POLITECNICO DI TORINO

SCUOLA DI DOTTORATO

Dottorato in Ingegneria Informatica e dei Sistemi – XXIV ciclo

Tesi di Dottorato

Speaker and Language Recognition Techniques

Sandro Cumani

Tutore
prof. Pietro Laface

Coordinatore del corso di dottorato
prof. Pietro Laface

Il Signore disse: "Ecco, essi sono un solo popolo e hanno tutti una lingua sola; questo è l'inizio della loro opera e ora quanto avranno in progetto di fare non sarà loro possibile. Scendiamo dunque e confondiamo la loro lingua, perché non comprendano più l'uno la lingua dell'altro".

Bibbia, Libro della Genesi 11, 1-9

Poi disse a me: "Elli stessi s'accusa;
questi è Nembrotto per lo cui mal coto
pur un linguaggio nel mondo non s'usa.

Lasciànlo stare e non parliamo a vòto;
ché così è a lui ciascun linguaggio
come 'l suo ad altrui, ch'a nullo è noto".

Dante Alighieri, La Divina Commedia - Inferno: C. XXXI

Oh freddled gruntbuggly thy micturations are to me
As plurdled gabbleblotchits on a lurgid bee.
Groop I implore thee, my foonting turlingdromes.
And hooptiously drangle me with crinkly bindlewurdles,
Or I will rend thee in the gobberwarts with my blurglecruncheon, see if I don't!

Prostetnic Vogon Jeltz, Douglas Adams, The Hitchiker's Guide to the Galaxy

Summary

In this work we give an overview of different state–of–the–art speaker and language recognition systems. We analyze some techniques to extract and model features from the acoustic signal and to model the speech content by means of phonetic decoding. We then present state–of–the–art generative systems based on latent variable models and discriminative techniques based on Support Vector Machines.

We also present the author's contributions to the field. These contributions cover the different topics presented in this work. First we propose an improvement to Neural Network training for speech decoding which is based on the use of General Purpose Graphic Processing Units computational framework. We also propose adaptations of latent variable models developed for speaker recognition to the field of language identification. A novel technique which enhances the generation of low–dimensional utterance representations for speaker verification is also presented. Finally, we give a detailed analysis of different training algorithms for SVM–based speaker verification and we propose a novel discriminative framework for speaker verification, the Pairwise SVM approach, which allows for fast utterance testing and allows to achieve very good recognition performance.

Acknowledgements

I would like to thank my parents, to whom I dedicate this work.

I would also like to thank professor Pietro Laface for his support during these last three years, and all the people I had the opportunity to work with:

- The people from Brno University of Technology, in particular Oldřich Plchot, Ondřej Glembek, Pavel Matějka, Lukáš Burget and Mehdi Soufifar, with whom I had the pleasure to collaborate sharing many interesting thoughts, ideas and great moments, Honza Cernocký, for hosting me twice at BUT, and all the other people from Brno who made me have a great and profitable time there.
- The guys from Agnition, Niko Brümmer and Edward de Villiers, for all the discussions about generative and discriminative models and many memorable moments in Brno,
- All the other people who participated to the Bosaris workshop.
- The people from Loquendo, in particular Daniele Colibro and Claudio Vair, for the profitable collaboration in the last three years.
- My former collegues, Stefano Scanzio and Fabio Castaldo, with whom I had the opportunity to profitably work.
- All the people who I had the pleasure to interact with during workshops and conferences.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The growth of possible applications and the increase of computational power of processors has produced in the last years an increasing interest of both scientific community, industries and governments in automatic systems able to extract significant information from spoken utterances. Research has developed on three main branches, namely speech, speaker and language recognition.
Speech recognition is involved with the creation of automatic transcriptions of speech, whose applications include device control through spoken commands or virtual typing.
Speaker recognition can be summarized as the creation of automatic systems which are able to answer questions regarding the identity of the person who is talking. Among speaker recognition applications we can cite authentication procedures, audio indexing, forensic activities.
Language recognition deals with identification of the language (e.g. English, Italian) used in a given utterance. This field includes applications in multilingual answering systems or as front–end for language–dependent speech recognizers.

The goal of this work is to present the author's contributions to speaker verification and language recognition and, at the same time, offer an overview of state–of–the–art technologies for these fields.

1.1 Speaker Recognition

The goal of speaker recognition systems is to automatically make inferences about the identity of the speaker of a test utterance. Two main branches belong to speaker recognition, namely speaker identification and speaker verification.

Speaker identification can be described as a multiclass classification problem. Given a test utterance, we want to identify which, among a set of enrollment speakers, is the speaker of that utterance. The assumption of whether the test speaker

belongs or not to the enrollment set gives places to two different classification problems, closed–set speaker identification and open–set speaker identification, the latter being more difficult.

Speaker verification requires a system to answer whether a test utterance belongs to a given speaker, or, equivalently, whether a set of recordings (e.g. one enrollment and one test segment) belong to the same speaker. While these two formulations are very similar, they correspond to two completely different discriminative speaker verification approaches.

1.2 Language Recognition

Similar to speaker identification, language recognition can be described as a multi-class classification problem where the goal is to classify an utterance according to its language. Again we can have both open–set and closed–set problems, depending on whether the test utterance is known to belong to given set of languages.

In this work we focus on the speaker verification problem and on closed–set language identification. These tasks are characterized by similar problems and share some modeling tools such as Gaussian Mixture Models, Factor Analysis or Support Vector Machines. The systems described in this work are characterized by a set of common modules, namely feature extraction, creation of models (e.g. speaker or language models, background models), test utterance scoring, normalization and calibration of scores.

This work details the contributions of the author to the state–of–the–art, and at the same time provides an overview of different language and speaker recognition technologies related to the first three modules. While score normalization and score calibration play an important role in real applications, they fall out of the scope of this work. References and details about score normalization techniques can be found in [1, 2, 3, 4, 5], while [6] is an interesting treatise about calibration.

The outline of this work is the following.

- Chapter 2 describes the feature extraction process both for speaker recognition and language recognition systems and details some basic feature modeling techniques such as Gaussian Mixture Models, Hidden Markov Models and Neural Networks.

- Chapter 3 describes generative techniques for speaker recognition based on latent variable models.

- Discriminative techniques are the main focus of Chapter 4, where Support

Vector Machines (SVM) are introduced and applied to language identification problems.

- Chapter 5 presents two frameworks for SVM–based speaker verification.
- Experimental results regarding the author's contributions to the state–of–the–art are presented in Chapter 6.
- Conclusions are drawn in Chapter 7.

Chapter 2

Modeling the acoustic signal

In order to perform automatic speech recognition it is necessary to build mathematical models of the acoustic signal which can be used by a machine. In this chapter we analyze different techniques to transform the acoustic signal into a set of features which can be used to model the acoustic characteristics of speakers and languages. The feature extraction techniques presented in this chapter form a common front–end to many different speaker and language recognition systems.

2.1 Acoustic Features

The first section of this chapter is devoted to acoustic feature extraction, i.e. to the steps which allow transforming an analog signal into a discrete set of features suited for speaker and language recognition systems.

2.1.1 Sampling, quantization and filtering

The first step in audio analysis consists in the transformation of the analog acoustic signal into a discrete version that can be processed by a machine. This representation is obtained by sampling and quantization of the acoustic analog signal.

Time–domain sampling corresponds to a multiplication of the input signal by a sequence of impulses $\sum_k \delta(t - kt_s)$ where t_s is the sampling interval. Formally

$$y_s(k) = \sum_k [y(kt_s)\delta(t - kt_s)] \tag{2.1}$$

where $y(t)$ is the input signal and $y_s(k)$ is the sampled signal. This is equivalent, in the frequency domain, to the convolution of a sequence of impulses with the spectrum of the analog signal, which gives

$$Y_s(\omega) = \frac{1}{t_s} \sum_k Y\left(\omega + \frac{2\pi k}{t_s}\right) \tag{2.2}$$

where $Y(\omega)$ is the Fourier transform of $y(t)$ and $Y_s(\omega)$ is the Fourier transform of $y_s(t)$. In general the signal cannot be completely reconstructed due to the overlapping of replicas of the original spectrum (aliasing). However, since the human apparatus is sensitive only to frequencies lower than 4 kHz, it follows from Nyquist theorem that a sufficient sampling frequency for a speech signal corresponds to 8 kHz.

Quantization maps continuous values to discrete ones, thus it is always a lossy process. The simplest way to perform quantization consists in uniformly dividing the input range and assigning to each value the index of the corresponding interval (linear quantization). With this method, the quantization error corresponds to one least significant bit and, more important, its absolute value is constant for any given input value. The amplitude distribution of the acoustic signal signal is highly non–linear. To cope with this problem, a logarithmic quantization is usually performed, that is the logarithm of the acoustic signal is linearly quantized. In this way, the relative quantization error becomes constant. Since the logarithmic function is not defined in zero, slightly different functions are used in practice, as the μ–law (used in American communication nets) or the A–law (used in European communication nets) [7]. For telephone speech usually values are represented on 8 bits. In practice,

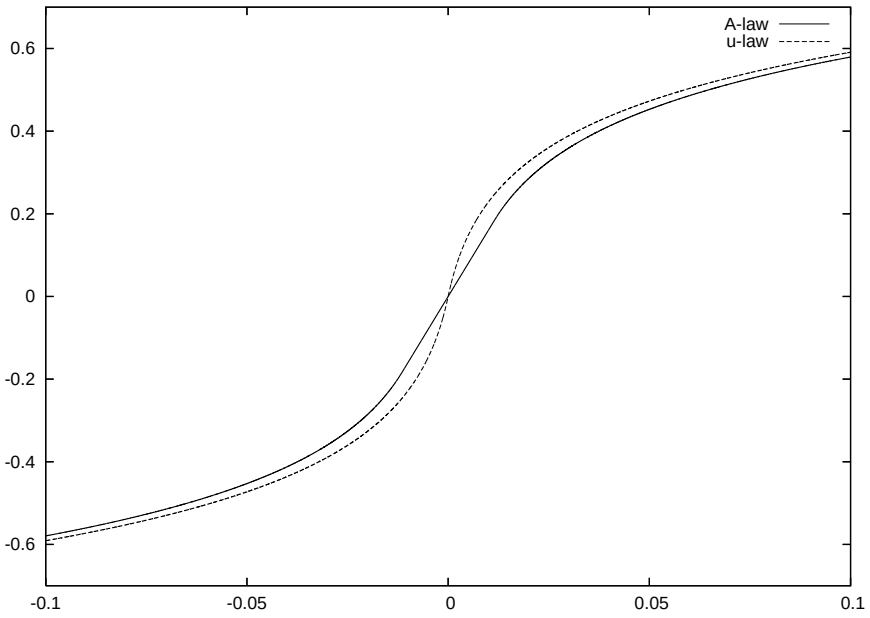


Figure 2.1: The A and μ laws

logarithmic quantization is done by performing linear quantization using a greater number of bits and then applying one of the laws in Figure 2.1 to map them to 8 bits.

Finally, in order to make the signal spectrum flatter in the given frequency band, the discrete samples are filtered by a first order pre–emphasis filter with transfer function

$$H(z) = 1 - az^{-1} \tag{2.3}$$

which is equivalent, in the time domain, to

$$\hat{Y}(k) = Y(k) - aY(k-1) \tag{2.4}$$

where a is a constant. Usually $a = 0.95$ in real applications.

2.1.2 Mel–Frequency Cepstral Coefficients

A standard representation of acoustic signal used in state–of–the–art speaker recognition systems is given by Mel–Frequency Cepstral Coefficients (MFCC) [8, 7], which provide a short–term representation of the power spectrum of the acoustic signal.

Since the acoustic signal can be considered stationary over time spans of the order of milliseconds, it can be split into frames (usually covering about 10 ms) which group together a set of samples. This operation can be described as

$$X_t(n) = \hat{Y}(M_a t + n) \quad 0 \leq n \leq N_a - 1, \quad 0 \leq t \leq T - 1 \tag{2.5}$$

where N_a is the grouping window size, M_a is the size of the shift (i.e. $M_a = N_a/2$), T is the duration in frames of the signal (the number of frames we split the signal into).

Since the grouping in frames results in a distortion of the spectrum of the samples of each frame (Gibbs phenomenon), the influence of samples near the borders of a frame is reduced, usually by means of a Hamming window

$$\hat{X}_t(n) = X_t(n)W(n) \quad 0 \leq n \leq N_a - 1, \quad 0 \leq t \leq T - 1 \tag{2.6}$$

where

$$W(n) = \begin{cases} c + (1-c)\cos\left(\frac{\pi n}{N-1} - \frac{\pi}{2}\right) & \text{if } 0 \leq n \leq N-1 \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

In real applications a typical value for c is $c = 0.54$. While this approach allows to obtain a better approximation of the original signal spectrum, it penalizes information contained in border samples of each frame. To compensate for this problem the window is shifted only by half its size. In this way, samples which are on the border of a frame are in the middle of either the previous or the next one.

The following step consists in performing the Fourier transform over the data of each frame

$$X_f(j) = \mathcal{F}\left(\hat{X}_t(n)\right)(j) \quad 0 \leq n, j \leq N_a - 1, \quad 0 \leq t, f \leq T - 1 \tag{2.8}$$

This spectrum is processed using filters which emulate the human apparatus. A simple but effective model represents the human auditory system as a filter bank. The acoustic signal spectrum is therefore divided according to frequency bands. We consider a filter bank where bands are constructed according to the Mel scale [7]. For each band the energy of corresponding samples is evaluated as

$$E_i(f) = \sum_{j=L_i}^{H_i} |X_f(j)|^2 \quad 1 \leq i \leq N_f \tag{2.9}$$

where $E_i(f)$ is the energy of the i–th band, L_i is the lower bound of the corresponding band and H_i is its higher bound. N_f is the number of bands (e.g. usually $N_f = 13$). Finally, MFCCs are computed as the Discrete Cosine Transform of the logarithm of the energy parameters $E_i(f)$

$$C_i(k) = \sum_{j=1}^{N_f} \log(E_j(k)) \cos\left[i\left(j - \frac{1}{2}\right)\frac{\pi}{N_f}\right] \quad 0 \leq i \leq N_f - 1 \tag{2.10}$$

where $C_i(k)$ denotes the i–th MFCC for frame k [7].

The total energy of the frame can be evaluated as

$$E(k) = \sum_{j=1}^{N_f} E_j(k) \tag{2.11}$$

Since the energy information contained in $C_0(k)$ is the same information given by $E(k)$, usually the cepstral parameter $C_0(k)$ is discarded. Moreover, it can be shown that cepstral parameters have decreasing variance as their indices grow, so high index parameters can be discarded, since they convey less information. Usually, the number p of cepstral parameters used ranges from $p = 12$ to $p = 24$.

In order to better model the acoustic signal, generally MFCCs are combined with their differential counterparts (depstral parameters). These parameters are evaluated through an approximation of the temporal derivative of MFCCs, for example using a polynomial expression over a given number of consecutive frames such as

$$\Delta\bar{C}_i(k) = G \sum_{j=-N}^{N} j\bar{C}_i(k-j) \quad 1 \leq i \leq p \tag{2.12}$$

where G is a gain used in order to have a similar variance between the set of cepstral and the one of depstral parameters and N is half the size of the window used to approximate the derivative. In the same way it is also possible to evaluate the differential energy as

$$\Delta E(k) = \sum_{j=-N}^{N} jE(k-j) \tag{2.13}$$

State–of–the–art systems usually also compute second–order derivatives of cepstral coefficients in a similar way. The set of cepstral parameters and their first and second order derivatives becomes then observed feature vector

$$\begin{aligned} O_t &= \{\bar{C}_1(t), \ldots, \bar{C}_p(t), \Delta\bar{C}_1(t), \ldots, \Delta\bar{C}_p(t), \Delta\Delta\bar{C}_1(t), \ldots, \Delta\Delta\bar{C}_p(t), \\ &\quad E(t), \Delta E(t), \Delta\Delta E(t)\} \end{aligned} \tag{2.14}$$

To make systems more robust, often post–processing techniques are applied to MFCCs, as for example feature warping [9]. These techniques try to compensate short–term distortions due to noise and channel mismatches.

2.1.3 Shifted Delta Coefficients

While MFCCs allow very good results in speaker recognition, language identification model performance can be improved by using Shifted Delta Cepstral (SDC) [10, 11]. SDCs allow including additional temporal information with respect to standard MFCCs and have been motivated by the success of phonotactic approaches, whose features are based on longer temporal time spans than MFCCs (Section 2.5). SDCs are specified by 4 parameters, N, d, P and k, where N is the number of cepstral coefficients for each frame, d is the size of the delay for delta computations, k is the number of blocks whose delta coefficients are concatenated in the final feature vectors and P is the time shift between consecutive blocks [10, 11]. The SDC coefficients $\Delta c_j(i,t)$ at time t are computes as

$$\Delta c_j(i,t) = \bar{C}_j(t + iP + d) - \bar{C}_j(t + iP - d) \tag{2.15}$$

where $\bar{C}_j(t)$ is the j–th MFCC coefficient for time t.

2.2 Gaussian Mixture Models

The acoustic signal can be interpreted as a piecewise stationary stochastic process. Acoustic features can be interpreted as realizations (observations) of some random variables. An effective technique to model the underlying distribution of such variables is given by Gaussian Mixture Models (GMMs) [12].

2.2.1 Gaussian Mixture Model

Given a (multivariate) random variable X with its own probability density function (pdf) $f(x)$, a Gaussian Mixture Model (GMM) can be used to approximate an estimate of $f(x)$ from a set $X_s = \{x_1, \ldots, x_n\}$ of samples of X. A GMM [12] is

a weighted sum (mixture) of a set of m (multivariate) normal distributions of the form

$$P(x|M) = \sum_{i=1}^{m} w_i \mathcal{N}(x|\mu_i, \Sigma_i) \tag{2.16}$$

where $P(x|M)$ is the probability of x given the GMM model M and c_i are the mixture weights. $\mathcal{N}(x|\mu_i, \Sigma_i)$ is a normal pdf with mean μ_i and covariance matrix Σ_i

$$\mathcal{N}_i(x|\mu_i, \Sigma_i) = \frac{1}{(2\pi)^{D/2}|\Sigma_i|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)\right) \tag{2.17}$$

where D denotes the dimensionality of input space. The weights w_i are also called mixing coefficients and are constrained by

$$\sum_{i=1}^{m} w_i = 1 \tag{2.18}$$

$$w_i \geq 0, \quad i = 1 \ldots m \tag{2.19}$$

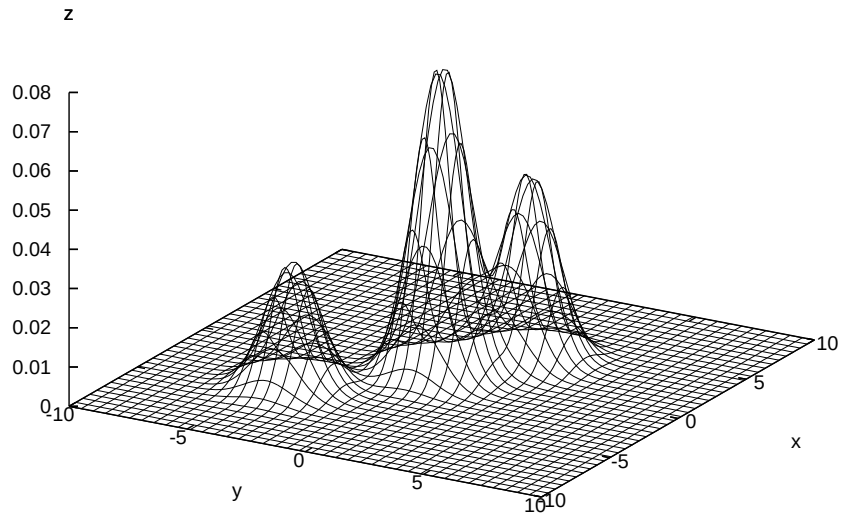


Figure 2.2: Gaussian Mixture Model

An interesting formulation of a GMM can be given in terms of discrete latent variables [12]. Let $Z = [z_1, \ldots, z_m]$ be a m–dimensional random variable such that $z_i \in \{0, 1\}$ and $\sum_i^m z_i = 1$, i.e. one single element of Z is equal to 1 while all the others are equal to 0. Let the distribution of z_k be defined in terms of the mixing coefficients w_i as

$$p(z_k = 1) = w_k \tag{2.20}$$

that is, since only one term of Z is equal to one and the others are equal to zero,

$$p(Z) = \prod_{i=1}^{m} w_i^{z_i} \tag{2.21}$$

The realizations of Z can be interpreted as representations of m mutually exclusive states, of which only one is active. Assuming that, for each state, the conditional probability $p(x|z)$ of random variable X given state $z_k = 1$ is

$$p(x|z_k = 1) = \mathcal{N}(x|\mu_k, \Sigma_k) \tag{2.22}$$

the marginal probability of X can then be obtained by summing the joint distribution $p(x|z)p(z)$ over all values of Z

$$p(x) = \sum_z p(z)p(x|z) = \sum_{i=1}^{m} w_i \mathcal{N}(x|\mu_i, \Sigma_i) \tag{2.23}$$

which has the same form of our original Gaussian Mixture model (2.16).

The main advantage in using GMMs is their capability to accurately approximate any probability density function, given enough components in the mixture. However, we have to estimate GMMs parameters $\vartheta = \{c_i, \mu_i, \Sigma_i\}$ describing model M from a sufficiently large set of samples of the random value. A possible approach consists in performing a Maximum–Likelihood (ML) estimate [13, 12], which looks for the solution of

$$\vartheta_{ML} = \arg\max_{\vartheta} P(X_s|\vartheta) \tag{2.24}$$

One of the advantages of representing a GMM as a latent variable model is that it allows for an easy derivation of a ML training procedure through the Expectation–Maximization algorithm (more details about the EM algorithm can be found in Appendix A.1).

2.2.2 Maximum likelihood estimate of a GMM

The Maximum Likelihood estimate of a GMM can be computed by means the Expectation–Maximization (EM) algorithm [12, 14, 15, 16, 13]. Let w^0, μ^0, Σ^0 denote the initial (random) values for the GMM parameters. The EM algorithm iterates

between the E and M steps as follows. During the E–step, for each observation x_i the posterior distributions of hidden variables Z_i associated to observation x_i, given the current model parameters w^c, μ^c, Σ^c, is computed as

$$p(z_{ik} = 1|x_i, \theta_c) = \gamma_{ik}(\theta_c) = \frac{w_k^c \mathcal{N}(x_i|\mu_k^c, \Sigma_k^c)}{\sum_{j=1}^m w_j^c \mathcal{N}(x_i|\mu_j^c, \Sigma_j^c)} \tag{2.25}$$

The parameters are updated in the M step to maximize the EM auxiliary function $Q(w, \mu, \Sigma, w^c, \mu^c, \Sigma^c) = \sum_i p(z_i|x_i, w^c, \mu^c, \Sigma^c) \log p(x_i, z_i|w, \mu, \Sigma)$. The update formulas are

$$w_k = \frac{\sum_{i=1}^n \gamma_{ik}(\theta_c)}{\sum_{i=1}^n \sum_{j=1}^m \gamma_{ij}(\theta_c)} \tag{2.26}$$

$$\mu_k = \frac{\sum_{i=1}^n \gamma_{ik}(\theta_c) x_i}{\sum_{i=1}^n \gamma_{ik}(\theta_c)} \tag{2.27}$$

$$\Sigma_k = \frac{\sum_{i=1}^n \gamma_{ik}(\theta_c)(x_i - \mu_k)^T (x_i - \mu_k)}{\sum_{i=1}^n \gamma_{ik}(\theta_c)} \tag{2.28}$$

A commonly used stopping criterion for the algorithm is the convergence of the likelihood $P(X|w, \mu, \Sigma)$.

Since estimating full covariance matrices is often difficult due to the scarcity of data and a increasing computational load, usually GMMs are estimated under the assumption that covariance matrices are diagonal. The loss in terms of modeling capabilities of the GMM is compensated by an increase in the number of components in the mixture [13].

2.3 Hidden Markov Models

Gaussian Mixture Models provide a compact and powerful representation of a set of acoustic features. However, temporal information is lost, since all frames are considered independent. A possible solution to account for temporal evolution of the acoustic signal consists in the use of Hidden Markov Models (HMM) [17, 12].

A HMM is a finite state machine characterized by two stochastic processes. The first one is responsible for the discrete temporal evolution across the system states, while the second generates the observation samples which form the acoustic event. The first process cannot be directly observed (hidden), but can be estimated from the frame sequence generated by the model.

2.3.1 Topological structure

From a topological view, a HMM is a directed graph $G(S, E)$ where $S = \{S_1, \ldots, S_N\}$ are the nodes or states of the model and E denotes the edges. Although the general

Markov model allows each state to reach any other state, in speaker and language recognition systems usually a simplified structure is adopted, which follows the temporal evolution of the acoustic event.

This simplified model, named Bakis linear model or left–to–right model [17], is characterized by a well established topological order of the nodes and presents only three kinds of edges: self–loop edges, forward edges (those ending in adjacent states) and skip edges (those ending in non adjacent states). Backward edges are not present, so the graph does not present any loop apart from self–loops. Each state identifies a stationary interval of the acoustic event, while the presence of self–loops allows to remain on the same state for longer periods, thus granting this model the capability to dynamically align the acoustic segment to the model. Moreover, this model imposes a minimal duration of the acoustic event, corresponding to the shortest path linking the first node to the final one.

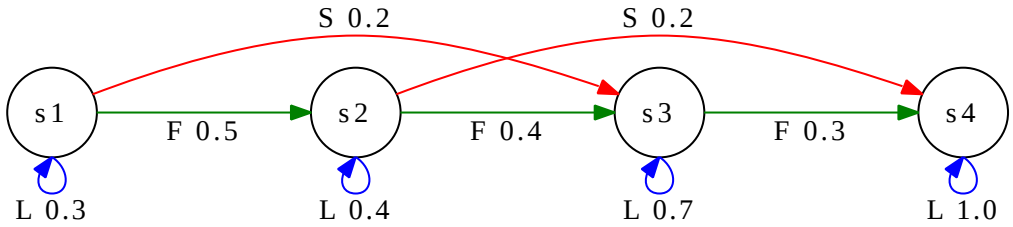


Figure 2.3: Left–to–right (Bakis) model

2.3.2 Probabilistic structure

The evolution through the states of a HMM is controlled by a stochastic process described by transition probabilities. Given q_t, the system state activated at time t, we can define $\pi = (\pi_1, \ldots, \pi_N)$ as the array of initial probabilities, that is the probabilities of being in state S_i at time $t = 1$

$$\pi_i = P(q_1 = S_i) \quad 1 \leq i \leq N \tag{2.29}$$

$$\sum_{i=1}^{N} \pi_i = 1 \tag{2.30}$$

Transition probabilities are described by a matrix

$$A = \{a_{ij}\} \quad 1 \leq i, j \leq N \tag{2.31}$$

where $a_{ij} = P(q_{t+1} = S_j | q_t = S_i)$ is the probability of reaching state S_j from S_i in a single step. As such, the following properties hold

$$a_{ij} \geq 0 \quad \forall (i,j) \tag{2.32}$$

$$\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i \tag{2.33}$$

In theory transition probabilities could also depend on the time t and on the observed sequence X. However, a simplified approach considers the transition probabilities as a stationary, first order Markovian process, for which the following properties hold

$$P(q_t = S_i | q_{t-1} = S_j, x_1, \ldots, x_{t-1}) = P(q_t = S_i | q_{t-1} = S_j) \tag{2.34}$$

and

$$P(q_t = S_i | q_{t+1} = S_j) = a_{ij} \quad \forall t = 1 \ldots N-1 \tag{2.35}$$

that is, transition probabilities do not depend neither on time nor on observed features.

Each state is associated to a stochastic function $f_i(x_t)$ which represents the probability of generating feature x_t when the system is in S_i. Usually it is assumed that acoustic features are generated independently given the state, that is

$$P(x_t | q_t = S_i, x_1, \ldots, x_{t-1}) = P(x_t | q_t = S_i) \tag{2.36}$$

In many real systems the functions f_i are, in fact, GMMs. An example of applications of these systems is speech decoding, where the GMMs are used to model the distribution of acoustic features given the HMM state [7]. Often, in these models the state–dependent GMMs share mean and covariance values. In this case, each $f_i(x)$ has the form

$$f_i(x) = \sum_{j=1}^{N} w_{ij} \mathcal{N}(x | \mu_j, U_j) \quad 1 \leq i \leq N \tag{2.37}$$

where the mean vectors μ_j and covariance matrices U_j do not depend on the state, while the weights w_{ij} still do. This approach allows for a reduction in the number of parameters with respect to a model where mean vectors and covariance matrices depend on the state.

In the following three sections we analyze the three main problems related to the use of HMMs [17], namely how to compute the probability of a set of observed features given the model parameters, how to evaluate the state sequence which best explains the observations (where "best" is defined according to a Maximum Likelihood criterion) and how to estimate the model parameters as to maximize the likelihood of some observations.

2.3.3 Forward-Backward algorithm

The first problem we address is how to evaluate the probability of a sequence of observed features $O = \{o_1, \ldots, o_T\}$ given the model M. This could, in theory, be done by summing over all possible T–long sequences of states $Q = \{q_1, \ldots, q_T\}$ the joint probability of the sequence Q and of the observations O

$$P(O|M) = \sum_Q P(O|Q, M)P(Q|M) \tag{2.38}$$

Under the assumption of statistically independent observed features, this becomes

$$\begin{aligned} P(O|M) &= \sum_Q \prod_{t=1}^{T} P(o_t|Q, M)P(Q|M) \\ &= \sum_Q \prod_{t=1}^{T} P(o_t|q_t, M)P(Q|M) \\ &= \sum_Q \prod_{t=1}^{T} f_{q_t}(o_t)P(Q|M) \end{aligned} \tag{2.39}$$

where $P(Q|M)$ can be evaluated as

$$P(Q|M) = \pi_{q_1} \prod_{t=1}^{T-1} a_{q_t q_{t+1}} \tag{2.40}$$

Assuming independent features is conceptually incorrect, since the acoustic signal presents a high degree of correlation between consecutive frames. However, this assumption is often used because it greatly simplifies the computations related to these kind of models. In the following we will assume feature independence. The approach of (2.39) is computationally unfeasible, since it requires a number of operations which grows exponentially with T (its complexity is $O\left(TN^T\right)$ [17]). The Forward–Backward algorithm [17, 18] allows a much faster evaluation of $P(O|M)$ (complexity $O\left(TN^2\right)$ [17]), exploiting that the probability of being in state S_i at time t having observed the sequence $o_1, \ldots, o_t$ can be evaluated from the probability of being in each state S_j at time $t-1$.

Let $\alpha_t(i)$ denote the *forward probabilities* of being in state S_i at time t having observed the sequence $o_1, \ldots, o_t$ and $\beta_t(i)$ denote the *backward probabilities* of observing, from time $t+1$ to T, the sequence $o_{t+1} \ldots o_T$ given that the system is in state S_i at time t, i.e.

$$\alpha_t(i) = P(q_t = S_i, o_1, \ldots, o_t|M), \quad \alpha_1(1) = f_1(o_1) \tag{2.41}$$

$$\beta_t(i) = P(o_{t+1}, \ldots, o_T|q_t = S_i, M), \quad \beta_T(f) = 1 \tag{2.42}$$

where S_1 is the initial state and S_f is the final state. The forward probabilities at time $t+1$ can be computed from the forward probabilities at time t as

$$\alpha_{t+1}(j) = \sum_{i=1}^{N} \alpha_t(i) a_{ij} f_j(o_{t+1}) \tag{2.43}$$

while for the backward probabilities the following holds

$$\beta_t(i) = \sum_{j=1}^{N} \beta_{t+1}(j) a_{ij} f_j(o_{t+1}) \tag{2.44}$$

The computation of the forward and backward probabilities can then be done in $O\left(TN^2\right)$. The quantity $P(O|M)$ can be obtained from the forward and backward probabilities as

$$P(O|M) = \sum_{j=1}^{N} \alpha_{t_0}(j) \beta_{t_0}(j) \tag{2.45}$$

where t_0 represents any frame, thus the overall complexity is $O\left(TN^2\right)$. More details about the forward–backward algorithm are given in Appendix A.2.

2.3.4 Viterbi algorithm

The second problem involving HMMs consists in finding the sequence of states $Q = \{q_1, \ldots, q_T\}$ which best explains the observed features $O = \{o_1, \ldots, o_T\}$. Different criteria might be chosen to define optimality. In this context we assume that the best sequence refers to the sequence of states which is jointly most likely, that is the sequence S^* defined as

$$S^* = \arg\max_S P(Q = S, O|M) = \arg\max_S P(Q = S|O, M) \tag{2.46}$$

A possible solution can be computed through the Viterbi algorithm [17, 18], which is equivalent to a minimum path search algorithm over the trellis associated to the HMM (for more details and the derivation of the algorithm see Appendix A.3)

Assuming that the HMM has a single starting node S_1, the algorithm evaluates, for each time frame, the probability of the best path ending in state k at time t (denoted as $\delta_t(k)$) and keeps the pointer $\psi_t(k)$ to the node for which the probability of reaching state k at time t is maximum.

The algorithm proceeds as follows

1. initialize $\delta_1(1)$ and $\psi_1(i)$ as

$$\delta_1(1) = f_1(o_1) \tag{2.47}$$
$$\psi_1(i) = 0 \qquad 1 \leq i \leq N \tag{2.48}$$

since the starting state has no parent node

2. iteratively evaluate the δ value of each node as t increases as

$$\delta_t(i) = \max_j \left(\delta_{t-1}(j) a_{ji}\right) f_i(o_t) \tag{2.49}$$

and update the corresponding backpointer $\psi_t(i)$ using

$$\psi_t(i) = \arg\max_j \left(\delta_{t-1}(j) a_{ji}\right) \tag{2.50}$$

The best sequence $\hat{Q} = \{\hat{q}_1, \hat{q}_2, \ldots, \hat{q}_T\}$ can be evaluated by setting $\hat{q}_T = S_f$ and following the backpointers up to the first frame $\hat{q}_t = S_{\psi_{t+1}(\hat{q}_{t+1})}$. The probability of the best path is $\delta_T(f)$, where f is the index of the final state S_f.

2.3.5 Training

Training a HMM consists in evaluating the model parameters $\theta = (A, B, \pi)$ with $A = \{a_{ij}\}$, $B = \{f_i(x)\}$ and $\pi = \{\pi_1, \ldots \pi_N\}$ in order to maximize the likelihood of an observed sequence given the model $P(O|\theta)$, this to adapt the model parameters so that they best describe the training data. Once again, this can be done by performing Maximum–Likelihood estimation through the Expectation–Maximization algorithm [17, 12].

In this section we give the re–estimation formulas for the model parameters at each iteration. The full derivation of these formulas can be found in Appendix A.4.

Let θ_c represent the current estimate of the model parameters (which are identified by the superscript c) and $\alpha_t^c(i)$, $\beta_t^c(i)$ be the forward and backward probabilities of state S_i at time t respectively, with $\alpha_0(j) = \pi_j$ and $\beta_T(j) = 1$, computed through the forward–backward algorithm using the current estimate of the model parameters. The new parameters can be evaluated as

$$a_{ij} = \frac{\sum_{t=1}^{T} \alpha_{t-1}^c(i) a_{ij}^c \beta_t^c(j) \left[\sum_k w_{k,j}^c \mathcal{N}(o_t | \mu_{k,j}^c, \Sigma_{k,j}^c)\right]}{\sum_{t=1}^{T} \alpha_{t-1}^c(i) \beta_{t-1}^c(i)} \tag{2.51}$$

$$w_{i,j} = \frac{\sum_{t=1}^{T} \alpha_t^c(j) \beta_t^c(j)}{\sum_{t=1}^{T} \alpha_t^c(j) \beta_t^c(j) \frac{w_{i,j}^c \mathcal{N}(o_t | \mu_{i,j}^c, \Sigma_{i,j}^c)}{\sum_{k=1}^{m} w_{i,k}^c \mathcal{N}(o_t | \mu_{i,k}^c, \Sigma_{i,k}^c)}} \tag{2.52}$$

$$\mu_{i,j} = \frac{\sum_{t=1}^{T} \alpha_t^c(j) \beta_t^c(j) \frac{w_{i,j}^c \mathcal{N}(o_t | \mu_{i,j}^c, \Sigma_{i,j}^c)}{\sum_{k=1}^{m} w_{i,k}^c \mathcal{N}(o_t | \mu_{i,k}^c, \Sigma_{i,k}^c)} o_t}{\sum_{t=1}^{T} \alpha_t^c(j) \beta_t^c(j) \frac{w_{i,j}^c \mathcal{N}(o_t | \mu_{i,j}^c, \Sigma_{i,j}^c)}{\sum_{k=1}^{m} w_{i,k}^c \mathcal{N}(o_t | \mu_{i,k}^c, \Sigma_{i,k}^c)}} \tag{2.53}$$

$$\Sigma_{i,j} = \frac{\sum_{t=1}^{T} \alpha_t^c(j)\beta_t^c(j)\frac{w_{i,j}^c \mathcal{N}(o_t|\mu_{i,j}^c,\Sigma_{i,j}^c)}{\sum_{k=1}^{m} w_{i,k}^c \mathcal{N}(o_t|\mu_{i,k}^c,\Sigma_{i,k}^c)}(x_t-\mu_{i,j})^T(x_t-\mu_{i,j})}{\sum_{t=1}^{T} \alpha_t^c(j)\beta_t^c(j)\frac{w_{i,j}^c \mathcal{N}(o_t|\mu_{i,j}^c,\Sigma_{i,j}^c)}{\sum_{k=1}^{m} w_{i,k}^c \mathcal{N}(o_t|\mu_{i,k}^c,\Sigma_{i,k}^c)}} \tag{2.54}$$

2.4 Artificial Neural Networks

Another set of models which have broad applications in speech technology are Artificial Neural Networks. In particular, in the context of speech recognition Artificial Neural Networks are often used in combination with HMM to provide a fast and discriminative way to perform speech decoding [19].

2.4.1 Structure

An Artificial Neural Network [20] (ANN) is a directed graph $G(I, N, O, E)$ where I denotes the set of input nodes, O denotes the set of output nodes, N is the set of internal nodes and E is the set of edges. Input nodes receive their inputs from outside the net, while output nodes expose values out of the net. Each non–input node has associated a set of parent nodes as described by the graph edges. An edge from node n_i to node n_j will be denoted as $\xi_{i,j}$, the set of input units of a given node as $S(i)$ and the set of output units of a given node as $D(i)$.

Each unit of the net n_i is characterized by an input value $n_i(t)$, an activation value $a_i(t)$ and an output value $o_i(t)$. These values depend on the input pattern of the net x. Each edge $\xi_{i,j}$ has associated a weight $w_{i,j}$ (or, which is equivalent, each unit has associated a set of weights, one for each of its input units).

The output value is evaluated as a function of the current activation value

$$o_i(t) = F_i(a_i(t)) \tag{2.55}$$

while the new activation value is updated using a function of the current activation value and the inputs of the unit as

$$a_i(t+1) = f_i(a_i(t), n_i(t)) \tag{2.56}$$

where $f_i(x)$ is called activation function. The output function $F_i(x)$ is usually either the identity function $F_i(x) = x$, a threshold function or a stochastic function (e.g. Gaussian). The activation function $f_i(x)$ usually is either a threshold function, a semi–linear function (e.g. sigmoid, hyperbolic tangent) or a stochastic function (e.g. Gaussian).

Artificial Neural Networks are able to "learn" an association between input patterns and output patterns by adjusting the weights of the net in such a way that

the net output corresponds to a given pattern when the net is fed a given input. Several types of network exist, the main differences lying in the topological structure and in the learning algorithm used. Since in speech recognition we mostly deal with feed–forward networks usually derived from the Multi–Layer Perceptron, the following sections will focus on this kind of networks.

2.4.2 Feed–forward Neural Network and Perceptron

Feed–forward neural networks are characterized by the absence of cycles. The units are organized in layers (Figure 2.4)

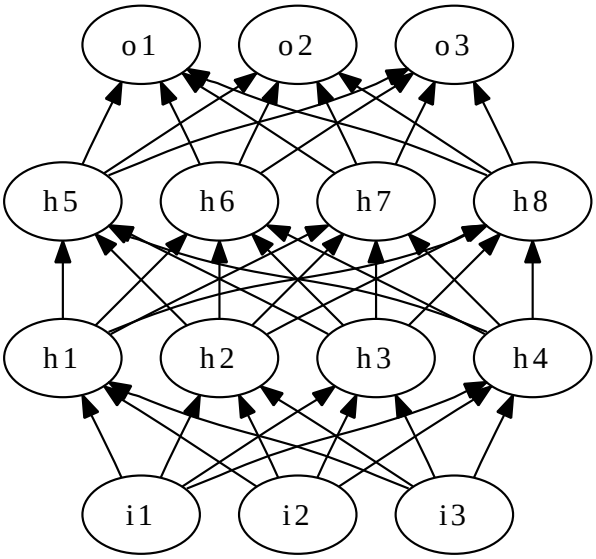


Figure 2.4: Topological structure of a Multi–Layer Perceptron net

A very popular feed–forward network is the Multi–Layer Perceptron [20], which is characterized by the presence of one or more hidden layers between the input and the output layer. When used in speech recognition the activation functions are usually sigmoid functions. Sometimes output units have a soft–max activation function (Section 2.4.4).

Each unit evaluates its input value as

$$n_i(t) = \sum_{j \in S(i)} w_{j,i} o_j(t) + b_i \tag{2.57}$$

where b_i is a fixed bias. The output is evaluated as

$$o_i(t) = a_i(t) = f(n_i(t)) \tag{2.58}$$

with

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.59}$$

The strength of the MLP consists in being able to approximate any kind of division surface given enough hidden nodes and enough hidden layers (i.e. two hidden layers [21]). Usually, the number of hidden layers is limited to 1 or 2.

2.4.3 Training

The training algorithm used with feed–forward networks goes under the name of backward propagation [22] and tries to minimize the classification error of the network. Let $x_p = (x_{1p}, \ldots, x_{Np})$ be an input pattern p of the net (which has N input units), $t_p = (t_{1p}, \ldots, t_{Mp})$ be the M target output values corresponding to x_p and $o_p = (o_{1p}, \ldots, o_{Mp})$ the output values of the net given input pattern x_p. The set of output units will be denoted as O. The error function can be expressed as

$$E = \sum_p E(p) \tag{2.60}$$

where the error $E(p)$ for a given pattern p can be expressed as the sum of the errors $e_j(p)$ of each output unit n_j

$$E(p) = \sum_{j \in O} e_j(p) \tag{2.61}$$

The back-propagation algorithm minimizes the error function using gradient descent in the weights space. Weights are adjusted according to

$$\Delta w_{i,j} = -k \frac{\partial E}{\partial w_{i,j}} = -k \delta_j \frac{\partial n_j}{\partial w_{i,j}} = -k \delta_j o_i \tag{2.62}$$

with

$$\delta_j = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial n_j} \tag{2.63}$$

The update rule for the output layer weights is straightforward, since $\frac{\partial E}{\partial o_j}$ depends only on the form of $E(p)$.

