned}
\mathbf{\Gamma}_i &= b_i^2\mathbf{W}\mathbf{F}\big[((b_i+b_j)\mathbf{F}'\mathbf{W}\mathbf{F}+\mathbf{I})^{-1} - (b_i\mathbf{F}'\mathbf{W}\mathbf{F}+\mathbf{I})^{-1}\big]\mathbf{F}'\mathbf{W}, \\
\mathbf{\Gamma}_j &= b_j^2\mathbf{W}\mathbf{F}\big[((b_i+b_j)\mathbf{F}'\mathbf{W}\mathbf{F}+\mathbf{I})^{-1} - (b_j\mathbf{F}'\mathbf{W}\mathbf{F}+\mathbf{I})^{-1}\big]\mathbf{F}'\mathbf{W}, \\
\mathbf{\Lambda}_{ij} &= b_ib_j\mathbf{W}\mathbf{F}\big((b_i+b_j)\mathbf{F}'\mathbf{W}\mathbf{F}+\mathbf{I})^{-1}\mathbf{F}'\mathbf{W}, \\
k_{ij} &= -\frac{1}{2}\log|(b_i+b_j)\mathbf{F}'\mathbf{W}\mathbf{F}+\mathbf{I}| + \frac{1}{2}\log|b_i\mathbf{F}'\mathbf{W}\mathbf{F}+\mathbf{I}| + \frac{1}{2}\log|b_j\mathbf{F}'\mathbf{W}\mathbf{F}+\mathbf{I}|.
\end{aligned} \tag{C.6}$$

Appendix D

How to check the sampling algorithm

In Sections 5.4.2 to 5.4.5, we have described various methods to sample from the partition posterior $P(\mathcal{L} \mid \mathcal{R})$. The posterior over partitions is a complex distribution that is impossible even to evaluate for real-world large-scale tasks. Also, the sampling methods are very elaborated. These two factors make it difficult to verify whether the sampling method was implemented correctly and provides the samples from the correct distribution. Below, we discuss some approaches addressing this problem. However, notice that verifying the sampler is a hard problem that until recently was unsolved (see, e.g., [Brooks et al., 2011]); nowadays, several new methods have become available [Sriperumbudur et al., 2009, Rossi, 2015, Yang et al., 2018] but it is still the area of active research.

D.1 Small-scale synthetic experiment

One of the simplest ways to spot the problem when implementing a sampler is to generate a small-scale synthetic dataset such that it is feasible to compute the posterior over the partitions $P(\mathcal{L} \mid \mathcal{R})$ for this set exactly. Then, collecting the partitions produced by the sampler at hand and comparing their distribution to the true one allows us to see if the sampler generates samples from the correct posterior. We follow this approach and perform the following experiment.

First, we run a generative process to generate a synthetic dataset of eight 10-dimensional data points: first, we fix the parameters of the HT-PLDA model $\theta^* = \{\mathbf{F}^*, \mathbf{W}^*, \nu^*\}$. Then, we sample the simulated "true" labels $\mathcal{L}^*$ from the prior (in our case, CRP with fixed concentration and discount parameters). Finally, given the parameters θ^* and the labels $\mathcal{L}^*$, sample the observed data $\mathcal{R}$. Since the dataset has 8 points, there are $B_8 = 4140$ ways of partitioning them into speaker clusters. The true model and prior parameters are known exactly; consequently, it is possible to compute the posterior for each of the 4140 partitions. The true posterior is displayed on the top graph of Figure D.1. On the graph, the horizontal axis shows indices of possible partitions, and the vertical one displays corresponding values of the true posterior.

Then, using the true model parameters, we run the sampler (in the case of this experiment, SDDS with GS) and collect 10000 samples from it. To collect these samples, we keep the output of every 100th step of the SDDS chain. The lower graph of Figure D.1 shows the normalized histogram of the partitions that were sampled in this way. Visually comparing the true posterior and the sample histogram, we see that the graphs are very similar. However, there are minor differences between them. A possible source of these differences is that

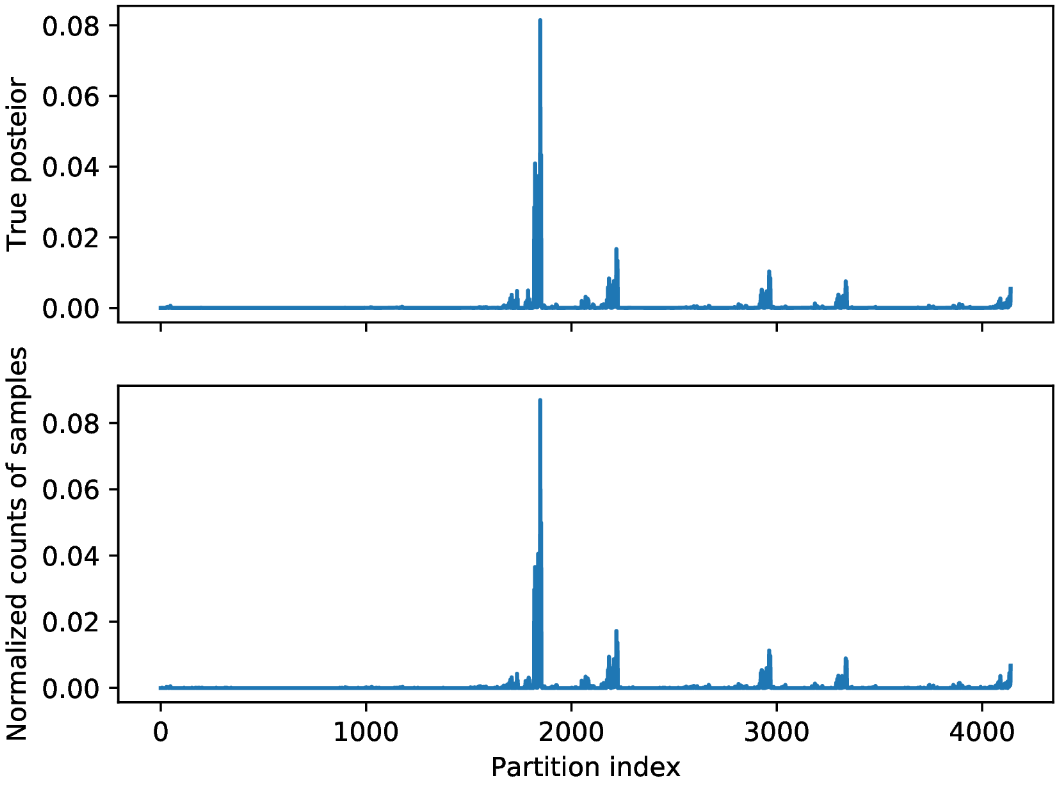

Figure D.1: Top: True posterior estimated with the correct model and prior parameters for each of 4140 partitions of the dataset consisting of 8 points. Bottom: Normalized histogram of 10000 sampled partitions retrieved by SDDS with GS sampling algorithm when correct model parameters were used. Since the graphs are similar, we can conclude that it is likely that the sampling algorithm produces samples from the correct posterior.