The back–propagation algorithm allows to update hidden layer weights by back–propagating (hence the name) the output layer error through the network. In particular, in order to update weights for nodes in layer $\mathcal{L}$ we need to compute the values of δ_j for such nodes. The term $\frac{\partial o_j}{\partial n_j}$ depends only on the node structure (i.e. its activation function). For the error term $\frac{\partial E}{\partial o_j}$ we make explicit its dependency on input values of layer $\mathcal{L} + 1$ as

$$\frac{\partial E}{\partial o_j} = \sum_{i \in D(j)} \left(\frac{\partial E}{\partial n_i} \frac{\partial n_i}{\partial o_j} \right) \tag{2.64}$$

Since $\frac{\partial E}{\partial n_i} = \delta_i$ and $\frac{\partial n_i}{\partial o_j} = w_{i,j}$ we can rewrite the derivative of the error function as

$$\frac{\partial E}{\partial o_j} = \sum_{i \in D(j)} (\delta_i w_{i,j}) \tag{2.65}$$

Therefore, in order to compute δ_j for a layer in $\mathcal{L}$ we only need errors δ_i of nodes in layer $\mathcal{L}+1$. The backpropagation consists then in a gradient descent optimization where the weights are updated according to

$$\Delta w_{i,j}^{t+1} = -\eta_t \frac{\partial E}{\partial w_{i,j}^t} = -\eta_t \delta_j^t o_i^t \tag{2.66}$$

$$w_{i,j}^{t+1} = w_{i,j}^t + \Delta w_{i,j}^{t+1} \tag{2.67}$$

where η_t is the learning rate at time t and $w_{i,j}^t$ is the value of weight $w_{i,j}$ at iteration t. In order to improve the convergence rate often a *momentum* term [20] is added, which also partially allows to avoid getting stuck in local minima. Let β denote the momentum coefficient. The update rule then becomes

$$\Delta w_{i,j}^{t+1} = -\eta_t \delta_j^t o_i^t + \beta \Delta w_{i,j}^t \tag{2.68}$$

$$w_{i,j}^{t+1} = w_{i,j}^t + \Delta w_{i,j}^{t+1} \tag{2.69}$$

The back–propagation algorithm can be initialized using small random values as initial guess of the net weights.

2.4.4 ANN–HMM

An attractive alternative to GMM–based HMM is given by hybrid HMM–ANN models [19]. In these models an ANN is used in place of a GMM to estimate posterior probabilities of states given the observations. ANNs allow for a faster computation of posteriors than GMMs and are intrinsically discriminative. However, neural networks are designed to classify static patterns, which makes training more complex, and, moreover, require a large amount of training data. On the other hand, while GMMs are based on the assumption of statistical independence between consecutive frames, this is not the case of ANN–HMMs. In fact ANN–HMM models are often trained using a frame together with its left and right contexts [23].

Structure

A classical topology for the ANN part of the model consists in a Multi–Layer Perceptron (MLP) where consecutive layers are fully connected (Figure 2.5).

In this case, the net is fed a frame together with its context and each node of the net is associated to a node of the HMM.

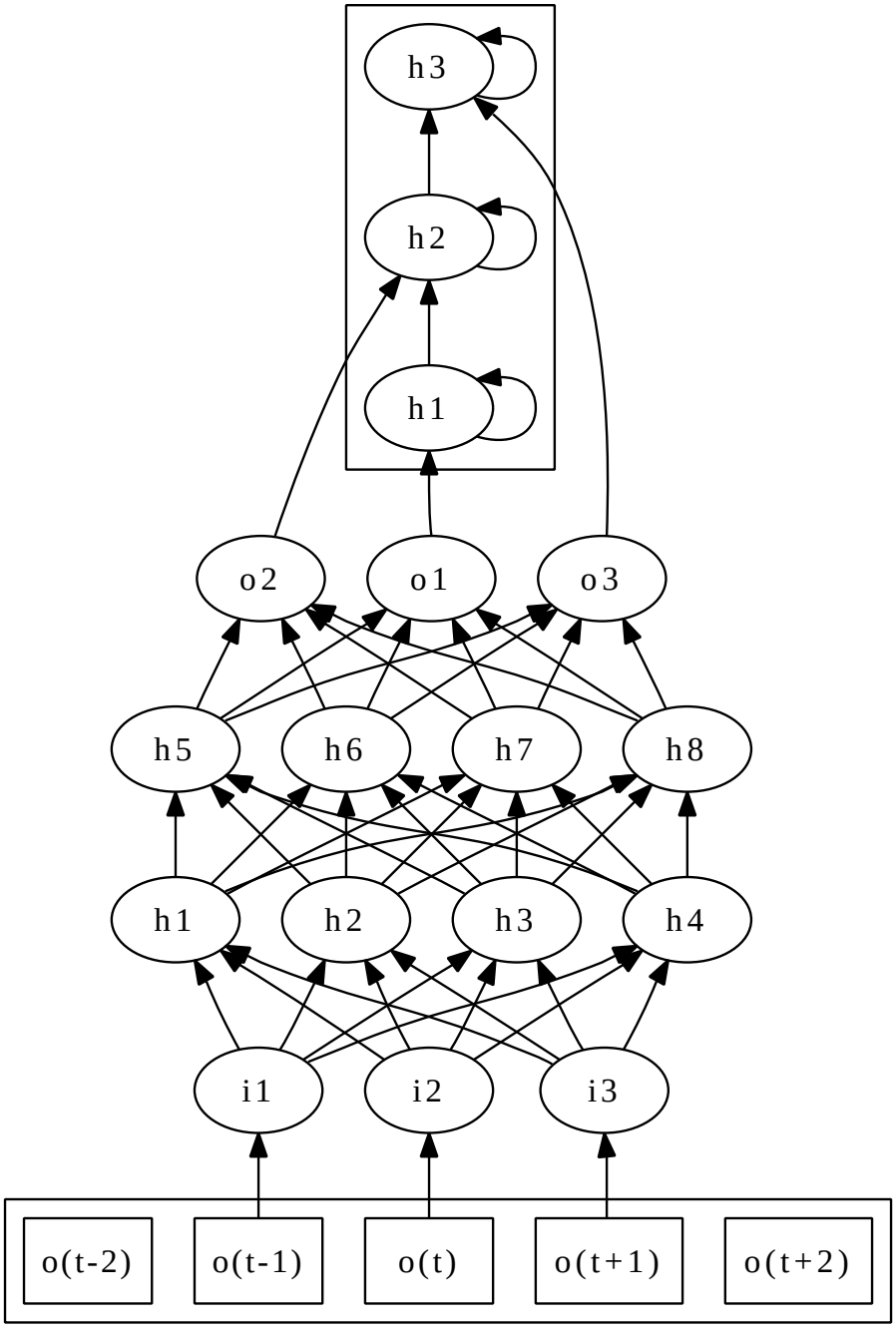


Figure 2.5: Example of a HMM used in combination with a MLP

The output value of the net is the probability associated to the corresponding HMM node for the given frame (thus it has the same role of GMMs in GMM–based HMMs). Usually, input and hidden layers have sigmoidal activation functions.

Output units, instead, are often characterized by softmax activation functions, which allow to interpret the network outputs as posterior class probabilities $P(c_i|x_t)$ for class c_i given the observed feature vector x_t. The output of the networks are computed as

$$o_i = \frac{e^{n_i}}{\sum_{j \in O} e^{n_j}} \tag{2.70}$$

so that

$$0 \leq o_i \leq 1 \quad \forall i \in O \tag{2.71}$$

and

$$\sum_{i \in O} o_i = 1 \tag{2.72}$$

The main disadvantage of these kind of models is the need to retrain the complete model when a new class is added. In speaker recognition, for example, where it is usual to add new speakers, this can pose some issues since training a neural network is a time consuming task.

Usually we are interested in the class–conditional probability of observed features $P(x_t|c_i)$. This can be evaluated resorting to the Bayes theorem as

$$P(x_t|c_i) = \frac{P(c_i|x_t)P(x_t)}{P(c_i)} \tag{2.73}$$

If we make the hypothesis of equiprobable distribution for $P(x_t)$ we have

$$P(x_t|c_i) \propto \frac{P(c_i|x_t)}{P(c_i)} \tag{2.74}$$

This way, we can compute the posterior probability of an observed vector given a class up to a multiplicative factor. The prior probability can be taken into account by including it in the biases of the last layer as

$$b_i^{new} = b_i - \log P(C_i) \tag{2.75}$$

Training

The main problem when using hybrid ANN–HMM models lies in the training stage, since training an ANN would require to have fixed targets for each pattern, while in speech recognition this is not the case. The change of target values is due to the segmentation process used to label the given utterance (i.e., when using ANN–HMM models as phonetic recognizers, we need to identify phonemes corresponding to each

part of the sentence). Since manual segmentation is not feasible due to the high cost it would require, automatic segmentation is used, which, in turn, is not completely accurate but depends on the model used to perform it: the better the model, the better the segmentation. As we refine our ANN–HMM model, the segmentation itself usually changes, thus causing the change of target values (i.e. if we label a given segment with a different phoneme, then the ANN units corresponding to that phoneme would have one as target value; when we change the label, those units will have zero as target value). Training such a system requires an initial segmentation, often obtained by using other kind of models such as GMM–HMMs.

The procedure used to perform training can be described as

1. initialize the network with small random weights
2. load the actual segmentation (or the initial one if this is the first iteration)
3. train the model with several iterations of the back propagation algorithm in order to obtain a model which approximates the targets given by the actual segmentation
4. compute the new segmentation using the current model
5. update the current segmentation by taking into account both the old and the new segmentation (e.g. the new segmentation target values can be evaluated as a weighted sum of the old target values and of those computed during the previous step)
6. repeat from 2.

Computing the actual segmentation can be done using the Viterbi algorithm and labeling the sentence with the symbols associated with the states of the best path of the Markov model [7].

2.5 Phonotactic features

Acoustic features are often directly used to create speaker and language models, as described in the next chapters. While acoustic models allow very good performance both for speaker and language verification, state–of–the–art systems often employ models based on higher level features which provide useful complementary information to acoustic models. In this section we will focus on the use of phonotactic features and phonetic models for language recognition.

While acoustic techniques are based on direct modeling of acoustic features, phonetic techniques build models based on the phonetic content of sentences, and

in particular on the occurrences of phoneme sequences (called n–gram, where n is the sequence length). These techniques share a common front–end, which is based on speech recognition technology used to perform phonetic decoding of utterances. Various implementations of n–grams have been presented. They may use a single phone recognizer followed by language dependent modeling (PRLM) or multiple phone recognizers followed by the same kind of language modeling (PPRLM), n–gram evaluation can be based on the first hypothesis made by the phone recognizer (1–best decoding) or on a set of hypothesis (lattice decoding), the decision rule could be implemented using maximum likelihood or discriminative classifiers such as Support Vector Machines.

2.5.1 Bags of n-grams features for language identification

Language identification can be described as looking for the language with highest posterior probability $P(L|X, \Lambda, \Phi)$ given the speech segment X, the phone acoustic models Λ and the phonotactic models Φ (phonotactic models describe a–priori relations between phonemes). Under the assumption of equiprobable languages, the standard approach consists in finding

$$L^* = \arg\max_L \sum_H f(X|H, L, \Lambda) P(H|L) \tag{2.76}$$

where L^* is the hypothesized language, H is a hypothesized sequence of phonemes or words (transcription), f is the likelihood of speech segment X given H, L and the phone models Λ and $P(H|L)$ is the prior probability of H estimated through a phone n–gram model [24]. The model can be simplified by taking into account only the most relevant term of the summation,

$$L^* \simeq \arg\max_L \max_H f(X|H, L, \Lambda) P(H|L) \tag{2.77}$$

which leads to the so called parallel phone recognition (PPR) approach [25]. Since PPR systems don't give better results than similar phonotactic models, they will not be described in this work. By making the phone models language–independent we can simplify equation (2.77) by replacing $f(X|H, L, \Lambda)$ with $f(X|H, \Lambda)$. The resulting model, however, still consider all possible phone labellings. Further simplifications can be obtained by considering only the most likely phone sequence $H^* = \arg\max_H f(X|H, \Lambda)$ and by replacing (2.77) with

$$L^* \simeq \arg\max_L P(H^*|L) \tag{2.78}$$

Sometimes the best hypothesis is evaluated as the one maximizing the posterior probability $P(H|X, \Lambda)$, The two formulations would be equivalent if we could assume

that all hypotheses are a–priori equiprobable. Although this is not the case, usually the latter expression is used when dealing with HMM–based phone recognizer, since it can be evaluated using the standard algorithms for HMMs (Section 2.4.4). To compensate for the simplification of phone models independence from language, multiple parallel phone recognizers are often used in practice [26] (PPRLM systems).

Best–hypothesis systems are attractive due to the relative simplicity of the evaluation of the most likely labeling hypothesis. However, neglecting less likely hypothesis is a potential source of performance degradation [24]. In order to cope with this problem phonetic lattices are used to take into account also less likely hypothesis. Phone lattices are directed acyclic graphs whose nodes represent timing constraints and edges represent phone hypotheses and have associated an acoustic score. The idea behind lattices is to replace the approximation leading to 1–best decoding by Expectation–Maximization of $\log P(H|L)$ over all the possible phone sequences in the speech segment [24]

$$L^* \simeq \arg\max_L E_H[\log P(H|L)|X, \Lambda, L] \tag{2.79}$$

Bags of n-grams

Standard PRLM systems use the best hypothesis generated by a single phone recognizer to evaluate the (log)–likelihood of that hypothesis given a language L (2.78). From (2.78) we can also write that the target language is

$$\begin{aligned} L^* &= \arg\max_L \log P(H^*|L) \\ &= \arg\max_L \sum_{i=1}^{N} \log P(h_i|\hat{h}_i, L) \end{aligned} \tag{2.80}$$

where $\hat{h}_i$ is the set of phones that precedes h_i in H. Assuming that a phoneme only depends on the m preceding ones, this becomes

$$L^* = \arg\max_L \sum_{i=1}^{L} \log P(h_i|h_{i-(m)}, \ldots, h_{i-1}, L) \tag{2.81}$$

where L is the length of the sequence. This is equivalent to

$$L^* = \arg\max_L \sum_{\hat{h},h} P(\hat{h}, h|H^*) \log P(h|\hat{h}, L) \tag{2.82}$$

where the summation is taken over all the possible distinct n–grams $(\hat{h}, h)$.

This can be evaluated by estimating n–gram occurrences in a large training set. First of all, we evaluate context–dependent n–grams probabilities for each speech

segment in the training set (that is, the probability of a symbol given its trailing context which, together with the symbol, forms a n–gram) as

$$P(h_i|\hat{h}_i^n, H_k) = \frac{count(\hat{h}_i^n, h_i|H_k)}{count(\hat{h}_i^n|H_k)} \tag{2.83}$$

where H_k is the best phone hypothesis for speech segment k, h_i identifies the i–th symbol of H_k, $\hat{h}_i^n$ is the sequence of $n-1$ symbols which precede h_i in H_k, n is the n–gram order and $count(x|H_k)$ is the number of occurrences of sequence x in H_k. Different methods can be used to approximate $P(h_i|\hat{h}_i^n)$ from the statistics gathered from the training set. The simplest way consists in averaging the context dependent n–gram frequencies over all the (M–sized) training set, that is evaluating $P(h_i|\hat{h}_i^n)$ as

$$P(h_i|\hat{h}_i^n) = \frac{1}{M}\sum_{j=1}^{M} P(h_i|\hat{h}_i^n, H_j) \tag{2.84}$$

Another possibility consists in computing the sum of the same n–gram frequencies weighted by the hypothesis probability for each speech segment X_j

$$P(h_i|\hat{h}_i^n) = \sum_{j=1}^{M} P(h_i|\hat{h}_i^n, H_j)P(H_j|X_j) \tag{2.85}$$

In [27] the authors describe a way to approximate n–gram distribution by an interpolated model where $P(h_i|h_i^n)$ in (2.81) is substituted by

$$\tilde{P}(h_i|\hat{h}_i^n) = \sum_{k=0}^{n} \alpha_k P_k(i) \tag{2.86}$$

where $P_0(i)$ is the reciprocal of the number of different symbol types, $P_1(i) = P(h_i)$ and $P_k(i) = P(h_i|\hat{h}_i^k)$ for $2 \leq k \leq n$ evaluated as in (2.83). The coefficients α_j can be evaluated through the Expectation–Maximization algorithm.

The (joint) probability of an n–gram given the phone hypothesis H_k can be evaluated as

$$P(\hat{h}_i^n, h_i|H_k) = \frac{count(\hat{h}_i^n, h_i|H_k)}{\sum_{j=1}^{M} count(\hat{h}_j^n, h_j|H_k)} \tag{2.87}$$

While n–gram counts can be used to evaluate language models and directly to perform classification of unknown utterances, state–of–the–art systems tend to use bags of n–grams as high–level features which are combined with different classifiers (e.g. Support Vector Machines). Some of these approaches are presented in Section 4.2.4

2.5.2 Phonetic decoders

In order to create models based on n–gram frequencies it is necessary to associate phonetic labels to a given utterance. This process is called phonetic decoding. In this context we are interested in a model that allows estimating the phone labeling H which maximizes the likelihood of the observed features $P(X|H,\Lambda,L) = P(X|H,\Lambda)$ (for 1–best decoding) or which allows evaluating the posterior probability of an hypothesis given the observations $P(H|X,\Lambda,L)$ (for lattice based decoding). An answer to this problem is given by HMMs.

State–of–the–art phone recognizers are implemented through the use of HMMs in conjunction with either GMMs [7] or ANNs [19]. A possible approach to HMM–based phonetic decoding consists in associating a small left–to–right Markov model to each phoneme. The global HMM is obtained by linking the final state of each phoneme to the starting state of each other [23, 28]. Moreover, along with phonemes, these models often present units associated to transitions between different phonemes in order to improve recognition accuracy. In this case, usually grammatical constraints are imposed between transitions and stationary states (those corresponding to phonemes).

While a complete analysis of speech recognition technology is beyond the scope of this work, in the next sections we briefly present how speech recognizers can be used to extract high–level features suitable for a language recognition tasks and we briefly detail the speech decoders used in our experiments.

1–best decoding

1–best decoding [25] looks for the most likely labeling hypothesis, that is, the most likely sequence of states of the HMM. This can be computed by means of the Viterbi algorithm [25].

Lattice decoding

In order to evaluate (2.79) we could compute, over all possible phoneme sequences, the probabilities of a particular sequence together with its n–gram statistics. This corresponds to applying a PRLM scheme on each possible hypothesis and summing up the results weighted by the sequence probability. However, this approach is unfeasible since due to the prohibitive computational load when the n–gram order grows, even for short utterances and low–order n–grams. Dynamic programming algorithms exist that allow for a much faster creation of the lattice through a modified Viterbi algorithm in which the most promising paths only are expanded and added to the lattice at each frame (e.g. through a beam search).

A lattice comprises most of the labeling hypothesis for a given utterance. The great number of paths does not allow directly computing n–gram statistics by graph

inspection. However, n–gram statistics can be efficiently computed by means of the forward–backward algorithm [24].

2.5.3 Loquendo ASR

The decoder used in our experiments is the Loquendo ASR [29, 23, 28]. The decoder replaces the GMM part of the HMM with an ANN. The use of an ANN is mainly motivated by the small amount of time required to compute the class posteriors required during decoding. Moreover, the neural network is inherently a discriminative system. The ANN part of the decoder is a 3–layers Multi Layer Perceptron. Stationary Transitional Units (STU) [23] are used to model the phonetic content of utterances. The net is fed with a sequence of frames. The hidden layers have between 300 and 500 nodes with sigmoid activation function, while the output layer has between 700 and 1000 nodes (the number is language dependent) with softmax activation units. The network is fed with patterns consisting of seven frames, the central frame and three frames for the left and right contexts, respectively. Training is done by means of back–propagation. The main back–propagation flavors are batch mode and online mode [30]. In the former weights are updated after all patterns have been processed, in the latter weights are updated after each pattern. Online mode usually allows for faster convergence of the algorithm and better accuracy. However, online training is intrinsically sequential and as such cannot be parallelized. In order to allow for some degree of parallelism our ANNs are trained in *bunch* mode [31], that is, weights re–estimation is carried out after a (small) bunch of patterns have been processed. The HMM part of the network consists in left–to–right models with self loops for each class. These models are then connected together to form the full HMM. All states transition probabilities are equal [28]. The complete training steps are similar to the ones detailed in Section 2.4.4.

2.5.4 Speeding up ANN training

ANN–HMMs allow good accuracy with a decoding time which is much lower than with GMM–HMMs. However, ANNs require large amounts of data to avoid overfitting problems. Training of a ANN–HMM can thus require very long time (e.g. even in the orders of one month [32]). Speeding–up ANN training would allow to either reduce the training time or to achieve better performance by using larger datasets. In this section we present some results about ANN training speed–up using single core and multicore CPUs and Graphic Processing Units (GPU). These results were first published in [33, 32].

Single–core and multi–core CPU

The first step towards our fast training algorithm for ANN training consists in rewriting forward and backward procedures in matrix notations. Let $\mathbf{W}_{\mathcal{L}}$ be the matrix of weights between layer $\mathcal{L}-1$ an $\mathcal{L}$, i.e. $\mathbf{W}_{\mathcal{L}i,j} = w^{\mathcal{L}}_{j,i}$. Assume we want to process a bunch of B patterns. Let $\mathbf{o}_{\mathcal{L}}$ be the matrix of outputs of layer $\mathcal{L}$ for the considered patterns, i.e. $\mathbf{o}_{\mathcal{L}i,j}$ is the output of node i for pattern j, and let $\mathbf{n}_{\mathcal{L}}$ be the matrix of inputs of layer $\mathcal{L}$, i.e. $\mathbf{n}_{\mathcal{L}i,j}$ is the input value for node i corresponding to pattern j The forward step computations can be evaluated as

$$\mathbf{n}_{\mathcal{L}} = \mathbf{W}_{\mathcal{L}}\mathbf{o}_{\mathcal{L}-1} \tag{2.88}$$

while the activation function can be applied independently on all elements of n in a vectorized fashion. For the backpropagation step we use as error function the cross–entropy between the net outputs $\mathbf{o}$ and the target values $\mathbf{t}$. The negative cross–entropy is defined as

$$e(p) = -\mathbf{t}_{i,p}\log\mathbf{o}_{i,p} - (1-\mathbf{t}_{i,p})\log(1-\mathbf{o}_{i,p}) \tag{2.89}$$

The derivative of the error function with respect to the outputs for the last layer $\mathcal{L}_N$ node is then given by

$$\mathbf{e}_{\mathcal{L}_{\mathcal{N}}} = (\mathbf{1}-\mathbf{t})\circ(\mathbf{1}-\mathbf{o}_{\mathcal{L}})^{\circ-1} - \mathbf{t}\circ\mathbf{o}_{\mathcal{L}}^{\circ-1} \tag{2.90}$$

where symbol $\circ$ denotes elementwise operations (multiplication and inverse). The backpropagation update rules for the different layers can then be computed as

$$\begin{aligned}
\boldsymbol{\delta}_{\mathcal{L}} &= (\mathbf{1}-\mathbf{o}_{\mathcal{L}})\circ\mathbf{o}_{\mathcal{L}}\circ\mathbf{e}_{\mathcal{L}} && (2.91)\\
\mathbf{e}_{\mathcal{L}-1} &= \mathbf{W}^T_{\mathcal{L}}\boldsymbol{\delta}_{\mathcal{L}} && (2.92)\\
\boldsymbol{\Delta W}^{t+1}_{\mathcal{L}} &= -\eta\boldsymbol{\delta}_{\mathcal{L}}\mathbf{o}^T_{\mathcal{L}-1} + \beta\boldsymbol{\Delta W}^{t}_{\mathcal{L}} && (2.93)\\
\mathbf{W}^{t+1}_{\mathcal{L}} &= \mathbf{W}^{t}_{\mathcal{L}} + \boldsymbol{\Delta W}^{t+1}_{\mathcal{L}} && (2.94)
\end{aligned}$$

where η is the learning rate and β is the momentum coefficient. The simplest optimization of ANN training consists then in the use of highly optimized matrix–vector multiplication routines [34].

A different and complementary approach to speed up the training is the so called focused–attention back–propagation (FABP) learning strategy [35]. In FABP the attention is focused on patterns which are more difficult to learn while discarding the easy patterns. The forward step is computed for all patterns, while only those patterns having Mean Square Error (MSE) greater than a given threshold are used in the back–propagation step. In our applications FABP reaches a skip ratio of almost 80%. For this reason, the bunch size of the forward step is larger than the

one used for back–propagation. The bound of the forward step bunch size is due to the need to perform the forward computations for the same pattern more than once, should the FABP select more patterns than the back–propagation bunch size (this is due to the change in the weights that takes place after the back–propagation step).

Straightforward multi–core implementations can be obtained by using multi-thread matrix multiplication routines as well as by parallelizing vectorized operations. In the next section, however, we show how a faster implementation can be obtained by exploitation of the multi–processing capabilities of Graphical Processing Units.

Graphical Processing Units for ANN training

Graphical Processing Units (GPUs) are graphics-oriented dedicated processors suited to computationally expensive but highly parallelizable tasks such as 3D graphic rendering. GPUs are characterized by the presence of a high number of floating core processors able to perform parallel computations in a vectorized fashion. Since most of the ANN computations are characterized by fine–grain parallelism, GPUs provide an interesting tool to speed–up ANN training. The General Purpose computing on Graphical Processing Units (GPGPU) framework has been boosted in recent years by the development of high–level programming languages which facilitate GPU–based algorithm implementations. Among these frameworks, we use the NVIDIA CUDA (Compute Unified Device Architecture [36]) programming language, which provides both a C–like high–level language for GPU programming and an efficient implementation of the BLAS library (CuBLAS). In the CUDA framework the programmer has access to a grid of thread blocks where each thread executes a single instruction set called kernel [36].

CUDA–based optimizations of ANN training have been proposed in the past, e.g. in [37] for recurrent networks, [38] for the forward step, or [39] for a MLP under the assumption that all patterns can be stored on the GPU memory. The approach we presented in [33, 32] is similar to the latter, however it takes into account problems related to the use of ANNs for speech recognition. First of all, training patterns easily exceed some millions, thus they cannot be stored in main memory. Moreover, convergence problems suggest that bunch–mode back–propagation with FABP is a better training strategy for phonetic decoders. Finally, the presence of a softmax output layer poses some problems due to the presence of a sum over all output units of the network.

The implementation we presented in [33, 32] allows keeping bunch–mode FABP while still allowing good performance in terms of speed–up. First of all, matrix multiplication operations are mapped onto CuBLAS equivalents (e.g. cublasSgemm, a fast and hardware-optimized implementation of matrix product functions) both in

the forward and backward steps. Carefully tailored kernels have been implemented for the softmax and sigmoid functions, the MSE evaluation, and the update of the network weights. Zero–padding has been adopted to have matrices whose dimensions are multiples of powers of 2. This allows achieving better performance when using CuBLAS routines. Due to the limited amount of GPU memory, patterns are loaded in bunches and only weights are kept in the GPU RAM. Finally, the computation of the softmax sum is performed in parallel with respect to the patterns, although we adopt a simple sequential approach for each pattern.

The core steps of the training algorithm can be summarized as follows:

- Load a bunch of F input patterns on the GPU memory.
- Compute the net outputs (forward step) and the MSE for each pattern.
- Transfer both the bunch MSEs and the output patterns from the GPU to the main memory (the outputs are needed by the Viterbi algorithm to refine the segmentation).
- Test bunch MSEs to select patterns to be back–propagated (FABP).
- Stack the selected patterns on the GPU into a matrix used for backpropagation (the copy of the vectors allows for a faster back–propagation step).
- Repeat from beginning until a sufficient number of patterns B have been selected.
- Execute the back–propagation procedure.
- Discard selected patterns which did not fit the back–propagation bunch (they have to be re–submitted to the forward step).
- Repeat from beginning until all patterns have been processed.

These steps are iterated until convergence criteria are met.

Experimental results covering the speed–up obtained by the different implementations (single core, multicore, GPU) are given in Section 6.1. Here we observe that our implementation proves to be memory efficient, in that less than 10% of training time is spent in GPU/main memory transfers. Moreover, most of the training time is spent in CuBLAS functions, while a fraction is required to transfer patterns selected by FABP to the back–propagation structure. This step is mandatory because only a fraction of the patterns are selected and memory contiguity allows for a much faster back–propagation step. Finally, the time spent in computation kernels covers about only one forth of all training time. This motivates our decision not to push any further kernel optimizations. These results are summarized in Table 2.1, which

shows how training time is split among the different steps and where "type" refers to the kind of function used to perform the operations (*cudaMemcpy* for memory transfers, *cublasSgemm* for matrix multiplications and *kernel* for tailored kernels).

Table 2.1: Timing profiles for MLP training using CUDA

Function	Time (%)	Type
Memory transfer CPU/GPU	8.7	cudaMemcpy
Forward step	32.5	
Computation of network inputs **n**	22.8	cublasSgemm
Softmax function	7.2	kernel
Sigmoid function	2.5	kernel
Memory transfer GPU/GPU	10.0	kernel
Backpropagation step	48.8	
Cross–entropy	1.2	kernel
Error derivative	2.4	kernel
Error propagation	20.1	cublasSgemm
Weights gradient	17.0	cublasSgemm
Weights update	8.1	kernel

Chapter 3

Latent variable models for speaker and language recognition

Latent (or hidden) variable models offer an interesting approach to build probabilistic models of speakers and languages. Latent variable models provide a probabilistic framework to model observed features in terms of more tractable conditional distributions instead of complex marginal distributions [12]. Often these variables have associated strong meanings, e.g. they can interpreted as representing the speaker identity or the channel.

Another interesting use of hidden variable is dimensionality reduction. In fact, latent variables provide a probabilistic framework to estimate low–dimensional manifolds where most of the data points live.

The first set of models we describe use latent variables in the GMM space to build robust speaker models. We then describe how these models can be applied to perform dimensionality reduction and how latent variable models can be used to perform inference using these low–dimensional features.

3.1 Speaker verification problem

The task we address in this chapter is speaker verification. We are therefore interested in systems which can be used to asses whether a test utterance is spoken by a target speaker or not. In particular, we are interested in systems which are able to produce verification scores or, even better, (log–)likelihoods [6]. Two main flavors of the problem exist. In the first case models for the target speakers are built using a set of enrollment utterances and then test utterances are scored against these models. In this case the role of enrollment and test utterances are not symmetric. More recent approaches, instead, rely on background models of speaker and noise distributions to directly estimate log–likelihood ratios given the hypotheses that

both enrollment and test utterances are from the same speaker and the hypothesis that they are from different speakers. These systems, where enrollment and test utterances play the same role, prove to be more effective in terms of recognition accuracy.

The next sections focus on generative speaker recognition models.

3.2 Universal Background Models and GMMs

GMM–based systems use mixture models to represent the distribution $P(X|s)$ of observed acoustic features for a given speaker s. This distribution can be used to compute the likelihood of acoustic features for a test utterance, and the resulting likelihood can be interpreted as the probability that utterance $\mathcal{X}$ with associated acoustic observations $X = [x_1, \ldots, x_n]$ is spoken by speaker s [13], i.e.

$$\begin{aligned} \log P(\mathcal{X}|s) &= \log P(X|s) \\ &= \sum_{j=1}^{n} \log P(x_j|s) \end{aligned} \tag{3.1}$$

Log–likelihood ratios between target and non–target hypotheses can then be computed once we are able to evaluate the likelihood of the data under non–target hypothesis $P(X|\bar{s})$ (for the importance of log–likelihood ratios see [6]). However, the computation of this term poses some issued due to the difficulty in the definition of a suitable model. A possible solution consists in training separate *background* models for many different background speakers and combining their likelihoods [40, 41]. Another approach, called GMM–UBM [41, 42], pools together a large set of utterances coming from many different speakers and trains a single GMM over these data. The resulting GMM, referred to as Universal Background Model, can also be interpreted as representing the common characteristics of the acoustic space. Training the UBM is usually done by means of the EM algorithm (Section 2.2.2).

The simplest approach to estimate a speaker model in the GMM framework consists in training a GMM for the target speaker to maximize the likelihood of the observed features for this speaker [13]. Given enough training data this approach would accurately capture the speaker's characteristics. However, most of the time enrollment data is limited. In this case, better models can be built by Maximum–a–Posteriori (MAP) adaptation [43, 42] , i.e. by estimating the models incorporating prior information on the model parameters. If we denote $g(\theta)$ the prior probability of the parameter set θ, ML estimate evaluates the parameters as

$$\theta_{ML} = \arg\max_{\theta} P(X|\theta) \tag{3.2}$$

while MAP adaptation evaluates the parameters as [43]

$$\theta_{MAP} = \arg\max_{\theta} P(X|\theta)g(\theta) \tag{3.3}$$

Classical GMM MAP adaptation estimates the speaker dependent GMMs by adapting the speaker independent UBM. MAP estimates can be computed by means of the EM algorithm [43]. Usually only mean vector adaptation is performed, so that each speaker can be represented by the set of his adapted means and shares with all other speaker models the remaining UBM parameters. Define

$$N_i = \sum_{t=1}^{n} \gamma_{it} \tag{3.4}$$

$$F_{X,i} = \frac{1}{N_i} \sum_{t=1}^{n} \gamma_{it} x_t \tag{3.5}$$

where the terms γ_{it} are the occupation probabilities defined as

$$\gamma_{it} = \frac{w_i \mathcal{N}(x_t|\mu_i, \Sigma_i)}{\sum_j w_j \mathcal{N}(x_t|\mu_j, \Sigma_j)} \tag{3.6}$$

and μ_i, Σ_i are the UBM mean and covariance matrices. Relevance MAP [42] mean adaptation can be computed as

$$\hat{\mu}_i = \alpha_i F_{X,i} + (1 - \alpha_i)\mu_i \tag{3.7}$$

where $\hat{\mu}_i$ is the adapted mean for Gaussian i, μ_i is the UBM mean for Gaussian i and

$$\alpha_i = \frac{N_i}{N_i + r} \tag{3.8}$$

The term r is a fixed relevance factor that acts as a trade–off between the ML estimate and the UBM.

MAP adaptation provides a way to estimate speaker models when training data is scarce, while also allowing to asymptotically reach the ML solution when enough training data is available. However, when a speaker lacks observations for a particular mixture Gaussian, MAP adaptation is forced to fall back on the speaker–independent UBM supervector values for that component. A solution to this problem can then consist in performing Eigenvoice–MAP adaptation [44, 45] (Section 3.3.3).

3.3 Factor analysis models

In this Section we introduce a set of techniques based on Factor Analysis [12]. These techniques provide a probabilistic framework to perform MAP estimate of speaker–dependent GMMs while taking into account inter–session variability [46, 45, 47, 48]. JFA models were first introduced for HMMs. However, they can easily be extended

to Gaussian Mixture Models. In this and subsequent sections we will assume that acoustic features are F–dimensional column vectors. The number of components of the mixture models will be denoted as C.

Before venturing into the details of Factor Analysis models we give some definitions which will be used through this section.

3.3.1 Statistics and likelihoods

It is useful to define a set of statistics for a given utterance. Assume that, given a set of observations $X(s) = \{x_1, \dots x_n\}$ for speaker s, we have computed the alignment of X over the components of a GMM, i.e. we have associated each observation to a single mixture component. Let $X_i(s)$ denote the set of observations associated to component i. Following [47] we define the (Viterbi) zero–order statistic $N_i(s)$ as the number of frames associated to Gaussian i. We also denote with $F_i(s)$ and $F_{X,i}(s)$ the Viterbi first order and centered first order statistics respectively defined as

$$F_i(s) = \sum_{t|x_t \in X_i(s)} x_t \tag{3.9}$$

and

$$\begin{aligned} F_{X,i}(s) &= \sum_{t|x_t \in X_i(s)} (x_t - \mu_i) \\ &= F_i(s) - N_i(s)\mu_i \end{aligned} \tag{3.10}$$

where μ_i is the mean of the GMM used to estimate the statistics (usually the UBM), The Viterbi second order and centered second order statistics are given by

$$S_i(s) = \sum_{t|x_t \in X_i(s)} x_t x_t^T \tag{3.11}$$

and

$$\begin{aligned} S_{X,i}(s) &= \sum_{t|x_t \in X_i(s)} (x_t - \mu_i)(x_t - \mu_i)^T \\ &= S_i(s) - 2F_i(s)\mu_i + N_i(s)\mu_i\mu_i^T \end{aligned} \tag{3.12}$$

Finally, we denote by $\mathbf{N}(s)$ the $CF \times CF$ block–diagonal matrix whose blocks are $F \times F$ matrices $N_i\mathbf{I}$, by $\mathbf{F}(s)$ and $\mathbf{F}_X(s)$ the stacking of vectors $F(s)$ and $F_X(s)$ respectively and by $\mathbf{S}(s)$, $\mathbf{S}_X(s)$ the block–diagonal matrices whose elements are the matrices $S_i(s)$ and $S_{X,i}(s)$, respectively.

Given these statistics and the observations alignment the data log–likelihood for a given GMM supervector $\mathbf{g}$ can be expressed as

$$\begin{aligned}\log P(X) &= \sum_{c=1}^{C}\left[N_c(s)\log\frac{1}{(2\pi)^{\frac{F}{2}}|\Sigma_c|^{\frac{1}{2}}} - \frac{1}{2}\sum_{t|x_t\in X_i(s)}(x_t-\mu_c)^T\Sigma_c^{-1}(x_t-\mu_c)\right]\\ &= \sum_{c=1}^{C}\left[N_c(s)\log\frac{1}{(2\pi)^{\frac{F}{2}}|\Sigma_c|^{\frac{1}{2}}} - \frac{1}{2}\mathrm{tr}\left(\Sigma_c^{-1}S_{X,c}(s)\right)\right]\\ &\triangleq G_{\mathbf{\Sigma}}(s)\end{aligned} \tag{3.13}$$

Assume now that the alignment of features with Gaussian components is not known, so that the full GMM model has to be used in the likelihood computations. In this case, the structure of the log–likelihood would require as to deal with sums of exponentials inside a logarithm. An approximation of the correct likelihood, which still allows us to use the formal expressions given in this chapter, consists then in replacing the Viterbi statistics with Baum–Welch statistics [47], defined by

$$\gamma_{it} = \frac{w_i\mathcal{N}(x_t|\mu_i,\Sigma_i)}{\sum_j w_j\mathcal{N}(x_t|\mu_j,\Sigma_j)} \tag{3.14}$$

$$N_i(s) = \sum_{t=1}^{n}\gamma_{it} \tag{3.15}$$

$$F_i(s) = \sum_{t=1}^{n}\gamma_{it}x_t \tag{3.16}$$

$$S_i(s) = \sum_{t=1}^{n}\gamma_{it}x_tx_t^T \tag{3.17}$$

$$F_{X,i}(s) = F_X(s) - N_i(s)\mu_i \tag{3.18}$$

$$S_{X,i}(s) = S_i(s) - 2F_i(s)\mu_i + N_i(s)\mu_i\mu_i^T \tag{3.19}$$

Again, we denote by $\mathbf{N}(s)$ the $CF\times CF$ block–diagonal matrix whose blocks are $F\times F$ matrices $N_i\mathbf{I}$, by $\mathbf{F}(s)$ and $\mathbf{F}_X(s)$ the stacked vectors of $F(s)$ and $F_X(s)$ respectively and by $\mathbf{S}(s)$, $\mathbf{S}_X(s)$ the block–diagonal matrices whose elements are the matrices $S_i(s)$ and $S_{X,i}(s)$ respectively. In this case, the likelihood is still given by

$$\begin{aligned}\log P(X) &= \sum_{c=1}^{C}\left[N_c(s)\log\frac{1}{(2\pi)^{\frac{F}{2}}|\Sigma_c|^{\frac{1}{2}}} - \frac{1}{2}\mathrm{tr}\left(\Sigma_c^{-1}S_{X,c}(s)\right)\right]\\ &\triangleq G_{\mathbf{\Sigma}}(s)\end{aligned} \tag{3.20}$$

where the Viterbi statistics are replaced by the Baum–Welch statistics. In the following we will use Baum–Welch statistics and assume a likelihood of the form (3.20).

3.3.2 MAP adaptation

An interesting interpretation of MAP adaptation in terms of hidden variable models has been given in [46]. Consider the following latent variable model

$$\mathbf{g}(s) = \boldsymbol{\mu} + \mathbf{D}\mathbf{z}(s) \tag{3.21}$$

where $\mathbf{g}(s)$ is the speaker–dependent supervector, $\boldsymbol{\mu}$ is a supervector (usually it is assumed to be equal to the UBM supervector), $\mathbf{D}$ is a diagonal $CF \times CF$ matrix and $\mathbf{z}(s)$ are CF speaker–dependent hidden variables. Let the prior distribution for $\mathbf{z}(s)$ be standard normal

$$\mathbf{z}(s) \sim \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I}) \tag{3.22}$$

The posterior for $\mathbf{z}(s)$ can be obtained as follows.

Let $\mathbf{N}(s)$ and $\mathbf{F}(s)$ denote the zero and centered first–order statistics for speaker s, defined as in 3.3.1. Dropping the reference to s, the joint likelihood of the observed data X and the hidden variables is given by

$$P(X, \mathbf{z}) = P(X|\mathbf{z})P(\mathbf{z}) \propto \left(\mathbf{z}^T\mathbf{D}^T\boldsymbol{\Sigma}^{-1}\mathbf{F}_X - \frac{1}{2}\mathbf{z}^T\mathbf{D}^T\boldsymbol{\Sigma}^{-1}\mathbf{N}\mathbf{D}\mathbf{z} - \frac{1}{2}\mathbf{z}^T\mathbf{z}\right) \tag{3.23}$$

The posterior for $\mathbf{z}$ is then

$$P(\mathbf{z}|X) \propto P(X, \mathbf{z}) \propto \left(\mathbf{z}^T\mathbf{D}^T\boldsymbol{\Sigma}^{-1}\mathbf{F}_X - \frac{1}{2}\mathbf{z}^T\mathbf{D}^T\boldsymbol{\Sigma}^{-1}\mathbf{N}\mathbf{D}\mathbf{z} - \frac{1}{2}\mathbf{z}^T\mathbf{z}\right) \tag{3.24}$$

By inspection we recover that the posterior for $\mathbf{z}$ is also Gaussian

$$\mathbf{z}|X \sim \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_\mathbf{z}, \Lambda_\mathbf{z}^{-1}) \tag{3.25}$$

with mean and precision matrix given by

$$\Lambda_\mathbf{z} = \left(\mathbf{D}^T\boldsymbol{\Sigma}^{-1}\mathbf{N}\mathbf{D} + \mathbf{I}\right) = \left(\mathbf{D}^2\boldsymbol{\Sigma}^{-1}\mathbf{N} + \mathbf{I}\right) \tag{3.26}$$

$$\boldsymbol{\mu}_\mathbf{z} = \Lambda_\mathbf{z}^{-1}\mathbf{D}^T\boldsymbol{\Sigma}^{-1}\mathbf{F}_X \tag{3.27}$$

The mean of the posterior for the speaker dependent GMM $\mathbf{g}$ is then

$$\mathbb{E}[\mathbf{g}] = \boldsymbol{\mu} + \mathbf{D}\mathbb{E}[\mathbf{z}] = \boldsymbol{\mu} + \left(\mathbf{D}^2\boldsymbol{\Sigma}^{-1}\mathbf{N} + \mathbf{I}\right)^{-1}\mathbf{D}^2\boldsymbol{\Sigma}^{-1}\mathbf{F}_X \tag{3.28}$$

Let $\mathbf{r} = \mathbf{D}^{-2}\boldsymbol{\Sigma}$. We can rewrite the adapted GMM as

$$\mathbb{E}[\mathbf{g}] = \boldsymbol{\mu} + \mathbf{D}\mathbb{E}[\mathbf{z}] = \boldsymbol{\mu} + (\mathbf{N} + \mathbf{r})^{-1}\mathbf{F}_X \tag{3.29}$$

Assuming that the first order statistics are the Baum–Welch statistics, this corresponds to the MAP adaptation formula (3.7).