we used a finite number of samples to approximate the posterior. To see whether this is the case, we sample from the multinomial distribution with the number of trials parameter set to 10000 (same as the collection of samples obtained with SDDS) and the true posterior used as event distribution. We collect 1000 samples from a such multinomial distribution. Normalizing each sample retrieved from the multinomial distribution by 10000 (number of trials), we get 1000 approximate posteriors. We compute the KL divergence between the true posterior and each of these 1000 approximate ones. We also compute the KL divergence between the true posterior and the approximate posterior estimated with the samples obtained by SDDS. Figure D.2 shows the histogram of all these KL divergences, the red star on the horizontal axis indicates the KL divergence computed with the output of the SDDS sampler. From the graph, we see that this KL divergence is located among others showing that the approximate posterior estimated with the SDDS sample is similar to those approximate posteriors estimated from the finite number of samples from the true posterior distribution.

This experiment shows that there is unlikely a mistake in the implementation of the SDDS sampling algorithm. Similar experiments can be done for other sampling methods; however, we do not show their results here.

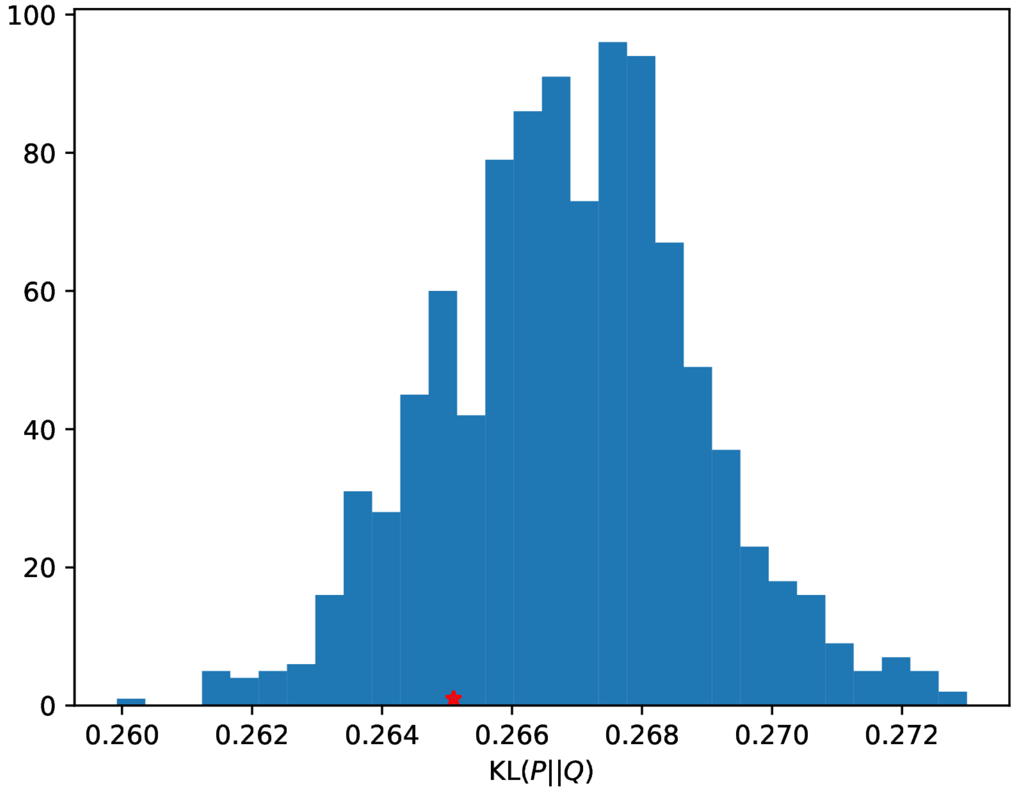

Figure D.2: Histogram of KL divergences between the true (P) and approximate (Q) posteriors. The approximate posteriors are estimated with the finite samples of 10000 elements from the true posterior. KL divergence between the true posterior and the approximate one estimated with 10000 SDDS samples is shown as the red star.

D.2 Sampler quality measure

The experiments described above allow us to conclude that the sampling method was implemented correctly. Still, it is impossible to say when the sampler has warmed up or to compare several samplers this way. Consequently, we propose to consider the following quantity:

$$\begin{aligned}\mathcal{O} &= \left\langle \log P(\mathcal{L} \mid \mathcal{R}) \right\rangle_{Q(\mathcal{L}|\mathcal{R})P(\mathcal{R})} - \left\langle \log P(\mathcal{L} \mid \mathcal{R}) \right\rangle_{P(\mathcal{L}|\mathcal{R})P(\mathcal{R})} \\ &= \left\langle \log P(\mathcal{L}, \mathcal{R}) - \log P(\mathcal{R}) \right\rangle_{Q(\mathcal{L},\mathcal{R})} - \left\langle \log P(\mathcal{L}, \mathcal{R}) - \log P(\mathcal{R}) \right\rangle_{P(\mathcal{L},\mathcal{R})} \\ &= \left\langle \log P(\mathcal{L}, \mathcal{R}) \right\rangle_{Q(\mathcal{L},\mathcal{R})} - \left\langle \log P(\mathcal{R}) \right\rangle_{P(\mathcal{R})} - \left\langle \log P(\mathcal{L}, \mathcal{R}) \right\rangle_{P(\mathcal{L},\mathcal{R})} + \left\langle \log P(\mathcal{R}) \right\rangle_{P(\mathcal{R})} \\ &= \left\langle \log P(\mathcal{L}, \mathcal{R}) \right\rangle_{Q(\mathcal{L},\mathcal{R})} - \left\langle \log P(\mathcal{L}, \mathcal{R}) \right\rangle_{P(\mathcal{L},\mathcal{R})}.\end{aligned} \tag{D.1}$$