3.3.3 Eigenvoice models

A different approach to MAP estimation of GMM supervectors, more suitable when sparse enrollment data is available, consists in eigenvoice MAP adaptation [44, 45]. The eigenvoice MAP model assumes that the covariance matrix of speaker GMMs has a low rank compared to the supervector space dimensions. Speaker GMMs are then constrained by the model to live in a low–dimensional manifold corresponding to a translation of the range subspace of the supervectors covariance matrix. The name eigenvoice comes from the fact that this subspaces is identified by the eigenvectors of the covariance matrix corresponding to non–zero eigenvalues, which are referred to as eigenvoices. The model can be formalized as

$$\mathbf{g}(s) = \boldsymbol{\mu} + \mathbf{V}\mathbf{y}(s) \tag{3.30}$$

where $\mathbf{V}$ is an $CF \times M$ matrix, with $M \ll CF$. As in classical MAP, $\mathbf{y}(s)$ is a latent variable assumed to have a standard Gaussian prior distribution. However, compared to classical MAP, the dimension of $\mathbf{y}(s)$ is much smaller than the dimension of $\mathbf{z}(s)$. Again, we can compute the posterior of $\mathbf{y}(s)$ for a given speaker. Since the derivations are almost the same as for classical MAP, we just give the final expression for the posterior

$$\mathbf{y}|X \sim \mathcal{N}(\mathbf{y}|\boldsymbol{\mu}_\mathbf{y}, \Lambda_\mathbf{y}^{-1}) \tag{3.31}$$

with mean and precision matrix given by

$$\Lambda_\mathbf{y} = \left(\mathbf{V}^T\boldsymbol{\Sigma}^{-1}\mathbf{N}\mathbf{V} + \mathbf{I}\right) \tag{3.32}$$

$$\boldsymbol{\mu}_\mathbf{y} = \Lambda_\mathbf{y}^{-1}\mathbf{V}^T\boldsymbol{\Sigma}^{-1}\mathbf{F}_X \tag{3.33}$$

3.3.4 Eigenchannels

So far we have not considered the effects of session mismatch between recordings, i.e. we have presented models which do not explicitly model inter–session variability. Inter–session variability is assumed to be mostly due to channel effects, although other factors may be responsible for differences between enrollment and test utterances. We therefore refer to these kinds of noise as channel effects. While techniques exist to perform channel compensation directly in the feature space the Factor Analysis framework allows performing model–level compensation.

A popular set of techniques to compensate channel effects for GMM–based models consists in feature mapping [49]. These techniques can be interpreted as performing model–based channel compensation under the assumption that a GMM supervector for a given recording h can be expressed as the contribution of two terms [47]

$$\mathbf{g}_h(s) = \mathbf{s}(s) + \mathbf{c}_h(s) \tag{3.34}$$

where $\mathbf{s}(s)$ denotes a speaker–dependent component and $\mathbf{c}_h(s)$ denotes the channel component for recording h. Again, different approaches can be used to estimate and compensate channel effects. One of them is eigenchannel adaptation [44]. This technique assumes that the channel component $\mathbf{c}_h(s)$ lies in a small dimensional subspace [44, 46, 47], i.e.

$$\mathbf{c}_h(s) = \mathbf{U}\mathbf{x}_h(s) \tag{3.35}$$

where $\mathbf{U}$ is a rectangular $CF \times R_C$ matrix with $R_C \ll CF$ and $\mathbf{x}_h(s)$ is a standard normal distributed hidden variable representing the channel effects.

3.3.5 Joint factor analysis of speaker and channel

Classical, eigenvoice and eigenchannel MAP can be combined in a single model, referred to as Joint Factor Analysis (JFA) of speaker and session variability [46, 48]. JFA provides a unified framework to estimate model parameters and posterior distributions for the different components.

The JFA model can be summarized as

$$\mathbf{g}_h(s) = \boldsymbol{\mu} + \mathbf{V}\mathbf{y}(s) + \mathbf{U}\mathbf{x}_h(s) + \mathbf{D}\mathbf{z}(s) \tag{3.36}$$

where the different terms have the same meaning as in the previous sections. We will therefore refer to $\mathbf{y}(s)$ as speaker factor, to $\mathbf{x}(s)$ as channel factors and to $\mathbf{z}(s)$ as common factors. Different strategies to learn matrices $\mathbf{V}$, $\mathbf{U}$ and $\mathbf{D}$ have been proposed, for example decoupled estimation of the different terms, to simplify the computational load of training the parameters. The JFA framework allows to directly perform channel compensation by integrating out the channel effects as in [47]. However, classical JFA is very demanding from a computational point of view. Different techniques have thus been proposed which make use of the JFA model to perform feature level channel compensation [50, 51], and which approximate the JFA scoring with different and faster techniques [52, 51]. After the introduction of i–vectors (Section 3.5), however, the popularity of these techniques has lowered. Since most state–of–the–art speaker recognition systems are nowadays based on i–vector technology, we skip the details of the different flavors of JFA and refer to [53] for a much broader overview of JFA–based techniques. In the following, we will only focus on the aspects of JFA connected to i–vectors.

3.4 Front–end JFA

JFA provides a framework for speaker modeling and scoring of test utterances while taking into account channel effects. In recent years, however, the focus has shifted from JFA as a full framework to JFA as a feature extractor. The reasons can be

traced back to the success of discriminative techniques in the GMM space, which can benefit from the low dimensionality of speaker factors.

In [54] the authors propose to combine the JFA models with discriminative classifiers based on Support Vector Machines (SVM). This approach has been motivated by the success of GMM supervector–based SVM systems with Nuisance Attribute Projection (NAP) [55] to compensate for channel effects (Section 5.1). The systems proposed in [54] are conceptually very similar to the GMM supervectors – SVM approach [56, 55]. However, instead of training a SVM in the GMM space the classifier is trained in the much smaller speaker factors space. Nuisance Attribute Projection is replaced by a two–step channel compensation which makes use of JFA channel factors estimates (i.e. both speaker and channel factors are estimated) and then applies within–class covariance normalization [57] in the speaker factors space. Finally, the authors show how a simple linear kernel or a cosine distance kernel allow obtaing results which are comparable to classical JFA–based scoring in terms of accuracy with a much lower computational load.

The success of JFA as front–end for discriminative speaker identification motivated the development of similar approaches for language verification [58]. These techniques apply JFA models to language GMMs, assuming that speaker information can be mostly compensated in the feature space so that the factors $\mathbf{y}$ of the JFA model (3.36) can be interpreted as language factors. More details about this approach are given in Section 4.2.2.

3.5 I–vectors

JFA–based speaker factors proved to be effective at modeling speakers by means of low–dimensional vectors. JFA provides an explicit mechanism to estimate and compensate channel effects. It was shown, however, that channel factors can contain useful speaker information, although the model assumes that channel factors model only channel and noise effects [59]. To overcome this problem the JFA model was adapted so that a single subspace is estimated, which is assumed to contain most of the useful information [60, 61]. Channel compensation is then deferred to successive post–processing (e.g. Within–Class Covariance Normalization [62, 3]).

The underlying latent variable model can be formalized as

$$\mathbf{g}(s) = \boldsymbol{\mu} + \mathbf{T}\mathbf{w}(s) \tag{3.37}$$

where $\mathbf{T}$ is a $CF \times M$ matrix whose columns span the subspace where the GMMs are assumed to live (and should include both channel and speaker information) and $\mathbf{w}(s)$ is a latent variable representing a GMM in the subspace associated to $\mathbf{T}$. The $\mathbf{T}$ matrix is sometimes referred to as total variability matrix. As in previous JFA models, the prior for $\mathbf{w}$ is assumed to be standard normal distributed. The

point–estimate of the posterior is called i–vector (in some cases we will use the term i–vector also to refer to the hidden variable $\mathbf{w}$).

The i–vector model is formally equivalent to the eigenvoice model of Section 3.3.3. However, while in eigenvoice MAP recordings from the same speakers are assumed to share the same values for the hidden variables, the i–vector approach assumes that each recording has its own hidden variables.

3.5.1 I–vector posterior distribution

The derivations of the posterior distribution for an i–vector given the $\mathbf{T}$ matrix and assuming a standard normal prior are very similar to the ones given in Section 3.3.2 for classical MAP. However, we briefly recall some aspects of this derivation which will be useful in the next sections.

Again, let $\mathbf{N}(s)$ and $\mathbf{F}_X(s)$ denote the zero and centered first–order statistics for a given utterance, defined as in 3.3.1. Denoting with $\mathbf{T}_c$ the $F \times M$ matrix whose entries are the elements of $\mathbf{T}$ corresponding to Gaussian c, the conditional likelihood of the observed data X given an i–vector $\mathbf{w}$ and the subspace matrix $\mathbf{T}$ is given by

$$\begin{aligned}
\log P(X|\mathbf{w},\mathbf{T}) &= \sum_{c=1}^{C}\left[N_c(s)\log\frac{1}{(2\pi)^{\frac{F}{2}}|\Sigma_c|^{\frac{1}{2}}}\right. \\
&\quad \left.-\frac{1}{2}\sum_t (x_t-\mu_c-\mathbf{T}_c\mathbf{w})^T\Sigma_c^{-1}(x_t-\mu_c-\mathbf{T}_c\mathbf{w})\right] \\
&= \sum_{c=1}^{C}\left[N_c(s)\log\frac{1}{(2\pi)^{\frac{F}{2}}|\Sigma_c|^{\frac{1}{2}}}-\frac{1}{2}\mathrm{tr}(S_{X,c}(s)\Sigma_c^{-1})\right. \\
&\quad \left.-\frac{1}{2}\left(-2\mathbf{w}^T\mathbf{T}_c^T\Sigma_c^{-1}F_{X,c}(s)+N_c(s)\mathbf{w}^T\mathbf{T}_c^T\Sigma_c^{-1}\mathbf{T}_c\mathbf{w}^T\right)\right] \\
&= G_{\mathbf{\Sigma}}(s)+\mathbf{w}^T\mathbf{T}^T\mathbf{\Sigma}^{-1}\mathbf{F}_X(s)-\frac{1}{2}\mathbf{w}^T\mathbf{T}^T\mathbf{N}(s)\mathbf{\Sigma}^{-1}\mathbf{T}\mathbf{w} \qquad (3.38)
\end{aligned}$$

The joint likelihood of the observed data X and the hidden variable is given by

$$\begin{aligned}
P(X,\mathbf{w}|\mathbf{T}) &= P(X|\mathbf{w},\mathbf{T})P(\mathbf{w}) \\
&= K_{\mathbf{\Sigma}}\exp\left(\mathbf{w}^T\mathbf{T}^T\mathbf{\Sigma}^{-1}\mathbf{F}_X-\frac{1}{2}\mathbf{w}^T\mathbf{T}^T\mathbf{\Sigma}^{-1}\mathbf{N}\mathbf{T}\mathbf{w}-\frac{1}{2}\mathbf{w}^T\mathbf{w}\right) \qquad (3.39)
\end{aligned}$$

where the term $K_{\mathbf{\Sigma}}$ is a normalization term which does not depend on $\mathbf{w}$ and on $\mathbf{T}$. Thus, the posterior for $\mathbf{w}$ is proportional to

$$P(\mathbf{w}|X)\propto P(X,\mathbf{w})\propto\exp\left(\mathbf{w}^T\mathbf{T}^T\mathbf{\Sigma}^{-1}\mathbf{F}_X-\frac{1}{2}\mathbf{w}^T\mathbf{T}^T\mathbf{\Sigma}^{-1}\mathbf{N}\mathbf{D}\mathbf{w}-\frac{1}{2}\mathbf{w}^T\mathbf{w}\right) \qquad (3.40)$$

that is, the posterior for $\mathbf{w}$ is Gaussian distributed

$$\mathbf{w}|X \sim \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}_{\mathbf{w}}, \Lambda_{\mathbf{w}}^{-1}) \tag{3.41}$$

with mean and precision matrix given by

$$\Lambda_{\mathbf{w}} = \left(\mathbf{T}^T\boldsymbol{\Sigma}^{-1}\mathbf{N}\mathbf{T} + \mathbf{I}\right) \tag{3.42}$$

$$\boldsymbol{\mu}_{\mathbf{w}} = \Lambda_{\mathbf{w}}^{-1}\mathbf{T}^T\boldsymbol{\Sigma}^{-1}\mathbf{F}_X \tag{3.43}$$

The i–vector corresponds to the MAP estimate of $\mathbf{w}$, i.e. it is given by the mode of the posterior distribution for $\mathbf{w}$.

3.5.2 Training the T matrix

Similar to classical JFA, training of the i–vector extractor is usually done through Maximum–Likelihood. While the model parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ can, in theory, be also re–estimated, we don't have evidence that such re–estimation significantly improves i–vector extraction in terms of recognition accuracy. Moreover, a full re–estimation would entail a bigger computational load. Therefore, we will restrict our analysis to training $\mathbf{T}$ matrix only. The UBM mean $\boldsymbol{\mu}_{ubm}$ and covariance matrices $\boldsymbol{\Sigma}_{ubm}$ will be used for the terms $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ of the model. Moreover, we assume that statistics are always computed from the UBM.

The ML estimate of $\mathbf{T}$ can be computed by means of the EM algorithm [45]. We already showed that the posterior for $\mathbf{w}$ corresponds to a multivariate Gaussian with mean and covariance matrix given by (3.41). In order to derive the EM update formulas we define the EM auxiliary function using $\mathbf{w}$ as hidden variables [45]. The EM function to maximize is given by

$$Q(\mathbf{T}|\mathbf{T}^{old}) = \sum_s \mathbb{E}_{\mathbf{w}|X(s),\mathbf{T}^{old}}[\log P(X(s), \mathbf{w}|\mathbf{T})] \tag{3.44}$$

where $P(X(s), \mathbf{z}|\mathbf{T}^{old})$ is the joint probability of the observations $X(s)$ and the hidden variables (here s denotes an utterance, not a speaker). Since for each utterance the joint probability (3.39) can be written as

$$P(X(s), \mathbf{w}|\mathbf{T}) = \log P(X(s)|\mathbf{w}, \mathbf{T}) + \log P(\mathbf{w}) \tag{3.45}$$

and

$$P(\mathbf{w}) \sim \mathcal{N}(0, \mathbf{I}) \tag{3.46}$$

then maximizing (3.44) is equivalent to maximizing

$$\sum_s \mathbb{E}_{\mathbf{w}|X(s),\mathbf{T}^{old}}[P(X(s), |\mathbf{z}, \mathbf{T}^{old})] = \sum_s \int \log P(X(s)|\mathbf{w}, \mathbf{T}) P(\mathbf{w}|X(s), \mathbf{T}^{old}) \tag{3.47}$$

Considering the expression for the likelihood (3.39), we know that the term $K_{\mathbf{\Sigma}}$ doesn't depend on $\mathbf{T}$, thus the maximizer of (3.47) corresponds to the maximizer of

$$\sum_s \mathbb{E}_{\mathbf{w}|X(s),\mathbf{T}^{old}} \left[H_{\mathbf{T},\mathbf{\Sigma}}(s,\mathbf{w}) \right] \tag{3.48}$$

$$= \sum_s \mathbb{E}_{\mathbf{w}|X(s),\mathbf{T}^{old}} \left[\mathbf{w}^T \mathbf{T}^T \mathbf{\Sigma}^{-1} \mathbf{F}_X - \frac{1}{2} \mathbf{w}^T \mathbf{T}^T \mathbf{N} \mathbf{\Sigma}^{-1} \mathbf{T} \mathbf{w} \right] \tag{3.49}$$

which, dropping the reference to $\left(\mathbf{w}|X(s), \mathbf{T}^{old}\right)$ for sake of readability, corresponds to

$$\sum_s \mathrm{tr} \left[\mathbf{\Sigma}^{-1} \left(\mathbf{F}_X \mathbb{E} \left[\mathbf{w}^T \right] \mathbf{T}^T - \frac{1}{2} \mathbf{N} \mathbf{T} \mathbb{E} \left[\mathbf{w} \mathbf{w}^T \right] \mathbf{T}^T \right) \right] \tag{3.50}$$

In order to maximize this function we take the derivatives with respect to the elements of $\mathbf{T}$ and equal them to zero. In matrix form, we need to solve

$$\frac{\partial}{\partial \mathbf{T}} \sum_s \mathrm{tr} \left[\mathbf{\Sigma}^{-1} \left(\mathbf{F}_X \mathbb{E} \left[\mathbf{w}^T \right] \mathbf{T}^T - \frac{1}{2} \mathbf{N} \mathbf{T} \mathbb{E} \left[\mathbf{w} \mathbf{w}^T \right] \mathbf{T}^T \right) \right] = \mathbf{0} \tag{3.51}$$

that is,

$$\sum_s \mathbf{\Sigma}^{-1} \left(\mathbf{F}_X \mathbb{E} \left[\mathbf{w}^T \right] - \mathbf{N} \mathbf{T} \mathbb{E} \left[\mathbf{w} \mathbf{w}^T \right] \right) = 0 \tag{3.52}$$

The solution is given by

$$\mathbf{T}^i \sum_s \mathbf{N}^i(s) \mathbb{E}[\mathbf{w}(s) \mathbf{w}^T(s)] = \sum_s \mathbf{F}_X^i(s) \mathbb{E}[\mathbf{w}^T(s)] \tag{3.53}$$

where $\mathbf{T}^i$, $\mathbf{F}_X^i(s)$ and $\mathbf{N}^i(s)$ are the i-th row of $\mathbf{T}$, $\mathbf{F}_X(s)$ and $\mathbf{N}(s)$ respectively. The solution for $\mathbf{T}$ can be found by simply solving the linear systems (3.53), where the expectation $\mathbb{E}[\mathbf{w}]$ is computed as the mean of the posterior of $\mathbf{w}$ given $\mathbf{T}^{old}$ and

$$\mathbb{E}[\mathbf{w}\mathbf{w}^T] = \mathbb{E}[\mathbf{w}] \mathbb{E}[\mathbf{w}^T] + \Lambda_{\mathbf{w}}^{-1} \tag{3.54}$$

Again, $\Lambda_{\mathbf{w}}$ is the covariance matrix of the posterior for $\mathbf{w}$ given $\mathbf{T}^{old}$.

3.5.3 Speeding up the i–vector extraction

I–vectors extraction as detailed in the previous section requires the computation of the full covariance matrix of the posterior. The time complexity of a naïve implementation of (3.42) requires $O(CFM^2)$ operations and, in practice, can represent a bottleneck in the i–vector extraction process. A faster implementation can be obtained by observing that the computation of the posterior covariance can be expressed as the combination of C $M \times M$ matrices as

$$\Lambda_{\mathbf{w}} = \left(\mathbf{T}^T \mathbf{\Sigma}^{-1} \mathbf{N} \mathbf{T} + \mathbf{I} \right) = \mathbf{I} + \sum_{c=1}^{C} \mathbf{T}_c^T \Sigma_c^{-1} N_c \mathbf{T}_c \tag{3.55}$$

where, again, $\mathbf{T}_c$ is the part of matrix $\mathbf{T}$ associated to Gaussian c, Σ_c is the UBM covariance matrix for Gaussian c and N_c is the zero–order statistics for Gaussian c. Since we can pre–compute matrices $\mathbf{T}_c^T \Sigma_c \mathbf{T}_c$, this approach allows reducing the time complexity to $O(CM^2)$, however the memory requirements increase from $O(CMF)$ to $O(CMF + CM^2)$, making this approach less attractive for limited–memory devices. Moreover, both approaches still need $O(M^3)$ time to perform matrix inversion and $O(CMF)$ to compute the i–vector given the covariance matrix. In this section we analyze a set of techniques which allow to improve i–vector extraction performance in terms of required time and memory, while at the same time trying to keep the i–vector accuracy as high as possible.

Before describing the two main techniques which allow achieving these goals, we note that we can apply a transformation to first–order statistics and to the T matrix which allows to effectively remove all $\mathbf{\Sigma}$ terms from the i–vector posterior expressions. The transformation is given by

$$\widetilde{\mathbf{F}}_{X,c} = \mathbf{\Sigma}_c^{-\frac{1}{2}} \mathbf{F}_{X,c} \tag{3.56}$$

$$\widetilde{\mathbf{T}}_c = \mathbf{\Sigma}_c^{-\frac{1}{2}} \mathbf{T}_c \tag{3.57}$$

where $\mathbf{\Sigma}_c^{-\frac{1}{2}}$ can be computed from the Cholesky decomposition of $\mathbf{\Sigma}_c$. In this way, the posterior for the i–vector is given by

$$\mathbf{w}|X \sim \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}_\mathbf{w}, \Lambda_\mathbf{w}^{-1}) \tag{3.58}$$

with mean and precision matrix given by

$$\Lambda_\mathbf{w} = \left(\widetilde{\mathbf{T}}^T \mathbf{N} \widetilde{\mathbf{T}} + \mathbf{I}\right) = \mathbf{I} + \sum_{c=1}^{C} N_c \widetilde{\mathbf{T}}_c^T \widetilde{\mathbf{T}}_c \tag{3.59}$$

$$\boldsymbol{\mu}_\mathbf{w} = \Lambda_\mathbf{w}^{-1} \widetilde{\mathbf{T}}^T \widetilde{\mathbf{F}} \tag{3.60}$$

Diagonalized i–vectors

The first approach we consider tries to reduce all the $\widetilde{\mathbf{T}}_c^T \widetilde{\mathbf{T}}_c$ matrices to diagonal form, so that the memory requirements for storing such matrices becomes $O(CM)$ and the time required to compute the covariance matrix becomes $O(CM)$ including the time required for the inversion. The extraction time would therefore be dominated by the $O(CMF)$ operations required to compute the i–vector mean given the covariance matrix $\Lambda_\mathbf{y}^{-1}$.

In order to achieve this goal the authors of [63] assume that all the $\widetilde{\mathbf{T}}_c^T \widetilde{\mathbf{T}}_c$ matrices can be simultaneously diagonalized, i.e. they assume the existence of an (orthogonal) matrix $\mathbf{Q}$ such that

$$\mathbf{Q}^{-1} \widetilde{\mathbf{T}}_c^T \widetilde{\mathbf{T}}_c \mathbf{Q} \tag{3.61}$$

is diagonal for any c. While this is not true in practice, the authors propose a way to estimate an orthogonal projection matrix such that the $\widetilde{\mathbf{T}}_c^T\widetilde{\mathbf{T}}_c$ are "almost" diagonal, so that approximating them with their diagonal does not cause too much degradation of the i–vectors. The eigen–decomposition of the weighted average covariance matrix

$$\mathbf{W} = \sum_{c=1}^{C} \omega_c \widetilde{\mathbf{T}}_c^T \widetilde{\mathbf{T}}_c = \mathbf{Q}\mathbf{\Lambda}_\mathbf{W}\mathbf{Q}^T \tag{3.62}$$

where ω_c is the weight of the c–th GMM in the UBM supervector and $\mathbf{Q}\mathbf{\Lambda}_\mathbf{W}\mathbf{Q}^T$ is the eigenvalue decomposition of $\mathbf{W}$, allows obtaining a suitable orthogonal transformation in this sense [63]. Therefore, $\Lambda_\mathbf{y}$ can be computed as

$$\Lambda_\mathbf{y} = \mathbf{I} + \sum_{c=1}^{C} N_c \widetilde{\mathbf{T}}_c^T \widetilde{\mathbf{T}}_c \approx \mathbf{Q}\left(\mathbf{I} + \sum_{c=1}^{C} N_c \mathbf{D}_c\right)\mathbf{Q}^T \tag{3.63}$$

where

$$\mathbf{D}_c = \mathbf{I} \circ \mathbf{Q}^T \widetilde{\mathbf{T}}_c^T \widetilde{\mathbf{T}}_c \mathbf{Q} \tag{3.64}$$

Note that $\mathbf{D}_c$ is diagonal, therefore the computation of $\Lambda_\mathbf{y}$ only requires to compute a linear combination of C M–dimensional vectors. The inverse of $\Lambda_\mathbf{y}$ can be computed as

$$\Lambda_\mathbf{y}^{-1} \approx \mathbf{Q}\left(\mathbf{I} + \sum_{c=1}^{C} \mathbf{D}_c\right)^{-1}\mathbf{Q}^T \tag{3.65}$$

therefore requiring only the inversion of a diagonal matrix.

While this technique allows very fast i–vector extraction with low memory requirements (the required memory is essentially the memory needed to store the $\widetilde{\mathbf{T}}$ matrix), the resulting i–vectors prove to be slightly inferior in terms of performance when used as features for state–of–the–art systems such as PLDA.

In [63] the authors also propose better estimates of the $\mathbf{Q}$ matrix based in Heteroscedastic Linear Discriminant Analysis [64, 65]. This technique allows improving the i–vector quality, however some degree of performance degradation is still present.

Variational Bayes approximation of the posterior

Another set of i–vector extraction techniques which allow for faster and accurate i–vectors are based on variational approximations of the i–vector posteriors [12]. Variational inference offers a framework to estimate an approximation of the posterior distribution of random variables given some data which are optimal in the sense that they minimize the KL divergence between the approximation and the true posterior [12]. The approach is similar to the Expectation Maximization algorithm: given a random variable $\mathbf{X}$ and hidden variables $\mathbf{Z}$ we decompose the log–likelihood of the data as

$$\log P(\mathbf{X}) = \mathcal{L}(q) + D(q\|p) \tag{3.66}$$

where

$$\mathcal{L}(q) = \int q(\mathbf{Z}) \log \frac{P(\mathbf{X}, \mathbf{Z})}{q(\mathbf{Z})} d\mathbf{Z} \tag{3.67}$$

and

$$D(q\|p) = -\int q(\mathbf{Z}) \log \frac{p(\mathbf{Z}|\mathbf{X})}{q(\mathbf{Z})} d\mathbf{Z} \tag{3.68}$$

is the KL divergence between the approximated posterior $q(\mathbf{Z})$ and the true posterior $p(\mathbf{Z}|\mathbf{X})$. The constraints on the form of the posterior allow us to estimate approximations which might be more tractable than the true posterior.

In order to extract i–vectors we assume a variational approximation of the posterior $q(\mathbf{w}) \approx P_{\mathbf{T}}(\mathbf{w}|X)$ such that

$$q(\mathbf{w}) = \prod_i q_i(\mathbf{w}_i) \tag{3.69}$$

that is, an approximation which factorizes over the different i–vector dimensions. The VB solution for the distributions $q_i(\mathbf{w}_i)$ is given by

$$\log q_i(\mathbf{w}_i) = \mathbb{E}_{j \neq i} \left[\log P(X, \mathbf{w})\right] + c \tag{3.70}$$

for some constant c [12]. The expectation is taken over all variables $\mathbf{w}_j$ with $j \neq i$. Let

$$\begin{aligned} \overline{\boldsymbol{w}}_i &= [\boldsymbol{w}_1, \dots \boldsymbol{w}_{i-1}, \boldsymbol{w}_{i+1}, \dots \boldsymbol{w}_M]^T \\ \overline{\boldsymbol{T}}_i &= [\boldsymbol{T}_1, \dots \boldsymbol{T}_{i-1}, \boldsymbol{T}_{i+1} \dots \boldsymbol{T}_M] \end{aligned} \tag{3.71}$$

so that the product $\boldsymbol{T}\boldsymbol{w}$ can be written as

$$\boldsymbol{T}\boldsymbol{w} = \overline{\boldsymbol{T}}_i \overline{\boldsymbol{w}}_i + \boldsymbol{T}_i \boldsymbol{w}_i \tag{3.72}$$

Substituting the expression for the joint likelihood of the data and the hidden variables in (3.70) and ignoring all terms which do not depend on $\mathbf{w}$ we obtain

$$\begin{aligned} \log q(\boldsymbol{w}_i) = \; & \mathbb{E}_{j \neq i} \Big[\overline{\boldsymbol{T}}_i \overline{\boldsymbol{w}}_i \boldsymbol{\Sigma}^{-1} \mathbf{F}_X + \boldsymbol{T}_i \boldsymbol{w}_i \boldsymbol{\Sigma}^{-1} \boldsymbol{F}_X \\ & - \frac{1}{2} \overline{\boldsymbol{w}}_i^T \overline{\boldsymbol{T}}_i^T \mathbf{N} \boldsymbol{\Sigma}^{-1} \overline{\boldsymbol{T}}_i \overline{\boldsymbol{w}}_i - \boldsymbol{w}_i \boldsymbol{T}_i \mathbf{N} \boldsymbol{\Sigma}^{-1} \overline{\boldsymbol{T}}_i \overline{\boldsymbol{w}}_i \\ & - \frac{1}{2} \boldsymbol{w}_i^T \boldsymbol{T}_i^T \mathbf{N} \boldsymbol{\Sigma}^{-1} \boldsymbol{T}_i \boldsymbol{w}_i - \frac{1}{2} \overline{\boldsymbol{w}}_i^T \overline{\boldsymbol{w}}_i - \frac{1}{2} \boldsymbol{w}_i^T \boldsymbol{w}_i \Big] + c \end{aligned} \tag{3.73}$$

By inspection of (3.73) we can notice that the distribution $q_i(\mathbf{w}_i)$ is again Gaussian

$$q(\boldsymbol{w}_i) \sim \mathcal{N}(\boldsymbol{w}_i | \boldsymbol{\mu}_i, \boldsymbol{\Lambda}_i^{-1}) \tag{3.74}$$

with mean and precision matrix given by

$$\boldsymbol{\Lambda}_i = \left(\boldsymbol{T}_i^T \mathbf{N} \boldsymbol{\Sigma}^{-1} \boldsymbol{T}_i + \mathbf{I}\right) \tag{3.75}$$

and

$$\begin{aligned} \boldsymbol{\mu}_i &= \boldsymbol{\Lambda}_i^{-1} \boldsymbol{T}_i^T \boldsymbol{\Sigma}^{-1} \left(\boldsymbol{F}_X - \mathbf{N} \overline{\boldsymbol{T}}_i \mathbb{E}\left[\overline{\boldsymbol{w}}_i\right]\right) \\ &= \boldsymbol{\Lambda}_i^{-1} \boldsymbol{T}_i^T \boldsymbol{\Sigma}^{-1} \left(\boldsymbol{F}_X - \mathbf{N} \overline{\boldsymbol{T}}_i \overline{\boldsymbol{\mu}}_i\right) \end{aligned} \tag{3.76}$$

Variational Bayes thus allows the posterior distribution to be computed by iterating the estimation of the means of the different distributions $q_i(\mathbf{w}_i)$. Denoting $\overline{\boldsymbol{F}}_{X,i}$ the first–order statistics centered around the new supervector mean $\mathbf{m} + \overline{\boldsymbol{T}}_i \overline{\boldsymbol{\mu}}_i$, $\boldsymbol{\mu}_i$ can be computed as

$$\boldsymbol{\mu}_i = \boldsymbol{\Lambda}_i^{-1} \boldsymbol{T}_i^T \boldsymbol{\Sigma}^{-1} \overline{\boldsymbol{F}}_{X,i} \tag{3.77}$$

Observe that similar results can be obtained by considering a factorized distribution where the $\mathbf{w}_i$ are blocks of variables instead of single variables. The formal expression for the posterior remains the same.

Given enough iterations, this approach converges to the original i–vector estimate. We can rewrite equation (3.76) as

$$\mathbb{E}\left[\boldsymbol{w}_i\right] = \boldsymbol{\Lambda}_i^{-1} \boldsymbol{T}_i^T \boldsymbol{\Sigma}_i^{-1} (\boldsymbol{F}_X - \boldsymbol{N} \boldsymbol{T} \mathbb{E}\left[\boldsymbol{w}\right] + \boldsymbol{N} \boldsymbol{T}_i \mathbb{E}\left[\boldsymbol{w}_i\right]) \tag{3.78}$$

Multiplying both sides by $\boldsymbol{\Lambda}_i$ and rearranging we obtain

$$\boldsymbol{T}_i^T \boldsymbol{N} \boldsymbol{\Sigma}^{-1} \boldsymbol{T} \mathbb{E}\left[\boldsymbol{w}\right] + \left(\boldsymbol{\Lambda}_i - \boldsymbol{T}_i^T \boldsymbol{N} \boldsymbol{\Sigma}^{-1} \boldsymbol{T}_i\right) \mathbb{E}\left[\boldsymbol{w}_i\right] = \boldsymbol{T}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{F}_X \tag{3.79}$$

Replacing $\boldsymbol{\Lambda}_i$ by (3.75) we finally get

$$\boldsymbol{T}_i^T \boldsymbol{N} \boldsymbol{\Sigma}^{-1} \boldsymbol{T} \mathbb{E}\left[\boldsymbol{w}\right] + \mathbb{E}\left[\boldsymbol{w}_i\right] = \boldsymbol{T}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{F}_X \tag{3.80}$$

Thus, the optimal values for the set of $\boldsymbol{\mu}_i = \mathbb{E}\left[\boldsymbol{w}_i\right]$ are given by the solution of the linear system

$$\left(\boldsymbol{T}^T \boldsymbol{N} \boldsymbol{\Sigma}^{-1} \boldsymbol{T} + \boldsymbol{I}\right) \mathbf{w}_X = \boldsymbol{T}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{F}_X \tag{3.81}$$

that is,

$$\mathbf{w}_X = \left(\boldsymbol{T}^T \boldsymbol{N} \boldsymbol{\Sigma}^{-1} \boldsymbol{T} + \boldsymbol{I}\right)^{-1} \boldsymbol{T}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{F}_X \tag{3.82}$$

which corresponds to the mean of the posterior $P_{\boldsymbol{T},\boldsymbol{\Sigma}}(\boldsymbol{w}|X)$ given in (3.41).

Efficient VB estimate of i–vectors

A naïve implementation of (3.77) with a block size equal to 1 would make the complexity of this approach $O(CFM^2K)$, where K is the number of performed iterations. This is due to the computation load of $\boldsymbol{T}_i^T\boldsymbol{\Sigma}^{-1}\overline{\boldsymbol{F}}_{X,i}$, which has a complexity of $O(CF(M-1))$.

An efficient implementation is, however, possible by defining and updating a vector $\mathbf{F}_c$ that stores the first order statistics centered around the current supervector mean. At the beginning of the iterations the vector is set to

$$\begin{aligned}\mathbf{F}_c^{old} &= \boldsymbol{F}_X - \mathbf{N}\boldsymbol{T}\boldsymbol{\mu}^0 \\ &= \boldsymbol{F}_X - \sum_{j>0}\mathbf{T}_j\boldsymbol{\mu}_j^0\end{aligned} \tag{3.83}$$

Assume a variational approximation where the $\mathbf{w}$ variables are split among B blocks of size b. Let $\mathbf{K}_i = \boldsymbol{\Lambda}_i^{-1}\boldsymbol{T}_i^T\boldsymbol{\Sigma}^{-1}$, The i-th component of the new i–vector is then computed at iteration k as

$$\begin{aligned}\boldsymbol{\mu}_i^{k+1} &= \mathbf{K}_i\left(\boldsymbol{F}_X - \mathbf{N}\overline{\boldsymbol{T}}_i\overline{\boldsymbol{\mu}}_i\right) \\ &= \mathbf{K}_i\left(\boldsymbol{F}_X - \mathbf{N}\sum_{j<i}\mathbf{T}_j\boldsymbol{\mu}_j^{k+1} - \mathbf{N}\sum_{j>i}\mathbf{T}_j\boldsymbol{\mu}_j^{k}\right) \\ &= \mathbf{K}_i\left(\mathbf{F}_c + \mathbf{N}\mathbf{T}_i\boldsymbol{\mu}_i^k\right)\end{aligned} \tag{3.84}$$

and the new vector of the centered first order statistics becomes

$$\begin{aligned}\mathbf{F}_c^{new} &= \boldsymbol{F}_X - \mathbf{N}\sum_{j\leq i}\mathbf{T}_j\boldsymbol{\mu}_j^{k+1} - \mathbf{N}\sum_{j>i}\mathbf{T}_j\boldsymbol{\mu}_j^{k} \\ &= \mathbf{F}_c^{old} + \mathbf{N}\sum_{j<i}\mathbf{T}_j\boldsymbol{\mu}_j^{k+1} + \mathbf{N}\sum_{j\geq i}\mathbf{T}_j\boldsymbol{\mu}_j^{k} - \\ &\quad \mathbf{N}\sum_{j\leq i}\mathbf{T}_j\boldsymbol{\mu}_j^{k+1} - \mathbf{N}\sum_{j>i}\mathbf{T}_j\boldsymbol{\mu}_j^{k} \\ &= \mathbf{F}_c^{old} + \mathbf{N}\mathbf{T}_i\boldsymbol{\mu}_i^k - \mathbf{N}\mathbf{T}_i\boldsymbol{\mu}_i^{k+1}\end{aligned} \tag{3.85}$$

Without considering the complexity needed to estimate the variances $\boldsymbol{\Lambda}_i$ and their inverses, this approach requires $O(KCFM)$ operations to update the first order statistics and project them over the $\mathbf{T}$ matrix and $O(KbM)$ to compute the product between the projected statistics and the covariance matrices $\boldsymbol{\Lambda}^{-1}$. This complexity has to be compared with the $O(KCF(M-b)M/b)$ operations needed by the naïve VB implementation to compute the terms $\boldsymbol{F}_X - \mathbf{N}_X\overline{\boldsymbol{T}}_i\overline{\boldsymbol{\mu}}_i$ and the $O(KCFM)$ operations needed to multiply the $\mathbf{T}$ matrix with the centered statistics $\mathbf{F}_X$.

As far as the computation of the covariance matrices is concerned, the same approaches used for the classical posterior estimate can be adopted. However, instead of computing a single $M \times M$ covariance matrix, we only need to compute B $b \times b$ covariances matrices. The complexity of this approach therefore is $O(CFMb)$ for the slow approach, which does not require additional memory, and $O(CMb)$ for the faster approach, which, however, has memory requirements of the order of $O(CMb)$. In both cases, the inverse of the matrices can be computed in $O(Mb^2)$ operations.

A very small block size would result in faster iterations. However, in practice we have to take care not to make the size too small, because this would indirectly affect the number of iterations required to reach accurate results. In Section 6.5 we will show that the fast VB technique allows to achieve the same performance of the standard approach using less than 20% of its memory for standard–size size GMMs.

VB estimates as the solution of a linear system

Consider again equations (3.81) and (3.82). VB provides a way to iteratively solve the system in an efficient way. The algorithm itself is very similar to a classical algorithm for solving linear systems, the Gauss–Seidel method [66]. In fact, if we denote as b the product $\mathbf{T}^T\mathbf{\Sigma}^{-1}\mathbf{F}_X$, the Gauss–Seidel algorithm allows to compute an estimate of the i–vector $\mathbf{w} = \mathbf{\Lambda}^{-1}\mathbf{d}$ by iterating the update rule

$$w_i^{k+1} = \frac{1}{\Lambda_{ii}} \left[d_i - \sum_{j<i} \Lambda_{ij} w_j^{k+1} - \sum_{j>i} \Lambda_{ij} w_j^k \right], \quad i = 1, \ldots, n. \tag{3.86}$$

where the superscript denotes the algorithm iteration. The algorithm is guaranteed to converge since matrix $\mathbf{\Lambda}$ is positive definite. While the VB algorithm essentially implements the Gauss–Seidel method, however it directly provides an elegant and efficient way to avoid the computation of the off–diagonal terms Λ_{ij} of the precision matrix.

Other techniques exist in literature to iteratively solve a linear system without the need to compute and invert the system matrix. Of particular interest in our case is the Conjugate Gradient (CG) technique [67, 66]. CG proceeds iterating from an initial guess $\mathbf{w}_0$ and generating successive vectors that are closer to the solution $\mathbf{w}$ that minimizes the quadratic function

$$f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{\Lambda}\mathbf{w} - \mathbf{w}^{\mathrm{T}}\mathbf{b} \tag{3.87}$$

An interesting characteristic of CG is that it does not require neither the system matrix nor its inverse. Instead, the algorithm only needs to compute the residual, defined as

$$\mathbf{r}_k = \mathbf{b} - \mathbf{\Lambda}_X \mathbf{w}_k \tag{3.88}$$

Note that, while in expression (3.88) the precision matrix is present, what we really need is just to be able to compute the matrix product between $\mathbf{\Lambda}$ and a generic vector. This product can be efficiently computed as

$$\mathbf{\Lambda}_X \mathbf{w}_k = (\boldsymbol{T}^T \boldsymbol{N} \mathbf{\Sigma}^{-1} \boldsymbol{T}) \mathbf{w}_k + \boldsymbol{I} \mathbf{w}_k \tag{3.89}$$

where the first term is computed by the sequence of operations

$$\begin{aligned} \mathbf{Z} &= \boldsymbol{T} \mathbf{w}_k \\ \mathbf{Z} &\leftarrow \boldsymbol{N} \mathbf{\Sigma}^{-1} \mathbf{Z} \\ \mathbf{Z} &\leftarrow \boldsymbol{T}^T \mathbf{Z} \\ \mathbf{\Lambda}_X \mathbf{w}_k &= \mathbf{Z} + \mathbf{w}_k \end{aligned} \tag{3.90}$$

The order of the operations is important, since it allows to avoid to explicitly compute the dot product $\mathbf{T}^T \mathbf{N} \mathbf{\Sigma}^{-1} \mathbf{T}$. Note that in this way, matrix Λ is never actually computed or inverted. Since the variance matrix is positive definite, the CG algorithm is guaranteed to converge [66].

The computation complexity of the CG method is $O(KCFM)$ for the first and third operations, plus $O(KCF)$ for the second and fourth ones in (3.90). The memory requirements of the algorithm are $O(CFM)$, since only the $\mathbf{T}$ matrix has to be stored.

3.6 Probabilistic Linear Discriminant Analysis

The use of i–vectors as features poses again the problem of channel and nuisance compensation. The first works introducing i–vectors solve this problem by a simple Linear Discriminant Analysis (LDA) projection step followed by Within–Class Covariance Normalization (WCCN) [68, 62, 60].

Linear Discriminant Analysis allows to further reduce the dimensionality of the i–vectors by removing dimensions which have high intra–speaker variability due to channel effects and low variability between speakers. Formally, given a set of D–dimensional patterns $\mathbf{x}_i$ belonging to K classes $\mathcal{C}_k$, we look for a linear transformation

$$\mathbf{y}_i = \mathbf{W}^T \mathbf{x}_i \tag{3.91}$$

where $\mathbf{W}$ is a $D' \times D$ matrix, with $D' \leq K-1$, which maximizes the Fisher criterion [12]

$$J(\mathbf{W}) = \mathrm{tr}\left[(\mathbf{W} \mathbf{S}_W^{-1} \mathbf{W}^T)(\mathbf{W} \mathbf{S}_B \mathbf{W}^T) \right] \tag{3.92}$$

$\mathbf{S}_W$ and $\mathbf{S}_B$ are the within–class and between–class covariance matrices, defined as

$$\mathbf{S}_W = \sum_{k=1}^{K} \sum_{i|\mathbf{x}_i \in \mathcal{C}_k} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T \tag{3.93}$$

$$\mathbf{S}_B = \sum_{k=1}^{K} N_k (\boldsymbol{\mu}_k - \boldsymbol{\mu})(\boldsymbol{\mu}_k - \boldsymbol{\mu})^T \tag{3.94}$$

with

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i|\mathbf{x}_i \in \mathcal{C}_k} \mathbf{x}_i \tag{3.95}$$

$$\boldsymbol{\mu} = \frac{1}{N} \sum_{k=1}^{K} N_k \boldsymbol{\mu}_k \tag{3.96}$$

and N_k, N denote the number of patterns for class k and the total number of patterns, respectively. The solution for $\mathbf{W}$ is then given by the eigenvectors of $\mathbf{S}_W^{-1}\mathbf{S}_B$ corresponding to the $D^{'}$ largest eigenvalues [12].

Within–Class Covariance Normalization (WCCN) was first introduced in [57] as the minimizer of false positive and false negatives error bounds in the context of one–vs–all Support Vector Machine–based speaker recognition with generalized linear kernels (more details about SVMs and speaker recognition are given in the next chapters) and was then adapted to the i–vector framework. WCCN essentially estimates a linear transform

$$\mathbf{z} = \mathbf{B}\mathbf{y} \tag{3.97}$$

where $\mathbf{B}$ is a square matrix which allows whitening the within–class covariance of the transformed features $\mathbf{z}$. $\mathbf{B}$ can be computed as the Cholesky decomposition $\mathbf{B}^T\mathbf{B} = \mathbf{S}_W^{-1}$ of $\mathbf{S}_W^{-1}$, where $\mathbf{S}_W$ is defined as in (3.94)

While LDA and WCCN prove to be quite effective for noise compensation, more advanced models have been proposed which allow for better modeling of nuisance and improve recognition accuracy. One of these models is the Probabilistic Linear Discriminant Analysis (PLDA) model [69, 70]. PLDA is a generative model which tries to describe the i–vector generation process in terms of probability distributions which can then be used to infer log–likelihood scores. In particular, the model assumes that an i–vector can be represented as the sum of different terms [70]

$$\phi_r = m + U_1 x_1 + U_2 x_{2_r} + \epsilon_r \tag{3.98}$$

where r denotes the speaker segment. In analogy with JFA, the terms x_1, x_{2_r} and ϵ_r are latent variables representing, respectively, the speaker identity, the channel effects and the residual noise. Similar to JFA, it is assumed that speaker and channel

factors live in a small subspace. Moreover, recordings from the same speaker are assumed to share the same hidden variable x_1.

In its simplest form PLDA assumes Gaussian priors for the latent variables [69, 70], i.e.

$$\begin{aligned} x_1 &\sim \mathcal{N}(0, I) \\ x_{2_r} &\sim \mathcal{N}(0, I) \\ \epsilon_r &\sim \mathcal{N}(0, \Lambda^{-1}) \end{aligned} \tag{3.99}$$

with Λ diagonal. We will refer to this model as Gaussian PLDA (GPLDA).

A more complex model for PLDA was proposed in [70], where the priors where assumed to be Student's t distributed, i.e.

$$\begin{aligned} x_1 &\sim \mathcal{N}(0, u_1^{-1} I), \quad u_1 \sim \mathcal{G}(n_1/2, n_1/2) \\ x_{2_4} &\sim \mathcal{N}(0, u_{2_r}^{-1} I), \quad u_{2_r} \sim \mathcal{G}(n_2/2, n_2/2) \\ \epsilon_r &\sim \mathcal{N}(0, v_r^{-1} \Lambda^{-1}), \quad v_r \sim \mathcal{G}(\nu/2, \nu/2) \end{aligned} \tag{3.100}$$

This heavy–tailed model (HTPLDA in the following) allows much better accuracy when no preprocessing of i–vectors is performed [70]. However, since posteriors cannot be computed in closed form, the time required both for training and testing greatly increases. Since it was shown that proper normalization of the i–vectors, e.g. length normalization [71], allows the simpler GPLDA model to achieve the same level of accuracy in the following we will focus only on the GPLDA model.

3.6.1 Two covariance model

Before analyzing the GPLDA model we consider the simplified model known as two–covariance model [72], where both the speaker and channel subspaces are assumed full rank. In this case, the residual noise term can be neglected, since it is already accounted for by the channel term (or, equivalently, we can neglect the channel factors and assume a full covariance noise term). The resulting model can be formalized as

$$\phi = \mathbf{y} + \mathbf{z}_r \tag{3.101}$$

where $\mathbf{y}$ is the speaker factor and $\mathbf{z}_r$ accounts for all the noise components. The priors for $\mathbf{y}$ and $\mathbf{z}_r$ are assumed to be Gaussian

$$P(\mathbf{y}) \sim \mathcal{N}(\mathbf{y}|\mu, B^{-1}) \tag{3.102}$$

$$P(\phi|\mathbf{y}) \sim \mathcal{N}(\phi|\mathbf{y}, W^{-1}) \tag{3.103}$$

characterized by the (between–speaker) covariance matrix B^{-1} and the (within–speaker) covariance matrix W^{-1} (hence the name of the model).

The posterior for $\mathbf{y}$ given a set of i–vectors $\{\phi_1, \ldots, \phi_n\}$ belonging to the same speaker can be computed in closed form by observing that the prior (3.102) is conjugate to the likelihood (3.103). This results in a normal posterior given by

$$P(\mathbf{y}|\phi_1, \ldots, \phi_n) \sim \mathcal{N}(\mathbf{y}|L^{-1}\gamma, L^{-1}) \tag{3.104}$$

with

$$\gamma = B\mu + W\sum_{i=1}^{n}\phi_i \tag{3.105}$$

$$L = B + nW \tag{3.106}$$

3.6.2 Speaker verification likelihood

Before moving to a more general PLDA model we analyze how these kind of models can be used to perform inference about the speaker identity.

Given a set of enrollment i–vectors $\phi_{e_1}, \ldots, \phi_{e_n}$ for a given speaker s_e and a set of test segments (usually one) of speaker s_t with corresponding i–vectors $\phi_{t_1}, \ldots, \phi_{t_m}$ the question whether the test segments and enrollment segments are from the same speaker can be solved by computing the likelihood of the observed i–vectors under the *same speaker* and *different speaker* hypothesis [72, 70]. Formally, we want to compute the log–likelihood ratio

$$l = \log \frac{P(\phi_{e_1}, \ldots, \phi_{e_n}, \phi_{t_1}, \ldots, \phi_{t_m}|H_s)}{P(\phi_{e_1}, \ldots, \phi_{e_n}, \phi_{t_1}, \ldots, \phi_{t_m}|H_d)} \tag{3.107}$$

where H_s and H_d represent the same and different speaker hypotheses respectively.

Considering, for simplicity, the case where we have a single enrollment i–vector ϕ_e and a single test i–vector ϕ_t, the log–likelihood ratio (3.107) can be computed as

$$\begin{aligned} l &= \log \frac{P(\phi_e, \phi_t|H_s)}{P(\phi_e, \phi_t|H_d)} \\ &= \log \frac{\int P(\phi_e|\mathbf{y})P(\phi_t|\mathbf{y})P(\mathbf{y})d\mathbf{y}}{[\int P(\phi_e|\mathbf{y})P(\mathbf{y})d\mathbf{y}]\,[\int P(\phi_t|\mathbf{y})P(\mathbf{y})d\mathbf{y}]} \\ &= \log \frac{\int P(\phi_e|\mathbf{y})P(\phi_t|\mathbf{y})P(\mathbf{y})d\mathbf{y}}{P(\phi_e)P(\phi_t)} \end{aligned} \tag{3.108}$$

where we used the assumption of conditional i–vector independence given the speaker factors. Given more than one enrollment or test utterances the log–likelihood ratio between same–speaker and different speaker hypothesis becomes

$$l = \log \frac{\int \prod_{i=1}^{n} P(\phi_{e_i}|\mathbf{y}) \prod_{j=1}^{m} P(\phi_{t_j}|\mathbf{y})P(\mathbf{y})d\mathbf{y}}{[\int \prod_{i=1}^{n} P(\phi_{e_i}|\mathbf{y})P(\mathbf{y})d\mathbf{y}]\,[\int \prod_{i=1}^{m} P(\phi_{e_i}|\mathbf{y})P(\mathbf{y})d\mathbf{y}]} \tag{3.109}$$

Note that the log–likelihood ratio computation entails the evaluation of integrals of the form

$$\int P(\phi_1, \ldots, \phi_N|\mathbf{y})P(\mathbf{y})d\mathbf{y} = P(\phi_1, \ldots, \phi_N|H_s) \tag{3.110}$$

which correspond to the probability of observing a set of i–vectors under the assumption that they all belong to the same speaker. In fact, the speaker verification problem can be cast as a particular instance of the speaker partitioning problem [72] and the same–speaker and different speaker hypotheses correspond to the choice of a single set for all i–vectors and to the choice of two different sets for enrollment and test i–vectors.

The integral (3.110) does not have a closed form in the general case. The use of conjugate priors however allows us to avoid explicit computation of this integral observing that the probabilities $P(\phi_1, \ldots, \phi_k|H_s)$ can be expressed as

$$P(\phi_1, \ldots, \phi_k|H_s) = \frac{P(\phi_1, \ldots, \phi_k|\mathbf{y}_0)P(\mathbf{y}_0)}{P(\mathbf{y}_0|\phi_1, \ldots, \phi_k)} \tag{3.111}$$

where $\mathbf{y}_0$ is *any* value of the hidden variable which does not cause the denominator to be zero. The integral has now implicitly moved in the computation of the posterior for $\mathbf{y}$. However, we already showed how to compute the posterior for $\mathbf{y}$ given a set of i–vectors in the particular case of Gaussian priors.

The log–likelihood can then be expressed as

$$l = \log \frac{Q(\phi_{e_1}, \ldots, \phi_{e_n}, \phi_{t_1}, \ldots, \phi_{t_m}))}{Q(\phi_{e_1}, \ldots, \phi_{e_n})Q(\phi_{t_1}, \ldots, \phi_{t_m})} \tag{3.112}$$

with

$$Q(\phi_1, \ldots, \phi_k) = \frac{P(\mathbf{y}_0)}{P(\mathbf{y}_0|\phi_1, \ldots, \phi_k)} \tag{3.113}$$

Note that in Q we dropped the (irrelevant) term $P(\phi_1, \ldots, \phi_k|\mathbf{y}_0)$, which would cancel out anyway in the likelihood computation.

Finally, by fixing a value for $\mathbf{y}_0$ (e.g. $\mathbf{y}_0 = 0$) we can derive a closed form for Q (and therefore for the log–likelihood ratio)

$$\log Q(\phi_1, \ldots, \phi_k) = \frac{1}{2}\left(\log|B| - \mu^T B\mu - \log|L_S| + \gamma_S^T L_S^{-1}\gamma_S\right) \tag{3.114}$$

where S denotes the set of i–vectors $\phi_1, \ldots, \phi_k$ and L_S, γ_S are the parameters (3.106) for the posterior distribution of $\mathbf{y}$ given the i–vectors in S [72].

3.6.3 PLDA

We now consider a slightly general model, where the channel factors still occupy the whole space, but the speaker factors are entailed to live in a smaller subspace. The

model can be expressed formally as

$$\phi = \mu + Uy + z \tag{3.115}$$

where the matrix U is a $M \times K$ matrix whose columns span the speaker factor subspace.

Again, we assume Gaussian priors for both hidden variables. As in the two–covariance model case, the posterior for $\mathbf{y}$ given a set S of i–vectors $\phi_1, \ldots, \phi_k$ can still be computed in closed form by observing that

$$\begin{aligned} \log P(\mathbf{y}|S) &= \log P(S|\mathbf{y}) + \log P(\mathbf{y}) + c \\ &= -\frac{1}{2}\sum_{i=1}^{k} (\phi_i - \mu - U\mathbf{y})^T \Lambda (\phi_i - \mu - U\mathbf{y}) - \frac{1}{2}\mathbf{y}^T\mathbf{y} + c \end{aligned} \tag{3.116}$$

where Λ^{-1} is the covariance matrix of the prior for $\mathbf{z}$ and c is some normalization constant. The posterior is then again Gaussian distributed

$$P(\mathbf{y}|S) \sim \mathcal{N}(\mathbf{y}|\mu_{\mathbf{y}}, \Lambda_{\mathbf{y}}^{-1}) \tag{3.117}$$

with

$$\Lambda_{\mathbf{y}} = \left(kU^T\Lambda U + I\right) \tag{3.118}$$

$$\mu_{\mathbf{y}} = \Lambda_{\mathbf{y}}^{-1}U^T\Lambda\sum_{i=1}^{k}(\phi_k - \mu) \tag{3.119}$$

The case where the channel factors are not merged with the residual noise (i.e. $U_2 \neq \mathbf{0}$) is slightly more complex, since the hidden variables x_1 and x_2 are correlated in the posterior. A closed form solution can still be computed [69], although it requires the inversion of $R_s \times R_s$ block matrix, where R_s is the number of recordings for speaker S. A Variational Bayes solution to this problem was proposed in [70], together with the extension to non–Gaussian priors. However, since for most applications the GPLDA with merged channel and noise factors model achieves very good performance, we will skip the analysis of these more complex models.

3.6.4 Training the PLDA hyperparameters

So far we have discussed how the PLDA model can be used to compute verification log–likelihood scores given the model parameters. In the following we will focus on estimating the hyperparameters of the model. A broadly used technique to solve this problem consists in training the PLDA model in a maximum–likelihood fashion, that is, estimating the speaker subspace U and the noise precision matrix Λ as to maximize the likelihood of a given set of i–vectors with known speaker label [69, 70].

A complete solution in the general heavy tailed case was given in [70]. In this section we apply the same derivations to the simpler single subspace PLDA model.

Let $P(S)$ denote the likelihood of the set of i–vectors belonging to speaker s. The objective function we want to maximize is given by

$$\sum_s P(S) \tag{3.120}$$

where the sum extends over all speakers in the training set. It is possible to rewrite this likelihood for a single speaker s as [12]

$$\log P(S) = \mathcal{L}(q) + KL(q(\mathbf{y}) \| p(\mathbf{y}|S)) \tag{3.121}$$

where $q(\mathbf{y})$ is a distribution for $\mathbf{y}$ and $KL(q\|p)$ denotes the Kullback–Leibler divergence between the distribution q and the posterior for $\mathbf{y}$. $\mathcal{L}$ is defined as

$$\mathcal{L}(q) = \int q(\mathbf{y}) \log \left(\frac{P(S, \mathbf{y})}{q(\mathbf{y})} \right) d\mathbf{y} \tag{3.122}$$

Observe that, if the distribution $q(\mathbf{y})$ equals the posterior for $p(\mathbf{y}|S)$ then the divergence goes to zero and

$$\begin{aligned} \log P(S) &= \mathcal{L}(p(\mathbf{y}|S)) \\ &= \int p(\mathbf{y}|S) \log \left(\frac{P(S, \mathbf{y})}{p(\mathbf{y}|S)} \right) d\mathbf{y} \\ &= \mathbb{E}_{\mathbf{y}|S} \left[\log \left(\frac{P(S, \mathbf{y})}{p(\mathbf{y}|S)} \right) \right] \end{aligned} \tag{3.123}$$

In [70] the authors assume that q is a variational approximation to the posterior. However, in our case this is not necessary since the posterior can be computed in closed form.

Maximum Likelihood

We want to maximize

$$\sum_s \mathcal{L}(s) \tag{3.124}$$

Dropping the reference to the speaker we can write

$$\begin{aligned}
\mathcal{L}(s) &= \mathbb{E}_{\mathbf{y}|S}\left[\log P(S|\mathbf{y})\right] \\
&= \mathbb{E}_{\mathbf{y}|S}\left[\sum_{i=1}^{R_s} \log \frac{1}{(2\pi)^{\frac{F}{2}}|\Lambda^{-1}|^{\frac{1}{2}}} - \frac{1}{2}(\phi_r - U\mathbf{y} - m)^T\Lambda(\phi_r - U\mathbf{y} - m)\right] \\
&= \mathbb{E}_{\mathbf{y}|S}\left[\sum_{i=1}^{R_s} \log \frac{1}{(2\pi)^{\frac{F}{2}}|\Lambda^{-1}|^{\frac{1}{2}}} - \frac{1}{2}(\phi_r - \bar{U}\bar{\mathbf{y}})^T\Lambda(\phi_r - \bar{U}\bar{\mathbf{y}})\right] \\
&= \sum_{r=1}^{R_s} log \frac{1}{(2\pi)^{\frac{F}{2}}|\Lambda^{-1}|^{\frac{1}{2}}} - \frac{1}{2}\mathrm{tr}\left(\phi_r^T\Lambda\phi_r\right) \\
&\quad + \mathrm{tr}\left(E_{\mathbf{y}|S}\left[\bar{\mathbf{y}}^T\right]\bar{U}^T\Lambda\phi_r\right) - \frac{1}{2}\mathrm{tr}\left(\bar{U}^T\Lambda\bar{U}E_{\mathbf{y}|S}\left[\bar{\mathbf{y}}\bar{\mathbf{y}}^T\right]\right)
\end{aligned} \tag{3.125}$$

where we set

$$\bar{\mathbf{y}} = \begin{bmatrix}\mathbf{y}\\ 1\end{bmatrix} \quad \bar{U} = [U\ m] \tag{3.126}$$

The maximum of $\mathcal{L}_1$ with respect to $\bar{U}$ can be obtained by setting to zero the derivative of (3.125) with respect to $\bar{U}$, which gives

$$\bar{U}\sum_s\sum_{r=1}^{R_s}\mathbb{E}_{\mathbf{y}(s)|S(s)}\left[\bar{\mathbf{y}}(s)\bar{\mathbf{y}}^T(s)\right] = \sum_s\sum_{r=1}^{R_s}\phi_r\mathbb{E}_{\mathbf{y}(s)|S(s)}\left[\bar{\mathbf{y}}(s)\right] \tag{3.127}$$

which can be easily solved for $\bar{U}$ given the first and second order moments for the posterior of $\bar{\mathbf{y}}(s)$

$$\begin{aligned}
E_{\mathbf{y}|S}\left[\bar{\mathbf{y}}\right] &= \begin{bmatrix}\mu_{\mathbf{y}}\\ 1\end{bmatrix} \\
E_{\mathbf{y}|S}\left[\bar{\mathbf{y}}\bar{\mathbf{y}}^T\right] &= \begin{bmatrix}\Lambda_{\mathbf{y}}^{-1} + \mu_{\mathbf{y}}\mu_{\mathbf{y}}^T & \mu_{\mathbf{y}}\\ \mu_{\mathbf{y}}^T & 1\end{bmatrix}
\end{aligned} \tag{3.128}$$

where $\mu_{\mathbf{y}}$ and $\Lambda_{\mathbf{y}}$ are defined as in (3.119).

Following the success of the PLDA model a discriminative approach to PLDA parameters training has been proposed in [73, 74]. This approach will be discussed in Chapter 5.

Chapter 4

Discriminative Training and Support Vector Machines

In this chapter we analyze discriminative models based on linear classifiers like Support Vector Machines and linear Logistic Regression. While generative models try to describe the process which generates the observed features, discriminative techniques try to estimate a decision rule which optimizes some criterion over a set of training patterns.

4.1 Support Vector Machines and Logistic Regression

Among all possible discriminative optimization criteria two important ones are optimization of the classification margin between classes, which gives place to Support Vector Machines, and the cross–entropy between classifier scores and target labels, which corresponds to Logistic Regression. Support Vector Machines and logistic regression are among the most successful discriminative classifiers thanks to relatively easy training procedures and good classification performance.

4.1.1 Support Vector Machines

A Support Vector Machine (SVM) [75, 76, 12] is a two–class (linear) classifier which looks for the hyperplane that best discriminates two given classes of patterns according to a maximum separation margin criterion.

Linear Classifier

Given a set of n m–dimensional pattern vectors $\mathbf{x}_k$ belonging to either of two classes $\mathcal{C}_1$ and $\mathcal{C}_2$ a linear classifier is a hyperplane with form

$$\mathbf{w}^T\mathbf{x} + b = 0 \tag{4.1}$$

where $\mathbf{w} \in \mathbb{R}^m$ and $b \in \mathbb{R}$. The decision function $\mathcal{D}(\mathbf{x})$ is

$$\mathcal{D}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = \sum_{i=1}^{m} w_i x_i + b \tag{4.2}$$

and the distance between the hyperplane and a vector $\mathbf{x}$ is $\frac{\mathcal{D}(\mathbf{x})}{\|\mathbf{w}\|}$. If we assume that a separation margin M between the class boundary and the pattern vectors belonging to that class exists, for each pattern the following inequality is verified [77]

$$\frac{y_k \mathcal{D}(\mathbf{x}_k)}{\|\mathbf{w}\|} \geq M \tag{4.3}$$

where y_k is defined as

$$y_k = \begin{cases} 1 & \text{if } \mathbf{x}_k \in \mathcal{C}_1 \\ -1 & \text{if } \mathbf{x}_k \in \mathcal{C}_2 \end{cases} \tag{4.4}$$

The objective of a maximum–margin linear classifier is to find the parameter vector $\mathbf{w}$ that maximizes M

$$\mathbf{w} = \arg\max_{\mathbf{w}} M \tag{4.5}$$

subject to

$$y_k \mathcal{D}(\mathbf{x}_k) \geq M, \quad k = 1, \ldots, n \tag{4.6}$$

and

$$\|\mathbf{w}\| = 1 \tag{4.7}$$

which corresponds to the margin

$$M^* = \max_{\mathbf{w}, \|\mathbf{w}\|=1} M \tag{4.8}$$

The bound M^* is attained for those patterns x_k satisfying

$$y_k \mathcal{D}(\mathbf{x}_k) = M^* = \min_j y_j \mathcal{D}(\mathbf{x}_j) \tag{4.9}$$

which are called support vectors. To find the maximum margin hyperplane therefore requires to solve a minimax problem of the form

$$\|\mathbf{w}\| = \arg \max_{\mathbf{w}, \|\mathbf{w}\|=1} \min_k y_k \mathcal{D}(\mathbf{x}_k) \tag{4.10}$$

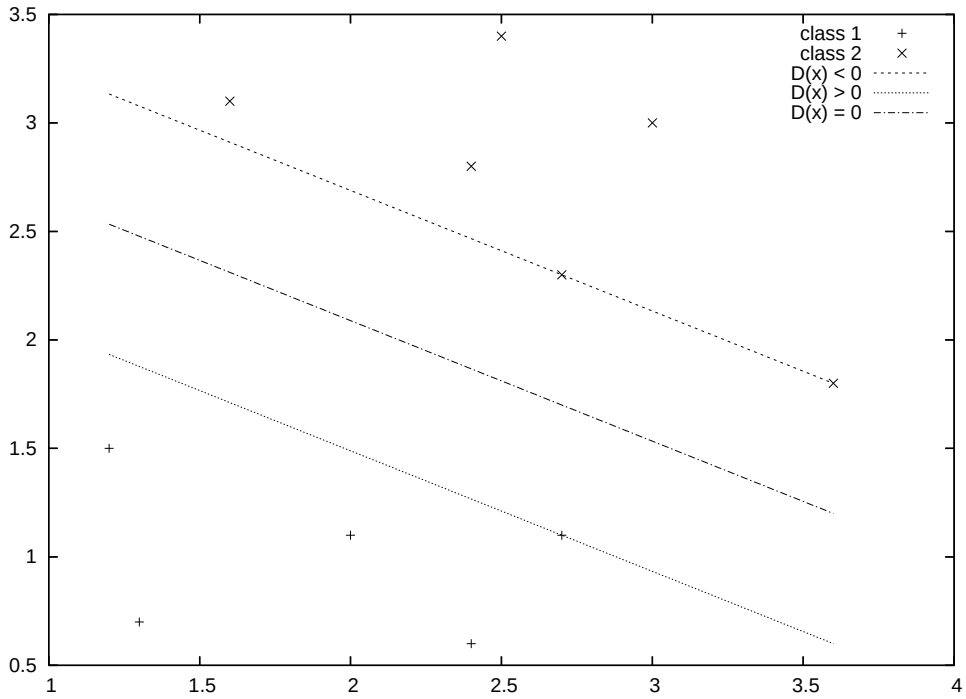


Figure 4.1: A maximum margin hyperplane and corresponding support vectors

The condition (4.7) is imposed in order to find a solution among the infinite set of solutions differing only in scaling. However, instead of fixing the norm of $\mathbf{w}$, it is possible to fix the value of the product of the margin M and of $\mathbf{w}$ such that

$$M\|\mathbf{w}\| = 1 \tag{4.11}$$

In this way, maximizing the margin is equivalent to minimizing the norm of $\mathbf{w}$, so the problem becomes

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \|\mathbf{w}\|^2 \tag{4.12}$$

under constraints

$$y_k \mathcal{D}(\mathbf{x}_k) \geq 1, \quad k = 1, \ldots, n \tag{4.13}$$

and the margin M^* becomes

$$M^* = \frac{1}{\|\mathbf{w}\|} \tag{4.14}$$

This problem can be solved using numerical methods, however, when the set dimensionality increases, this approach can become unfeasible, and, moreover, it does not give any information about support vectors.

Another way to solve the problem consists in considering the dual Lagrangian problem [78, 12, 77]

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{k=1}^{n} \alpha_k (y_k \mathcal{D}(\mathbf{x}_k) - 1)$$
$$\text{s.t. } \alpha_k \geq 0, \quad k = 1, \ldots, n \tag{4.15}$$

where α_k are the Lagrange multipliers and satisfy

$$\alpha_k (y_k \mathcal{D}(\mathbf{x}_l) - 1) = 0, \quad k = 1, \ldots, n \tag{4.16}$$

The fraction $\frac{1}{2}$ does not change the solution and has been introduced to ease successive simplifications.

The hyperplane $\mathbf{w}^*$ can now be found by looking for a saddle point of $L(\mathbf{w}, b, \boldsymbol{\alpha})$ which is a minimum of $L(\mathbf{w}, b, \boldsymbol{\alpha})$ with respect to $\mathbf{w}$ and a maximum with respect to $\boldsymbol{\alpha}$. Let $\boldsymbol{\alpha}^*$ and $\mathbf{w}^*$ denote the value of $\boldsymbol{\alpha}$ and of $\mathbf{w}$ respectively at the saddle point. In correspondence of $\mathbf{w}^*$ and $\boldsymbol{\alpha}^*$ we have

$$\left.\frac{\partial L}{\partial \mathbf{w}}\right|_{\mathbf{w}^*} = \mathbf{w}^* - \sum_{k=1}^{n} \alpha_k^* y_k \mathbf{x}_k \tag{4.17}$$

that is

$$\mathbf{w}^* = \sum_{k=1}^{n} \alpha_k^* y_k \mathbf{x}_k \tag{4.18}$$

From (4.16) we have $\alpha_k^* = 0$ for each pattern that does not satisfy $y_k \mathcal{D}(\mathbf{x}_k) = 1$, so $\mathbf{w}^*$ only depends on the remaining vectors, which are the support vectors.

Using (4.2) and (4.18), we can remove the dependence of $L(\mathbf{w}, b, \boldsymbol{\alpha})$ from $\mathbf{w}$ by substituting $\mathbf{w}$ with $\mathbf{w}^*$, which leads to

$$\begin{aligned} L(\boldsymbol{\alpha}, b) &= \frac{1}{2} \sum_{k=1}^{n} \alpha_k y_k \mathbf{x}_k^T \mathbf{x}_k - \sum_{k=1}^{n} \sum_{j=1}^{n} \alpha_k y_k \alpha_j y_j \mathbf{x}_k^T \mathbf{x}_j - \sum_{k=1}^{n} \alpha_k (1 - b y_k) \\ &= \left[\sum_{k=1}^{n} \alpha_k (1 - b y_k) \right] - \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{H} \boldsymbol{\alpha} \end{aligned} \tag{4.19}$$

$$\tag{4.20}$$

subject to

$$\alpha_k \geq 0, \quad k = 1, \ldots, n \tag{4.21}$$

Here $\mathbf{H}$ is a square $m \times m$ matrix with elements

$$h_{kj} = y_k y_j \mathbf{x}_k^T \mathbf{x}_j \tag{4.22}$$

In order for a unique solution to exist, $\mathbf{H}$ must be positive definite. In this case, the solution is obtained, for fixed b, by maximizing $L(\boldsymbol{\alpha}, b)$ under the given constraints. The optimal bias b^* can be computed as

$$\begin{aligned} b^* &= -\frac{1}{2}(\mathbf{w}^{*T}\mathbf{x}_{c_1} + \mathbf{w}^{*T}\mathbf{x}_{c_2}) \\ &= -\frac{1}{2}\sum_{k=1}^{n} y_k \alpha_k^* (\mathbf{x}_{c_1}^T \mathbf{x}_k + \mathbf{x}_{c_2}^T \mathbf{x}_k) \end{aligned} \tag{4.23}$$

where $\mathbf{x}_{c_1}$ and $\mathbf{x}_{c_2}$ are, respectively, a support vector of class $\mathcal{C}_1$ and one of class $\mathcal{C}_2$. This is due to the fact that the maximum margin is obtained when the decision boundary is halfway between the two classes. Since the decision function $\mathcal{D}(\mathbf{x})$ becomes

$$\begin{aligned} \mathcal{D}(\mathbf{x}) &= \mathbf{w}^{*T}\mathbf{x} + b \\ &= \sum_{k=1}^{n} y_k \alpha_k^* \mathbf{x}_k^T \mathbf{x} + b, \end{aligned} \tag{4.24}$$

with

$$\alpha_k^* \geq 0 \quad k = 1, \ldots, n \tag{4.25}$$

Finally, b^* can be can be obtained by evaluating $\mathcal{D}(\mathbf{x})$ in correspondence of $\mathbf{x}_{c_1}$ and $\mathbf{x}_{c_2}$, which leads to (4.23) (more robust estimates might, however, be obtained by taking an average over all support vectors [12])

Non linear classifier

Non linear classification in input space can be achieved through to linear classification in an expanded feature space. The use of the *kernel trick* allows then to compute the decision boundary without the need to explicitly expand the features [79, 77]. From Mercer's theorem [80] we have that a positive semi–definite function of two arguments in a measurable space $K(\mathbf{x}_i, \mathbf{x}_j)$ can be evaluated as the inner product of the mapping of its arguments in a higher dimensionality space, that is there exists a function

$$\varphi : \Re^N \rightarrow \Re^F \tag{4.26}$$

where F is the feature space dimensionality, such that

$$K(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}_j) \tag{4.27}$$

The support vector machine decision function becomes

$$\mathcal{D}(\mathbf{x}) = \mathbf{w}^T \varphi(\mathbf{x}) + b = \sum_{k=1}^{F} w_i \varphi(\mathbf{x}_i) + b \tag{4.28}$$

Here $\mathbf{w}$ is a vector representing the direction of the optimal hyperplane in the expanded feature space. Using the same approach as for linear classification, we can find the optimal value of $\mathbf{w}$, obtaining

$$\mathbf{w}^* = \sum_{k=1}^{F} \alpha_k^* y_k \varphi(\mathbf{x}_k) \tag{4.29}$$

Equation (4.19) still holds if we replace $\mathbf{H}$ with a $F \times F$ matrix $\mathbf{H}'$ whose elements are

$$\mathbf{H}'_{jk} = y_j y_k \varphi(\mathbf{x}_j)^T \varphi(\mathbf{x}_k) = y_j y_k K(\mathbf{x}_j, \mathbf{x}_k) \tag{4.30}$$

and the resulting decision function becomes

$$\begin{aligned} \mathcal{D}(\mathbf{x}) &= \mathbf{w}^{*T} \varphi(\mathbf{x}) + b \\ &= \sum_{k=1}^{F} y_k \alpha_k^* K(\mathbf{x}_k, \mathbf{x}) + b \end{aligned} \tag{4.31}$$

where $\mathbf{x}_k$ are the support vectors. As far as the bias b is concerned, it can be estimated in the same way as in linear classification, which leads to

$$\begin{aligned} b^* &= -\frac{1}{2}(\mathbf{w}^{*T} \varphi(\mathbf{x}_{c_1}) + \mathbf{w}^{*T} \varphi(\mathbf{x}_{c_2})) \\ &= -\frac{1}{2} \sum_{k=1}^{F} y_k \alpha_k^* [K(\mathbf{x}_{c_1}, \mathbf{x}_k) + K(\mathbf{x}_{c_2}, \mathbf{x}_k)] \end{aligned} \tag{4.32}$$

where $\mathbf{x}_{c_1}$ and $\mathbf{x}_{c_2}$ have the same meanings as in (4.23).

It is interesting to observe that no explicit expansion of input space patterns has to be made in order to evaluate the decision function of the optimal hyperplane in the expanded feature space, since all we need is the kernel evaluation. As an example, a classifier with polynomial decision surfaces of order d [12] could be implemented using the kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + 1)^d \tag{4.33}$$

without having to explicitly evaluate the polynomial expansion of input vectors.

Soft margin hyperplane

It is not unusual that the classes we want to discriminate between are not completely separable, that is we cannot find a hyperplane which discriminates the training samples of the classes without errors. However, in this case, we can still look for the hyperplane separating the two classes with a minimal number of errors. To solve

this problem we introduce slack variables $\xi_i \geq 0$ for each pattern [81, 82] and we consider the functional

$$\Phi(\xi) = \sum_{i=1}^{n} \xi_i^{\sigma} \tag{4.34}$$

with $\sigma > 0$ and subject to the constraints

$$y_i \mathcal{D}(\mathbf{x}_i) \geq 1 - \xi_i, \quad i = 1, \ldots, n \tag{4.35}$$

and

$$\xi_i \geq 0, \quad i = 1, \ldots, n \tag{4.36}$$

For small values of σ the functional $\Phi(\xi)$ can be interpreted as the number of training errors. By minimizing it we can find a minimal subset of training errors and, excluding them from the training set, we can separate without errors the remaining data using the maximum margin hyperplane. Formally, this can be expressed as minimizing the functional [82]

$$\frac{1}{2}\|\mathbf{w}\|^2 + Cf\left(\sum_{i=1}^{n} \xi_i^{\sigma}\right) \tag{4.37}$$

subject to the constraints (4.35) and (4.36), where c is a constant and f is a monotonic convex function. However, in the general form the problem is NP–complete. The minimum value for σ which allows to avoid NP–completeness is $\sigma = 1$. It can be shown that, for sufficiently large values of c, letting $\sigma = 1$ gives the hyperplane minimizing the sum of deviations ξ of training errors and maximizing the margin of correctly classified vectors. If the classes are completely separable, this hyperplane coincides with the maximum margin hyperplane [82].

The same approach used for the maximum margin hyperplane can be followed. However, this time the Lagrangian of the functional becomes

$$L(\mathbf{w}, b, \boldsymbol{\alpha}, \xi, \mathbf{r}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{k=1}^{n} \alpha_k (y_k \mathcal{D}(\mathbf{x}_k) - 1 + \xi_k) + Cf(\xi) - \sum_{k=1}^{n} r_k \xi_k \tag{4.38}$$

subject to

$$\begin{aligned} \alpha_k &\geq 0, \quad k = 1, \ldots, n \\ r_k &\geq 0, \quad k = 1, \ldots, n \end{aligned} \tag{4.39}$$

To minimize this functional with respect to $\mathbf{w}_i$, b and ξ_i we look for a saddle

point of the Lagrangian

$$\frac{\partial L}{\partial \mathbf{w}}(\mathbf{w}^*) = \mathbf{w}^* - \sum_{k=1}^{n} \alpha_k y_k \mathbf{x_k} = 0 \tag{4.40}$$

$$\frac{\partial L}{\partial b}(b^*) = \sum_{k=1}^{n} \alpha_k y_k = 0 \tag{4.41}$$

$$\frac{\partial L}{\partial \xi_i}(\xi_i^*) = c \left. \frac{\partial f(\xi)}{\partial \xi} \frac{\partial \xi}{\partial \xi_i} \right|_{\xi_i^*} - \alpha_i - r_i = 0 \tag{4.42}$$

If we define $\delta = C\frac{\partial f(\xi)}{\partial \xi}(\xi^*)$, since $\xi = \sum_{i=1}^{n} \xi_i^{\sigma}$ we have

$$\delta\sigma\xi_i^{*\sigma-1} - \alpha_i - r_i = 0 \tag{4.43}$$

Substituting these expressions in the Lagrangian functional we have

$$L(\boldsymbol{\alpha}, \xi, \mathbf{r}) = \sum_{k=1}^{n} \alpha_k - \frac{1}{2}\boldsymbol{\alpha}^T \mathbf{H} \boldsymbol{\alpha} + Cf(\xi^*) - \sum_{k=1}^{n} \delta\sigma\xi_k^{*\sigma} \tag{4.44}$$

Since $\delta = C\frac{\partial f(\xi)}{\partial \xi}(\xi^*) = Cf'(\xi^*)$ we can write that

$$\xi^* = \sum_{k=1}^{n} \xi_k^{*\sigma} = f'^{-1}\left(\frac{\delta}{C}\right) \tag{4.45}$$

and the Lagrangian becomes

$$L(\boldsymbol{\alpha}, \delta) = \sum_{k=1}^{n} \alpha_k - \frac{1}{2}\boldsymbol{\alpha}^T \mathbf{H} \boldsymbol{\alpha} + Cf\left(f'^{-1}\left(\frac{\delta}{C}\right)\right) - f'^{-1}\left(\frac{\delta}{C}\right)\delta\sigma \tag{4.46}$$

From the constraints $\alpha_i + r_i = \delta$ and $r_i \geq 0$ it is possible to conclude that the maximum value for δ corresponds to $\delta = \alpha_{max} = \max_i(\alpha_1, \ldots, \alpha_n)$ Therefore, the soft margin hyperplane, which has the form $\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$ can be evaluated by solving the convex programming problem [82]

$$\begin{aligned} \max_{\boldsymbol{\alpha}} L(\boldsymbol{\alpha}) = \max_{\boldsymbol{\alpha}} \sum_{k=1}^{n} \alpha_k - \frac{1}{2}\boldsymbol{\alpha}^T \mathbf{H} \boldsymbol{\alpha} + Cf\left(f'^{-1}\left(\frac{\alpha_{max}}{C}\right)\right) - f'^{-1}\left(\frac{\alpha_{max}}{C}\right)\alpha_{max}\sigma \\ \text{s.t. } \alpha_k \geq 0, \quad \forall k \end{aligned} \tag{4.47}$$

If we assume $\sigma = 1$ and we choose as function f the linear function $f(x) = x$ we recover the standard hinge–loss (L1–loss) SVM formulation whose corresponding primal is given by

$$\begin{aligned} \mathbf{w}^*, b^* = \arg\max_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i \\ \text{s.t. } y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) \geq 1 - \xi_i, \qquad i = 1, \ldots, n \\ \xi_i \geq 0, \qquad i = 1, \ldots, n \end{aligned} \tag{4.48}$$

or, equivalently,

$$\mathbf{w}^*, b^* = \arg\max_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \max\left(0, 1 - y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right)\right) \tag{4.49}$$

and the corresponding dual problem is

$$\begin{aligned} \boldsymbol{\alpha}^* &= \arg\max_{\boldsymbol{\alpha}} \boldsymbol{\alpha}^T\mathbf{e} - \frac{1}{2}\boldsymbol{\alpha}^T\mathbf{H}\boldsymbol{\alpha} \\ &\text{s.t. } 0 \leq \alpha_k \leq C, \quad \forall k \end{aligned} \tag{4.50}$$

where $\mathbf{e}$ denotes a vector of all ones. The relationship between primal and dual solution is given by

$$\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \tag{4.51}$$

Choosing the regularization coefficient

Training an SVM entails the selection of an appropriate value for the regularizer term C. A complete discussion of optimal selection of the SVM regularization parameter and the connections between regularizers and SVM generalizations capabilities are beyond the scope of this work. Although different approaches can be undertaken to estimate a good regularizer, for example cross–validation or the approach of [83], where SVMs are fitted for all possible regularizers, we found that in practice a simple expression for the regularizer which can be computed from the data is sufficient to produce accurate models. The expression for the regularizer we used is the same adopted by SVM$^{\text{Light}}$ [84] for the default regularizer parameter, and corresponds to

$$C = \left(\frac{1}{N}\sum_{i=1}^{N} \|\mathbf{x}_i\|\right)^{-2} \tag{4.52}$$

where $\mathbf{x}_i$ are the N patterns in the training set. We believe that the reasons this regularizer allows to obtain good results are connected to theoretical results on generalization properties of SVM [76], however we have no theoretical proofs to support this statement.

4.1.2 Logistic Regression

Another widely used linear classifier is the Logistic Regression (LR) classifier [12]. LR allows to evaluate a classification boundary which can be interpreted as a log–likelihood ratio between class–conditional likelihoods.

LR model

Let $\mathbf{x}$ be the pattern to classify, belonging to one of two classes C_1, C_2. Let also $P(\mathbf{x}|C_i)$ denote the likelihood of pattern $\mathbf{x}$ given the class C_i. The class posterior can then be evaluated using Bayes rule as

$$p(C_j|\mathbf{x}_i) = \frac{p(\mathbf{x}_i|C_j)p(C_j)}{p(\mathbf{x}_i|C_1)p(C_1) + p(\mathbf{x}_i|C_2)p(C_2)} \tag{4.53}$$

which can be rewritten, for class C_1, as

$$p(C_1|\mathbf{x}_i) = \left(1 + e^{(-s)}\right)^{-1} \tag{4.54}$$

where s is the log–posterior ratio

$$s = \log \frac{p(C_1|\mathbf{x})}{p(C_2|\mathbf{x})} = \log \frac{p(\mathbf{x}|C_1)}{p(\mathbf{x}|C_2)} + \log \frac{p(C_1)}{p(C_2)} \tag{4.55}$$

and σ is the logistic sigmoid function. Under the assumption of uniform priors s reduces to the log–likelihood ratio. Assume that s be represented as a linear function of some (unknown) parameters $\mathbf{w}$ and b, that is $s = \mathbf{w}^T\mathbf{x} + b$. Of course this is not true in the general case. However, if we assume that the class–conditional likelihoods $P(x|C_i)$ are Gaussian distributed with common covariance matrix it is possible to show that the functional form for s becomes exact [12]. LR provides then a discriminative framework to compute an estimate of parameters $\mathbf{w}$ and b which maximizes the likelihood of the training set class labels.

Estimate of the LR decision boundary

As with the SVM we consider a set of m N–dimensional pattern vectors $\mathbf{x}_k$ belonging to either of two classes C_1, C_2 with associated labels t_i. In this case, we assume that labels belong to the set $\{0, 1\}$. Let $\gamma_i = \sigma(\mathbf{w}^T\mathbf{x} + b)$ denote the class posterior probability $p(C_1|\mathbf{x}_i)$. The log–likelihood of target labels can then be expressed as [12]

$$\log p(t_1, \ldots, t_n|\mathbf{w}) = -\sum_{i=1}^{n} \left(t_i \log \gamma_i + (1 - t_i) \log (1 - \gamma_i)\right) \tag{4.56}$$

which corresponds to the cross–entropy between target labels and class posteriors. ML estimate of hyperparameters $\mathbf{w}$ is obtained by minimizing this function.

Simple ML estimate of LR hyperparameters may incur in extreme overfit in the separable classes case [12]. In order to avoid this phenomenon we can put a prior over the hyperparameters and perform MAP estimate of the class boundary. The resulting model is called regularized logistic regression (the reason is that a prior over $\mathbf{w}$ can be interpreted also as a regularization term).