Above, $Q(\mathcal{L} \mid \mathcal{R})$ is used to denote the distribution from which the sampling algorithm is effectively sampling, and $P(\mathcal{L} \mid \mathcal{R})$ is the true partition posterior. For the perfect sampler, these two are equal $Q(\mathcal{L} \mid \mathcal{R}) = P(\mathcal{L} \mid \mathcal{R})$. For brevity, in (D.1), we used the following notation:

$$Q(\mathcal{L},\mathcal{R}) = Q(\mathcal{L} \mid \mathcal{R})P(\mathcal{R}).$$

Approximate estimation of $\mathcal{O}$ can be done by noticing that it is possible to sample from $P(\mathcal{L})$ and $P(\mathcal{R} \mid \mathcal{L})$ by running the generative process described above and, consequently, it is possible to sample from $P(\mathcal{R},\mathcal{L})$ by ancestral sampling (first, sampling $\mathcal{L}^* \sim P(\mathcal{L})$ and, then, $\mathcal{R} \sim P(\mathcal{R} \mid \mathcal{L}^*)$). Also, all of these distributions can be evaluated. Besides, if, when sampling from $P(\mathcal{R},\mathcal{L})$, we discard $\mathcal{L}$, we effectively sample from $P(\mathcal{R})$, however it is impossible to evaluate it. Thus, one can approximate $\mathcal{O}$ by drawing N samples $(\mathcal{R}_i, \mathcal{L}^*_i)$ from $P(\mathcal{R},\mathcal{L})$; given these samples, one can estimate the second component of (D.1). By running the sampler (drawing samples $\mathcal{L}_i$ from $Q(\mathcal{L} \mid \mathcal{R}_i)$) for each of the $\mathcal{R}_i$, we can approximate the first expectation in the last line of (D.1). Thus, $\mathcal{O}$ can be approximated as:

$$\mathcal{O} \approx \frac{1}{N}\sum_{i=1}^{N}\bigl[\log P(\mathcal{L}_i,\mathcal{R}_i) - \log P(\mathcal{L}^*_i,\mathcal{R}_i)\bigr] = \frac{1}{N}\sum_{i=1}^{N}\delta_i = \bar{\delta}, \tag{D.2}$$

where, for future references, we have defined:

$$\delta = \log P(\mathcal{L},\mathcal{R}) - \log P(\mathcal{L}^*,\mathcal{R}). \tag{D.3}$$

Now, when we know that it is possible to estimate $\mathcal{O}$ approximately, let us look closer at it to see what properties $\mathcal{O}$ has.

$$\begin{aligned}
\mathcal{O} &= \bigl\langle \log P(\mathcal{L},\mathcal{R})\bigr\rangle_{Q(\mathcal{L},\mathcal{R})} - \bigl\langle \log P(\mathcal{L},\mathcal{R})\bigr\rangle_{P(\mathcal{L},\mathcal{R})} \\
&= \bigl\langle \log P(\mathcal{L},\mathcal{R})\bigr\rangle_{Q(\mathcal{L},\mathcal{R})} + \bigl\langle \log \frac{Q(\mathcal{L},\mathcal{R})}{Q(\mathcal{L},\mathcal{R})}\bigr\rangle_{Q(\mathcal{L},\mathcal{R})} - \bigl\langle \log P(\mathcal{L},\mathcal{R})\bigr\rangle_{P(\mathcal{L},\mathcal{R})} \\
&= \bigl\langle \log \frac{P(\mathcal{L},\mathcal{R})}{Q(\mathcal{L},\mathcal{R})}\bigr\rangle_{Q(\mathcal{L},\mathcal{R})} + \bigl\langle \log Q(\mathcal{L},\mathcal{R})\bigr\rangle_{Q(\mathcal{L},\mathcal{R})} - \bigl\langle \log P(\mathcal{L},\mathcal{R})\bigr\rangle_{P(\mathcal{L},\mathcal{R})} \\
&= -\operatorname{KL}(Q(\mathcal{L},\mathcal{R}) \mid\mid P(\mathcal{L},\mathcal{R})) + \bigl\langle \log Q(\mathcal{L},\mathcal{R})\bigr\rangle_{Q(\mathcal{L},\mathcal{R})} - \bigl\langle \log P(\mathcal{L},\mathcal{R})\bigr\rangle_{P(\mathcal{L},\mathcal{R})}.
\end{aligned} \tag{D.4}$$

The last line of (D.4) shows that $\mathcal{O}$ can be decomposed into three terms: negative KL divergence between Q and P and two entropy terms. The first term is equal to zero if and only if $Q(\mathcal{L},\mathcal{R}) = P(\mathcal{L},\mathcal{R})$ (the sampler is sampling from the correct distribution). Also, if the sampler is correct, the two other terms cancel. Thus, when the sampler is implemented correctly and it draws samples from the correct partition posterior, $\mathcal{O}$ becomes zero. However, the opposite is not true: we can not be sure that if $\mathcal{O} = 0$, then the sampler is correct. The following example illustrates this issue.

Let us assume that $P(\mathcal{L},\mathcal{R})$ is fixed, from this follows that $P(\mathcal{L} \mid \mathcal{R})$ and $P(\mathcal{R})$ are also fixed. The only thing that we allow to change in this example is $Q(\mathcal{L} \mid \mathcal{R})$. We can show that it is possible to define $Q(\mathcal{L} \mid \mathcal{R}) \neq P(\mathcal{L} \mid \mathcal{R})$ so that $\mathcal{O}$ becomes zero. For every $\mathcal{R}$, there may exist such a partition $\mathcal{L}_\mathcal{R}$ that the partition posterior evaluated at $\mathcal{L}_\mathcal{R}$ will be equal to the expected value of the posterior:

$$\bigl\langle \log P(\mathcal{L} \mid \mathcal{R})\bigr\rangle_{P(\mathcal{L}\mid\mathcal{R})} = \log P(\mathcal{L}_\mathcal{R} \mid \mathcal{R}). \tag{D.5}$$

Let us assume that (D.5) is true for the model under consideration. Then, if we define $Q(\mathcal{L} \mid \mathcal{R})$ as a collapsed distribution with the only possible value $\mathcal{L} = \mathcal{L}_\mathcal{R}$ for every $\mathcal{R}$, $\mathcal{O}$

becomes:

$$\begin{aligned}\mathcal{O} &= \big\langle \log P(\mathcal{L} \mid \mathcal{R}) \big\rangle_{Q(\mathcal{L}|\mathcal{R})P(\mathcal{R})} - \big\langle \log P(\mathcal{L} \mid \mathcal{R}) \big\rangle_{P(\mathcal{L}|\mathcal{R})P(\mathcal{R})} \\ &= \big\langle \log P(\mathcal{L}_{\mathcal{R}} \mid \mathcal{R}) \big\rangle_{P(\mathcal{R})} - \big\langle \big\langle \log P(\mathcal{L} \mid \mathcal{R}) \big\rangle_{P(\mathcal{L}|\mathcal{R})} \big\rangle_{P(\mathcal{R})} \\ &= \big\langle \log P(\mathcal{L}_{\mathcal{R}} \mid \mathcal{R}) - \big\langle \log P(\mathcal{L} \mid \mathcal{R}) \big\rangle_{P(\mathcal{L}|\mathcal{R})} \big\rangle_{P(\mathcal{R})}.\end{aligned} \tag{D.6}$$

By taking into account (D.5), we notice that for $Q(\mathcal{L} \mid \mathcal{R})$ defined in this way $\mathcal{O}$ becomes zero.

By this example, we have shown that $\mathcal{O}$ does not define a proper metric. However, it still possesses some useful properties that we can utilize. First, as shown above, there is a convenient way to approximate $\mathcal{O}$. Second, we know that if the value of $\mathcal{O}$ is large, then the sampler is definitely sampling from the wrong distribution either because it was not implemented correctly or because it did not warm up yet. Finally, if we want to compare two samplers using $\mathcal{O}$, we should trust more the one for which its value is lower as it is more likely to be correct. On the other hand, we have to notice that using $\mathcal{O}$ to train the parameters of the sampler automatically would probably lead to a degenerate solution similar to the example above; thus, one should not use $\mathcal{O}$ for this purpose.

Finally, before moving to the experiments using $\mathcal{O}$, let us notice that (D.1) can be considered as a degenerate case of Integral Probability Metric (IPM) [Sriperumbudur et al., 2009]. IPM is given by:

$$D(Q, P \mid \mathcal{R}) = \sup_{f \in \mathcal{F}} \big[\big\langle f(\mathcal{L}) \big\rangle_{Q(\mathcal{L}|\mathcal{R})} - \big\langle f(\mathcal{L}) \big\rangle_{P(\mathcal{L}|\mathcal{R})} \big], \tag{D.7}$$

where the supremum is taken with respect to the functions f belonging to the set $\mathcal{F}$. Set $\mathcal{F}$ should be chosen so that it is rich enough to observe when Q is different from P. On the other hand, it has to be restrictive enough to keep the computation of IPM feasible and prevent the metric from growing arbitrarily large. Finally, it is desirable for $\mathcal{F}$ to consist of functions that are smooth enough not to select individual data points. Otherwise, two distributions might look different only because function f picked up different data points for two of them. Depending on the definition of $\mathcal{F}$, different flavors of IPM can be introduced. For example, Maximum Mean Discrepancy (MMD) [Sriperumbudur et al., 2009,Rossi, 2015] is an example of IPM with $\mathcal{F}$ defined as Reproducing Kernel Hilbert Space (RKHS).

Comparing (D.1) and (D.7), one can notice that $\mathcal{O} = \big\langle D(Q, P \mid \mathcal{R}) \big\rangle_{P(\mathcal{R})}$ is a special case of IPM when the function set $\mathcal{F}$ consists of a single element $f(x) = \log P(x \mid \mathcal{R})$.

D.2.1 Using $\mathcal{O}$ to compare samplers for the true model

We track approximate $\mathcal{O}$ to compare Gibbs sampling, Split-Merge, and SDDS. Also, we include the combination of SDDS and Gibbs Sampling as was proposed by the authors of SDDS. To do the visual comparison of the algorithms, we do the following:

We create 20 independent datasets of 1000 points. First, we generate 20 different HT-PLDA models: matrix $\mathbf{W}$ in each case is the identity matrix with random noise added to its diagonal elements; elements of factor loading matrix $\mathbf{F}$ are sampled from a normal distribution with zero mean and variance set to 3, degrees of freedom parameter ν is set to 2 in all cases. Then, we sample 20 correct partitions $\mathcal{L}_i$ from CRP prior with fixed parameters. For each partition, we sample the necessary number of hidden speaker variables from a standard normal distribution and 1000 precision scaling factors α_{ij} from Gamma distribution with both shape and rate parameters set to $\nu/2 = 1$. Then, for each partition, we select one of the HT-PLDA models, sample noise variables $\boldsymbol{\eta}_{ij}$ from a normal distribution with precision $\alpha_{ij}\mathbf{W}$ and generate the observations with equation (3.13).

We run four sampling algorithms for each of the generated datasets; each time, the correct model parameters are used when sampling. The initial state of the sampler (initial partition) is just a random partition of the dataset, and it is the same for each sampling method. We compute δ (see (D.3)) for each 100-th sample generated by each method. Then, δ values are plotted, and the results are shown in Figure D.3. The upper half of the figure shows δ graphs for each dataset separately. Different colors of the graphs correspond to different sampling methods. The lower half of the figure shows the average of δ across different sets $\bar{\delta}$ for each algorithm, i.e., it shows the approximate $\mathcal{O}$ as defined in (D.2). The horizontal axis in both cases corresponds to the number of steps done by the sampling algorithm. For GS (and SDDS with GS), we take a sample of a single label as a single sampling step, i.e., to go over the whole dataset, GS has to run at least 1000 steps. Notice that the rejected proposal in SDDS and SM also counts as one step.

In the beginning, all sampling methods start from the same values of δ (this is because of the fixed initial partition used for different methods). The average δ across datasets is far below zero for all of the algorithms; this is because the random partitions we use to initialize the samplers are highly unlikely for the model with given parameters, i.e., the algorithms have to warm up before they start sampling from the correct distribution. Then, values of $\bar{\delta}$ are growing for all four algorithms. Notice, however, that the growth rate for Gibbs Sampling, SDDS, and their combination is much higher than for the Split-Merge. SM still does not pass the warm-up period by the time other methods presumably sample from the correct distribution. Steps seen on the graphs of individual datasets in the case of SM come from the fact that most of the samples proposed by SM are getting rejected, and consequently, the value of δ stays the same for a long time. And only eventually, the proposed sample is good enough to pass the acceptance test. Then, we see a rapid change in the value of δ.

Looking at the other three methods, we see that SDDS provides the fastest growth of $\bar{\delta}$. On the other hand, two other methods slightly outperform it in the long run. That is, after some time of running the samplers, GS and SDDS with GS have a value of $\bar{\delta}$ closer to zero and still improving, while SDDS alone seems to get stuck. We explain it by the inability of SDDS to make fine adjustments, as was discussed above.

As expected, SDDS with GS combines the advantages of both methods. The warm-up stage for it passes almost as quickly as for SDDS, and it does not get trapped in a local optimum as SDDS does. From its graph, we see that δ metric for SDDS with GS follows the desired behavior, and we can assume that this method indeed samples from the correct distribution and the burn-in stage for it passes reasonably fast. Hence, we adopt this algorithm for the experiments with the real data.