4.1.3 Regularized LR and SVM

Let consider again the LR objective function (4.56) and assume the same label encoding used for the SVM, i.e. let $y_i = -1$ whenever $t_i = 0$ and $y_i = 1$ when $t_i = 1$. The optimal hyperplane can be evaluated as

$$\begin{aligned}
\mathbf{w}^* &= \arg\min_{\mathbf{w},b} -\sum_{i=1}^{n} \left(t_i \log \gamma_i + (1-t_i)\log\left(1-\gamma_i\right)\right) \\
&= \arg\min_{\mathbf{w},b} -\sum_{i=1}^{n} \left\{t_i \log\left[\sigma\left(\mathbf{w}^T\mathbf{x}+b\right)\right] + (1-t_i)\log\left[-\sigma\left(\mathbf{w}^T\mathbf{x}+b\right)\right]\right\} \\
&= \arg\min_{\mathbf{w},b} \frac{1}{n}\sum_{i=1}^{n} \log\left(1+e^{-y_i\left(\mathbf{w}^T\mathbf{x}_i+b\right)}\right) \qquad (4.57)
\end{aligned}$$

where we have used the properties of the sigmoid function $\sigma(a) = \frac{1}{1+e^{-a}}$

$$\sigma(-a) = 1 - \sigma(a) \qquad (4.58)$$

$$a = \log\left(\frac{\sigma(a)}{1-\sigma(a)}\right) \qquad (4.59)$$

and the fact that $t_i = \frac{y_i+1}{2}$. We also scaled the function by the number of patterns, which does not change the result in the case of unregularized LR.

Finally, we consider a regularizer of the form $\frac{\lambda}{2}\|\mathbf{w}\|^2$, which corresponds to an isotropic Gaussian prior on $\mathbf{w}$. The regularized LR objective function $f_{LR}(\mathbf{w}, b)$ is then

$$f_{LR}(\mathbf{w},b) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n} \log\left(1+e^{-y_i\left(\mathbf{w}^T\mathbf{x}_i+b\right)}\right) \qquad (4.60)$$

We can observe the similarities with the SVM objective function

$$f_{SVM}(\mathbf{w},b) = \|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \max\left(0, 1-y_i\left(\mathbf{w}^T\mathbf{x}+b\right)\right) \qquad (4.61)$$

Let $\lambda = \frac{1}{nC}$. A new, equivalent objective function can be devised for the SVM as

$$f'_{SVM}(\mathbf{w},b) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n} \max\left(0, 1-y_i\left(\mathbf{w}^T\mathbf{x}+b\right)\right) \qquad (4.62)$$

LR and SVM objective functions then differ only for the form of the *loss* term $l(\mathbf{w_i}^T\mathbf{x_i}, y_i)$. The difference between the two loss functions is depicted in Figure 4.2.

In particular, both the SVM and regularized LR can be interpreted as instances of a broader class of problems which go under the name of *regularized risk minimization* problems, i.e. problems of the form

$$\min_{\mathbf{w}} \lambda\Omega(\mathbf{w}) + R_{emp}(\mathbf{w}) \qquad (4.63)$$

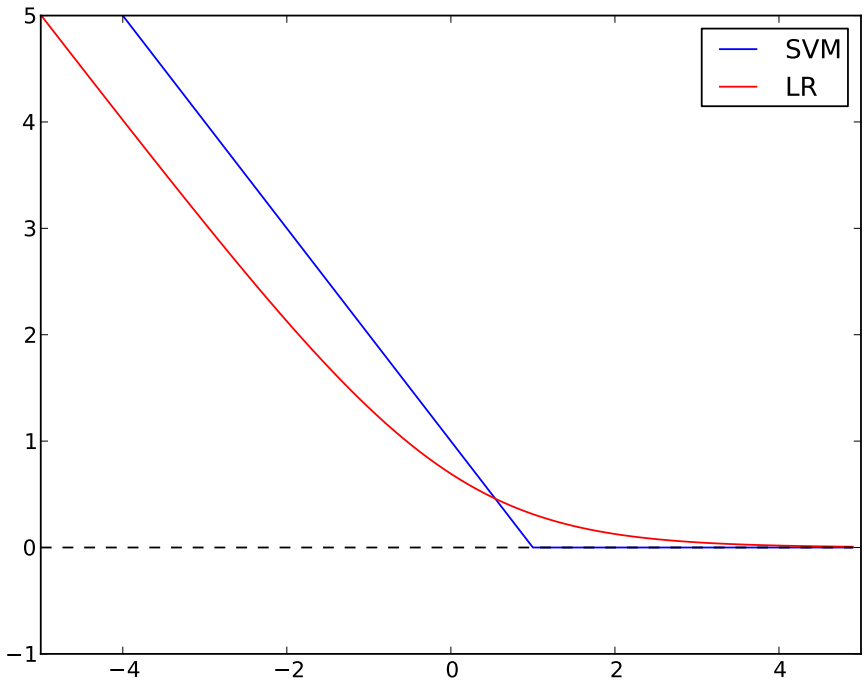


Figure 4.2: LR and SVM loss functions

where Ω is a convex regularization term and R_{emp} is the empirical *risk*, i.e.

$$R_{emp} = \frac{1}{n}\sum_{i=1}^{n} l(\mathbf{w}, \mathbf{x}_i, y_i) \tag{4.64}$$

with $l(\mathbf{w}, \mathbf{x}_i, y_i)$ being a convex loss function. This allows to elaborate a training framework which can to accommodate both SVM and regularized LR [85, 86]. In Section 4.3.2 we will present a techniques based on bundle methods. This technique, however, might fail when the regularizer term λ is small. The LR optimization problem, on the other hand, can efficiently be solved also with different second–order optimization techniques, e.g. Newton–Rhapson updates [12, 87].

4.1.4 Multiclass SVM and LR for language recognition

SVM and LR can be generalized to multiclass problems [12]. For multiclass SVM and multiclass LR, the parameter $\mathbf{w}$ is actually a matrix $\mathbf{W} = [\mathbf{w}_1, \dots \mathbf{w}_r]$ representing

a set of hyperplanes, one for each class. Labels are no more binary since each class has its own label.

The class scores for test patterns are given by the projection of test patterns over set of hyperplanes $\mathbf{w}_i$.

Multiclass SVM

Let the r classes be labeled as $\{y_1, \dots y_r\}$. The loss function for multiclass SVM [88] is an extension of the binary loss function given by

$$l(\mathbf{W}, \mathbf{x}_i, y_i) = \max_{y'} \mathbf{w}_{y'}^T \mathbf{x}_i - \mathbf{w}_{y_i}^T \mathbf{x}_i + \Delta(y', y_i) \tag{4.65}$$

where $\Delta(y_1, y_2)$ is the cost of misclassifying class y_1 for y_2. In this work we use $\Delta(y_i, y_j) = 1 - \delta_{ij}$, where δ_{ij} is the Kronecker delta. Multiclass SVM can be interpreted as a joint optimization of r SVMs where the hyperplanes are trained as to maximize the margin between each class and all the remaining classes [12].

Multiclass LR

As for multiclass SVM, multiclass logistic regression is an extension of binary logistic regression. While binary LR minimizes the binary cross entropy between labels, multiclass LR optimizes the multiclass cross entropy for training labels [12]. Multiclass LR assumes that the posterior for each class can be computed as

$$P(C_i|\mathbf{x}) = \frac{e^{\mathbf{w}_i^T \mathbf{x}}}{\sum_k e^{\mathbf{w}_k^T \mathbf{x}}} \tag{4.66}$$

LR then allows to estimate the parameters $\mathbf{W}$ which maximize the likelihood of the training labels, which corresponds to the minimization of the multiclass cross–entropy between the posteriors and the training labels. This corresponds to the loss function

$$l(\mathbf{W}, \mathbf{x}_i, y_i) = \left(\log \sum_{y'} e^{\mathbf{w}_{y'}^T \mathbf{x}_i} \right) - \mathbf{w}_{y_i} \mathbf{x} \tag{4.67}$$

4.1.5 Multiclass Score Backprojection

The last approach to multiclass classification we consider consists in training binary classifiers for each pair of classes (i.e. we train $\frac{1}{2} r(r-1)$ classifiers for r classes) and then transform the scores so that they can be interpreted as multiclass scores [89].

Assuming that class–conditional log–likelihoods are available, log–likelihood ratios between a set of two classes can, in fact, be computed as the difference between the scores for the two languages. Therefore, we can map multiclass scores to binary

scores with a simple linear transformation represented by a rectangular $N \times \frac{1}{2}r(r-1)$ matrix whose entries are in $\{-1, 0, +1\}$. Each row of the matrix maps the multiclass scores to a binary score and thus has exactly one element valued $+1$ and one element valued -1. Starting from binary scores, we can recover the original multiclass log–likelihoods by simply solving an overdetermined linear system.

In general, when binary scores are produced by different classifiers the solution to this system does not exist. However, we can interpret the binary scores as affected by noise and estimate the multiclass scores as the least squares solution of the system. The mapping between binary scores and multiclass scores is then simply given by the projection of the binary scores on the pseudo inverse of the multiclass–to–binary transformation matrix.

4.2 SVM–based language identification

In this section we present several state–of–the–art SVM–based techniques for language recognition. The next section is dedicated to an analysis of different SVM algorithms and their usability for the different techniques described in this section.

Language identification (LID) usually comes under the form of a closed set identification problem [90, 91, 92], where the goal is to estimate, for a given utterance, the conditional log–likelihood of that utterance given the different target languages. Log–likelihood scores allow to derive decision rules for different sets of problems by simply applying Bayes decision rule with different language priors. While speaker verification can be cast as a two–class problem, where the classes correspond to "utterances from the same speakers" and "utterances from different speakers" (see also Chapter 5), language recognition is naturally posed as a multiclass problem where each language corresponds to a single class. While LR can be easily extended to model multiclass posteriors, SVMs are usually cast as binary classifications problems. In Section 4.1.4 we showed some extensions for multiclass SVM, however state–of–the–art systems tend to stick with the binary SVM formulation. In order to produce multiclass scores SVMs are therefore trained in a one–versus–all fashion, i.e. for each language one SVM model is trained. The SVM classes are defined according to the "target language" versus "non–target" language division, i.e. all non–target language utterances are pooled together against the utterances of the target language [26]. Since SVMs do not produce log–likelihood scores, some transformations (e.g. score calibration [6]) might be required before the scores can be used to perform language inference. The systems presented in the next two sections are based on this framework.

4.2.1 GSV–SVM and pushed–GMM

The first two techniques we present are based on acoustic modeling of utterances by means of GMM supervectors (GSV). The first approach builds an SVM model for each target language using MAP–adapted GMM supervectors as features [93, 94, 95]. Test utterance GMMs are then directly scored against the different SVM hyperplanes. A suitable distance function between GMM supervectors has been proposed in [93]. Assuming diagonal covariance UBM/GMMs, this distance can be interpreted as an upper bound of the Kullback–Leibler divergence [96] between the considered GMMs $\mathbf{g}_a, \mathbf{g}_b$

$$D(\mathbf{g}_a \| \mathbf{g}_b) = \int_{\mathbb{R}^n} \mathbf{g}_a \log \frac{\mathbf{g}_a}{\mathbf{g}_b} \tag{4.68}$$

This upper bound is defined as [97, 93]

$$d(\mathbf{g}_a \| \mathbf{g}_b) = \sum_{i=1}^{m} w_i (\mu_{ai} - \mu_{bi})^T \Sigma_i^{-1} (\mu_{ai} - \mu_{bi}) \tag{4.69}$$

From distance (4.69) a corresponding kernel can be derived as [93]

$$K(\mathbf{g}_a, \mathbf{g}_b) = \sum_{i=1}^{m} \left(w_i^{\frac{1}{2}} \Sigma_i^{-\frac{1}{2}} \mu_{ai} \right)^T \left(w_i^{\frac{1}{2}} \Sigma_i^{-\frac{1}{2}} \mu_{bi} \right) \tag{4.70}$$

Observe that the corresponding kernel is linear, and, as such, can be interpreted as performing a normalization of the feature vectors. In this way, a simple linear SVM model can be trained in the normalized feature space.

While GSV–SVM models allow obtaining reasonable results, more effective models based on this approach combine SVMs and GMMs to provide discriminatively trained generative models.

The main flavor of this approach is the so–called pushed–GMM approach [98, 95], where the discriminative GMM models are derived from the generative ones, by exploiting the information provided by the non–null Lagrange multipliers obtained by training a SVM. In particular, SVM training is first performed using the GMM-SVM approach, then two GMMs are created for each language: one for the target language $\mathbf{g}^+$ and the other for the non–target languages (anti–model) $\mathbf{g}^-$ according to

$$\mathbf{g}^+ = \frac{1}{\sum_{i|y_i>0} \alpha_i} \sum_{i|y_i>0} \alpha_i \mathbf{g_i} \tag{4.71}$$

$$\mathbf{g}^- = \frac{1}{\sum_{i|y_i<0} \alpha_i} \sum_{i|y_i<0} \alpha_i \mathbf{g_i} \tag{4.72}$$

where $\mathbf{g_i}$ is the GMM of the i–th utterance of the target language. Thus, the target model is a weighted combination of the GMMs belonging to the target language,

and the weights are the Lagrange multipliers $\boldsymbol{\alpha}^*$ obtained from the dual solution of the SVM problem. The target language anti–model is a weighted combination of the GMMs of the non–target languages utterances. Language log–likelihood ratios are then computed from model and anti–model. The pushed–GMM approach requires the dual solution of the SVM problem, which, depending on the algorithm, may not be available when the SVM problem is solved in its primal form.

4.2.2 Language factors

The success of eigenvoice models in speaker recognition has motivated the analysis and development of similar systems for language recognition. In [99] the authors propose to use a simple factor analysis model to represent language GMMs. The model is similar to that of JFA introduced in Section 3.3.5, that is, a GMM $\mathbf{g}$ is decomposed as

$$\mathbf{g} = \boldsymbol{\mu} + \mathbf{V}\mathbf{y} + \mathbf{U}\mathbf{x} \tag{4.73}$$

In [99] the authors propose two methods to train the subspace $\mathbf{V}$. The first approach, called *speaker compensated eigenvoices*, trains the subspace matrix from speaker–compensated features. In particular, an inter–speaker (and partially intersession) subspace matrix $\mathbf{U}$ is estimated using a large set of differences between models generated by different speakers of the same language. Feature–level compensation is then carried out as in [100]. Given the occupation probabilities for each Gaussian γ_{it} for observation o_t, the compensated features are computed as

$$\hat{o}_t = o_t - \sum_{i=1}^{m} \gamma_{it} \mathbf{U}_i \mathbf{x} \tag{4.74}$$

where $\mathbf{x}$ is the channel factor associated to the utterance. The subspace matrix $\mathbf{V}$ is then computed as to estimate the eigenvoices of the feature–compensated utterances. The assumption behind this approach is that speaker–compensated eigenvoices would retain less speaker (and channel) information, and would therefore capture more language information. The second approach starts from speaker–compensated features and tries to find directions that further enhance the discrimination among languages. To find these directions the authors consider a polyglot speaker that utters a set of phonetically rich sentences in different languages. GMM supervector differences for utterances spoken by such speaker in different languages would factor out the speaker characteristics and enhance the acoustic components of a language with respect to the others. To cope with the difficulty in finding such a speaker, the authors propose to consider differences between GMMs of speakers of two different languages without caring about speaker identity, assuming that speaker–compensation has successfully removed most of the speaker information. PCA is then used to estimate the $\mathbf{V}$ matrix. In order to reduce the complexity

of estimating all cross–pair differences, the authors propose to compute only the difference of each GMM from the centroid of the GMMs of each other language.

One of the main advantages of language factors consists in their low dimensionality, which makes SVM training easier when the datasets are large. In [99] a way to adapt the standard GSV–SVM with KL kernel to language factors is shown. SVM training is performed in the language factor space, which has very few dimensions compared to the GMM one. Moreover, the KL kernel can be effectively implemented as a linear kernel in the language factor space [99]. To show this, consider UBM–centered GMMs, i.e. $\hat{\mathbf{g}} = \mathbf{g} - \boldsymbol{\mu}_{ubm}$. The KL distance (4.69) is invariant to space translations, thus we can keep the expression for the kernel given by (4.70). We can rewrite (4.70) as

$$K(\hat{\mathbf{g}}_a, \hat{\mathbf{g}}_b) = \left(\mathbf{W}^{\frac{1}{2}}\boldsymbol{\Sigma}^{\frac{1}{2}}\mathbf{V}\mathbf{y}_a\right)^T\left(\mathbf{W}^{\frac{1}{2}}\boldsymbol{\Sigma}^{\frac{1}{2}}\mathbf{V}\mathbf{y}_b\right) = \mathbf{y}_a^T\hat{\mathbf{V}}^T\hat{\mathbf{V}}\mathbf{y}_b \tag{4.75}$$

where $\mathbf{W}$ is a diagonal matrix whose values are the UBM weights repeated F times (F denotes the feature dimensionality), $\boldsymbol{\Sigma}$ is a block–diagonal matrix whose elements are the UBM covariance matrices and $\hat{\mathbf{V}} = \mathbf{W}^{\frac{1}{2}}\boldsymbol{\Sigma}^{\frac{1}{2}}\mathbf{V}$. If we denote with $\boldsymbol{\Lambda}^T\boldsymbol{\Lambda} = \hat{\mathbf{V}}^T\hat{\mathbf{V}}$ the Cholesky decomposition of $\hat{\mathbf{V}}^T\hat{\mathbf{V}}$ the kernel (4.75) can be rewritten as

$$K(\hat{\mathbf{g}}_a, \hat{\mathbf{g}}_b) = \mathbf{y}_a^T\boldsymbol{\Lambda}^T\boldsymbol{\Lambda}\mathbf{y}_b = \hat{\mathbf{y}}_a^T\hat{\mathbf{y}}_b \tag{4.76}$$

where $\hat{\mathbf{y}}_i = \mathbf{y}_i\boldsymbol{\Lambda}$. The KL kernel thus corresponds to a linear kernel in the language factor space.

Finally, the pushed–GMM approach can be also applied by simply computing the language model and anti–model as

$$\begin{aligned}\mathbf{g}^+ &= \boldsymbol{\mu}_{ubm} + \mathbf{V}\sum_{i|y_i>0}\frac{1}{\sum_{i|y_i>0}\alpha_i}\alpha_i\mathbf{y}_i \\ \mathbf{g}^- &= \boldsymbol{\mu}_{ubm} + \mathbf{V}\sum_{i|y_i<0}\frac{1}{\sum_{i|y_i<0}\alpha_i}\alpha_i\mathbf{y}_i\end{aligned} \tag{4.77}$$

4.2.3 Acoustic i–vectors

Following the success in speaker modeling, language recognition systems have seen the recent introduction of language i–vectors. Language i–vectors share all the modeling techniques used for speaker i–vectors, however they are usually built over SDC features instead of MFCCs. The low–dimensionality of i–vectors allows to directly build multiclass SVM and LR models where the i–vectors are the features. Although these systems are very simple, since just a single multiclass LR or SVM has to be trained over all set of i–vectors, they proved to be effective in recent NIST evaluations.

4.2.4 Phonetic models

Acoustic models are often combined with another set of powerful models which provide complementary information. These models are based on phonetic features extracted by a phone or word tokenizer. Most of these models represent utterances as *bags of n–grams*, that is, the occurrences of different sequences of tokens (usually up to third or fourth order) are stacked in a single vector associated to the utterance. The n–grams (Section 2.5.1) are extracted either by 1–best decoding or through lattice decoding.

SVM–based phonetic models

While language models can be built using directly n–gram counts (Section 2.5.1), a successful approach to phonotactic language verification consists in training a discriminative classifier in the n–gram vector space [101, 102, 103, 104]. The concept is similar to the GSV–SVM approach (Section 4.2.1), that is, for each language a SVM is trained in a one–versus–all fashion using as features the n–gram statistics. Many different kernels have been proposed in the last years, a popular one being the Term Frequency Log Likelihood Ratio (TFLLR) kernel [101, 103]. The TFLLR kernel is a linear kernel and can therefore be expressed as a normalization of n–gram features

$$\hat{\mathbf{x}}_k^i = \sqrt{\frac{1}{f_i}}\mathbf{x}_k^i, \quad f_i = \sum_{j=1}^{n} \mathbf{x}_j^i \tag{4.78}$$

where $\mathbf{x}_k^i$ denotes the i–th component of the n–gram vector $\mathbf{x}_k$ associated to utterance k.

Phonotactic i–vectors

Following the success of low–dimensional subspace representations for acoustic GMMs, subspace models have been recently extended to phonetic systems. In analogy with the acoustic i–vector approach the phonotactic i–vector framework [89, 105] assumes that stacked n–gram vectors can be represented by a small dimensional vector which can be estimated in a maximum likelihood fashion. The main idea consists in interpreting n–gram counts as counts of exclusive events, so that they can be modeled by a multinomial distribution whose underlying parameters are constrained to live in a small subspace [89, 105, 106]. The model can be formalized as follows. Let ν_i denote a supervector of n–gram counts (i.e. the vector of stacked counts) for utterance i. We assume that this supervector has been produced by a discrete process governed by a multinomial distribution with parameters ϕ_i, so that the likelihood of the observed data is given by

$$\log P(\nu_i|\phi_i) = \sum_c \nu_{ic} \log \phi_{ic} \tag{4.79}$$

where ν_{ic} and ϕ_{ic} denote the c–th element of supervector ν_i and parameter vector ϕ_i respectively. The complete data log–likelihood is given by

$$\log P(D) = \sum_j \sum_c \nu_{jc} \log \phi_{jc} \tag{4.80}$$

where D denotes the complete set of utterances. The subspace model proposed in [105, 106] assumes that the utterance–dependent multinomial parameters ϕ_i actually lie in a small r–dimensional subspace defined by

$$\phi_{ic} = \frac{e^{m_c + \mathbf{t}_c \mathbf{w}_i}}{\sum_j e^{m_j + \mathbf{t}_j \mathbf{w}_i}} \tag{4.81}$$

where $\mathbf{w}$ is an r–dimensional vector which, in analogy to acoustic i–vectors, represents channel and speaker information of utterance i. The row vectors $\mathbf{t}_c$ are the rows of the subspace matrix $\mathbf{T}$ and m_c is a bias term. This expression can be simplified replacing $\mathbf{w}_i$ by $\bar{\mathbf{w}}_i = \left[\mathbf{w}_i^T 1\right]^T$ and $\mathbf{t}_c$ by $\bar{\mathbf{t}}_c = [\mathbf{t}_c m_c]$ as

$$\phi_{ic} = \frac{e^{\bar{\mathbf{t}}_c \bar{\mathbf{w}}_i}}{\sum_j e^{\bar{\mathbf{t}}_j \bar{\mathbf{w}}_i}} \tag{4.82}$$

Maximum Likelihood estimation can be used to estimate the model parameters $\mathbf{T}$. In [105, 106] the authors propose an iterative algorithm which alternates estimates of $\mathbf{T}$ and $\mathbf{w}_i$. Each step is then optimized with an iterative algorithm similar to Newton–Rhapson updates [12, 87].

4.3 Large–scale SVM algorithms

In this chapter we have presented many successful language recognition techniques based on SVMs and other discriminative classifiers. We did not, however, consider the issues involved in training the SVM classifiers for these systems. The increase of the size of corpora and the dimensions of the feature patterns do not, in general, allow the use medium–scale training algorithms, which assume that all the datasets can be stored into main memory. While this is not a problem for i–vector and language factor systems, both the GSV–SVM and the classical phonotactic approaches have to deal with patterns which can easily exceed some tens of thousand dimensions. Moreover, classical SVM algorithms are characterized by a squared time complexity with respect to the database size. Thus, linear memory and linear time algorithms become more and more appealing. Many algorithms have been proposed to handle SVM optimization for large–scale problems. While most of them are efficient only for linear kernels, this does not cause many problems since the most used kernels in language recognition are in fact linear (Section 4.2). Following the work in [107, 108]

this section details five large–scale training algorithms and their usability in language (and speaker) recognition systems. In the following sections n denotes the number of patterns in the training set and d is the pattern dimensionality. Following the convention in Section 4.1, C denotes the regularization parameter of the SVM. We also denote by ε the optimization accuracy.

4.3.1 Dual solvers

In this section we analyze three dual SVM problem solvers, whereas in Section 4.3.2 we will describe two primal solvers and the steps to derive their corresponding dual solutions, needed by the pushed–GMM approach. For ease of reading, we rewrite the dual SVM problem (4.50)

$$\begin{aligned} \boldsymbol{\alpha}^* = \arg\max_{\boldsymbol{\alpha}} \boldsymbol{\alpha}^T \mathbf{e} - \frac{1}{2}\boldsymbol{\alpha}^T \mathbf{H} \boldsymbol{\alpha} \\ \text{s.t. } 0 \leq \alpha_k \leq C, \quad \forall k \end{aligned} \tag{4.83}$$

SVM$^{\text{Light}}$

A popular "fast" linear–space SVM solver is SVM$^{\text{Light}}$ [84]. SVM$^{\text{Light}}$ decomposes the SVM problem into a set of subproblems and iteratively optimizes these subproblems. Its memory occupation scales linearly with the number of training patterns and of support vectors. Since SVM$^{\text{Light}}$ solves the dual problem it provides the Lagrange multipliers needed in language recognition by the pushed–GMM approach.

The main limitation of this algorithm comes from its time complexity, which has been empirically shown to be $O(n^2 d)$. If memory is not a constraint, a fast implementation of SVM$^{\text{Light}}$ can be obtained by caching all the kernel evaluations. Of course, the kernel matrix has a size, and thus a computational cost, which still grows quadratically with the training set size.

Dual Coordinate Descent

In [109] the authors propose to solve the dual problem by means of a coordinate descent approach, referred to in the following as Dual Coordinate Descent Method (DCDM). The multivariate problem is split into a sequence of univariate optimizations which are iteratively solved until convergence to the optimal multivariate solution is attained. Assuming that a sub–optimal solution $\boldsymbol{\alpha}$ of the dual problem (4.83) is known, the i–th component of the optimal solution, given all the other coordinates, can be evaluated by solving

$$\min_h f(\boldsymbol{\alpha} + h\mathbf{e}_i) \quad \text{subject to } 0 \leq \alpha_i + h \leq C \tag{4.84}$$

where $\mathbf{e}_i$ denotes the unit vector with i–the element equal to one. The objective function is quadratic in h

$$f(\boldsymbol{\alpha} + h\mathbf{e}_i) = \frac{1}{2}H_{ii}h^2 + \nabla_i f(\boldsymbol{\alpha}) + K \tag{4.85}$$

for a given constant K. The minimization of this function leads to the update rule [109]

$$\alpha_i \leftarrow \min\left[\max\left(\alpha_i - \frac{\nabla_i f(\boldsymbol{\alpha})}{H_{ii}}, 0\right), C\right] \tag{4.86}$$

where the gradient $\nabla_i f(\boldsymbol{\alpha})$ is

$$\nabla_i f(\boldsymbol{\alpha}) = \sum_{j=1}^{n} H_{ij}\alpha_j - 1 \tag{4.87}$$

The computation of the gradient is in general expensive, but for linear SVMs it simplifies to

$$\nabla_i f(\boldsymbol{\alpha}) = y_i \mathbf{w}^T \mathbf{x}_i - 1 \tag{4.88}$$

The cost of evaluating $\mathbf{w}$ given $\boldsymbol{\alpha}$ would be linear with the size of the training set. However, by keeping the previous value of $\mathbf{w}$ it can be updated according to

$$\mathbf{w} \leftarrow \mathbf{w} + (\alpha_i - \alpha_i^{old}) y_i \mathbf{x}_i \tag{4.89}$$

where α_i^{old} refers to the value the parameter α_i had before being updated. Since DCDM solves the dual formulation of the SVM problem, it directly provides the Lagrange multipliers required by the pushed–GMM approach. The time complexity of the algorithm is $O\left(nd \log\left(\frac{1}{\varepsilon}\right)\right)$. DCDM is very fast, however it cannot take advantage of a distributed environment because the solution is updated after each pattern is processed.

SVM$^{\text{Perf}}$

One of the most effective linear–time SVM solvers is SVM$^{\text{Perf}}$ [110, 111, 112]. Though the package provides different algorithms for solving the SVM problem, its main innovation is the Cutting–Plane Subspace–Pursuit (CPSP) approach [112]. Cutting–Plane algorithms are based on a different formulation of the primal problem (4.49)

$$\begin{aligned} &\min_{\mathbf{w},\xi} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\xi \\ &\text{subject to} \quad \forall \hat{y}_1 \dots \hat{y}_n \in \{-1, +1\} : \\ &\frac{1}{2}\mathbf{w}^T\left[\sum_{i=1}^{n}(y_i\mathbf{x}_i - \hat{y}_i\mathbf{x}_i)\right] \geq \sum_{i=1}^{n}\Delta(y_i, \hat{y}_i) - \xi \end{aligned} \tag{4.90}$$

where $\Delta(y, \hat{y})$ is the zero–one loss function, which takes value 1 when its arguments are equal, and value 0 otherwise.
The solution is found by iteratively building a working set of constraints over which a Quadratic Problem (QP) is solved. An accuracy of ε can be obtained using at most $O\left(\frac{1}{\varepsilon}\right)$ constraints. The CPSP algorithm modifies the traditional Cutting–Plane algorithm by iteratively building a set of *basis vectors* $\{\mathbf{b}_1, \dots \mathbf{b}_k\}$ whose span is approximatively the sub–space where the optimal solution lies [112]. The approximate solution is thus given by

$$\mathbf{w}^* \approx \sum_{i=1}^{k} \beta_i \mathbf{b}_i \tag{4.91}$$

The rationale for the introduction of the basis vectors is to reduce the number of kernel evaluations for non–linear classification. Since basis vectors are associated with the Cutting Plane constraints, which are supposed to be a constant number with respect to the training set, the hyperplane can be represented similarly to (4.51) but using a much smaller set of patterns. It is worth noting that since the basis vectors do not belong to the training set, it is not possible to exploit the possible speedup coming from the pre-computation of the kernel matrix. Moreover, computing the kernel matrix is unfeasible for large datasets.

Due to the nature of the algorithm, it can be easily modified to be executed in a distributed environment.

4.3.2 Primal solvers

In this section we describe two primal solvers and the steps to derive their corresponding dual solutions, which are necessary for the pushed–GMM approach in language recognition (Section 4.2.1). For ease of reading we rewrite the primal SVM problem (4.50)

$$\mathbf{w}^*, b^* = \arg\max_{\mathbf{w}, b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \max\left(0, 1 - y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right)\right) \tag{4.92}$$

In the following we will not consider the bias term b. In order to estimate the bias we can nevertheless append to the features a constant and then include the bias in the hyperplane. Although this slightly modifies the optimization problem, since the regularizer shouldn't depend on the bias, choosing a large constant to append to the training patterns allows to reduce the impact of the bias on the regularizer.

Pegasos

The first solver is based on gradient descent in the primal solution space. Standard gradient descent techniques try to reach the minimum of the objective function

by iteratively moving an approximate solution along the direction that gives the greatest decrease of the objective function. For the hinge–loss function this leads to the update rule

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \left[\mathbf{w}_t + C \sum_{i=1}^{n} \nabla_{\mathbf{w}} \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i)|_{\mathbf{w}_t} \right] \tag{4.93}$$

where η_t is the learning rate at iteration t. The selection of the learning rate values is crucial for fast convergence of the algorithm. Since the SVM loss function is not completely differentiable but still convex, a *subgradient* of the loss function can be computed as

$$\nabla_{\mathbf{w}} \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i) = \begin{cases} -y_i & \text{if } y_i \mathbf{w}^T \mathbf{x}_i \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.94}$$

Stochastic Gradient Descent (SGD) approximates the gradient computation step by evaluating the subgradient of the objective function on a pattern (or on a small subset of patterns)

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \left[\mathbf{w}_t + nC \nabla_{\mathbf{w}} l(\mathbf{w}, \mathbf{x}_{i_t}, y_{i_t})|_{\mathbf{w}_t} \right] \tag{4.95}$$

where i_t is chosen randomly for each iteration.

Pegasos [113] combines SGD descent of the SVM L1–loss function with a projection step ensuring faster convergence to the optimal solution. A set of training patterns $\mathcal{A}_t$ is randomly chosen at each iteration. The subgradient of the objective function is estimated from this subset as

$$\nabla_t = \mathbf{w}_t - \frac{nC}{|\mathcal{A}_t|} \sum_{\substack{i|\mathbf{x}_i \in \mathcal{A}_t \\ y_i \mathbf{w}^T \mathbf{x}_i < 1}} y_i \mathbf{x_i} \tag{4.96}$$

and the hyperplane is updated as

$$\mathbf{w}_{t+\frac{1}{2}} = \mathbf{w}_t - \eta_t \nabla_t \tag{4.97}$$

The optimal SVM solution is bounded by $\|\mathbf{w}\| \leq \sqrt{nC}$ [113]. Therefore, the current solution is projected onto a ball of radius $\sqrt{nC}$ by scaling $\mathbf{w}_{t+1/2}$ according to

$$s_t = \min \left\{ 1, \frac{\sqrt{nC}}{\|\mathbf{w}_{t+1/2}\|} \right\} \tag{4.98}$$

$$\mathbf{w}_{t+1} = s_t \mathbf{w}_{t+1/2} \tag{4.99}$$

This projection step, combined with a fast–decaying learning rate, allows bounding to $O\left(\frac{1}{\varepsilon}\right)$ the average number of iterations required to achieve ε optimization accuracy [113].
Since Pegasos solves the primal formulation of the SVM problem it does not produce the Lagrange multipliers, which are necessary in the pushed–GMM approach. In [113] the authors propose an extension of their algorithm that allows the hyperplane to be estimated as a linear combination of training patterns $\mathbf{w} = \sum_i \alpha_i \mathbf{x}_i$ where the set of $\boldsymbol{\alpha}$'s are iteratively obtained as

$$\boldsymbol{\alpha}_{t+1} = \left[\boldsymbol{\alpha}_t - \eta_t \left(\boldsymbol{\alpha}_t + \frac{nC}{|\mathcal{A}_t|}\chi_t^i\right) s_t\right] \tag{4.100}$$

with

$$\chi_t^i = \begin{cases} 1 & \text{if } x_i \in \mathcal{A}_t \\ 0 & \text{if } x_i \notin \mathcal{A}_t \end{cases} \tag{4.101}$$

Since Pegasos is based on stochastic gradient descent, which performs sequential updates, it cannot take advantage of a distributed environment.

Bundle Methods

Bundle methods approximate a convex function by means of a set of tangent hyperplanes (subgradients) and solve the simpler optimization problem on the approximated function. The approach is similar to SVM$^{\text{Perf}}$, where a small and incremental subset of constraints is built until the solution approximates the optimal solution up to a given error. Bundle Methods for Regularized Risk Minimization (BMRM) [85, 86] offer a general and easily extensible framework to general risk regularization problems, of which SVM is an example. In particular, an incremental working set of approximate solutions $\{\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2, \dots\}$ is built by defining, at each iteration, the set of hyperplanes which are tangent to the objective function in the working set points (starting from $\mathbf{w}_0 = \mathbf{0}$)

$$f_t(\mathbf{w}) = l_{emp}(\mathbf{w}_t) + \nabla l_{emp}(\mathbf{w}_t) \cdot (\mathbf{w} - \mathbf{w}_t) \tag{4.102}$$

where $l_{emp}(\mathbf{w}) = \sum_{i=1}^{n} l(\mathbf{w}, \mathbf{x_i}, y_i)$ is the empirical loss. At each iteration, a new working point is selected as the minimizer of the approximation function

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}} \left[\frac{1}{2}\|\mathbf{w}\|^2 + C \cdot \max\left(0, \max_{t' \le t+1} f_{t'}(\mathbf{w})\right)\right] \tag{4.103}$$

In [85] it is shown that this problem is equivalent to the dual quadratic problem

$$\begin{aligned} \min_{\beta} D_i(\beta) \quad &= \tfrac{C}{2}\beta^T A^T A \beta - \beta^T \mathbf{b} \\ \text{subject to} \quad &\beta \ge \mathbf{0}, \quad \mathbf{e}^T\beta \le 1 \end{aligned} \tag{4.104}$$

where $\mathbf{e}$ is a vector of ones, A is the matrix $[a_1 a_2 \dots a_i]$ of gradients $a_{t+1} = \nabla l_{emp}(\mathbf{w}_t)$ and $\mathbf{b}$ is the vector $[b_1 b_2 \dots b_i]^T$ of offsets $b_{t+1} = l_{emp}(\mathbf{w}_t) - a_{t+1}^T \mathbf{w}_t$. The new solution is obtained as

$$\mathbf{w_{t+1}} = -CA\beta \tag{4.105}$$

This quadratic problem is not expensive because its complexity does not increase with the size of the training dataset, but with the number of iterations only.
The BMRM algorithm does not directly provide the Lagrange multipliers for the dual SVM problem. However, the dual solution can still be computed as follows [107, 108]. The hinge loss function can be rewritten to make explicit its dependency on the dot product between $\mathbf{w}$ and a given pattern $\mathbf{x}$ as

$$l(\mathbf{w}, \mathbf{x}, y) = \widetilde{l}(\mathbf{w}^T \mathbf{x}, y) \tag{4.106}$$

and its gradient with respect to $\mathbf{w}$ as

$$\nabla_{\mathbf{w}} l(\mathbf{w}, \mathbf{x}, y) = \frac{\partial \widetilde{l}(\mathbf{w}^T \mathbf{x}, y)}{\partial \mathbf{w}^T \mathbf{x}} \mathbf{x} \tag{4.107}$$

Defining an array $\widetilde{a}_t = [\widetilde{l}_1 \widetilde{l}_2 \dots \widetilde{l}_n]^T$, where $\widetilde{l}_i = \frac{\partial \widetilde{l}(\mathbf{w}^T \mathbf{x_i}, y_i)}{\partial \mathbf{w}^T \mathbf{x}}$, we can express a_t as

$$a_t = \mathbf{X} \widetilde{a}_t \tag{4.108}$$

where $\mathbf{X}$ is the complete set of training patterns represented as a matrix. Matrix A in (4.104) can then be evaluated as $A = \mathbf{X}\widetilde{A}$ with $\widetilde{A} = [\widetilde{a}_1 \widetilde{a}_2 \dots \widetilde{a}_t]$, and (4.105) can be rewritten as

$$\mathbf{w_{t+1}} = -C\mathbf{X}\widetilde{A}\beta \tag{4.109}$$

Setting $\boldsymbol{\alpha} = -CY^{-1}\widetilde{A}\beta$, where Y is the diagonal matrix of the target labels $Y_{ii} = y_i$, allows obtaining the separation hyperplane in terms of a linear combination of the training patterns $\mathbf{w} = \mathbf{X}Y\boldsymbol{\alpha} = \sum_i y_i \mathbf{x}_i \alpha_i$ as in (4.51).

The BMRM algorithm converges to its optimal solution up to the accuracy ε in $O\left(\frac{1}{\varepsilon}\right)$ iterations. Usually the number of required iterations is small, thus the time required to solve sub–problems (4.104) can be neglected, and the global complexity of the algorithm is $O\left(\frac{nd}{\varepsilon}\right)$.
Similarly to SVM$^{\text{Perf}}$, BMRM incrementally builds a working set of approximate solutions, thus this algorithm can be easily modified to run in a distributed environment.

An extension to the BMRM algorithm has been proposed in [114]. The Optimized Cutting Plane Algorithm (OCAS) approach [114, 86] tries to simultaneously optimize the original and the BMRM approximated objective function and to select cutting planes that have higher chance to actively contribute to the approximation of the objective function around its optimum. This is done using a linear search

algorithm to optimize the SVM loss function over the line connecting the BMRM solution at a given iteration with the solution obtained in the previous step. OCAS choices of cutting planes allow to reduce the number of iterations needed for convergence, at the expense of a higher execution time per iteration. In particular, OCAS complexity is $O(n \log n)$, however usually $\log n \ll d$. In the latter case the global complexity of the algorithm is dominated by the factor $O(nd)$ needed for the computation of dot–products and gradients.

Chapter 5

SVM–based Speaker Recognition

In this chapter we present two frameworks for discriminative SVM–based speaker verification systems. The first approach, GMM–SVM [93, 94], consists in training a model for each target speaker that is able to discriminate between utterances from that speaker and utterances from different speakers. This technique shares many similarity with the GSV–SVM approach in language identification (Section 4.2.1). In particular, both techniques use GMMs as features for an SVM classifier trained in a one–versus–all fashion. However, while in language verification many utterances are available for each language, in speaker verification usually few recordings (as low as one in NIST evaluations) are available for a target speaker. This poses some issues on how to effectively train the discriminative classifiers. Some improvements have been proposed (e..g. Nuisance Attribute Projection [115, 94]), however recently the focus has shifted on a different training framework, called Pairwise SVM [73, 74], which solves this problem by completely changing the definitions of the classes.

5.1 GMM–SVM

The GMM–SVM approach [93] builds a model for each speaker using MAP adapted GMMs as features for an SVM–based discriminative classifier. Similarly to the LID GSV–SVM model (Section 4.2.1), a possible choice for the SVM kernel is an approximation of the Kullback–Leibler divergence between two GMMs

$$D(\mathbf{g}_a \| \mathbf{g}_b) = \int_{\mathbb{R}^n} \mathbf{g}_a \log \frac{\mathbf{g}_a}{\mathbf{g}_b} \tag{5.1}$$

The corresponding approximated distance is given by

$$d(\mathbf{g}_a \| \mathbf{g}_b) = \sum_{i=1}^{m} w_i (\mu_{ai} - \mu_{bi})^T \Sigma_i^{-1} (\mu_{ai} - \mu_{bi}) \tag{5.2}$$

which corresponds to the kernel function [93]

$$K(\mathbf{g}_a, \mathbf{g}_b) = \sum_{i=1}^{m} \left(w_i^{\frac{1}{2}} \Sigma_i^{-\frac{1}{2}} \mu_{ai} \right)^T \left(w_i^{\frac{1}{2}} \Sigma_i^{-\frac{1}{2}} \mu_{bi} \right) \tag{5.3}$$

A technique for channel compensation in these kind of models, Nuisance Attribute Projection (NAP), was introduced in [116, 115, 94]. NAP assumes that channel effects can be compensated by removing a low number of dimensions in the mapped feature space. Let $A = [\varphi(\mathbf{g}_0), \ldots, \varphi(\mathbf{g}_n)]$, where φ is the mapping function corresponding to a chosen kernel function. Formally, NAP looks for a projection matrix

$$P = I - \sum_i u_i u_i^T \quad \|u_i\| = 1 \; \forall i \tag{5.4}$$

where the u_i are vectors spanning the "noisy" dimensions. Projecting away those dimensions results in a new kernel function

$$K^{'}(\mathbf{g}_a, \mathbf{g}_b) = (P\varphi(\mathbf{g}_a))^T (P\varphi(\mathbf{g}_b)) \tag{5.5}$$

and a corresponding kernel matrix $K^{'}$ given by

$$K^{'} = K - Kv(Kv)^t \tag{5.6}$$

where K is the original kernel matrix $K_{a,b} = K(\mathbf{g}_a, \mathbf{g}_b)$ and $u = Av$. The condition on the norm of u becomes $v^T K v = 1$.

A possible criterion for selecting the discarded dimensions consists in finding the projection matrix which minimizes the distances between projected points having different channels [94]. This corresponds to finding P as

$$P = \arg\min_{\bar{P}} \sum_{i,j} U_{ij} \|P(\phi(\mathbf{g}_i) - \phi(\mathbf{g}_j))\|^2 \tag{5.7}$$

where the elements of matrix U are

$$U_{ij} = \begin{cases} 1 & x_i \text{ and } x_j \text{ have different channels} \\ 0 & \text{otherwise} \end{cases} \tag{5.8}$$

Let $\bar{U}$ denote the diagonal matrix whose diagonal is given by $U\mathbf{1}$, where $\mathbf{1}$ is a vector of all ones. It can be shown [116] that v can be computed as the solution to the generalized eigenvalue problem

$$KZKv = \lambda v \tag{5.9}$$

where $Z = \bar{U} - W$.

Some more refinement of the NAP approach have been proposed (e.g. weighted NAP [117]), however all this techniques suffer from the very limited amount of training data for the target speaker. In the next sections we propose a novel discriminative framework which allows solving some of the GMM–SVM approach issues and, similar to PLDA models, allows reachong state–of–the–art results without the need to train a different model for each speaker.

5.2 Pairwise SVM

Although, GSV–NAP based systems allow good performance, they require a careful selection of the background impostor cohorts, and they need to train an SVM system for each speaker. The main drawback of GSV–NAP systems is the small size of the target speaker class, since the enrollment segments can be as few as just a single utterance. This makes an estimate of the hyperplane unreliable, thus requiring some sort of "compensation" (which is achieved by NAP). This problem is due to the formulation of the task, i.e., the model can be interpreted as a natural solution to the problem of identifying whether a test segment belongs to a target speaker, given a set of enrollment utterances over which the model is built. This statement naturally allows to view speakers as classes, and can be translated directly into the one–vs–all training scheme of GSV–NAP systems. The roles of enrollment and test segments are therefore different, as happens with Factor Analysis based models illustrated in Section 3.3.

On the other hand, the speaker verification task can equivalently be defined as the problem of classifying whether two sets of utterances belong to the same speaker or to different speakers. For generative systems this translates into PLDA–like models, which score these sets of utterances without forcing any interpretation on them and perform generally better than JFA–based and GSV–SVM based systems. We can still refer to the two sets as enrollment and test set, however the role of the utterances has become symmetric. Assume now that we use as features for a classifier these sets of utterances. The natural classes of this problems become then the *same–speaker (target)* class and the *different speaker (non–target)* class [118, 73, 74]. Thus, the multiclass speaker verification problem can be recast as a binary problem where the features are pairs (or sets) of utterances. This is similar to what is done in the PLDA framework, where the models provide likelihood–ratios between same speaker and different speakers hypotheses.

In this section we describe how to build a non–linear SVM which is able to discriminate between these two classes. We also show how the SVM kernel is connected to the PLDA–like models of Section 3.6 and that it can be efficiently implemented for medium–size (around 20 thousand utterances) datasets [73, 74]. A similar approach holds for a LR model [74].

5.2.1 Two–covariance model and pairwise SVM

We consider a single–conversation recognition problem, i.e. where we have a single enrollment utterance with an associated i–vector ϕ_1 and a single test utterance with an associated i–vector ϕ_2 (the roles of ϕ_1 and ϕ_2 are interchangeable). We also consider a two–covariance model (Section 3.6.1) with within–covariance W and between–covariance B. The likelihood–ratio l between same speaker and different

speaker hypotheses can be written as

$$\begin{aligned}\log l = &\frac{1}{2}(\log|B| - \mu^T B\mu + \log|\tilde{\Lambda}| + \gamma_{1,2}^T\tilde{\Lambda}\gamma_{1,2}) \\ &- \frac{1}{2}(2\log|B| - 2\mu^T B\mu + 2\log|\tilde{\Gamma}| + {\gamma_1}^T\tilde{\Gamma}\gamma_1 \\ &+ {\gamma_2}^T\tilde{\Gamma}\gamma_2)\end{aligned} \tag{5.10}$$

where

$$\begin{aligned}\tilde{\Lambda} &= (B+2W)^{-1} & \tilde{\Gamma} &= (B+W)^{-1} \\ \gamma_{1,2} &= B\mu + W(\phi_1+\phi_2) & \gamma_i &= B\mu + W\phi_i\end{aligned} \tag{5.11}$$

Combining all the terms which do not depend on the i–vectors into a single term $\tilde{k}$ we can rewrite the log–likelihood ratio as

$$\log l = \tilde{k} + \gamma_{1,2}^T\tilde{\Lambda}\gamma_{1,2} - {\gamma_1}^T\tilde{\Gamma}\gamma_1 - {\gamma_2}^T\tilde{\Gamma}\gamma_2 \tag{5.12}$$

Replacing (5.11) in (5.12) to make explicit the role of the two i–vectors in the log–likelihood ratio computation we obtain

$$\log l = \frac{1}{2}\Big([B\mu + W(\phi_1+\phi_2)]^T\tilde{\Lambda}[B\mu + W(\phi_1+\phi_2)] \tag{5.13}$$

$$- (B\mu + W\phi_1)^T\tilde{\Gamma}(B\mu + W\phi_1) \tag{5.14}$$

$$- (B\mu + W\phi_2)^T\tilde{\Gamma}(B\mu + W\phi_2) + \tilde{k}\Big) \tag{5.15}$$

which can be rewritten as

$$\log l = \phi_1^T\Lambda\phi_2 + \phi_2^T\Lambda\phi_1 + \phi_1^T\Gamma\phi_1 + \phi_2^T\Gamma\phi_2 + (\phi_1+\phi_2)^T c + k \tag{5.16}$$

where

$$\begin{aligned}\Lambda &= \frac{1}{2}W^T\tilde{\Lambda}W \\ \Gamma &= \frac{1}{2}W^T\left(\tilde{\Lambda} - \tilde{\Gamma}\right)W \\ c &= W^T\left(\tilde{\Lambda} - \tilde{\Gamma}\right)B\mu \\ k &= \frac{1}{2}\tilde{k} + \frac{1}{2}(B\mu)^T\left(\tilde{\Lambda} - 2\tilde{\Gamma}\right)(B\mu)\end{aligned} \tag{5.17}$$

This formulation shows that the log–likelihood ratio can be expressed as a quadratic form of the two considered i–vectors, where the original model parameters are function of Λ, Γ, c and k.

The pairwise SVM approach presented in [73, 74] starts from this formal expression for the log–likelihood ratio and discriminatively trains the transformed model parameters Λ, Γ, c and k. The classes are assumed to be *same speaker pair (target)* and *different speaker pair (non–target)*, corresponding respectively to a positive and negative log–likelihood ratio, and features are pairs of i–vectors $\Phi_{1,2} = \left[\phi_1^T\phi_2^T\right]^T$

5.2.2 Pairwise SVM feature space

The score function (5.16) is clearly non–linear in $\Phi_{i,j}$, however it can be shown that there exists a transformation of the feature vectors $\varphi(\Phi_{i,j})$ such that the scoring function can be expressed as a dot–product between the model parameters and the transformed features. To find this transformation, we recall that the computation of the bilinear form $\mathbf{x}^T A\mathbf{y}$ can be expressed in terms of the Frobenius inner product as $\mathbf{x}^T A\mathbf{y} = \langle A, \mathbf{x}\mathbf{y}^T \rangle = vec(A)^T vec(\mathbf{x}\mathbf{y}^T)$, where $vec(\cdot)$ is is the operator that stacks the columns of a matrix into a vector and $\langle A, B \rangle$ denotes the dot–product $\sum_i \sum_j a_{i,j} b_{i,j}$ between matrix $A = \{a_{i,j}\}$ and matrix $B = \{b_{i,j}\}$. Hence, the expression for the speaker detection log-likelihood can be rewritten as

$$\begin{aligned} \log l = &\langle \Lambda, \phi_1 \phi_2^T + \phi_2 \phi_1^T \rangle + \langle \Gamma, \phi_1 \phi_1^T + \phi_2 \phi_2^T \rangle \\ &+ c^T(\phi_1 + \phi_2) + k \end{aligned} \tag{5.18}$$

If we stack the parameters as

$$w = \begin{bmatrix} vec(\Lambda) \\ vec(\Gamma) \\ c \\ k \end{bmatrix} = \begin{bmatrix} w_\Lambda \\ w_\Gamma \\ w_c \\ w_k \end{bmatrix} \tag{5.19}$$

and we expand the i-vectors pairs as

$$\varphi(\Phi_{1,2}) = \begin{bmatrix} vec(\phi_1 \phi_2^T + \phi_2 \phi_1^T) \\ vec(\phi_1 \phi_1^T + \phi_2 \phi_2^T) \\ \phi_1 + \phi_2 \\ 1 \end{bmatrix} = \begin{bmatrix} \varphi_\Lambda(\phi_1, \phi_2) \\ \varphi_\Gamma(\phi_1, \phi_2) \\ \varphi_c(\phi_1, \phi_2) \\ \varphi_k(\phi_1, \phi_2) \end{bmatrix} \tag{5.20}$$

the scoring (5.16) can be expressed as

$$S(\Phi_{1,2}) = S(\phi_1, \phi_2) = \log l(\phi_1, \phi_2) = w^T \varphi(\Phi_{1,2}) \tag{5.21}$$

that is, the score is computed as the dot–product between the hyperparameters w and the expanded features $\varphi(\Phi)$.

5.2.3 Pairwise SVM as likelihood approximation

We now derive the same formulation without considering the generative 2–covariance model. Suppose that the speaker detection log–likelihood ratio is some analytic function of the i–vectors $\Phi = (\phi_1, \phi_2)$ given by $s = s(\Phi) = s(\phi_1, \phi_2)$, invariant to

i–vector swapping, i.e. $s(\phi_1, \phi_2) = s(\phi_2, \phi_1)$. We can write the Taylor expansion for s around some generic point $\hat{\Phi}$ as

$$s(\Phi) = \sum_{k=0}^{+\infty} \frac{\left(\left(\Phi - \hat{\Phi}\right) \cdot \nabla\right)^k s|_{\hat{\Phi}}}{k!} \tag{5.22}$$

where ∇ is the gradient operator

$$\nabla = \left(\frac{\partial}{\partial \Phi_1}, \ldots, \frac{\partial}{\partial \Phi_d}\right) \tag{5.23}$$

A second order approximation of this function around $\hat{\Phi} = 0$ is

$$s(\Phi) = s(\hat{\Phi}) + (\Phi \cdot \nabla s|_{\hat{\Phi}}) + \Phi^T (H(s)|_{\hat{\Phi}})\Phi \tag{5.24}$$

where H(s) is the Hessian of function s. We can rewrite (5.24) as

$$s(\phi_1, \phi_2) = k + (\phi_1 + \phi_2)^T c + \phi_1^T \Lambda \phi_2^T + \phi_2^T \Lambda \phi_1 + \phi_1^T \Gamma \phi_1 + \phi_2^T \Gamma \phi_2 \tag{5.25}$$

where

$$\begin{aligned} s(\hat{\Phi}) &= k \\ H(s)|_{\hat{\Phi}} &= \begin{bmatrix} \Gamma & \Lambda \\ \Lambda & \Gamma \end{bmatrix} \\ \nabla s|_{\hat{\Phi}} &= c \end{aligned} \tag{5.26}$$

The problem now consists in estimating the coefficient matrices of the Taylor expansion. We note that the choice of $\hat{\Phi} = 0$ is not restrictive, since any other choice would lead to a formally equivalent expression for $s(\Phi)$. We obtained the same formal expression as in (5.16).

This derivation is interesting because it provides a rationale for a straightforward extension of the PSVM approach to Gender Independent training [119]. In particular, since PSVM can be interpreted as an approximation of the log–likelihood ratio function, we can assume that probabilistic modeling of gender labels can be implicitly learned by the classifier by simply using a mixed–gender training set. Some care, however, might be necessary to balance data according to gender.

5.2.4 Polynomial Feature Mapping

Finally, we show that the quadratic form (5.16) corresponds to a slightly modified polynomial kernel of degree 2. The kernel function describing polynomial kernels is

$$K(\mathbf{x}_a, \mathbf{x}_b) = (\mathbf{x}_a^T \mathbf{x}_b + 1)^d \tag{5.27}$$

where d is the kernel degree (in our case $d = 2$). Let $\Phi_{a,b} = [\phi_a\, \phi_b]$ and $\Phi_{w,z} = [\phi_w\, \phi_z]$, with $\phi_a, \phi_b, \phi_w, \phi_z$ i–vectors, i.e. the SVM patterns are i–vectors pairs. The kernel expression (5.27) becomes

$$K(\Phi_{a,b}, \Phi_{w.z}) = K([\phi_a\, \phi_b], [\phi_w\, \phi_z]) = (\phi_a^T \phi_w + \phi_b^T \phi_z + 1)^2 \tag{5.28}$$

that is,

$$\begin{aligned} K(\Phi_{a,b}, \Phi_{w,z}) &= \phi_a^T \phi_w \phi_w^T \phi_a + \phi_b^T \phi_z \phi_z^T \phi_b + 2\phi_a^T \phi_w \phi_z^T \phi_b + 2\phi_a^T \phi_w + 2\phi_b^T \phi_z + 1 \\ &= \langle \phi_a \phi_a^T, \phi_w \phi_w^T \rangle + \langle \phi_b \phi_b^T, \phi_z \phi_z^T \rangle \\ &\quad + 2\langle \phi_a \phi_b, \phi_w \phi_z \rangle + 2\phi_a^T \phi_w + 2\phi_b^T \phi_z + 1 \\ &= \langle [\phi_a \phi_b 1][\phi_a \phi_b 1]^T, [\phi_w \phi_z 1][\phi_w \phi_z 1]^T \rangle \\ &= \tilde{\varphi}(\phi_a \phi_b)^T \tilde{\varphi}(\phi_w \phi_z) \end{aligned} \tag{5.29}$$

where $\tilde{\varphi}(\Phi_{1,2})$ is the feature mapping

$$\tilde{\varphi}(\Phi_{1,2}) = vec([\phi_1 \phi_1 1][\phi_1 \phi_2 1]^T) \sim \begin{bmatrix} vec(\phi_1 \phi_2^T) \\ vec(\phi_2 \phi_1^T) \\ vec(\phi_1 \phi_1^T) \\ vec(\phi_2 \phi_2^T) \\ \phi_1 \\ \phi_1 \\ \phi_2 \\ \phi_2 \\ 1 \end{bmatrix} \tag{5.30}$$

Symbol $\sim$ is used to denote equivalence of vectors up to reordering of entries. Finally consider the log–likelihood (5.16) and transform the (unknown) parameter c as $\tilde{c} = c/2$, so that the linear term of the log–likelihood becomes $2\tilde{c}^T(\phi_1 + \phi_2)$. Then the feature expansion (5.20) becomes

$$\varphi(\Phi_{1,2}) = \begin{bmatrix} vec(\phi_1 \phi_2^T + \phi_2 \phi_1^T) \\ vec(\phi_1 \phi_1^T + \phi_2 \phi_2^T) \\ 2(\phi_1 + \phi_2) \\ 1 \end{bmatrix} \tag{5.31}$$

In this case it is easy to verify that

$$\varphi(\Phi_{a,b})^T \varphi(\Phi_{w,z}) = \tilde{\varphi}(\Phi_{a,b})^T \tilde{\varphi}(\Phi_{w,z}) \tag{5.32}$$

Therefore, apart from a scaling of the linear term, the PSVM kernel corresponds to a polynomial kernel of degree 2.

5.2.5 Fast scoring

Since the number of i-vector pairs is of the order of hundred of millions in our experiments, the evaluation of a kernel matrix would be clearly unfeasible. However, if we use a primal SVM solver, we need only to evaluate the SVM loss function and its gradient with respect to the hyperplane. Both evaluations require, in principle, a sum over all the i-vectors pairs, but in the next two subsections we show that for feature mapping (5.20) the loss function and the gradients can be computed without an explicit expansion of all the i-vectors pairs [73, 74]. This allows an efficient training technique based on primal solvers as, for example, BMRM (Section 4.3.2).

Loss function evaluation

Let D denote the matrix of all stacked training i-vectors ϕ_i

$$D = [\phi_1 \phi_2 \dots \phi_n] \tag{5.33}$$

Also, let $\Theta \in \{\Lambda, \Gamma, c, k\}$ denote a component of the hyperplane, and S_Θ be the score matrix of training patterns due to component Θ, i.e. $S_{\Theta i,j} = S_\Theta(\phi_i, \phi_j)$. From (5.16) the score matrices can be evaluated as

$$S_\Lambda(\phi_1, \phi_2) = \phi_1^T \Lambda \phi_2 + \phi_2^T \Lambda \phi_1 = 2 D^T \Lambda D \tag{5.34}$$

$$S_\Gamma(\phi_1, \phi_2) = \phi_1^T \Gamma \phi_1 + \phi_2^T \Gamma \phi_2 = \tilde{S}_\Gamma + \tilde{S}_\Gamma{}^T \tag{5.35}$$

$$S_c(\phi_1, \phi_2) = c^T D \tag{5.36}$$

$$S_k(\phi_1, \phi_2) = k \cdot \mathbf{1} \tag{5.37}$$

where

$$\tilde{S}_\Gamma = [\underbrace{d_\Gamma \dots d_\Gamma}_{n}] \tag{5.38}$$

$$d_\Gamma = \text{diag}\ (D^T \Gamma D) \tag{5.39}$$

diag is the operator that returns the diagonal of a matrix as a column vector, and $\mathbf{1}$ is an $n \times n$ matrix of ones. Let S be the sum of these partial score matrices. The SVM loss function can then be expressed as

$$\begin{aligned} L(D, Z) &= C \sum_{i,j} \max \left[0, 1 - \zeta_{i,j} w^T \varphi(\phi_i, \phi_j)\right] \\ &= C \langle \mathbf{1}, \max(\mathbf{0}, \mathbf{1} - (Z \circ S) \rangle \end{aligned} \tag{5.40}$$

where $\mathbf{0}$ is an $n \times n$ matrix of all zeros, Z is the $n \times n$ matrix of labels for trial (ϕ_i, ϕ_j), $Z_{i,j} = \zeta_{i,j} \in \{-1, +1\}$, and $\circ$ is the element-wise matrix multiplication operator.

Gradient Evaluation

The gradient of the loss function can be evaluated from its derivative with respect to the m-th dimension of w as

$$\begin{aligned}\frac{\partial L}{\partial w_m} &= \sum_{i,j} \frac{\partial l_{L1}(w,(\phi_i,\phi_j),\zeta_{i,j})}{\partial (w^T\varphi(\phi_j,\phi_j))} \frac{\partial w^T\varphi(\phi_j,\phi_j)}{\partial w_m} \\ &= \sum_{i,j} g_{i,j}\frac{\partial S_{i,j}}{\partial w_m} \\ &= \sum_{i,j} g_{i,j}\varphi(\phi_i,\phi_j)_m\end{aligned} \tag{5.41}$$

where $g_{i,j}$ is the derivative of the loss function with respect to the dot product

$$g_{i,j} = \begin{cases} 0 & \text{if } S_{i,j}\zeta_{i,j} \geq 1 \\ -\zeta_{i,j} & \text{otherwise} \end{cases} \tag{5.42}$$

Let G be the matrix $G_{i,j} = g_{i,j}$, then

$$\begin{aligned}\nabla L = \begin{bmatrix} \nabla_\Lambda L \\ \nabla_\Gamma L \\ \nabla_c L \\ \nabla_k L \end{bmatrix} &= \begin{bmatrix} vec\left(\sum_{i,j} g_{i,j}\left(\phi_i\phi_j^T + \phi_j\phi_i^T\right)\right) \\ vec\left(\sum_{i,j} g_{i,j}\left(\phi_i\phi_i^T + \phi_j\phi_j^T\right)\right) \\ \sum_{i,j} g_{i,j}\left(\phi_i + \phi_j\right) \\ \sum_{i,j} g_{i,j} \end{bmatrix} \\ &= \begin{bmatrix} 2\cdot vec\left(DGD^T\right) \\ 2\cdot vec\left(\left[D\circ(\mathbf{1_A}G)\right]\right)D^T \\ 2\cdot\left[D\circ(\mathbf{1_A}G)\right]\mathbf{1_B} \\ \mathbf{1_B}^T G\mathbf{1_B} \end{bmatrix}\end{aligned} \tag{5.43}$$

where $\mathbf{1_A}$ is a $n \times M$ matrix (M is the i–vector dimensionality) of ones and $\mathbf{1_B}$ is a size n column vector of ones.
Again, no explicit expansion of i-vectors is necessary for this evaluation.

Computing the regularization coefficient

Finally, we consider the problem of estimating the regularizer coefficient (Section 4.1.1). In order to estimate the norm of the expanded features we observe that, from (5.28), the norm of the expanded features $\varphi(\phi_1,\phi_2)$ for i–vector pair $\Phi = (\phi_1,\phi_2)$ can be computed as

$$\|\varphi(\phi_1,\phi_2)\| = K(\Phi_{1,2},\Phi_{1,2})^{\frac{1}{2}} = \phi_1^T\phi_1 + \phi_2^T\phi_2 + 1 \tag{5.44}$$

The regularization parameter C can then be computed as

$$\begin{aligned} C &= \left(\frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} \|\varphi(\phi_i, \phi_j)\| \right)^{-2} \\ &= \left(\frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} \left(\phi_i^T \phi_i + \phi_j^T \phi_j + 1 \right) \right)^{-2} \\ &= \left(1 + \frac{2}{n} \sum_{i=1}^{n} \|\phi_i\|^2 \right)^{-2} \end{aligned} \tag{5.45}$$

Chapter 6

Experimental Results

This chapter presents some experimental results for the different contributions of the authors to language and speaker verification presented in this work. Most of the speaker and language recognition techniques are evaluated according to the protocols defined by the American National Institute of Standards and Technology (NIST) for their Language and Speaker Recognition Evaluations [90, 91, 120].

6.1 GPU–based ANN training

The first set of results we present are related to the speed–up of ANN training for speech decoding. The reference architecture is detailed in Section 2.5.4. The experiments were run on an HP xw8600 workstation equipped with a quad-core 3.0 GHz CPU, 1600 MHz FSB, 8 GB RAM, NVIDIA GTX280 GPU, and running Linux RedHat RHEL 5.2 EM64T. The GPU–accelerated computations are compared to a single–thread MKL [121] implementation and to a multi–thread MKL implementation [33, 32]. The first set of experiments we report were done to asses the capabilities of the CUDA framework for medium–scale matrix multiplications. In particular, we measured the time required by the different techniques to perform matrix multiplications for square matrices of different sizes. The results of these experiments are shown in Figure 6.1 where the execution times are plotted in logarithmic scale. As expected, using a multi-threaded architecture increases the performance of the MKL implementation. CuBLAS matrix multiplication routines, on the other hand, outperform the MKL routines for matrices with medium to large sizes, however, for smaller matrices, the overhead due to memory transfers cannot be neglected (in Figure 6.1 we report times both with and without taking into account memory transfers). This suggests to use large bunch sizes for training the net to reduce the impact of memory transfers We can also observe that efficiency peaks are obtained, with the CUDA framework, in correspondence of matrix sizes which

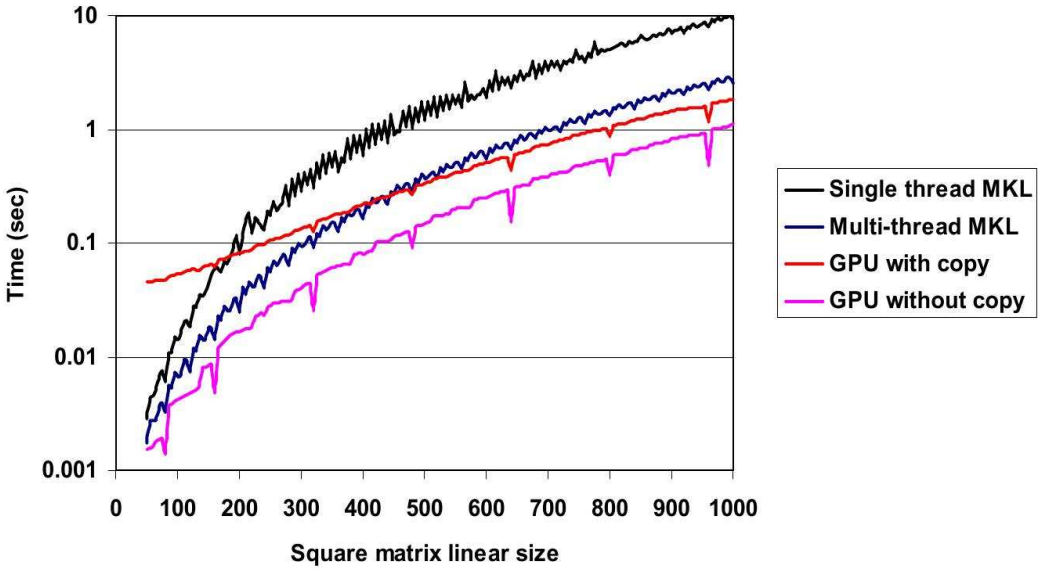


Figure 6.1: Time required for matrix–matrix multiplications for squared matrices of different sizes

are multiple of powers of two. This motivates the use of padding.

Next, we analyze the performance in terms of training time for the different implementations. The task consists in training an English model using the Wall Street Journal corpus [122], a relatively small database of 7236 files, which represents a popular case study in Automatic Speech Recognition. This corpus has been used as a validation set for the algorithms and for tuning the training procedure.

Table 6.1 shows the elapsed training time, and the relative speed–up of the three analyzed implementations together with a baseline standard C implementation. In order to obtain a fair comparison between MKL and GPU implementations the forward bunch size was set to 15 for MKL in order to reduce the significant overhead of re–processing patterns accepted by FABP but exceeding the BP bunch size. The GPU bunch, on the other hand, is set to 32 in order to reduce the memory transfer overhead. The MKL implementation achieves a speed–up of 3.7 times with respect to the naïve C implementation. On the other hand, the GPU accelerated training is about 19 times faster than the baseline and more than 5 times faster than the single–thread MKL implementation.

Table 6.2 shows the results of another set of experiments. This time different language models were trained with the same setup using large corpora collected for creating the models used by the Loquendo decoder. Since different bunch sizes can result in slightly different models (due to the presence of FABP), these experiments

Table 6.1: Training time for different ANN training algorithms

Training implementation	Elapsed time (hh:mm)	Speed–up w.r.t. standard C	Speed–up w.r.t. MKL
Standard C	42:35	-	-
Single–thread MKL	11:36	3.7×	-
Multi–thread MKL	9:41	4.4×	1.2×
GPU and CUDA w/o padding	2:29	17.1×	4.7×
GPU and CUDA with padding	2:14	19.1×	5.2×

were also used to asses the quality of the GPU–based decoders. Errors are measured in terms of Word Accuracy (WA) and compared to the standard Loquendo ASR models. The results show that GPU–based training allows to achieve the same accuracy as the MKL implementation and both achieve the same results as the baseline system. Using GPUs for training the ANN models allows to achieve an average of 6 times speed–up with respect to the single–thread MKL implementation.

Table 6.2: Speed–up and recognition accuracy for different algorithms on different datasets

Languages	Loquendo ASR	Single–thread MKL		GPU–CUDA		
	WA (%)	WA (%)	Δ WA	WA (%)	Δ WA	Speed–up
Italian	93.1	93.0	+0.1	93.0	+0.1	4.9×
Spanish	93.2	93.3	-0.1	93.2	0.0	5.4×
French	90.2	90.2	0.0	90.0	+0.2	6.5×
German	89.4	89.7	-0.3	90.0	-0.6	7.0×
English	84.6	84.5	+0.1	84.1	84.2	6.0×
Brazilian	84.1	84.7	-0.6	84.2	-0.1	6.7×
Average	89.1	89.2	-0.1	89.1	0.0	6.1×

6.2 Language Identification

In this section we present some results for FA–based techniques applied to language identification. In particular, we will analyze the language factor approach (Section 4.2.2) and the use of phonotactic i–vectors combined with discriminative classifiers (Section 4.2.4).

6.2.1 Language Factors

The first set of experiments we present are related to the language factors model (Section 4.2.2). The results are reported for the NIST 2007 Language Recognition Evaluation (LRE) [90] which comprises 14 languages, with approximately 6500 test utterances uniformly distributed for durations of 3, 10, and 30 seconds. The reported evaluation measures are the NIST–defined minimum Decision Cost Function score (minDCF) and the percent Equal Error Rate (% EER) uniformly weighted over the languages [90]. Raw scores are transformed by means of a SVM–based back–end. In particular, a small SVM has been trained over the raw scores using a held–out development set, taking care to balance the cost factor of the different classes [58].

In these experiments we asses the performance of language factors for both GMM–SVM and pushed GMM approaches (Section 4.2.1). In order to estimate language factors a gender independent 2048 components gender independent UBM was trained using the Callfriend corpus [123]. Acoustic features are Cepstral plus Shifted Delta coefficients (Section 2.1.3). For each utterance a MAP adapted GMM is estimated using a small relevance factor. Language and speaker compensated eigenvoice subspaces are estimated using all data from the 23 languages in the Callfriend Corpus [123], half of the NIST LRE07 development corpus, half of the OSHU corpus provided by NIST for LRE05 [124] and the Russian through switched telephone network corpus [123], resulting in a total of approximately 14000 segments. The backend was trained using the second half of the NIST LRE07 development corpus, the second half of the OSHU corpus and the development and test sets provided by NIST for LRE03. The backend data amounts to approximately 6000 segments for each of the 30s, 10s and 3s duration conditions defined by NIST.

The first set of experiments was done to asses the performance of a linear SVM classifier with respect to the type of subspace modeling used and the dimension of the subspaces. Figure 6.2 shows the minimum DCF (minDCF) for the two different subspace models for the 30s condition. The first thing to notice is that language factors perform always better than speaker compensated eigenvoice models. Moreover, the former models are more stable with respect to the subspace dimensionality: with 100 factors the performance are just 10% worse than using 600–dimensional factors. Therefore, we restrict out analysis only to the language factor subspace model.

A second set of experiments was conducted to compare the language factor performance with a classical GMM–SVM system. Table 6.3 summarizes the results. We can observe that the language factor approach outperforms the classical GMM–SVM approach for both short and long duration conditions. Moreover, a linear SVM proves to achieve the same performance as the more theoretically sound KL kernel. The second part of Table 6.3 shows the performance of the language factors in combination with the pushed–GMM approach. While the pushed–GMM approach performs better than the classical GMM–SVM approach when using MAP–adapted

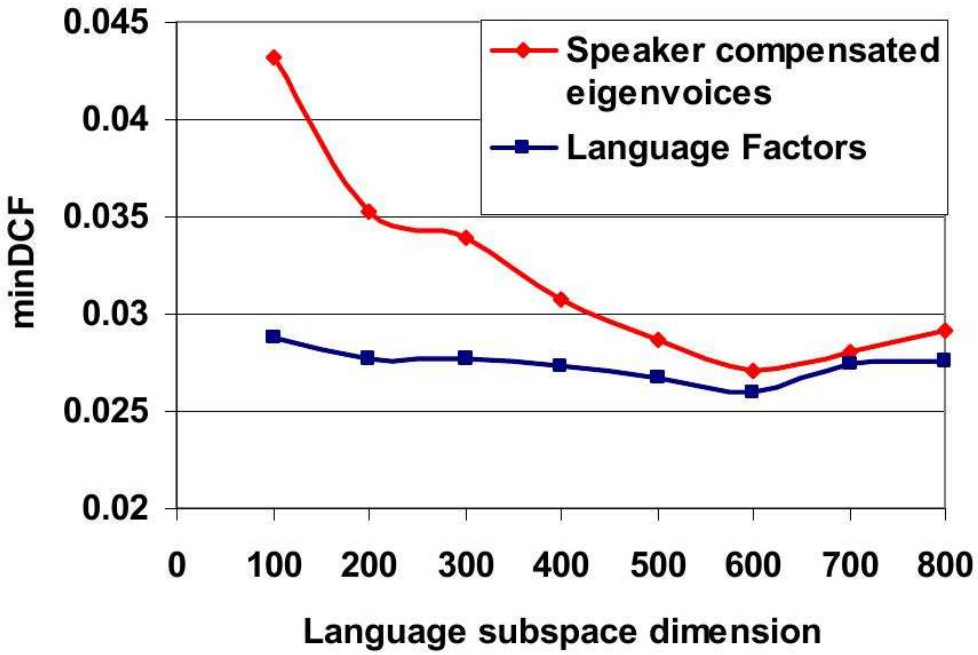


Figure 6.2: MinDCF as a function of the number of language factors for two different language subspaces

supervectors, the combination of language factors and GMM pushing does not improve the performance of the language factors models with respect to standard SVM modeling. However, in both cases language factors allow for much faster SVM model training due to the much smaller dimensionality of the features.

6.2.2 Phonotactic i–vectors

We now focus on the phonotactic i–vector approach. In this section we compare different classifiers based on SVM and Logistic Regression, both in a one–vs–all framework and in a multiclass framework (Section 4.2.4). In particular, for the multiclass problem we also analyze the performance of binary classification followed by score back–projection as described in Section 4.1.5. All models are trained using the default SVM regularizer (Section 4.1.1).

The classifier features are 600–dimensional i–vectors extracted from tri–gram counts using the BUT Hungarian phone recognizer [89]. Our baseline system is a binary one–vs–all LR classifier [89]. A preprocessing step is carried out which centers the i–vectors and perform Within–Class Covariance Normalization. After the classification step all scores are calibrated by means of a multiclass Logistic

Table 6.3: Min DCF and (%EER) for the core closed set tests in LRE07

Model	30s	10s	3s
GMM–SVM (KL kernel SVM)	0.029 (3.43)	0.085 (9.12)	0.201 (21.3)
GMM–SVM (Linear SVM)	0.031 (3.71)	0.087 (9.51)	0.200 (21.0)
LF–SVM (KL kernel SVM)	0.026 (3.13)	0.083 (9.02)	0.186 (20.4)
LF–SVM (Linear SVM)	0.026 (3.11)	0.083 (9.13)	0.187 (20.4)
Discriminative GMMs	0.021 (2.56)	0.069 (7.49)	0.174 (18.5)
Discriminative LF (KL kernel SVM)	0.025 (2.97)	0.084 (9.04)	0.186 (19.9)
Discriminative LF (Linear SVM)	0.025 (3.05)	0.084 (9.05)	0.186 (20.0)

Regression backend [125].

Results are given for the 2009 NIST Language Recognition Evaluation [91] in terms of C_{avg} as defined by NIST.

Table 6.4 shows the results in terms of C_{avg} [91] using the classifiers described in Section 4.2.4. LR and SVM entries refer to the one-versus-all systems, MC-LR and MC-SVM denote to the multiclass systems and SP-LR and SP-SVM refer to the binary pairwise systems with Score Projection. Following the success of LDA

Table 6.4: $C_{avg} \times 100$ for different systems on NIST LRE09 Evaluation task over 30s, 10s and 3s conditions.

System	Scores type	30s	10s	3s
LR	one-to-all	2.98	8.25	21.37
SVM	one-to-all	3.07	8.55	21.68
MC-LR	multiclass	3.16	8.66	21.82
MC-SVM	multiclass	3.89	10.60	23.92
MC-SP-LR	projection	2.93	8.14	21.32
MC-SP-SVM	projection	3.91	9.86	23.03

and length normalization for acoustic i–vectors in speaker verification problems, we also analyzed different preprocessing techniques. In particular, it was found out

that Heteroscedastic Discriminant Analysis [126] followed by WCCN and length normalization of the i–vectors allow effectively improving the performance for all the systems. Table 6.5 shows the results of the same classifiers used before with this particular preprocessing. We can observe that the different classifiers, trained in this 22 dimensional subspace, achieve almost the same performance, with multiclass systems giving slightly better results than binary systems. I–vector postprocessing proves to be valuable to improve the system performance.

Table 6.5: $C_{avg} \times 100$ for different systems on NIST LRE09 Evaluation task over 30s, 10s and 3s conditions after HDA, WCCN and length normalization.

System	Scores type	30s	10s	3s
LR	one-to-all	2.83	8.09	21.34
SVM	one-to-all	2.83	8.09	21.34
MC-LR	multiclass	2.79	8.06	21.35
MC-SVM	multiclass	2.81	8.05	21.33
SP-LR	projection	2.80	8.09	21.31
SP-SVM	projection	2.82	8.04	21.33

6.3 Large–scale linear SVM training

In Section 4.3 we have described different SVM training algorithms for large–scale linear classification problems. In this section we analyze the behavior of these different algorithms for a set of language and speaker verification tasks. In particular, we compare the algorithms in terms of time required to reach convergence, scalability towards large datasets and recognition performance. The tasks we take into consideration are the phonotactic GMM–SVM and the pushed–GMM approaches for language identification (Section 4.2.1) and the pairwise SVM model for speaker identification (Section 5.2).

We begin with an analysis of the implementation details of each algorithm.

6.3.1 SVM algorithms implementation

DCDM, BMRM and Pegasos algorithms have been implemented from scratch using a Python/C framework, where the modules requiring expensive computations are either written in C language or are evaluated by means of fast and parallelized NumPy/BLAS functions [127]. In order to reduce the mixed Python/C framework overhead, we adopted a bunch mode training approach, where bunches of patterns are loaded into main memory and processed together. In particular, Python was

used for preparing and loading the patterns, while the computation intensive tasks are performed by fast external libraries (NumPy/BLAS) and C code. Large bunches minimize the communication overhead between Python and the library routines.

6.3.2 Algorithms for language recognition

The language identification systems we presented require training of a single one–versus–all SVM model for each language. However, all the models share the same data, the difference being only in the label associated to each pattern. For this reason, all SVMs algorithms used for language identification, with the exception of $\text{SVM}^{\text{Light}}$, have been implemented as to jointly train all the language models in parallel in order to minimize disk accesses. $\text{SVM}^{\text{Light}}$ has been modified to effectively compute by means of multi threaded NumPy/BLAS libraries the kernel matrix, and to cache it in memory. Thus, all the language models share the same kernel matrix computed just once. This approach has proven to be faster than caching kernel computations on the fly.

Dual Coordinate Descent Method

The Dual Coordinate Descent Method has been implemented as described in Section 4.3.1. Shrinking [84] has been used to speed–up the training [109]. A multi–threaded implementation of DCDM (referred to as MTDCDM) was also developed. Since DCDM is intrinsically a sequential algorithm, the parallelism was obtained by allocating a thread to each of the different language models. The obtained speed–up, however, is not very significative.

Bundle Methods for Regularized Risk Minimization

The BMRM algorithm has been implemented as described in Section 4.3.2. The quadratic problem (4.104) is solved by means of the CVXOPT Python solver [128]. The shrinking technique was not adopted in the BMRM algorithm, which solves the problem in its primal formulation, thus each iteration of the algorithm performs a full scan of the training dataset. The $\boldsymbol{\alpha}$'s needed for the pushed–GMM approach have been evaluated as outlined in Section 4.3.2.
Since, at each iteration, the gradient computations can be performed in parallel, a distributed version of BMRM has been implemented (referred to as MPBMRM in the following) where the dataset is split among different processes, and each process exploits the multi–thread capabilities of the NumPy/BLAS functions.

Pegasos

Our implementation of Pegasos (Section 4.3.2) slightly modifies the stochastic gradient descent step by introducing the concept of epoch. Each epoch denotes a block of training iterations which are performed on a randomly selected subset of the full training dataset. In particular, at the beginning of each epoch a block of randomly selected training patterns is loaded in memory, and the hyperplane update is performed using only the patterns belonging to this subset. Pegasos provides the primal solution to the SVM problem. However, a set of $\boldsymbol{\alpha}$'s for the pushed–GMM approach was computed as described in Section 4.3.2. Note, however, that stopping the algorithm before convergence does not guarantee that the obtained values correspond to a dual feasible solution. Due to its sequential update rule we did not implement a multi–threaded version of Pegasos.

SVM$^{\mathbf{Perf}}$

Finally, SVM$^{\text{Perf}}$ (Section 4.3.1) was modified to fit the data format of the other implementations and to read the training patterns on demand rather than loading all the dataset in main memory. A multi–threaded implementation (referred to as MTSVM$^{\text{Perf}}$ in the following) was obtained by modifying the original code with the OpenMP [129] instructions that allow core computations to be parallelized.

6.3.3 Algorithms for speaker recognition

The pairwise SVM problem (Section 5.2) requires some additional considerations. First of all, patterns are pairs of i–vectors, thus the number of the training trials grows as $O(n^2)$, where n is the number of i–vectors in the training set. Moreover, if d denotes the i–vector dimensionality, the feature mapping described in Section 5.2 produces mapped features having $O(d^2)$ components, thus the global dataset size is $O(n^2d^2)$. Caching the complete kernel matrix is impractical even for relatively small sized datasets because it would require $O(n^4)$ memory. SVM training of the i–vector pairs by means of SVM$^{\text{Light}}$ is not viable because, as shown in Section 4.3.1, its time complexity is $O(n^4d)$. In DCDM the hyperplane is updated for each pattern, thus it requires either the complete dataset of mapped features ($O(n^2d^2)$ memory) or online feature mapping ($O(n^2d^2)$ operations for each iteration). Since in our experiments d equals 400 and n is approximately 20000, these dual algorithms cannot be directly used to train the models for our discriminative approach.

Training is feasible, instead, by using primal solvers such as BMRM and Pegasos because it is possible to efficiently evaluate the loss function and its gradient with respect to $\mathbf{w}$ over appropriate subsets of trials (Section 5.2.5.

The BMRM approach allows to easily plug–in the formulas given in Section 5.2.5. Pegasos, on the other hand, requires some more considerations. The expressions given in Section 5.2.5 are effective only when blocks of i–vectors are scored together considering all cross pairs. In fact, evaluation of the scores of all pairs would otherwise require $O(n^2d^2)$ time. On the other hand, in stochastic gradient descent usually a small number of patterns are scored before a hyperplane update. For this reason, Pegasos has been slightly modified: instead of randomly picking a bunch of trials among all possible trials, at each iteration all the i–vector trials for a set of random speakers are selected, and the hyperplane is adapted according to the approximation of the gradient evaluated on this set of trials only. This reduces the risk of selecting "different speaker pair" trials only.

6.3.4 Language Recognition task results

The NIST 2009 LRE [91] core condition task comprises 23 languages and three duration conditions, corresponding to 30s, 10s and 3s nominal utterance duration. Test data comprises both Conversational Telephone Speech and telephone bandwidth broadcast radio speech. The results reported in this section cannot be compared to those of state–of–the–art systems, which usually are the combination of different subsystems. Since the goal here is to compare the relative performance of different SVM algorithms, a single channel–independent system is considered.

Our training set for the acoustic system consists of 17521 utterances taken from the Callfriend corpus, the corpora provided by NIST for the 2003, 2005 and 2007 Language Recognition Evaluations [130, 124, 90], the Russian through switched telephone network [123], the OGI corpus [123] and the Voice of America corpora [91]. References for these data and the details on the selection process for training patterns are given in [131].

2048–Gaussian mixtures are used for the UBM and language GMMs, while features are 7 Mel frequency cepstral coefficients and their 7–1–3–7 Shifted Delta (SDC) coefficients, 56 acoustic features in total, compensated for nuisances in the feature domain as in [100].

The phonetic system is trained using the same dataset, though 2005 LRE utterances were split into chunks of approximately 30s. This results in 20543 training utterances.

The first issue in training SVM classifiers is class balancing. It can be faced by appropriately filtering the dataset, or by replicating the patterns of the less populated classes, or even better, by giving different weights to the loss function of patterns belonging to different classes. The first method is not attractive because it reduces the amount of training patterns. The second approach increases the secondary memory accesses, and makes difficult jointly training the different language models because language dependent datasets have to be generated from the full

dataset. In our experiments, all the SVMs have been trained with the third method of class balancing, except for SVM$^{\text{Perf}}$, which does not provide a simple and direct way to apply this technique.

Concerning the C parameter, the default one described in Section 4.1.1 was used, which proves to produce good models and reasonable results in a large variety of experiments.

System performance is presented in terms of Equal Error Rate (EER) and C_{avg} as defined by NIST [91]. Scores are normalized by means of a Gaussian back–end trained on a held–out set [132]. The performance is given as a function of time by testing models obtained after a variable number of iterations. Timings were evaluated on a HP DS160G5 server equipped with two Xeon X5472 3 GHz quad-core processors, 32 GB of DDR2-800 RAM and a SATA 7200 RPM hard disk. All results are given in terms of wall clock time.