D.2.2 Tracking the sampling for an unknown model

Suppose a sampler is known to draw samples from the correct posterior. Then, for the real dataset and the warmed-up sampler, the significant deviations of $\bar{\delta}$ from zero must come from the fact that the parameters of the model are not the true ones. Figure D.4 corresponds to the experiment emulating this scenario. We run SDDS-GS sampling for the same datasets as used in Figure D.3. This time, instead of using the correct model parameters for the sampler, we use different ones. That is, the sampling is effectively done from a distribution different from the one the data were generated from. We plot the progress of $\bar{\delta}$ for the

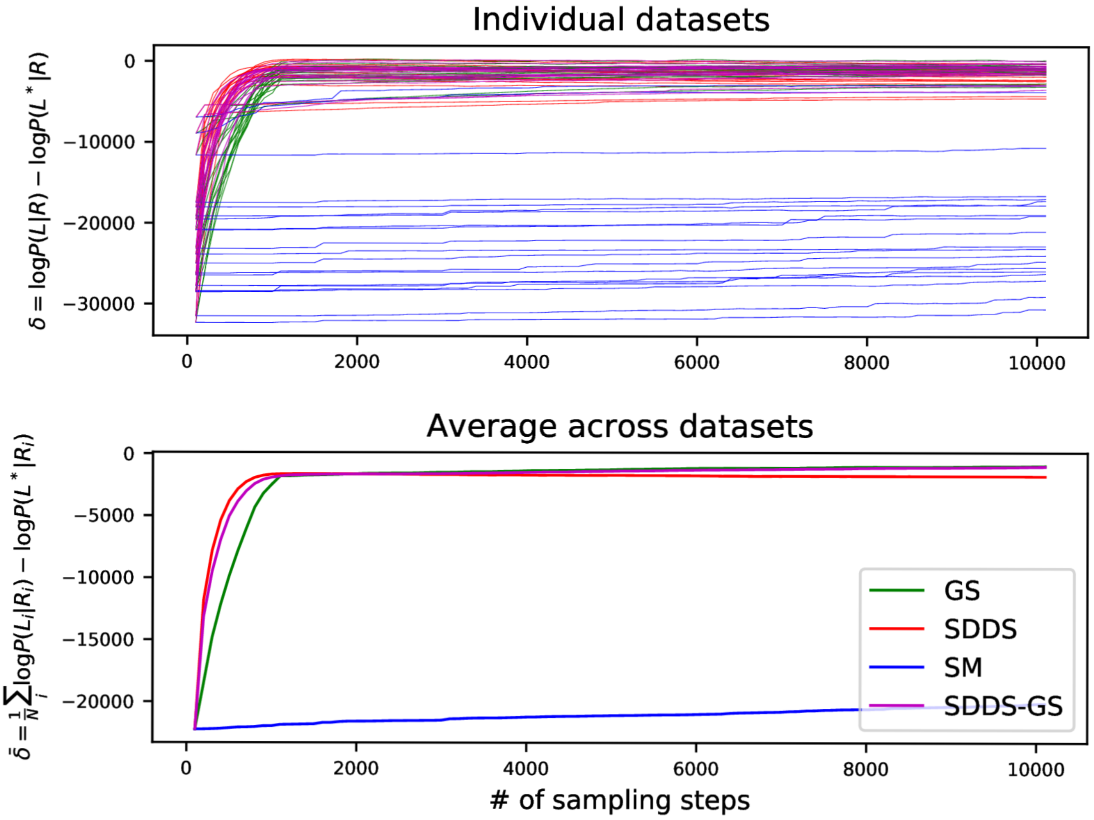

Figure D.3: Progress of δ for different datasets and sampling methods (Gibbs sampling, Smart-Dumb/Dumb-Smart, Split-Merge, and a combination of SDDS and GS). The results per dataset are shown in the upper plot. The averaged δ across different sets are shown in the lower plot. At the beginning of sampling, values of δ are much lower than zero, indicating that the burn-in stage is not passed yet for any of the methods (true partition has a higher posterior than accepted samples). By the end of sampling, all methods except Split-Merge produce samples resulting in δ closer to zero, suggesting that they are sampling from the true posterior. The average acceptance rate for SDDS sampling is 10.27%, and for SM is 3.06%. The acceptance rate for GS is 100% by definition.

sampling with the wrong model along with the graph from Figure D.3 (using the correct parameters). Comparing these, we see that in both cases, $\bar{\delta}$ approaches some fixed value after a burn-in stage. For the model with incorrect parameters, $\bar{\delta}$ converges to a value higher than zero (shown as a thin black line for reference). In contrast, as we saw before, for the correct parameters, it approaches zero.

The experiment above shows that when the model parameters are not known exactly, the sampled partitions are more likely than the true one.

Logically, minimizing δ as a function of θ should allow us to find such parameters that the correct partition appears among the likely ones. The gradient of δ is:

$$\begin{aligned} \nabla_\theta \delta(\theta) &= \nabla_\theta \log P(\mathcal{L} \mid \mathcal{R}, \theta) - \nabla_\theta \log P(\mathcal{L}^* \mid \mathcal{R}, \theta) \\ &= \nabla_\theta \log P(\mathcal{R} \mid \mathcal{L}, \theta) - \nabla_\theta \log P(\mathcal{R} \mid \mathcal{L}^*, \theta). \end{aligned} \tag{D.8}$$

Comparing (D.8) to (5.18), we see that the gradient of δ is almost the same as the

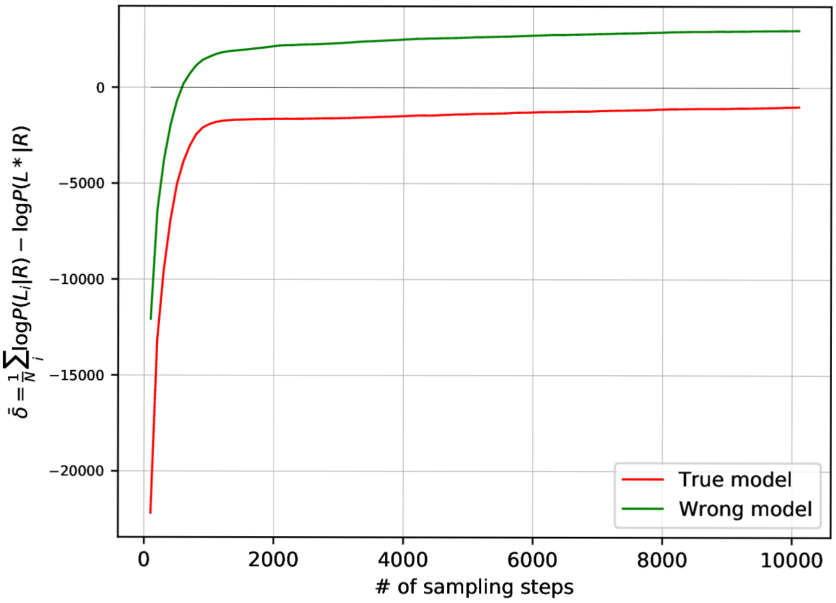

Figure D.4: Comparison of $\bar{\delta}$ when sampling from the true model generating the ground truth labels (red line) and from a different model (green line). When the wrong model is used, the true partition has a lower likelihood than the sampled ones; consequently, $\bar{\delta}$ converges to a value higher than zero. For the true model, the reference partition is among the likely ones, and $\bar{\delta}$ approaches zero.

contrastive divergence approximation of the log-posterior gradient with a single contrastive sample. The only difference between (D.8) and (5.18) is the sign: minimizing $\delta(\theta)$ is equivalent to maximizing $\log P(\mathcal{L}^* \mid \mathcal{R}, \theta)$ with contrastive divergence.