Phonetic system

The first set of experiments we present compares the performance of the SVM training algorithms for the phonetic GMM–SVM system (Section 4.2.1). We take as accuracy reference the models produced by SVM$^{\text{Light}}$ after convergence has been reached. The time required to obtain these models is used as reference time for this algorithm (we are not interested in measuring the time for less accurate SVM$^{\text{Light}}$ models since this algorithm would not scale linearly with the dataset size anyway).

As far as SVM$^{\text{Perf}}$ is concerned, again we consider only the final training time since this technique is less attractive for our applications due to the difficulties in class balancing, which lead to worse recognition results.

Table 6.6 compares the asymptotic accuracy (i.e. the accuracy obtained by models after a large number of iterations) of the different phonetic models in the 30, 10, and 3 seconds conditions for the NIST LRE09 tests. Both DCDM and BMRM models converge, in terms of C_{avg} and EER, to the results provided by the baseline SVM$^{\text{Light}}$–based model. Models trained with Pegasos give slightly worse performance, and SVM$^{\text{Perf}}$ does not produce models as good as the other approaches due to the lack of class balancing.

The convergence properties of the different techniques can be appreciated looking at Table 6.7, which reports the time required by each algorithm to estimate a model that reaches an accuracy within 1.0% of the C_{avg} value obtained using a model trained by SVM$^{\text{Light}}$, and in Fig. 6.3, which shows, in logarithm scale and for the 30s condition, the tradeoff between training time and C_{avg}. In Table 6.7 the conditions which did not reach convergence are indicated by a "–" sign. Similar trends have been observed for EER and for the different duration conditions.

Restricting our analysis to single–thread implementations, we can observe that DCDM is the fastest algorithm to reach convergence. Moreover, the estimated model

Table 6.6: Phonetic system: asymptotic values C_{avg} and EER

Algorithm	30s	10s	3s
SVM$^{\text{Light}}$	0.0375 3.972%	0.0858 8.881%	0.2037 20.772%
BMRM	0.0375 3.941%	0.0860 8.981%	0.2032 20.842%
DCDM	0.0376 3.965%	0.0861 8.948%	0.2031 20.785%
SVM$^{\text{Perf}}$	0.0434 4.583%	0.0944 9.782%	0.2061 21.057%
Pegasos	0.0392 4.198%	0.0879 9.116%	0.2032 20.979%

Table 6.7: Phonetic system: time required to achieve 1% SVM$^{\text{Light}}$ C_{avg} accuracy ("-" means not reached)

Algorithm	30s	10s	3s
SVM$^{\text{Light}}$	6991s	6991s	6991s
BMRM	6672s	5563s	1598s
MPBMRM	1493s	1148s	210s
DCDM	184s	143s	96s
MTDCDM	85s	66s	46
SVM$^{\text{Perf}}$	-	-	-
MTSVM$^{\text{Perf}}$	-	-	-
Pegasos	-	-	3364s

behaves even better than the asymptotic one, probably because of less overfitting of the training data. The other solvers are much slower, and BMRM is faster and slightly better than Pegasos.

The parallel implementation of DCDM and BMRM takes into account the different characteristics of the two approaches. For MTDCDM we use 23 threads (one for each language model), while in MPBMRM parallelism is exploited at computation level by distributing the load among 4 processes, each one using 8 threads for NumPy/BLAS operations. As expected, the DCDM algorithm is not able to gain much from the increased number of cores, whereas BMRM benefits from multiple

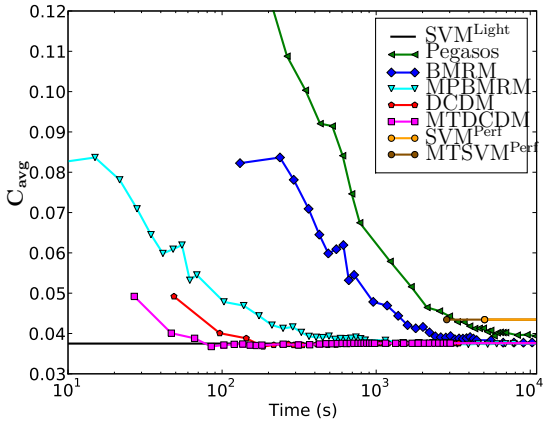


Figure 6.3: Phonetic system: C_{avg} as a function of the training time for the 30s condition

processes because training is done in batch mode. It takes also advantage of multi–threading for the NumPy/BLAS operations. While for this dataset single–thread DCDM has faster convergence rate than multi–process BMRM, massive parallelization of BMRM could easily outperform DCDM for larger datasets.

Pushed–GMM system

Similar to what was done for the phonetic system we measured the performance of pushed–GMM systems with respect to training time. However, due to the much larger time required for testing these kind of models, fewer samples were taken than for the phonotactic system. SVM^{Perf} cannot be used in this context since it does not provide the dual solution required to build the pushed–GMM model and anti–model.

Table 6.8 shows the results in terms of accuracy (C_{avg} and EER) and time required to reach 1% of reference (SVM^{Light}) accuracy. Note that the model used for testing is the same in all conditions (although for the different conditions the same algorithm might require a different number of iterations to provide accurate enough models). Also, training times are sampled in correspondence to algorithm iterations, thus the measured times are the same when the target C_{avg} is reached at the end of the same iteration.

The first thing we observe is that all models asymptotically achieve SVM^{Light} accuracy. DCDM is again the fastest algorithm to reach convergence, followed by the BMRM algorithm. Pegasos is much slower but its models give slightly better results, probably due to the fact that its algorithm for the evaluation of the Lagrange multipliers does not impose constraints to the $\boldsymbol{\alpha}$ values.

It is worth noting, however, than even the first BMRM model provides quite

Table 6.8: Pushed–GMM: asymptotic values for C_{avg} and EER, and time required to achieve 1% SVM^{Light} C_{avg} accuracy

Algorithm	30s	10s	3s
SVM^{Light}	0.0276 3.120% 2871s	0.0621 6.592% 2871s	0.1616 16.594% 2871s
BMRM	0.0276 3.153% 1194s	0.0626 6.587% 723s	0.1618 16.579% 311s
DCDM	0.0276 3.157% 655s	0.0626 6.596% 655s	0.1619 16.592% 121s
Pegasos	0.0272 3.112% 8433s	0.0617 6.543% 1800s	0.1605 16.623% 938s

good results. This is interesting since the Lagrange multiplier corresponding to this model are all equal, which means that model and anti–model are computed simply as the average of target and non–target GMMs[107].

6.4 Pairwise SVM

In this section we provide some experimental results concerning the pairwise SVM approach (Section 5.2). In particular, we compare the recognition performance of this model with respect to state–of–the–art PLDA models and we analyze how different SVM training algorithms behave in this particular context.

6.4.1 Pairwise SVM and PLDA

The first set of results we present are preliminary results comparing classical Gender Dependent (GD) GPLDA (i.e. without any preprocessing of i–vectors) and Heavy–Tailed PLDA models with a GD pairwise SVM system on the NIST 2010 SRE [120] tel–tel condition (condition 5) for both the male and female subsets. All the three systems are trained using the NIST SRE 2004, 2005 and 2006 train and test corpora [133, 134, 135], the Switchboard II Phase 2 and 3 corpora and the Switchboard Cellular part 1 and 2 corpora [123]. In these set of experiments the pairwise SVM regularization parameter was estimated using an approximated expression. The use of a worse regularizer also forced us to adopt some kind of class balancing, which, on the other hand, proves to be not necessary anymore when the default

regularizer is correctly computed. 400–dimensional i–vectors were extracted from a 60–dimensional features, full–covariance, 2048 Gaussians UBM [136]. While no normalization was used for PLDA, the presence of the regularizer term in SVM training requires some kind of normalization of the SVM features. For this reason, i–vectors were centered and whitened by means of WCCN before training the PSVM. PLDA systems were trained using 200 speaker factors and full rank channel factors.

The results are shown in Table 6.9 in terms of Equal Error Rate (EER), 2008 SRE DCF (oldDCF) and 2010 SRE minimum and actual Detection Cost Function (minDCF and actDCF respectively). All scores have been calibrated on using NIST SRE2008 data [137].

Table 6.9: EER, DCF as defined in SRE 2008 (oldDCF), minimum DCF (minDCF) and actual DCF (actDCF) as defined in SRE 2010 for the extended tel–tel core condition (condition 5) of NIST SRE10

Male Set

System	EER	oldDCF	minDCF	actDCF
GPLDA	3.82%	0.165	0.401	0.442
HTPLDA	1.55%	0.082	0.313	0.364
PSVM	1.50%	0.074	0.308	0.355

Female Set

System	EER	oldDCF	minDCF	actDCF
GPLDA	4.08%	0.179	0.448	0.531
HTPLDA	2.29%	0.118	0.412	0.415
PSVM	2.35%	0.108	0.394	0.398

All

System	EER	oldDCF	minDCF	actDCF
GPLDA	4.21%	0.183	0.470	0.498
HTPLDA	1.98%	0.102	0.379	0.393
PSVM	1.94%	0.095	0.373	0.378

We can observe that the pairwise SVM outperforms Gaussian PLDA and performs even slightly better than Heavy–Tailed PLDA. Moreover, the PSVM approach allows achieving testing time similar to that of GPLDA and much faster than with HTPLDA.

The second sets of experiments we present were performed to compare an improved PSVM system (i.e. with an exact evaluation of the regularization parameter

and without class balancing) with respect to the GPLDA models created using length–normalized i–vectors. In both cases WCCN was applied to whiten the i–vectors. The results shown in Table 6.10 show the performance of the improved GPLDA system trained with 120 speaker factors and the improved PSVM system.

Table 6.10: DCF and EER for improved GPLDA and PSVM systems

System	EER (%)	oldDCF	minDCF
	Female		
GPLDA	2.10	0.101	0.355
PSVM	2.21	0.102	0.342
	Male		
GPLDA	1.24	0.074	0.284
PSVM	1.96	0.082	0.262

In these conditions, the behavior of the two systems is similar (and in both cases much better than classical GPLDA). GPLDA performs slightly better than PSVM, however the choice of the number of speaker factors poses some problems and can influence the performance, as shown in Figure 6.4. The PSVM, on the other hand, does not need any parameter tuning.

6.4.2 Gender Independent PSVM

Another set of experiments was conducted to asses the performance of the PSVM approach in a Gender Independent (GI) framework [119]. In this case gender label is assumed to be unknown at test time, whereas the previous models assumed that gender information was available. While in NIST evaluations the gender label is given, in real applications it might not be available, thus we have to either estimate it from the data or to train a GI system which is able to model both genders. As described in Section 5.2.3 the PSVM approach can handle GI detection by simply training the system on a dataset with mixed gender utterances. In this section we compare the performance of this approach with the Gender Dependent (GD) PSVM system in order to asses the quality of the GI model. In particular, we compare the performance of three types of pairwise SVM systems: a fully GD system (GD), where both i–vector extraction and SVM training is gender dependent, a partially gender independent system (PGI) where the i–vectors are gender independent, whereas SVM is trained using GD trials, and finally a totally gender independent (GI) system, where both i–vectors and SVM are trained without using gender labels. For GD and PGI systems gender labels are provided at test time, while for the GI system no gender information is used to score trials. Pairwise SVMs

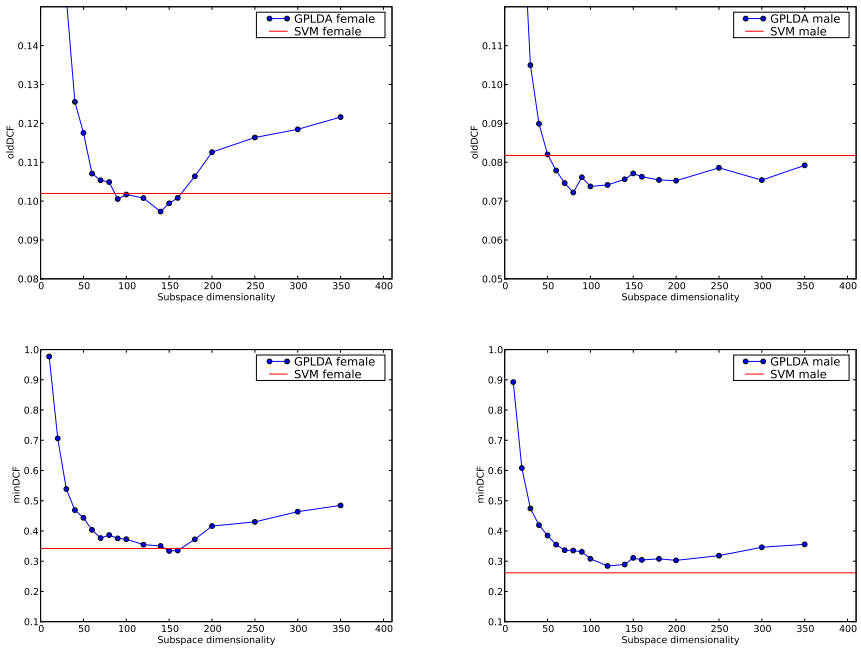


Figure 6.4: EER and DCF versus speaker subspace dimensionality for a GPLDA system

are trained according to [73], applying Within–Class Covariance Normalization to the i–vectors.

Results are reported in Tables 6.11 and 6.12 for the telephone condition in the NIST 2008 and for the extended tel–tel condition in the NIST 2010 evaluations, respectively, and for 400 and 600 dimension i–vectors. Note that, since a slightly different training dataset was used to estimate the i–vector subspace matrix, the results cannot be directly compared to the previous reported results for GD modeling.

Considering the performance of the 400-dimension i–vector systems on SRE08, the GI system gives results which are comparable to the GD and to the PGI systems. Surprising results were obtained with the 600-dimension i–vector system on the same data: the minDCF is similar to the 400 i–vector GI and the PGI systems, but the EER is worse. The 600 GI system, on the contrary shows a substantial improvement mostly for male speakers.

On the more meaningful extended telephone tests of SRE10 the GI systems, both with 400 and 600 i–vectors, are comparable to the PGI systems and not far from the GD systems.

Overall, these experiments show that the performance of a fully gender independent pairwise discriminative SVM model is comparable to that of a gender dependent

Table 6.11: Results for the SRE2008 tests in terms of % EER and minDCF08 with 400 and 600 dimension i–vectors

Gender	Female		Male	
System	EER	minDCF08	EER %	minDCF08
400 GD	2.65 %	0.081	1.26	0.053
400 PGI	2.69 %	0.077	1.34	0.056
400 GI	2.54 %	0.078	1.29	0.056
600 GD	2.64 %	0.078	1.74	0.055
600 PGI	2.59 %	0.076	1.42	0.052
600 GI	2.39 %	0.067	1.18	0.044

Table 6.12: EER and minDCFs for the SRE2010 tests with 400 and 600 dimension i–vectors

Gender	Female			Male		
System	EER	DCF08	DCF10	EER	DCF08	DCF10
400 GD	2.21 %	0.109	0.360	1.73 %	0.081	0.303
400 PGI	2.49 %	0.115	0.369	1.84 %	0.084	0.298
400 GI	2.51 %	0.115	0.382	1.82 %	0.087	0.309
600 GD	2.32 %	0.106	0.342	1.76 %	0.077	0.290
600 PGI	2.59 %	0.103	0.358	1.82 %	0.082	0.274
600 GI	2.51 %	0.108	0.383	1.80 %	0.078	0.307

PSVM trained on the same i–vectors, so that most of the degradation between GD and GI PSVM systems can be explained as due to the lower accuracy of GI i–vectors over GD i–vectors.

6.4.3 Performance on non–NIST datasets

Finally, to asses the quality of the PSVM models we presents the results of a comparative study of different speaker recognition technologies on datasets with different characteristics than those provided by NIST for SRE evaluations and using smaller UBM models which require less computational power and time. The complete analysis can be found in [5].

The compared systems are based on different acoustic features and different UBM sizes. In particular, we trained a 45–PLP 1024 components UBM system and a 25–MFCC 512–components UBM system, both gender–independent. 400–dimensional i–vectors have been extracted in both cases and used to train a GPLDA model with

WCCN and length normalization, a HTPLDA model, a cosine distance model based on LDA and WCCN and a PSVM model with WCCN. All the i–vector systems and the UBMs were trained using 1000 hours of speech data selected from the NIST SRE 2004, 2005 and 2006 [133, 134, 135], LDC Callfriend [123], and Italian, Portuguese and Swedish SpeechDat2 corpora [138].

The test dataset was created using 9 two-side conversational telephone speech corpora distributed by Appen Pty. Ltd. [139]. In each corpus the conversations are carried out between 200 native speakers of a given language. The primary use of the Appen databases is language identification of telephone speech. It is, however, possible to use these corpora also for speaker recognition evaluation, because almost all the speakers made two different calls. These calls can be used to create both target and impostor speaker trials. Each Appen conversation side typically lasts 5 minutes, twice as much as the SRE08 segments. On the other hand, all the target speaker's trials are affected by handset and channel mismatch, because the Appen specifications impose that each speaker makes two calls: one from a fixed telephone line, and the other one from a mobile phone. The Appen datasets we used include the following languages: Bulgarian, Dutch, Hebrew, Croatian, Italian, European Portuguese, Romanian, Russian and Turkish. One call of each speaker has been randomly selected as an enrollment segment, and the other call is used as a target speaker trial. The set of impostor trials is populated by all the segments having the same language (i.e. belonging to the same corpus) and same gender speakers, not previously selected as enrollment segments. The total number of female target and impostor speaker trials is 810 and 71817, respectively. These numbers increase to 1028 and 117365, respectively, for the male speakers. The trials are evenly distributed among the nine languages.

Table 6.13 shows the results of the different systems in terms of EER and 2008 SRE DCF. All scores have been normalized by means of AS–norm [5]. Again, the PSVM approach achieves very competitive results compared to other non–discriminative state–of–the–art approaches.

Table 6.13: EER and DCF for different systems on different datasets

	Male		Female		Male		Female	
System	45 PLP 1024G				25 MFCC 1024G			
	EER (%)	DCF08	EER (%)	DCF08	EER (%)	DCF08	EER (%)	DCF08
Cosine	4.47	0.163	5.19	0.171	5.74	0.188	5.43	0.231
GPLDA	4.28	0.165	4.69	0.169	4.98	0.185	4.79	0.218
HTPLDA	4.67	0.175	4.69	0.162	5.35	0.199	4.71	0.213
PSVM	3.99	0.166	4.45	0.144	4.56	0.180	5.18	0.197

6.4.4 Training the PSVM system

Finally, we resume the analysis of Section 6.3 about the performance of different SVM algorithms applied to the PSVM training problem. As already mentioned in Section 5.2.5 BMRM and Pegasos are the most attractive approaches to solve this problem, so we restrict our analysis to this two algorithms. The results we report are relative to the first PSVM system presented in the previous sections.

The first problem we address is the selection of an appropriate size of Pegasos bunches. Different bunch sizes can, in fact, affect performance: too small bunches do not allow to exploit the efficient scoring and gradient computation techniques described in Section 5.2.5, while too big bunches tend to reduce the algorithm to a standard gradient descent. The effect of the bunch size on training time is shown in Figure 6.5, where DCF10 versus training time is plotted for different bunch sizes (expressed as percentage of the full dataset). These results suggest to use moderately low bunch sizes (in the following we will use the 3% bunch size system).

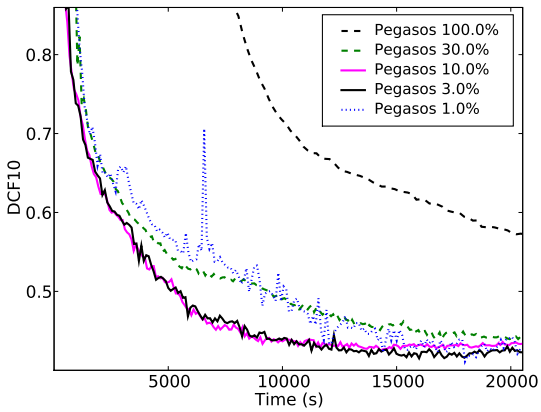
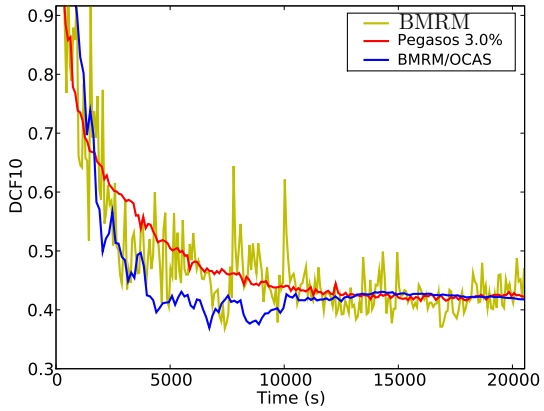


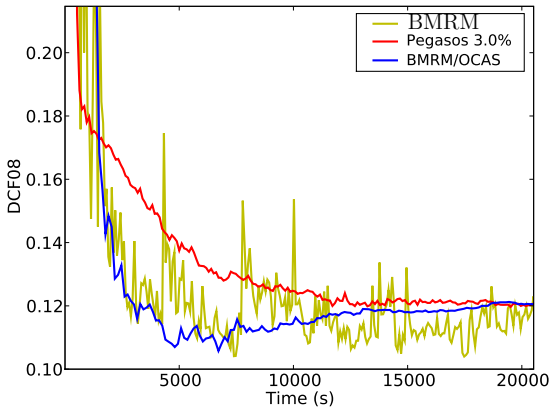
Figure 6.5: SRE-10 DCF of Pegasos models as a function of their training time for different bunch sizes (% of the training dataset)

Fig. 6.6 shows DCF10 versus training time for both Pegasos and BMRM. While good models can be obtained with BMRM after few iterations, the algorithm presents a rather unstable behavior, which was not evident with language recognition models. This is due to oscillations in the loss functions. This behavior is not surprising because at each iteration BMRM finds a cutting plane which approximates the objective function at the current solution. However the objective function does not necessarily decrease at each iteration,. These fluctuations reduce toward convergence and the algorithm reaches the asymptotic performance of Pegasos.
The third entry in Figure 6.6 shows how the OCAS approach (Section 4.3.2) allows to greatly improve BMRM convergence, although some fluctuations still remain.

This is done at the expense of a higher execution time per iteration.



(a)



(b)

Figure 6.6: DCF10 (a) and DCF08 (b) as a function of the training time for models produced by BMRM, Pegasos, and OCAS

As far as the DCF is concerned, the asymptotic performance of BMRM/OCAS and Pegasos is similar, as reported in the first two rows of Table 6.14. However, the comparison of the DCF10 and DCF08 in Figure 6.6 (a) and (b), respectively, shows that Pegasos reaches slowly its asymptotic performance, confirming the results obtained in the language recognition experiments. BMRM has large fluctuations, thus finding an early stopping criterion is difficult. However, BMRM has also the potential to give much better generalization results on the evaluation data if its models are trained with a reduced number of iterations. BMRM/OCAS allows to

Table 6.14: SRE 2010 female tel–tel condition performance for BMRM and Pegasos

Algorithm	Model	EER	DCF08	DCF10
BMRM/OCAS	asymptotic	2.54%	0.120	0.416
Pegasos	asymptotic	2.54%	0.119	0.420
BMRM/OCAS	$\approx 7000s$	2.32%	0.110	0.401
Pegasos	$\approx 7000s$	2.62%	0.121	0.434
BMRM/OCAS	best	2.19%	0.106	0.369

fit the best performance of BMRM using models obtained by stopping the iterations when the loss function decrease is less than 3%. This condition, which is usually met after a few tens of iterations, avoids over-fitting which manifesting in the asymptotic convergence region. The stopping criterion value was estimated using the NIST-SRE 2008 evaluation data as held–out development set. In these experiments the stopping criterion has been reached in 50 iterations corresponding to approximately 7000 seconds of training time. Table 6.14 compares the results of the models trained by BMRM/OCAS and by Pegasos in this same amount of time, showing that the performance of Pegasos is worse compared not only to BMRM/OCAS, but also to its asymptotic results. Last row of the table shows, for reference, the results of the best model. These results are slightly different with respect to the results presented in Section 6.4.1 due to the use of the OCAS extension to BMRM and, therefore, of a different stopping criterion.

6.5 I–vector extraction

The next set of experiments we present is related to the i–vector extraction process. In particular, we compare the performance of diagonalized i–vector, VB–based i–vector and CG–based i–vector extractors (Sections 3.5.3) in terms of extraction time and performance of the generated i–vectors. In order to asses i–vector quality, we present the results of a WCCN–LDA–cosine distance system and a GPLDA with WCCN and length normalization system. 400–dimensional i–vectors are extracted from a 60–MFCC 2048–Gaussians diagonal covariance GI UBM trained over NIST 2004, 2005 and 2006 SRE data [133, 134, 135]. The i–vector subspace matrix was estimated using the same data as for the UBM. LDA–WCCN system and GPLDA were trained using the NIST SRE 2004, 2005 and 2006 train and test corpora, the Switchboard II Phase 2 and 3 corpora and the Switchboard Cellular part 1 and 2 corpora [123]. Accuracy results are given in terms of EER, 2008 NIST SRE 2008 DCF (DCF08) [140] and NIST 2010 SRE DCF (DCF10) [120] for the female subset of the NIST 2010 SRE tel–tel condition [120]. Timings do not include feature extraction and statistics estimation.

The first two lines show the performance, average extraction time and required memory for the classical i–vector extraction approaches. The fast method pre-computes the $\mathbf{T}_c^T \boldsymbol{\Sigma}_c^{-1} \mathbf{T}_c$ matrices, while the slow method corresponds to the naïve posterior calculations. The third line presents the results of diagonalized i–vectors where the diagonalizing matrix is estimated as described in Section 3.5.3. We can observe that this technique has very fast extraction time with very limited memory requirements, however recognition accuracy is degraded significantly. The next two lines refer to a VB implementation using a block size of 20, and differ only in the number of iterations required. In particular, we used a stopping criterion based on the norm of the difference between two consecutive updates. The number shown in the system name column represents the stopping threshold. While not as fast as diagonalized i–vectors, the VB approach allows very good performance in terms of speed–up with almost no accuracy degradation. Finally, the last two lines refer to the CG–based i–vector extraction. In this case, the stopping criterion is given by the norm of the residual term and the number next to the system names shows the corresponding threshold. As for the VB, also the CG proves to be a fast and accurate technique for i–vector extraction. While slightly inferior to the VB in terms of processing time, this technique allows fast estimation of accurate i–vectors with the same memory requirements as diagonalized i–vectors.

Table 6.15: Results for the NIST SRE2010 tests in terms of % EER, minDCF08 and minDCF10 with different i–vector extractors

System	Memory	1 core 100 utterances 1127224 frames		12 cores 500 utterances 6168082 frames	
	(MB)	rel. speedup	cpu time	rel. speedup	cpu time
Fast baseline	2875	1	116.8 s	1	109.2 s
Slow baseline	375	0.27	435.9 s	0.36	306.5 s
Eigen	382	29.30	4.0 s	12.71	8.8 s
VB-20-2	500	4.23	27.5 s	2.98	37.5 s
VB-20-1	500	2.63	44.3 s	1.60	69.5 s
CG-2	375	0.85	136.6 s	1.48	75.4 s
CG-1	375	0.56	207.4 s	1.01	110.4 s

System	Cosine Scoring			PLDA		
	EER (%)	DCF08	DCF10	EER (%)	DCF08	DCF10
Fast baseline	4.97	230	612	3.59	180	567
Slow baseline	4.97	230	612	3.59	180	567
Eigen	5.67	252	697	4.26	202	685
VB-20-2	5.13	232	622	3.51	183	576
VB-20-1	4.93	229	621	3.46	182	569
CG-2	5.16	224	618	3.59	183	567
CG-1	4.96	230	612	3.59	179	564

Chapter 7

Conclusions

We presented an overview of several state–of–the–art technologies for speaker and language recognition, with a particular focus on Support Vector Machine based discriminative techniques. We analyzed some techniques to extract significant features from the acoustic signal and to model the speech content by means of phonetic decoding. We also detailed some of the basic models which are the basis of many different speaker and language systems, Gaussian Mixture Models and Hidden Markov Models. We then focused our attention on latent variable models, and in particular on the use of Factor Analysis derived techniques to build information–rich low–dimensional representations of utterances. We then described state–of–the–art generative techniques built on top of these low–dimensional representations.

We also presented the author's contributions to these fields, which cover different aspects of speaker and language modeling, starting from a technique to speed–up neural network training for speech decoding by means of a GPGPU computing framework, technique which allowed us to greatly improve training times.

As far as JFA models are concerned, we detailed a technique for discriminative language recognition using low–dimensional features extracted from a Factor Analysis front–end, the language factors. We proved that these low–dimensional features can be applied to different language recognition schemes and that they allow achieving good recognition performance while, at the same time, training simplifies thanks to the reduced size of patterns.

Still working with low–dimensional features, we showed an efficient way to compute i–vectors based on variational approximations of distributions. Experimental results show that these techniques allow for much faster i–vector extraction with low memory requirements while preserving recognition accuracy. Variational approximations, moreover, also provide a framework to extend i–vectors to use different prior and posterior distributions.

Finally, we analyzed discriminative techniques based on Support Vector Machines (SVM). We presented some of the concepts and issues related to SVM training, together with some solutions for large–scale linear problems. We then applied SVMs to both language recognition and speaker recognition problems. In the former case, we showed how different algorithms behave with different language recognition techniques. We also showed how multiclass classifiers can be used in a phonotactic language recognition system based on low–dimensional representations of phonetic information. We then presented a novel framework for discriminative training of speaker verification systems, Pairwise SVMs (PSVM). We showed that pairwise SVMs provide a more natural approach to discriminative speaker verification compared to the classical one–vs–all paradigm. We also showed that this technique has strong connections with state–of–the–art generative models and, at the same time, can be interpreted as a simple polynomial kernel classifier. While some issues are still open, for example extensions of the model to deal with more than two utterances or large–scale training, pairwise SVMs provide models which allow for fast scoring of test utterances and achieve state–of–the–art performance.

Bibliography

[1] R. Auckenthaler, M. Carey, and H. Lloyd-Thomas, "Score normalization for text-independent speaker verification systems," *Digital Signal Processing*, vol. 10, no. 1-3, pp. 42–54, 2000.

[2] D. E. Sturim and D. A. Reynolds, "Speaker adaptive cohort selection for tnorm in text-independent speaker verification," in *in Proceedings of ICASSP 2005*, pp. 741–744, 2005.

[3] N. Dehak, R. Dehak, J. Glass, D. Reynolds, and P. Kenny, "Cosine similarity scoring without score normalization techniques," 2010.

[4] Z. N. Karam, W. M. Campbell, and N. Dehak, "Towards reduced false-alarms using cohorts," in *ICASSP*, pp. 4512–4515, IEEE, 2011.

[5] S. Cumani, P. D. Batzu, D. Colibro, C. Vair, and V. Vasilakakis, "Comparison of speaker recognition approaches for real applications," in *Proc. of Interspeech 2011*, pp. 2365–2368, august 2011.

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

[7] L. Rabiner and B.-H. Juang, *Fundamentals of speech recognition*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993.

[8] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 28, pp. 357–366, Aug. 1980.

[9] J. Pelecanos and S. Sridharan, "Feature Warping for Robust Speaker Verification," in *Proceedings of Odyssey*, pp. 213–218, 2001.

[10] P. A. Torres-Carrasquillo, D. A. Reynolds, E. S. M. A. Kohler, R. J. Greene, and J. J. R. Deller, "Approaches to language identification using Gaussian Mixture Models and shifted delta cepstral features," in *Proceedings of ICSLP 2002*, pp. 89–92, 2002.

[11] W. Campbell, P. A. Torres-Carrasquillo, and D. Reynolds, "Language recognition with support vector machines," in *Proceedings of Odyssey:The Speaker and Language Recognition Workshop*, pp. 41–44, ISCA, 2004.

[12] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1st ed. 2006. corr. 2nd printing ed., Oct. 2007.

[13] D. Reynolds and R. Rose, "Robust text-independent speaker identification using gaussian mixture speaker models," *Speech and Audio Processing, IEEE Transactions on*, vol. 3, pp. 72–83, jan 1995.

[14] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum Likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society, Series B*, vol. 39, no. 1, pp. 1–38, 1977.

[15] S. Borman, "The Expectation Maximization algorithm – a short tutorial." Jul 2004.

[16] F. Dellaert, "The Expectation Maximization algorithm," tech. rep., 2002.

[17] R. Rabiner, "A tutorial on Hidden Markov Models and selected applications in speech recognition," *Proceedings of IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[18] M. A. Mohamed and P. Gader, "Generalized Hidden Markov Models – part i: Theoretical frameworks," *IEEE Transactions on Fuzzy Systems*, vol. 8, no. 1, pp. 67–81, 2000.

[19] H. A. Bourlard and N. Morgan, *Connectionist Speech Recognition: A Hybrid Approach*. Norwell, MA, USA: Kluwer Academic Publishers, 1993.

[20] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford University Press, USA, 1 ed., Jan. 1996.

[21] R. Lippmann, "An introduction to computing with neural nets," *ASSP Magazine, IEEE*, vol. 4, pp. 4–22, Apr 1987.

[22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning internal representations by error propagation*, pp. 318–362. Cambridge, MA, USA: MIT Press, 1986.

[23] L. Fissore, F. Ravera, and P. Laface, "Acoustic-phonetic modeling for flexible vocabulary speech recognition," in *EUROSPEECH*, ISCA, 1995.

[24] J.-L. Gauvain, A. Messaoudi, and H. Schwenk, "Language recognition using phone lattices," in *INTERSPEECH*, ISCA, 2004.

[25] M. A. Zissman, "Comparison of four approaches to automatic language identification of telephone speech," *IEEE Transactions on Speech and Audio Processing*, vol. 4, no. 1, pp. 31–44, 1996.

[26] F. Castaldo, D. Colibro, S. Cumani, E. Dalmasso, P. Laface, and C. Vair, "Loquendo-Politecnico di Torino system for the 2009 NIST Language Recognition Evaluation," in *Proceedings of ICASSP 2010*, pp. 5002–5005, 2010.

[27] F. Jelinek, "Self–organized language modeling for speech recognition," in *Readings in Speech Recognition* (A. Waibel and K. F. Lee, eds.), pp. 450–506, Morgan Kauffman, 1990.

[28] D. Albesano, R. Gemello, and F. Mana, "Hybrid hmm-nn modeling of stationary-transitional units for continuous speech recognition," *Inf. Sci.*, vol. 123, no. 1-2, pp. 3–11, 2000.

[29] "Loquendo ASR." http://www.loquendo.com/en/products/speech-recognition/.

[30] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*, vol. 1. Addison-Wesley, 1991.

[31] D. Anguita, G. Parodi, and R. Zunino, "An efficient implementation of bp on risc-based workstations.," *Neurocomputing*, vol. 6, no. 1, pp. 57–65, 1994.

[32] S. Scanzio, S. Cumani, R. Gemello, F. Mana, and P. Laface, "Parallel implementation of artificial neural network for speech recognition," *Pattern Recognition Letters*, vol. 31, no. 11, pp. 1302–1309, 2010.

[33] S. Scanzio, S. Cumani, R. Gemello, F. Mana, and P. Laface, "Parallel implementation of artificial neural network training," in *ICASSP*, pp. 4902–4905, IEEE, 2010.

[34] J. Bilmes, K. Asanović, C.-W. Chin, and J. Demmel, "Using PHiPAC to speed error back-propagation learning," in *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, pp. 4153–4156, April 1997.

[35] J. Hoskins, "Speeding up artificial neural networks in the 'real' world," in *Neural Networks, 1989. IJCNN., International Joint Conference on*, p. 626 vol.2, jun 1989.

[36] NVIDIA, *NVIDIA CUDA Programming Guide 2.0*. 2008.

[37] M. Černansky, "Training recurrent neural network using multistream extended kalman filter on multicore processor and cuda enabled graphic processor unit," *Artificial Neural Networks - ICANN 2009*, vol. 5768, pp. 381–390, 2009.

[38] H. Jang, A. Park, and K. Jung, "Neural network implementation using cuda and openmp," in *Proceedings of the 2008 Digital Image Computing: Techniques and Applications*, (Washington, DC, USA), pp. 155–161, IEEE Computer Society, 2008.

[39] S. Lahabar, P. Agrawal, and P. J. Narayanan, "High performance pattern recognition on gpu," in *Proceedings of the National Conference on Computer Vision, Pattern Recognition, Image Processing and Graphics 2008*, pp. 154–159, 2008.

[40] A. E. Rosenberg, J. DeLong, C.-H. Lee, B.-H. Juang, and F. K. Soong, "The use of cohort normalized scores for speaker verification," in *The Second International Conference on Spoken Language Processing, ICSLP 1992, Banff, Alberta, Canada, October 13-16, 1992*, ISCA, 1992.

[41] D. A. Reynolds, "Comparison of Background Normalization Methods for Text-Independent Speaker Verification," in *Proc. Eurospeech '97*, (Rhodes, Greece), pp. 963–966, Sept. 1997.

[42] D. A. Reynolds, T. F. Quatieri, and R. B. Dunn, "Speaker verification using adapted gaussian mixture models," vol. 10, pp. 19–41, 2000.

[43] J.-L. Gauvain and C.-H. Lee, "Maximum a posteriori estimation for multivariate gaussian mixture observations of markov chains," *Speech and Audio*

Processing, IEEE Transactions on, vol. 2, pp. 291–298, apr 1994.

[44] P. Kenny, M. Mihoubi, and P. Dumouchel, "New map estimators for speaker recognition," *Test*, pp. 2961–2964, 2003.

[45] P. Kenny, G. Boulianne, and P. Dumouchel, "Eigenvoice modeling with sparse training data," *Speech and Audio Processing, IEEE Transactions on*, vol. 13, pp. 345–354, may 2005.

[46] P. Kenny, "Joint factor analysis of speaker and session variability: Theory and algorithms," Tech. Rep. CRIM-06/08-13, CRIM, 2005.

[47] P. Kenny, G. Boulianne, P. Ouellet, and P. Dumouchel, "Joint factor analysis versus eigenchannels in speaker recognition," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 15, pp. 1435–1447, may 2007.

[48] P. Kenny, G. Boulianne, P. Ouellet, and P. Dumouchel, "Speaker and session variability in gmm-based speaker verification," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 15, pp. 1448–1460, may 2007.

[49] D. Reynolds, "Channel robust speaker verification via feature mapping," in *Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on*, vol. 2, pp. II – 53–6 vol.2, april 2003.

[50] C. Vair, D. Colibro, F. Castaldo, E. Dalmasso, and P. Laface, "Channel factors compensation in model and feature domain for speaker recognition," in *Speaker and Language Recognition Workshop, 2006. IEEE Odyssey 2006: The*, pp. 1–6, june 2006.

[51] N. Brümmer, A. Strasheim, V. Hubeika, P. Matějka, L. Burget, and O. Glembek, "Discriminative acoustic language recognition via channel-compensated gmm statistics," in *Proc. Interspeech 2009*, no. 9, pp. 2187–2190, International Speech Communication Association, 2009.

[52] O. Glembek, L. Burget, N. Dehak, N. Brummer, and P. Kenny, "Comparison of scoring methods used in speaker recognition with joint factor analysis," in *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, pp. 4057–4060, april 2009.

[53] J. G. Dominguez, *Session variability compensation in automatic language and speaker recognition*. PhD thesis, Universidad Autonoma de Madrid, 2011.

[54] N. Dehak, P. Kenny, R. Dehak, O. Glembek, P. Dumouchel, L. Burget, V. Hubeika, and F. Castaldo, "Support vector machines and joint factor analysis for speaker verification," in *ICASSP*, pp. 4237–4240, IEEE, 2009.

[55] W. M. Campbell, D. E. Sturim, D. A. Reynolds, and A. Solomonoff, "SVM based speaker verification using a gmm supervector kernel and nap variability compensation," *2006 IEEE International Conference on Acoustics Speed and Signal Processing Proceedings*, pp. 97–100, 2006.

[56] W. M. Campbell, D. E. Sturim, and D. A. Reynolds, "Support vector machines using GMM supervectors for speaker verification," *Signal Processing Letters, IEEE*, vol. 13, pp. 308–311, Apr. 2006.

[57] A. O. Hatch, S. S. Kajarekar, and A. Stolcke, "Within-class covariance normalization for SVM-based speaker recognition," in *INTERSPEECH*, ISCA, 2006.

[58] F. Castaldo, S. Cumani, P. Laface, and D. Colibro, "Language recognition using language factors," in *INTERSPEECH*, pp. 176–179, ISCA, 2009.

[59] N. Dehak, *Discriminative and generative approaches for long- and short-term speaker characteristics modeling: application to speaker verification.* PhD thesis, 2009. AAINR50490.

[60] N. Dehak, P. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, "Front-end factor analysis for speaker verification," *IEEE Transactions on Audio, Speech & Language Processing*, vol. 19, no. 4, pp. 788–798, 2011.

[61] P. Kenny, P. Ouellet, N. Dehak, V. Gupta, and P. Dumouchel, "A study of interspeaker variability in speaker verification," *IEEE Transactions on Audio, Speech & Language Processing*, vol. 16, no. 5, pp. 980–988, 2008.

[62] N. Dehak, R. Dehak, P. Kenny, N. Brümmer, P. Ouellet, and P. Dumouchel, "Support vector machines versus fast scoring in the low-dimensional total variability space for speaker verification," in *INTERSPEECH*, pp. 1559–1562, ISCA, 2009.

[63] O. Glembek, L. Burget, P. Matejka, M. Karafiát, and P. Kenny, "Simplification and optimization of i-vector extraction," in *ICASSP*, pp. 4516–4519, IEEE, 2011.

[64] N. Kumar, *Investigation of silicon auditory models and generalization of linear discriminant analysis for improved speech recognition.* PhD thesis, 1997. AAI9730738.

[65] M. Gales, "Semi-tied covariance matrices for hidden Markov models," in *IEEE Transaction on speech and Audio Processing*, vol. 7, pp. 272–281, 1999.

[66] G. Meurant, *Computer solution of large linear systems.* Studies in mathematics and its applications, North-Holland, 1999.

[67] M. R. Hestenes and E. Stiefel, "Methods of Conjugate Gradients for Solving Linear Systems," *Journal of Research of the National Bureau of Standards*, vol. 49, pp. 409–436, Dec. 1952.

[68] A. Hatch and A. Stolcke, "Generalized linear kernels for one-versus-all classification: Application to speaker recognition," in *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, vol. 5, p. V, may 2006.

[69] S. J. D. Prince and J. H. Elder, "Probabilistic linear discriminant analysis for inferences about identity," in *11th International Conference on Computer Vision*, pp. 1–8, 2007.

[70] P. Kenny, "Bayesian speaker verification with heavy–tailed priors," in *keynote presentation, Odyssey 2010*, 2010.

[71] D. Garcia-Romero and C. Y. Espy-Wilson, "Analysis of i-vector length normalization in speaker recognition systems," in *INTERSPEECH*, pp. 249–252, ISCA, 2011.

[72] N. Brümmer and E. de Villiers, "The speaker partitioning problem," in *Proc. Odyssey 2010*, pp. 194–201, 2010.

[73] S. Cumani, N. Brümmer, L. Burget, and P. Laface, "Fast discriminative speaker verification in the i–vector space," in *Proceedings of ICASSP 2011*, 2011.

[74] L. Burget, O. Plchot, S. Cumani, O. G. P. Matějka, and N. Brümmer, "Discriminatively trained Probabilistic Linear Discriminant Analysis for speaker verification," in *Proceedings of ICASSP 2011*, 2011.