Appendix E

Finding optimal settings of SDDS sampler for training HT-PLDA

Here, we present the experiments aimed at showing how to set the parameters of the SDDS sampler for training the HT-PLDA model with maximum posterior of the correct partition objective (5.4).

The setup for these experiments is the following: we use a Gaussian PLDA model trained on VoxCat-S (see Section 2.5.1) set as an initialization for the discriminative training of HT-PLDA with contrastive divergence. The embeddings we use are x-vectors of Section 2.4.2. The network was trained on the VoxCeleb2 dataset. The embeddings are centered, and their dimensionality is reduced to 300 by LDA (with speaker classes) prior to training models. No length normalization is done.

We track the training process on data from VoxCat-CV set (as described in Section 2.5.1) in the following way: after each parameter update, we draw N samples from the partition posterior of the cross-validation set, compute the value of δ_i as defined by (D.3) for each sample $\mathcal{L}_i$ from the posterior. Finally, we average δ_i across sampled partitions:

$$\bar{\delta} = \frac{1}{N}\sum_{i=1}^{N}\delta(\mathcal{L}_i, \mathcal{L}^*, \theta). \tag{E.1}$$

As described in Section D.1, lower absolute value of $\bar{\delta}$ corresponds to a posterior distribution where the true partition of the data is in a high-probability region. In other words, if we see that $|\bar{\delta}|$ on a cross-validation set decreases as the training progresses, it means that the model becomes better suited for the data and assigns a higher probability to the correct partition.

Sampling from the posterior over cross-validation set partitions is done by running a single chain of SDDS and collecting 50 samples from it. The chain is initialized with the correct partition. The warm-up stage is 100 samples, and we discard 100 samples between those that are used. The number of Gibbs sampling steps after each SDDS step is set in such a way that in the span of the sampling process, the label for each point has been sampled at least three times. The parameters of SDDS on the cross-validation set are kept fixed and stay the same in all of the experiments described below.

E.1 Initialization of SDDS Markov chain

First, we investigate how the initialization of the Markov chain affects the training with the approximate posterior of the correct partition objective. Theoretically, the initialization of the chain does not matter. The sampling algorithm must sample from the correct distribution after a sufficient number of steps are done. In other words, after the burn-in stage, we should be guaranteed to receive a sample from the desired posterior starting from any initial partition. However, the length of this burn-in could be very different for different initializations. For a bad initial partition, it might take a prohibitively long time before a sample from the desired distribution could be collected. From a practical point of view, it is desirable to select the initial partition as close as possible to the high probability region of the posterior so that the burn-in does not take too long.

Here, we experiment with two strategies to initialize the SDDS Markov chain. We train two HT-PLDA models. Both of them are initialized from the same G-PLDA. Each time, we run 30 iterations (30 parameter updates) of the contrastive divergence. At every iteration, we run a single SDDS chain and use 50 samples from this chain to approximate the gradient. These samples are separated by 100 SDDS accepted states, i.e., we use every 101st accepted sample of SDDS[1]. Also, the burn-in stage is fixed to 100 samples. We set the number of Gibbs sampling updates after each SDDS step so that GS passes through all the data at least three times during the sampling process. The difference between the two training strategies is how the SDDS sampler is initialized: in the first case, we initialize the chain from the true partition after each parameter update. For the second model, we use the last accepted sample from the previous iteration as the initialization for the new one. True partition is used to initialize the chain only at the very first iteration. This strategy is known as *Persistent Contrastive Divergence* (PCD) [Tieleman, 2008]. The motivation for using PCD is that parameters of the model for neighboring iterations are similar; consequently, posterior over partitions should not be too different. And the likely sample from the previous iteration should be in a high-probability region of the updated partition posterior.

Figure E.1 shows the progress of $\bar{\delta}$ computed on the cross-validation set for both initialization strategies. At the first iteration, the two approaches are still the same, as both of them use the true partition for the first chain initialization. After that, for several iterations, both approaches perform similarly, but eventually, PCD outperforms the other approach. For the following experiments, we adopt the PCD strategy as we have seen it to be more efficient.

E.2 Multiple SDDS chains

In the previous experiment, we used a single chain to collect the samples from the partition posterior. We saw that it is effective in terms of improving the objective on the cross-validation data. However, sampling in this way requires a large number of accepted states to be discarded for the collected samples to be independent of each other. Also, there is no possibility of parallelizing the sampling when only one chain is used. At the same time,

[1] According to MCMC literature [Brooks et al., 2011], it might be unnecessary to sub-sample MCMC chain when computing the expectations w.r.t. the sampled distribution. The fact that the samples are correlated is not crucial for such estimation. However, in our case, we compute the gradient $\nabla_\theta P(\mathcal{R} \mid \mathcal{L}_i, \theta)$ for each sampled partition $\mathcal{L}_i$. Computing the gradient is an expensive operation, so it makes computational sense to do it only for sufficiently different partitions. Thus, here and in the following experiments, we sub-sample the chain.

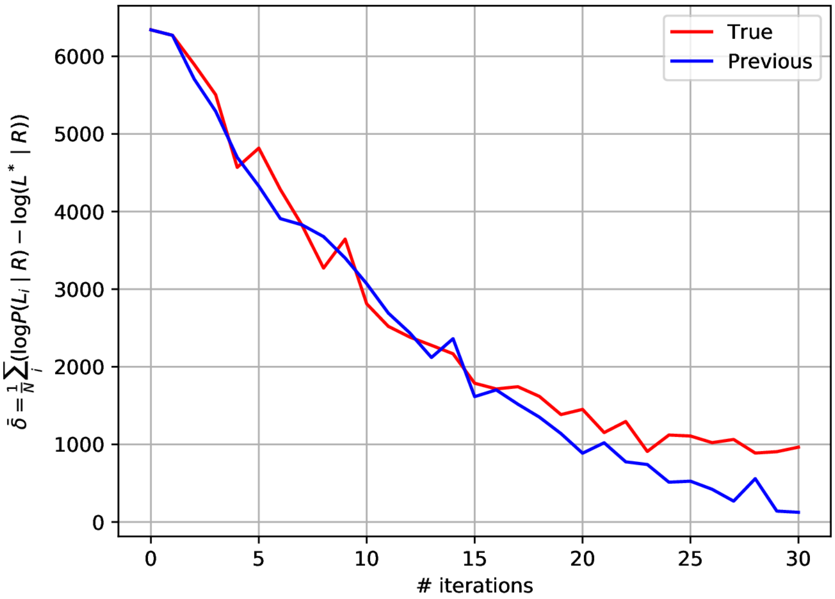

Figure E.1: Comparison of $\bar{\delta}$ on cross-validation set when initializing the sampling chain from the true partition at each training iteration (red line) or from the last sample from the previous iteration (blue line) - Persistent Contrastive Divergence.

running several Markov chains in parallel would allow us to collect the needed number of samples faster. Also, there is no need to run each chain for as long as the single one. We can keep only one sample from each of them. Here, we compare two such strategies. We check the training progress of the PCD with a single chain from the previous section. There, we use 50 samples from a single chain separated by 100 accepted states to compute the gradient. Alternatively, we approximate the gradient using 50 samples, each of them generated by an individual Markov chain. For each of these chains, we run 200 SDDS steps, and the last accepted state is used as a sample from the posterior. We use a higher number of burn-in steps to ensure that the only sample we use from each chain comes from the true posterior. When many samples from the same chain are used, we believe that even if the first few samples come from the burn-in period, there will be others that will overweight them. As in the case of the single chain, the many-chains approach also follows the PCD scheme, i.e., at each iteration, the chain is initialized from the last accepted state of the corresponding chain from the previous iteration. Notice that even though the approach with multiple chains requires more computations compared to a single chain, it allows collecting samples in parallel, thus providing faster training.

Figure E.2 demonstrates the comparison between these two strategies for collecting the samples. The graph shows that both training strategies provide comparable results for the whole training time.

As there is no performance advantage of the “one chain” approach, we proceed with multiple sampling chains in parallel in the following experiments.

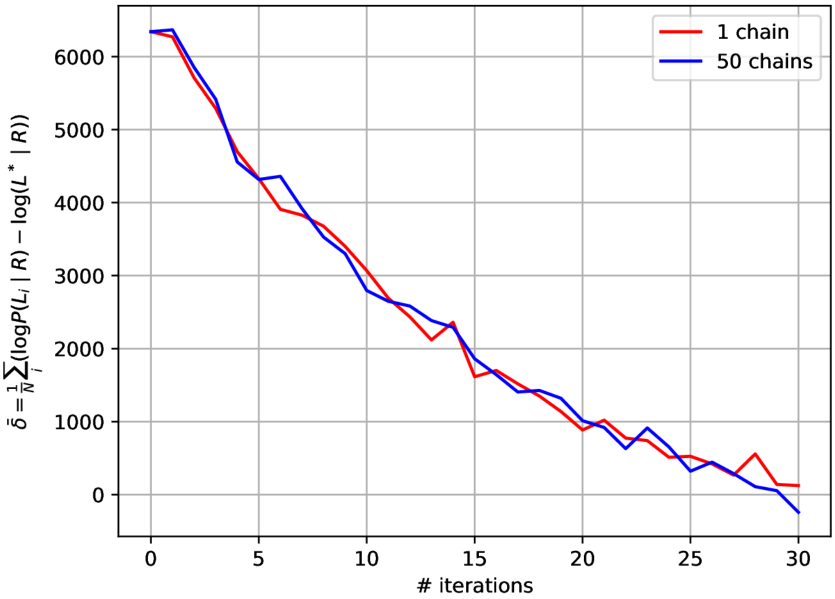

Figure E.2: Comparison of $\bar{\delta}$ on the cross-validation set when running a single sampling chain or running 50 independent chains for a shorter time. The number of samples to approximate the gradient is the same in both cases.

Gibbs Sampling steps to interleave SDDS moves

The next question is what is the optimal number of Gibbs sampling steps to insert between SDDS iterations. The original algorithm [Wang and Russell, 2015] proposes to interleave each SDDS iteration with one full GS iteration. Unfortunately, it would not be feasible in practice, as one GS iteration would take too long time to perform. Instead, we perform GS steps for some fixed number of points at a time. Moreover, we further simplify the correct GS scheme by simultaneously sampling all of the labels as discussed in Section 5.4.4. For all systems, we fix the training strategy to the PCD with samples collected from 50 independent Markov chains. We compare three settings of the number of GS updates per iteration: we set the number of GS steps so that all of the data are visited by the Gibbs sampler at least k times while running all SDDS iterations, and we vary k to be 1, 3, or 5 ($k = 3$ was the default value in the previous experiments). In all three cases, the number of SDDS steps is fixed at 200 per chain. As we have approximately 50k utterances in the training set, the number of points to sample the label after each SDDS step is around 200, 600, and 1000 for $k = 1, 3, 5$. The results are shown in Figure E.3.

As one can conclude from the plots, doing 1, 3, or 5 passes through the data with Gibbs sampling seems to provide comparable performance. As fewer GS steps lead to a faster sampling and we do not observe any significant benefit from increasing this number, we will set the number of GS steps between SDDS steps so that each point is visited by GS at least once during the sampling.

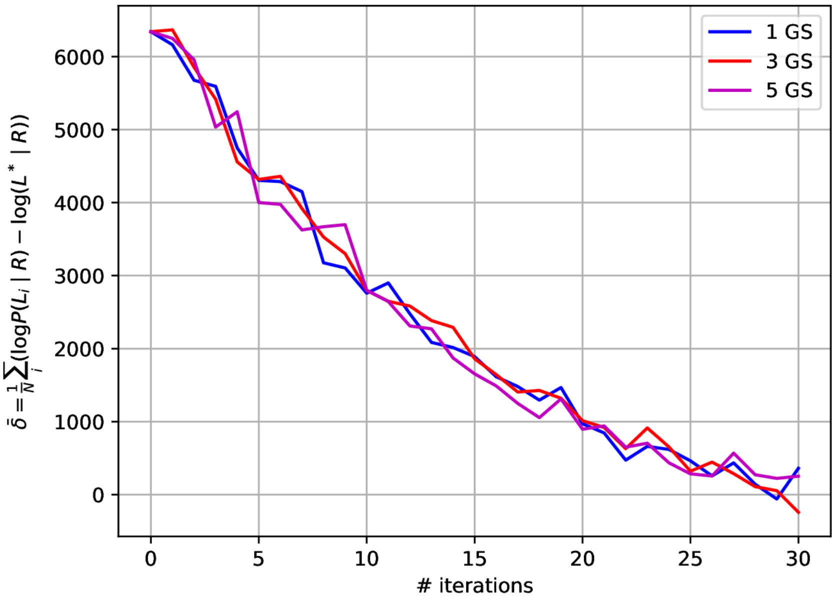

Figure E.3: Comparison of $\bar{\delta}$ on the cross-validation set when using different numbers of Gibbs samplings in between SDDS steps.

E.3 Number of samples

As seen from (5.18), the gradient of the log-posterior of the training data is approximated using gradients of n samples from the partition posterior. The question arises, how large should we set n. Theoretically, when n approaches infinity, the approximation becomes exact; it might seem logical to collect as many samples as possible for better training. However, as was mentioned earlier, the gradient approximation has the following effect: it pushes up the likelihood of the correct partition and lowers the likelihoods of the sampled partitions, which, with a high probability, are different from the correct one at least for the model at the beginning of the training. In other words, there is a single positive example and several likely negative examples. And the task of the training is to make the positive example more likely and negative examples less likely. Then, even a small number of negative samples should be sufficient to achieve the same behavior.

In our experiment, we compare the training when 50 samples from 50 independent Markov chains were used with the case when just a single sample is used for the gradient approximation. Notice that the second strategy corresponds to minimizing δ as was shown in (D.8). In both cases, we use the PCD. Figure E.4 displays the progress of the cross-validation performance for these two training strategies. As the plots indicate, using more samples results in a more stable training as could be expected. Eventually, using 50 samples to approximate the gradient leads to a slightly better performance in terms of $\bar{\delta}$ on the cross-validation set. However, using just a single sample allows for a significant decrease in the needed computations and consequent speed-up of the training process with a marginal loss in the final performance. Thus, in future experiments, we always use just one sample for gradient approximation.

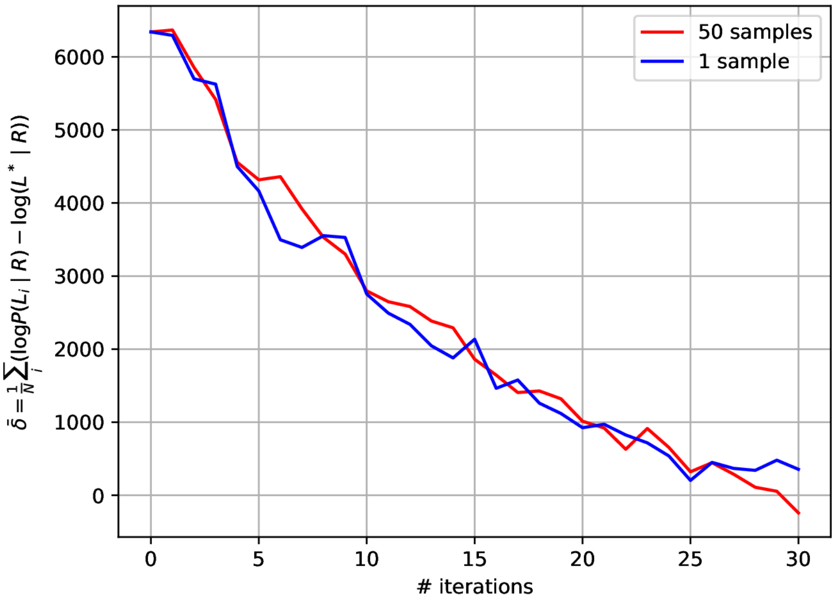

Figure E.4: Comparison of $\bar{\delta}$ on the cross-validation set when using a single sample or 50 independent samples to approximate the gradient.

Appendix F

Efficient computation of partitioning of tuples objective

Here, we describe how to efficiently compute objective (5.36) when training HT-PLDA or probabilistic embedding models. The objective is:

$$E(\theta) = \sum_{(\mathcal{R}_t, \mathcal{L}_t^*) \in \mathcal{T}} - \log \frac{P(\mathcal{R}_t \mid \mathcal{L}_t^*, \theta) P(\mathcal{L}_t^*)}{\sum_{\mathcal{L}_t'} P(\mathcal{R}_t \mid \mathcal{L}_t', \theta) P(\mathcal{L}_t')}, \tag{F.1}$$

where $\mathcal{R}_t$, $\mathcal{L}_t^*$ are t-th training example: set of several recordings (set of embeddings in case of HT-PLDA, set of speech segments, i.e., matrices of acoustic features for probabilistic embeddings) and its correct partition into speaker clusters, respectively. $P(\mathcal{L})$ is the prior probability of a given partition. Each component in the sum (F.1) is the negative log-posterior probability of the correct partition of the t-th example.

We approach the evaluation of (F.1) by the following procedure:

- Consider all of the training examples that have the same size (number of elements to partition) N, assume there are T of them. If training set $\mathcal{T}$ contains examples of different sizes, perform the following for each of them and sum the results.
- Compute sparse N-by-$(2^N - 1)$ matrix $\mathbf{Q}$ with 0/1 entries. Rows of the matrix correspond to individual points in the training example. Columns correspond to all possible subsets of points that can be created out of a set of size N. This matrix can be used to efficiently perform summations to compute statistics (3.34) (or (6.9) for probabilistic embeddings) for all possible subsets of N points.

$$\hat{\mathbf{b}} = \mathbf{bQ}, \tag{F.2}$$

$$\hat{\mathbf{A}} = \mathbf{AQ}. \tag{F.3}$$

Above, $\mathbf{b}$ is a matrix of size $T \times N$, each row of this matrix corresponds to one training example $t = 1 \ldots T$, i-th element of the t-th row is a scalar statistic b_{ti} computed with (3.22) (or with the second equation of (6.9)) for the i-th of N individual points $\mathbf{r}_{ti}$ from $\mathcal{R}_t$. $\mathbf{A}$ is a 3-dimensional tensor of shape $T \times d \times N$. Along the first dimension of $\mathbf{A}$ we stack matrices containing N d-dimensional column vector statistics $\mathbf{a}_i$ computed with (3.5) (or with the first equation of (6.9)) for the point $\mathbf{r}_{ti}$. Then, $\hat{\mathbf{b}}$ and $\hat{\mathbf{A}}$ contain statistics for each of $(2^N - 1)$ possible subsets of N points for each of T training examples. From these, one can compute $\log P(\mathcal{R}_{ti} \mid H_s)$ for every possible subset of $\mathcal{R}_t$ using (3.27) (or (6.11)). The result of this operation is T-by-$(2^N - 1)$ matrix $\mathbf{P}$.

- Compute sparse B_N-by-$(2^N - 1)$ matrix $\mathbf{U}$, with 0/1 entries. Rows of this matrix correspond to possible partitions of N points (N-th Bell number B_N of them), and columns correspond to the possible subsets of N elements. Thus, each matrix element indicates whether a particular subset is part of a particular partition. This matrix can be used to efficiently accumulate $\log P(\mathcal{R}_t \mid \mathcal{L})$, for every possible value of $\mathcal{L}$: $\mathbf{PU}'$ is a matrix of size $T \times B_N$ containing log-likelihoods for all possible partitions of N points into speaker clusters for each training example $t = 1, \ldots T$.

- Compute a vector of log-priors over partitions of N points of length B_N, let us denote it $\mathbf{p}$. Elements of $\mathbf{p}$ are $\log P(\mathcal{L}^i)$.

- By adding the vector of log-priors $\mathbf{p}$ to each row of matrix $\mathbf{PU}'$ containing log-likelihoods and passing it through a softmax function, we obtain the matrix of log-posteriors for every possible partition for every training example t. Selecting those elements of the matrix that correspond to the correct partitions of each example, summing them, and multiplying the result by -1 gives us the objective F.1.