[75] V. N. Vapnik, *The nature of statistical learning theory*. New York, NY, USA: Springer-Verlag New York, Inc., 1995.

[76] C. J. C. Burges, "A tutorial on Support Vector Machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, pp. 121–167, 1998.

[77] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pp. 144–152, 1992.

[78] D. Luenberger, *Linear and Nonlinear Programming*. Addison–Wesley, 1984.

[79] A. Aizerman, E. M. Braverman, and L. I. Rozoner, "Theoretical foundations of the potential function method in pattern recognition learning," *Automation and Remote Control*, vol. 25, pp. 821–837, 1964.

[80] J. Mercer, "Functions of positive and negative type, and their connection with the theory of integral equations," *Royal Society of London Proceedings Series A*, vol. 83, pp. 69–70, nov 1909.

[81] K. P. Bennett and O. L. Mangasarian, "Robust linear programming discrimination of two linearly inseparable sets," *Optimization Methods and Software*, vol. 1, no. 1, pp. 23–34, 1992.

[82] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[83] T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu, "The entire regularization path for the support vector machine," *J. Mach. Learn. Res.*, vol. 5, pp. 1391–1415, December 2004.

[84] T. Joachims, "Making large–scale Support Vector Machine learning practical," in *Advances in Kernel Methods – Support Vector Learning*, pp. 169–184, MIT–Press, 1999.

[85] C. H. Teo, A. Smola, S. V. Vishwanathan, and Q. V. Le, "A scalable modular convex solver for regularized risk minimization," in *Proceedings of KDD 2007*, pp. 727–736, 2007.

[86] C. H. Teo, A. Smola, S. V. Vishwanathan, and Q. V. Le, "Bundle methods for regularized risk minimization," *J. Mach. Learn. Res.*, vol. 11, pp. 311–365,

March 2010.

[87] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer, Aug. 2000.

[88] B. Taskar, C. Guestrin, and D. Koller, "Max-margin markov networks," in *NIPS* (S. Thrun, L. K. Saul, and B. Schölkopf, eds.), MIT Press, 2003.

[89] M. Soufifar, S. Cumani, L. Burget, and Černocký, "Discriminative classifier for phonotactic language recognition with ivectors," in *Proc. of ICASSP 2012*, 2012.

[90] Available at http://www.itl.nist.gov/iad/mig/tests/lre/2007/LRE07EvalPlan-v8b.pdf.

[91] Available at http://www.itl.nist.gov/iad/mig/tests/lre/2009/LRE09_Eval Plan_v6.pdf.

[92] Available at http://nist.gov/itl/iad/mig/upload/LRE11_EvalPlan_releasev1.pdf.

[93] W. M. Campbell, D. E. Sturim, and D. A. Reynolds, "Support vector machines using gmm supervectors for speaker verification," *IEEE Signal Processing Letters*, vol. 13, pp. 308–311, 2006.

[94] W. M. Campbell, D. E. Sturim, D. A. Reynolds, and A. Solomonoff, "SVM based speaker verification using a GMM supervector kernel and NAP variability compensation," in *Proceedings of ICASSP 2006*, pp. 97–100, 2006.

[95] W. M. Campbell, "A covariance kernel for SVM language recognition," in *Proceedings of ICASSP 2008*, pp. 4141–4144, 2008.

[96] S. Kullback and R. A. Leibler, "On information and sufficiency," *Annals of Mathematical Statistics*, vol. 22, pp. 49–86, 1951.

[97] M. N. Do, "Fast approximation of Kullback-Leibler distance for dependence trees and hidden Markov models," *Signal Processing Letters, IEEE*, vol. 10, pp. 115–118, Mar. 2003.

[98] F. Castaldo, D. Colibro, E. Dalmasso, P. Laface, and C. Vair, "Acoustic language identification using fast discriminative training," in *Proceedings of Interspeech 2007*, pp. 346–349, 2007.

[99] F. Castaldo, D. Colibro, E. Dalmasso, P. Laface, and C. Vair, "Compensation of nuisance factors for speaker and language recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, no. 7, pp. 1969–1978, 2007.

[100] F. Castaldo, D. Colibro, E. Dalmasso, P. Laface, and C. Vair, "Compensation of nuisance factors for speaker and language recognition," *IEEE Transactions on Audio, Speech & Language Processing*, vol. 15, no. 7, pp. 1969–1978, 2007.

[101] W. M. Campbell, J. P. Campbell, D. A. Reynolds, D. A. Jones, and T. R. Leek, "High-level speaker verification with Support Vector Machines," in *Proc. Int. Conf. Acoustics, Speech and Signal Processing*, pp. 73–76, 2004.

[102] A. O. Hatch, B. Peskin, and A. Stolcke, "Improved phonetic speaker recognition using lattice decoding," in *Proc. Int. Conf. Acoustics, Speech and Signal Processing*, pp. 169–172, 2005.

[103] W. M. Campbell, F. Richardson, and D. A. Reynolds, "Language recognition with word lattices and Support Vector Machines," in *Proc. Int. Conf. Acoustics, Speech and Signal Processing*, pp. 989–992, 2007.

[104] F. S. Richardson and W. M. Campbell, "Language recognition with discriminative keyword selection," in *Proc. Int. Conf. Acoustics, Speech and Signal Processing*, pp. 4145–4148, 2008.

[105] M. Soufifar, M. Kockmann, L. Burget, O. Plchot, O. Glembek, and T. Svendsen, "ivector approach to phonotactic language recognition," in *Proceedings of Interspeech 2011*, vol. 2011, pp. 2913–2916, International Speech Communication Association, 2011.

[106] M. Kockmann, L. Burget, O. Glembek, L. Ferrer, and J. Černocký, "Prosodic speaker verification using subspace multinomial models with intersession compensation," in *Proceedings of the 11th Annual Conference of the International Speech Communication Association (INTERSPEECH 2010)*, vol. 2010, pp. 1061–1064, International Speech Communication Association, 2010.

[107] S. Cumani, F. Castaldo, P. Laface, D. Colibro, and C. Vair, "Comparison of large–scale SVM training algorithms for language recognition," in *Proceedings of Odyssey 2010, The Speaker and Language Recognition Workshop*, pp. 222–229, 2010.

[108] S. Cumani and P. Laface, "Analysis of large-scale svm training algorithms for language and speaker recognition," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. PP, no. 99, 2012.

[109] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan, "A dual coordinate descent method for large–scale linear SVM," in *Proceedings of ICML 2008*, pp. 408–415, 2008.

[110] T. Joachims, "A support vector method for multivariate performance measures," in *Proceedings of ICML 2005*, pp. 377–384, 2005.

[111] T. Joachims, "Training linear SVMs in linear time," in *Proceedings of KDD 2006*, pp. 217–226, 2006.

[112] T. Joachims and C.-N. J. Yu, "Sparse kernel SVMs via cutting–plane training," *Machine Learning*, vol. 76, no. 2–3, pp. 179–193, 2009.

[113] S. Shalev-Shwartz, Y. Singer, and N. Srebro, "Primal estimated sub–gradient solver for SVM," in *Proceedings of ICML 2007*, pp. 807–814, 2007.

[114] V. Franc and S. Sonnenburg, "Optimized cutting plane algorithm for support vector machines," in *Proceedings of the 25th international conference on Machine learning*, ICML '08, (New York, NY, USA), pp. 320–327, ACM, 2008.

[115] A. Solomonoff, W. M. Campbell, and I. Boardman, "Advances in channel compensation for SVM speaker recognition," *Proc ICASSP vol I Philadelphia PA USA Mar*, no. 1, pp. 629–632, 2005.

[116] A. Solomonoff, C. Quillen, and W. Campbell, "Channel compensation for SVM speaker recognition," in *Proc. ˜Odyssey-04*, (Toledo, Spain), May 2004.

[117] W. Campbell, "Weighted nuisance attribute projection," in *Proceedings of Odyssey 2010, The Speaker and Language Recognition Workshop*, 2010.

[118] L. Burget, N. Brümmer, D. Reynolds, P. Kenny, J. Pelecanos, R. Vogt, F. Castaldo, N. Dehak, R. Dehak, O. Glembek, Z. Karam, J. J. Noecker, Y. H. Na, C. C. Costin, V. Hubeika, S. Kajarekar, N. Scheffer, and J. Černocký, "Robust speaker recognition over varying channels," tech. rep., 2008.

[119] S. Cumani, O. Glembek, N. Brümmer, E. de Villiers, and P. Laface, "Gender independent discriminative speaker recognition in ivector space," in *Proc. of ICASSP 2012*.

[120] Available at http://www.itl.nist.gov/iad/mig/tests/sre/2010/NIST_SRE10_evalplan.r6.pdf.

[121] INTEL, "MKL: Math kernel library." http://software.intel.com/en-us/articles/intel-mkl/.

[122] D. B. Paul and J. M. Baker, "The design for the wall street journal-based csr corpus," in *Proceedings of the workshop on Speech and Natural Language*, HLT '91, (Stroudsburg, PA, USA), pp. 357–362, Association for Computational Linguistics, 1992.

[123] Available at http://www.ldc.upenn.edu/Catalog/.

[124] http://www.itl.nist.gov/iad/mig/tests/lre/2005/LRE05EvalPlan-v5-2.pdf.

[125] N. Brümmer, L. Burget, O. Glembek, V. Hubeika, Z. Jančík, M. Karafiát, P. Matějka, T. Mikolov, O. Plchot, and A. Strasheim, "But-agnitio system description for nist language recognition evaluation 2009," in *Proceedings NIST 2009 Language Recognition Evaluation Workshop*, pp. 1–7, National Institute of Standards and Technology, 2009.

[126] G. Saon, M. Padmanabhan, R. Gopinath, and S. Chen, "Maximum likelihood discriminant feature spaces," in *Proceedings of ICASSP 2000*, pp. 1129–1132, 2000.

[127] Available at http://numpy.scipy.org.

[128] Available at http://abel.ee.ucla.edu/cvxopt.

[129] Available at http://www.openmp.org.

[130] http://www.itl.nist.gov/iad/mig/tests/lre/2003/LRE03EvalPlan-v1.pdf.

[131] F. Castaldo, D. Colibro, S. Cumani, E. Dalmasso, P. Laface, and C. Vair, "Loquendo-politecnico di Torino system for the 2009 NIST Language Recognition Evaluation," in *Proceedings of ICASSP 2010*, pp. 5002–5005, 2010.

[132] P. Matějka, L. Burget, O. Glembek, P. Schwarz, V. Hubeika, M. Fapso, T. Mikolov, O. Plchot, and J. Cernocky, "But language recognition system for NIST 2007 evaluations," in *Proceedings of Interspeech 2007*, pp. 739–742, 2007.

[133] http://www.itl.nist.gov/iad/mig/tests/spk/2004/SRE-04_evalplan-v1a.pdf.

[134] http://www.itl.nist.gov/iad/mig/tests/sre/2005/sre-05_evalplan-v6.pdf.

[135] http://www.itl.nist.gov/iad/mig/tests/sre/2006/sre-06_evalplan-v9.pdf.

[136] N. Brummer, L. Burget, P. Kenny, P. Matějka, E. V. de, M. Karafiát, M. Kockmann, O. Glembek, O. Plchot, D. Baum, and M. Senoussauoi, "Abc system description for nist sre 2010," in *Proc. NIST 2010 Speaker Recognition Evaluation*, pp. 1–20, National Institute of Standards and Technology, 2010.

[137] N. Brümmer and J. A. du Preez, "Application-independent evaluation of speaker detection," *Computer Speech & Language*, vol. 20, no. 2-3, pp. 230–275, 2006.

[138] Available at http://www.speechdat.org/.

[139] Available at http://www.appen.com.au.

[140] http://www.itl.nist.gov/iad/mig/tests/sre/2008/sre08_evalplan_release4.pdf.

[141] G. McLachlan and T. Krishnan, *The EM Algorithm and Extensions*. John Wiley & Sons, New York, 1996.

[142] L. A. Liporace, "Maximum Likelihood Estimation for multivariate observations of markov sources," *IEEE Transactions on Information Theory*, vol. 28, no. 5, pp. 729–734, 1982.

[143] B. H. Juang, S. E. Levinson, and M. M. Sondhi, "Maximum Likelihood Estimation for multivariate mixture observations of markov chains," *IEEE Transactions on Information Theory*, vol. 32, no. 2, pp. 307–309, 1986.

Appendix A

Expectation–Maximization and HMM algorithms

A.1 The EM algorithm

The EM algorithm [16, 15, 12] is an iterative procedure that to compute Maximum–Likelihood estimates in presence of missing or hidden data. The procedure is split in two steps: the Expectation (E) step, where missing data are estimated given the observed data and the current parameter estimate using conditional expectation, and the Maximization (M) step, where the likelihood function is maximized using the estimate of missing data obtained during the E–step.

Let X be a random vector which results from a parametrized family. We want to maximize the probability $P(X|\theta)$, where θ are the model parameters, or, which is the same, the log–likelihood $L(\theta) = \log P(X|\theta)$. In the following part of this section we will denote θ_n the model parameter estimate after n iterations of the EM algorithm. Since our goal consists in maximizing $L(\theta)$, we wish to evaluate a new estimate of θ such that

$$L(\theta) > L(\theta_n) \tag{A.1}$$

that is, we want to maximize the difference

$$L(\theta) - L(\theta_n) = \log P(X|\theta) - \log P(X|\theta_n) \tag{A.2}$$

We now introduce the hidden random vector Z. When the EM algorithm is used in problems where hidden variables exist, this allows for a natural framework for their inclusion. If this is not the case, hidden variables can be introduced just to make ML estimation tractable. Either way, we can now write the probability $P(X|\theta)$ as

$$P(X|\theta) = \sum_Z P(X|Z,\theta)P(Z|\theta) \tag{A.3}$$

Equation A.2 can now be rewritten as

$$\begin{aligned} L(\theta) - L(\theta_n) &= \log\left(\sum_Z P(X|Z,\theta)P(Z|\theta)\right) - \log P(X|\theta_n) \\ &= \log\left(\sum_Z P(Z|X,\theta_n)\frac{P(X|Z,\theta)P(Z|\theta)}{P(Z|X,\theta_n)}\right) - \log P(X|\theta_n) \\ &\geq \sum_Z P(Z|X,\theta_n)\log\left(\frac{P(X|Z,\theta)P(Z|\theta)}{P(Z|X,\theta_n)}\right) - \log P(X|\theta_n) \quad \text{(A.4)} \\ &= \sum_Z P(Z|X,\theta_n)\log\left(\frac{P(X|Z,\theta)P(Z|\theta)}{P(Z|X,\theta_n)P(X|\theta_n)}\right) \\ &\triangleq \Delta(\theta|\theta_n) \quad \text{(A.5)} \end{aligned}$$

where the inequality derives from Jensen's inequality and the fact that $\sum_Z P(Z|X,\theta_n) = 1$.

Now we can write that

$$L(\theta) \geq L(\theta_n) + \Delta(\theta|\theta_n) \triangleq l(\theta|\theta_n) \quad \text{(A.6)}$$

The function $l(\theta|\theta_n)$ is bounded above by $L(\theta)$. Moreover, it is easy to show that $l(\theta_n|\theta_n) = L(\theta_n)$. Therefore, any θ which increases $l(\theta|\theta_n)$ in turn increases $L(\theta)$. In order to achieve the greatest possible increase of $L(\theta)$ the EM looks for the values of θ maximizing $l(\theta|\theta_n)$. The next estimate of θ is therefore evaluated as

$$\begin{aligned} \theta_{n+1} &= \arg\max_\theta l(\theta|\theta_n) \\ &= \arg\max_\theta \left(L(\theta_n) + \sum_Z P(Z|X,\theta_n)\log\frac{P(X|Z,\theta)P(Z|\theta)}{P(Z|X,\theta_n)P(X|\theta_n)}\right) \\ &= \arg\max_\theta \left(\sum_Z P(Z|X,\theta_n)\log P(X|Z,\theta)P(Z|\theta)\right) \\ &= \arg\max_\theta \left(\sum_Z P(Z|X,\theta_n)\log P(X,Z|\theta)\right) \\ &= \arg\max_\theta \left(E_{Z|X,\theta_n}[\log P(X,Z|\theta)]\right) \quad \text{(A.7)} \end{aligned}$$

In (A.7) the E–step and the M–step are evident: the EM algorithm essentially iterates over

1. E–step: Determine the conditional expectation $E_{Z|X,\theta_n}[\log P(X,Z|\theta)]$ (or $E_{Z,X|\theta_n}[\log P(X,Z|\theta)]$, which is the same since maximization in the M–step is performed with respect to θ)

2. M–step: Maximize the expression determined during the E–step with respect to θ

The convergence of the algorithm is discussed in [141]. When initial parameter estimates are not available, usually random values are used.

A.2 The Forward–Backward algorithm

The forward–backward algorithm allows to efficiently compute the probability of a sequence of observed features $O = \{o_1, \ldots, o_T\}$ which are assumed to be generated by a Hidden Markov Model M. The algorithm implementation can be done resorting to the so–called trellis associated with the HMM. The trellis is essentially a developed representation of the HMM graph containing time information. Each node of the trellis corresponds to a node of the HMM at a given time. A trellis node $n_{i_1}(t_1)$ associated to HMM node s_{i_1} is linked to a trellis node $n_{i_2}(t_2)$ associated with HMM node s_{i_2} if and only if s_{i_1} is linked to s_{i_2} and $t_2 = t_1 + 1$. The probability of the edge is the same as the one of the edge linking s_{i_1} to s_{i_2}. Conceptually, the HMM and its trellis represent the same model.

In the following derivations we assume that the HMM has a single initial state and a single final state. This is not a restriction, since an initial probability distribution over states as well as many final states can be simulated by the addition of an initial and/or a final frame to the trellis and possibly to the observed sequence (Figures A.1 and A.2)

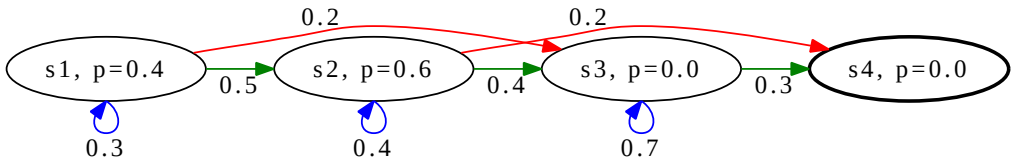


Figure A.1: A HMM with starting probabilities and one final state

The forward step of the forward–backward algorithm evaluates the forward probability $\alpha_t(i)$ of being in state S_i at time t having observed the sequence $o_1, \ldots, o_t$

$$\alpha_t(i) = P(q_t = S_i, o_1, \ldots, o_t | M) \tag{A.8}$$

Since the only starting state can be S_1 at time $t = 1$

$$\alpha_1(1) = f_1(o_1) \tag{A.9}$$

The probability of being in state S_j at time $t + 1$ having observed the sequence $o_1, \ldots, o_{t+1}$ can be evaluated as the product of the probability of being in state S_j

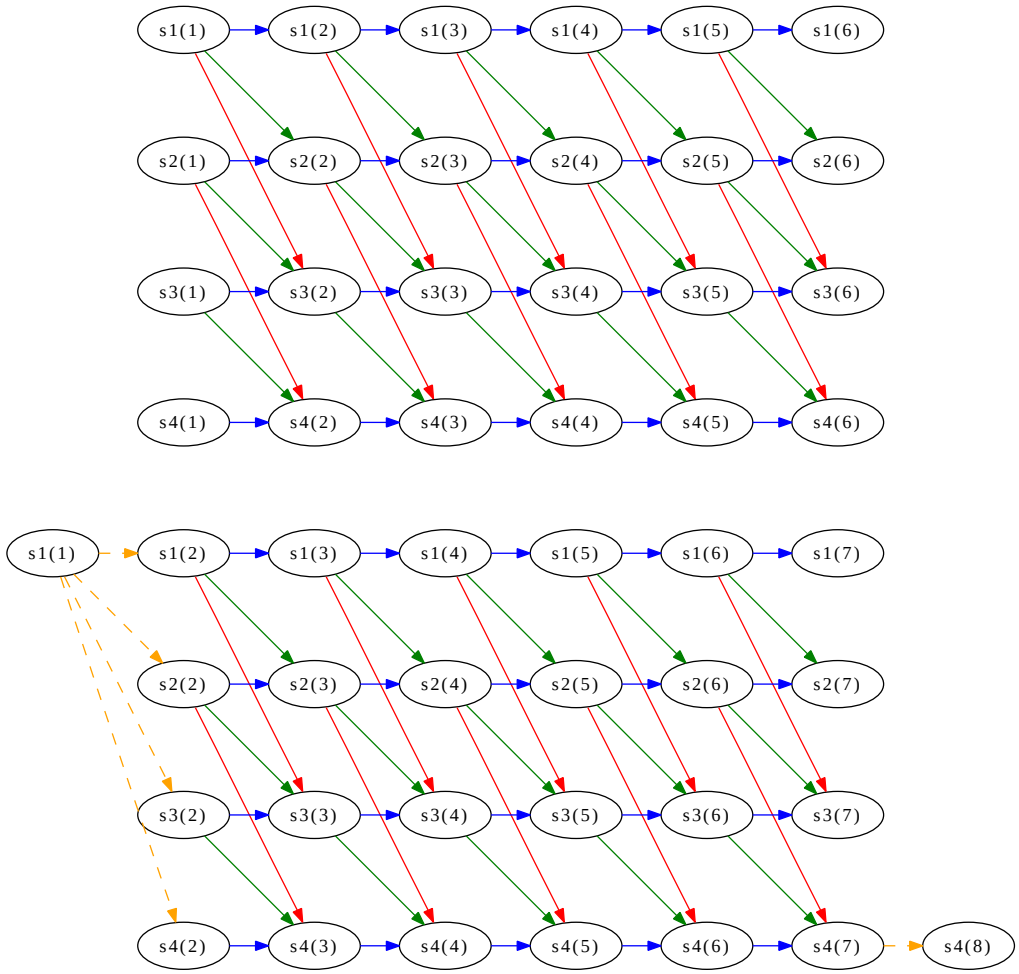


Figure A.2: Six frame long trellis of model shown in Figure A.1 and transformed trellis having one initial state and one final state

at time $t+1$ having observed $o_1, \ldots, o_t$ and the probability of observing o_{t+1} in state S_j. The first probability is the sum over all states for which a transition to S_i exists

multiplied by the probability of having that transition, that is

$$\begin{aligned}
\alpha_{t+1}(j) &= P(q_{t+1} = S_j, o_1, \ldots, o_{t+1}|M) \\
&= P(q_{t+1} = S_j, o_1, \ldots, o_t|M)P(o_{t+1}|q_{t+1} = S_j, M) \qquad \text{(A.10)} \\
&= \sum_{i=1}^{N} P(q_{t+1} = S_j, q_t = S_i, o_1, \ldots, o_t|M)P(o_{t+1}|q_{t+1} = S_j, M) \\
&= \sum_{i=1}^{N} P(q_t = S_i, o_1, \ldots, o_t|M)P(q_{t+1} = S_j|q_t = S_i, o_1, \ldots, o_t, M) \\
&\quad P(o_{t+1}|q_{t+1} = S_j, M) \\
&= \sum_{i=1}^{N} P(q_t = S_i, o_1, \ldots, o_t|M)P(q_{t+1} = S_j|q_t = S_i, M) \\
&\quad P(o_{t+1}|q_{t+1} = S_j, M) \qquad \text{(A.11)} \\
&= \sum_{i=1}^{N} \alpha_t(i)a_{ij}f_j(o_{t+1}) \qquad \text{(A.12)}
\end{aligned}$$

where (A.10) is justified by the assumption of statistically independent observed features and (A.11) and (A.12) come respectively from (2.34) and (2.35).

The backward step of the algorithm evaluates the backward probability $\beta_t(i)$ of observing, from time $t+1$ to T, the sequence $o_{t+1} \ldots o_T$ given that the system is in state S_i at time t

$$\beta_t(i) = P(o_{t+1}, \ldots, o_T|q_t = S_i, M) \qquad \text{(A.13)}$$

Having defined $\beta_T(f) = 1$, where S_f is the final state, $\beta_t(i)$ can be evaluated as follows:

$$\begin{aligned}
\beta_t(i) &= P(o_{t+1}, \ldots, o_T|q_t = S_i, M) \\
&= \sum_{j=1}^{N} P(o_{t+1}, \ldots, o_T|q_t = S_i, q_{t+1} = S_j, M)P(q_{t+1} = S_j|q_t = S_i, M) \\
&= \sum_{j=1}^{N} P(o_{t+1}, \ldots, o_T|q_{t+1} = S_j, M)P(q_{t+1} = S_j|q_t = S_i, M) \\
&= \sum_{j=1}^{N} P(o_{t+1}|o_{t+2}, \ldots, o_T, q_{t+1} = S_j, M)P(o_{t+2}, \ldots, o_T|q_{t+1} = S_j, M) \\
&\quad P(q_{t+1} = S_j|q_t = S_i, M) \\
&= \sum_{j=1}^{N} P(o_{t+1}|q_{t+1} = S_j, M)P(o_{t+2}, \ldots, o_T|q_{t+1} = S_j, M) \\
&\quad P(q_{t+1} = S_j|q_t = S_i, M) \\
&= \sum_{j=1}^{N} \beta_{t+1}(j)a_{ij}f_j(o_{t+1}) \qquad \text{(A.14)}
\end{aligned}$$

where the same assumptions for the forward step were made. Finally, $P(O|M)$ can be evaluated as

$$\begin{aligned} P(O|M) &= P(o_1, \ldots, o_T|M) \\ &= \sum_{j=1}^{N} P(o_1, \ldots, o_T, q_t = S_j) \\ &= \sum_{j=1}^{N} P(o_1, \ldots, o_t, q_t = S_j|M) P(o_{t+1}, \ldots, o_T|q_t = S_j, o_1, \ldots, o_t, M) \\ &= \sum_{j=1}^{N} P(o_1, \ldots, o_t, q_t = S_j|M) P(o_{t+1}, \ldots, o_T|q_t = S_j, M) \\ &= \sum_{j=1}^{N} \alpha_t(j)\beta_t(j) \qquad \text{(A.15)} \end{aligned}$$

$$\text{(A.16)}$$

for any value of t.

A.3 The Viterbi algorithm

The Viterbi algorithm provides a way to compute the most likely states sequence for a HMM S^* defined as

$$S^* = \arg\max_S P(Q = S, O|M) = \arg\max_S P(Q = S|O, M) \qquad \text{(A.17)}$$

The Viterbi algorithm essentially implements of a minimum path search algorithm on a trellis associated to the HMM using a suitable distance function. In order to find the most probable path, we look for the path having the minimum value of $-\log P(Q, O)$. Assuming a Markovian first order process, we have

$$\begin{aligned} P(O, Q|M) &= P(Q|M)P(O|Q, M) \\ &= \prod_{t=1}^{T-1} P(q_{t+1}|q_t, M) \prod_{t=1}^{T-1} P(o_t|q_t, q_{t+1}) \qquad \text{(A.18)} \end{aligned}$$

$$\text{(A.19)}$$

Let the transition from state S_i to state S_j at time t be defined as $\xi_t(i, j)$, so that

$$P(\xi_t(i, j)) = a_{ij} \quad \forall t, 1 \leq t \leq T - 1 \qquad \text{(A.20)}$$

By defining a transition length as

$$\begin{aligned} \lambda(\xi_t(i, j)) &= -\log P(q_{t+1} = S_j|q_t = S_i, M) - \log P(o_t|\xi_t(i, j), M) \\ &= -\log P(q_{t+1} = S_j|q_t = S_i, M) - \log P(o_t|q_t = S_i) \qquad \text{(A.21)} \end{aligned}$$

the total length of a path corresponds to

$$-\log P(O,Q|M) = \sum_{t=1}^{T-1} \lambda(\xi_t(i,j)) \tag{A.22}$$

Since it is possible to establish a topological order among the trellis nodes, finding the shortest path can be done by recursively evaluating the distance $d_t(i)$ of any given state S_i from the initial node as

$$d_t(i) = \min_j d_{t-1}(j) + \lambda(\xi_{t-1}(j,i)) \quad 2 \leq t \leq T \tag{A.23}$$

having defined the distance of the first state as $d_1(1) = -\log P(o_1|q_1 = S_1)$ (when the start state emits a NULL symbol this is equivalent to setting $d_1(1) = 0$). From (A.21) and (A.23) this becomes

$$d_t(i) = \min_j d_{t-1}(j) - \log P(q_t = S_i|q_{t-1} = S_j, M) - \log P(o_t|q_t = S_i) \tag{A.24}$$

Since the logarithm is a monotone increasing function it is possible to directly evaluate the minimum path as follows. Let $\delta_t(i) = e^{-d_t(i)}$ and $\psi_t(i)$ be the index of state activated at time $t-1$ in the optimal state sequence. The algorithm detailed previously is equivalent to

1. initialize $\delta_1(1)$ and $\psi_1(i)$ as follows

$$\delta_1(1) = f_1(o_1) \tag{A.25}$$
$$\psi_1(i) = 0 \quad 1 \leq i \leq N \tag{A.26}$$

   since the starting state has no parent node

2. iteratively evaluate the δ value of each node as t increases as

$$\delta_t(i) = \max_j \left(\delta_{t-1}(j) a_{ji}\right) f_i(o_t) \tag{A.27}$$

   and update the corresponding backpointer $\psi_t(i)$ using

$$\psi_t(i) = \arg\max_j \left(\delta_{t-1}(j) a_{ji}\right) \tag{A.28}$$

The best sequence $\hat{Q} = \{\hat{q}_1, \hat{q}_2, \ldots, \hat{q}_T\}$ can be evaluated by setting $\hat{q}_T = S_f$ and following the backpointers up to the first frame

$$\hat{q}_t = S_{\psi_{t+1}(\hat{q}_{t+1})} \tag{A.29}$$

It can be shown that the probability of the best path is $\delta_T(f)$, where f is the index of the final state S_f.

A.4 HMM training through the EM algorithm

We start by observing that the likelihood of a sequence can be decomposed as

$$P(O|M) = \sum_{j=1}^{N} P(O, q_t = s_j) \tag{A.30}$$

for a fixed value of t. Recalling the forward–backward algorithm (Section 2.3.3), we can express the probability $P(O, q_t = s_j)$ as

$$P(O, q_t = s_j) = \alpha_t(j)\beta_t(j) \tag{A.31}$$

The likelihood A.30 can also be written as

$$P(O|M) = \sum_{s} P(O, Q|M) = \sum_{s_{i_1},\ldots,s_{i_T}} P(O, q_1 = s_{i_1}, \ldots, q_T = s_{i_T}) \tag{A.32}$$

where the summation is taken over all possible state sequences of length T, which, in turn, can be expressed as

$$P(O|M) = \sum_{s} P(O, s|M) = \sum_{s} \prod_{t=1}^{T} a_{i_{t-1},i_t} \sum_{k=1}^{m} w_{k,i_t} \mathcal{N}(o_t|\mu_{k,i_t}, \Sigma_{k,i_t}) \tag{A.33}$$

where a_{i_{t-1},i_t} is the probability of transition from state $s_{i_{t-1}}$ to state s_{i_t}, with a_{i_0,i_1} being the starting probability of state s_{i_1}, and w_{k,i_t}, μ_{k,i_t} and Σ_{k,i_t} are the parameters of the k–th Gaussian of the mixture associated to state s_{i_t}.

The next step consists in partitioning the likelihood further by choosing a particular sequence $z = z_1, \ldots, z_T$ of mixture densities (which are the realizations of random variables $Z = Z_1, \ldots, Z_T$), which results in

$$P(O, Q, Z|M) = \prod_{t=1}^{T} a_{i_{t-1},i_t} w_{z_t,i_t} \mathcal{N}(o_t|\mu_{z_t,i_t}, \Sigma_{z_t,i_t}) \tag{A.34}$$

Thus, the complete likelihood can now be expressed as

$$P(O|M) = \sum_{s} \sum_{z} P(O, S, Z|M) \tag{A.35}$$

Now we can apply the EM algorithm to A.35 by taking as hidden variables S and Z. We can iteratively maximize A.35 by evaluating the expectation

$$E_{Z,S}\left[P(O, Z, S|M_c) \log P(O, Z, S|M)\right] \tag{A.36}$$

and maximizing it with respect to M.

Expression A.36 can be rewritten as

$$\begin{aligned} f(A,B) &= E_{Z,S}\left[P(O,Z,S|M_c)\log P(O,Z,S|M)\right] \\ &= \sum_s\sum_z P(O,S,Z|M_c)\left[\sum_{t=1}^{T}\log a_{i_{t-1},i_t}+\sum_{t=1}^{T}\log w_{z_1,i_t}+\right. \\ &\quad \left.+\sum_{t=1}^{T}\log C(\Sigma_{z_t,i_t})\sum_{t=1}^{T}\frac{1}{2}(x_t,\mu_{z_t,i_t})^T\Sigma_{z_t,i_t}{}^{-1}(x_t,\mu_{z_t,i_t})\right] \end{aligned} \tag{A.37}$$

where A is the transition matrix $A_{ij}=a_{ij}$, B is the set of probability functions associated to each state $B=\{f_i(x)\}$ and $C(\Sigma)=(2\pi)^{D/2}|\Sigma_i|^{1/2}$

The weights a_{ij} can be re–estimated by taking the maximum of A.37 with respect to a_{ij} under the constraint

$$\sum_j a_{ij}=1 \quad \forall i \tag{A.38}$$

Applying the Lagrange multiplier technique we look for a saddle point of the Lagrangian function $f_{\mathcal{L}}$. Taking the derivative of the Lagrangian with respect to a_{ij} gives

$$\frac{\partial f_{\mathcal{L}}}{\partial a_{ij}}=\sum_s\sum_z P(O,S,Z|M_c)\sum_{t\in T_{ij}(s)}\frac{1}{a_{ij}}-\lambda_i \tag{A.39}$$

where $T_{ij}(s)=\{t : s_{t-1}=i, s_t=j\}$ and λ_i is the i–th Lagrange multiplier. By interchanging the order of the summations this becomes

$$\frac{\partial f_{\mathcal{L}}}{\partial a_{ij}}=\frac{1}{a_{ij}}\sum_{t=1}^{T}\sum_{s\in S_{ij}(t)}\sum_z P(O,S,Z|M_c)-\lambda_i=0 \tag{A.40}$$

where $S_{ij}(t)=\{s : q_{t-1}=s_i, q_t=s_j\}$ is the set of paths which visit state s_i at time $t-1$ and state s_j at time t. The t–th term of the sum (A.40) can be recognized as the probability $P(O,q_{t-1}=s_i,q_t=s_j|M_c)$, thus we can write that

$$a_{ij}=\frac{1}{\lambda_i}\sum_{t=1}^{T}P(O,q_{t-1}=s_i,q_t=s_j|M_c) \tag{A.41}$$

If we take the summation of (A.41) over all the values of j we have

$$\begin{aligned} \sum_{j=1}^{N}a_{ij} &= 1 \\ &= \frac{1}{\lambda_i}\sum_{j=1}^{N}\sum_{t=1}^{T}P(O,q_{t-1}=s_i,q_t=s_j|M_c) \\ &= \frac{1}{\lambda_i}\sum_{t=1}^{T}P(O,q_{t-1}=s_i|M_c) \end{aligned} \tag{A.42}$$

$$\tag{A.43}$$

which yields

$$\lambda_i = \sum_{t=1}^{T} P(O, q_{t-1} = s_i | M_c) = \sum_{t=1}^{T} \alpha_{t-1}^c(i)\beta_{t-1}^c(i) \tag{A.44}$$

so the value of a_{ij} which maximizes the expectation A.37 can be evaluated as

$$\begin{aligned} a_{ij} &= \frac{\sum_{t=1}^{T} P(O, q_{t-1} = s_i, q_t = s_j | M_c)}{\sum_{t=1}^{T} \alpha_{t-1}^c(i)\beta_{t-1}^c(i)} \\ &= \frac{\sum_{t=1}^{T} \alpha_{t-1}^c(i) a_{ij}^c \beta_t^c(j) \sum_k w_{k,j}^c \left[\mathcal{N}(o_t | \mu_{k,j}^c, \Sigma_{k,j}^c)\right]}{\sum_{t=1}^{T} \alpha_{t-1}^c(i)\beta_{t-1}^c(i)} \end{aligned} \tag{A.45}$$

where the superscript c denotes the parameters values of the current iteration of the EM algorithm.

The same approach can be used to derive an expression to update the weights $w_{i,j}$. This time we look for the maximum of A.36 with respect to $w_{i,j}$ under the constraints $\sum_{i=1}^{m} w_{i,j} = 1$. Using the Lagrangian approach and taking the derivative with respect to $w_{i,j}$ yields

$$\frac{\partial f_{\mathcal{L}}}{\partial w_{i,j}} = \sum_s \sum_z P(O, S, Z | M_c) \sum_{t \in T'_{ij}(s)} \frac{1}{w_{i,j}} - \lambda_j \tag{A.46}$$

where $T'_{ij}(s) = \{t : z_t = i, s_t = j\}$. If we interchange the order of the summation we have

$$\frac{\partial f_{\mathcal{L}}}{\partial w_{i,j}} = \frac{1}{w_{i,j}} \sum_{t=1}^{T} \sum_{s \in S_j(t)} \sum_{z \in Z_i(t)} P(O, S, Z | M_c) - \lambda_j \tag{A.47}$$

$$\tag{A.48}$$

where $S_j(t) = \{s : q_t = s_j\}$ and $Z_i(t) = \{z : z_t = i\}$, which results in

$$\frac{\partial f_{\mathcal{L}}}{\partial w_{i,j}} = \frac{1}{w_{i,j}} \sum_{t=1}^{T} P(O, q_t = s_j, z_t = i | M_c) - \lambda_j = 0 \tag{A.49}$$

and multiplying by $w_{i,j}$ and taking the summation over i we have

$$\lambda_j = \sum_{t=1}^{T} P(O, q_t = s_j | M_c) = \sum_{t=1}^{T} \alpha_t^c(j)\beta_t^c(j) \tag{A.50}$$

Now we can observe that

$$P(O, q_t = s_j, z_t = i) = \alpha_t^c(j)\beta_t^c(j) \frac{w_{i,j}^c \mathcal{N}(o_t | \mu_{i,j}^c, \Sigma_{i,j}^c)}{\sum_{k=1}^{m} w_{i,k}^c \mathcal{N}(o_t | \mu_{i,k}^c, \Sigma_{i,k}^c)} \tag{A.51}$$

thus we can finally write the expression of the updated weights as

$$w_{i,j} = \frac{1}{\sum_{t=1}^{T} \alpha_t^c(j)\beta_t^c(j)\frac{w_{i,j}^c \mathcal{N}(o_t|\mu_{i,j}^c,\Sigma_{i,j}^c)}{\sum_{k=1}^{m} w_{i,k}^c \mathcal{N}(o_t|\mu_{i,k}^c,\Sigma_{i,k}^c)}} \sum_{t=1}^{T} \alpha_t^c(j)\beta_t^c(j) \tag{A.52}$$

Finally, we have to evaluate the next estimate of the mean vectors and of the covariance matrices. Letting $\frac{\partial f}{\partial \mu_{i,j}}$ denote the vector of the derivatives of the expectation (A.36) with respect to the components of the j–th mean vector of the GMM associated to state s_i we have

$$\frac{\partial f}{\partial \mu_{i,j}} = \sum_s \sum_z P(O,S,Z|M_c) \sum_{t \in T' ij(s)} \Sigma_{i,j}^{-1}(o_t - \mu_{i,j}) = \mathbf{0} \tag{A.53}$$

where $T_{ij}^{'}$ has the same meaning as above. Interchanging once again the order of the summation and premultiplying by $\Sigma_{i,j}$ we obtain

$$\sum_{t=1}^{T} \sum_{s \in S_j(t)} \sum_{z \in Z_i(t)} P(O,S,Z|M_c) o_t = \mu_{i,j} \sum_{t=1}^{T} \sum_{s \in S_j(t)} \sum_{z \in Z_i(t)} P(O,S,Z|M_c) \tag{A.54}$$

which gives

$$\begin{aligned}
\mu_{i,j} &= \frac{\sum_{t=1}^{T} \sum_{s \in S_j(t)} \sum_{z \in Z_i(t)} P(O,S,Z|M_c) o_t}{\sum_{t=1}^{T} \sum_{s \in S_j(t)} \sum_{z \in Z_i(t)} P(O,S,Z|M_c)} \\
&= \frac{\sum_{t=1}^{T} P(O, q_t = j, z_t = i) o_t}{\sum_{t=1}^{T} P(O, q_t = j, z_t = i)} \\
&= \frac{\sum_{t=1}^{T} \alpha_t^c(j)\beta_t^c(j)\frac{w_{i,j}^c \mathcal{N}(o_t|\mu_{i,j}^c,\Sigma_{i,j}^c)}{\sum_{k=1}^{m} w_{i,k}^c \mathcal{N}(o_t|\mu_{i,k}^c,\Sigma_{i,k}^c)} o_t}{\sum_{t=1}^{T} \alpha_t^c(j)\beta_t^c(j)\frac{w_{i,j}^c \mathcal{N}(o_t|\mu_{i,j}^c,\Sigma_{i,j}^c)}{\sum_{k=1}^{m} w_{i,k}^c \mathcal{N}(o_t|\mu_{i,k}^c,\Sigma_{i,k}^c)}}
\end{aligned} \tag{A.55}$$

As concerns the covariance matrices, let $\frac{\partial f}{\partial \Sigma_{i,j}^{-1}}$ denote the $D \times D$ matrix whose (x,y)–th element is $\frac{\partial f}{\partial \Sigma_{i,j}^{-1}(x,y)}$, that is the derivative of f with respect to the (x,y)–th element of $\Sigma_{i,j}^{-1}$. By recalling that $\frac{\partial \log \|\Sigma^{-1}\|}{\partial \Sigma^{-1}} = \Sigma$, we have

$$\begin{aligned}
\frac{\partial f}{\partial \Sigma_{i,j}^{-1}} &= \sum_s \sum_z P(O,S,Z|M_c) \sum_{t \in T_{ij}^{'}(s)}^{T} \left[\frac{1}{2}\Sigma_{i,j} - \frac{1}{2}(x_t - \mu_{i,j})^T (x_t - \mu_{i,j})\right] \\
&= \sum_{t=1}^{T} \sum_{s \in S_j(t)} \sum_{z \in Z_i(t)} P(O,S,Z|M_c) \left[\frac{1}{2}\Sigma_{i,j} - \frac{1}{2}(x_t - \mu_{i,j})^T (x_t - \mu_{i,j})\right] \\
&= 0
\end{aligned} \tag{A.56}$$

which yields

$$\Sigma_{ij} = \frac{\sum_{t=1}^{T} \alpha_t^c(j)\beta_t^c(j)\frac{w_{i,j}^c \mathcal{N}(o_t|\mu_{i,j}^c,\Sigma_{i,j}^c)}{\sum_{k=1}^{m} w_{i,k}^c \mathcal{N}(o_t|\mu_{i,k}^c,\Sigma_{i,k}^c)}(x_t-\mu_{i,j})^T(x_t-\mu_{i,j})}{\sum_{t=1}^{T} \alpha_t^c(j)\beta_t^c(j)\frac{w_{i,j}^c \mathcal{N}(o_t|\mu_{i,j}^c,\Sigma_{i,j}^c)}{\sum_{k=1}^{m} w_{i,k}^c \mathcal{N}(o_t|\mu_{i,k}^c,\Sigma_{i,k}^c)}} \tag{A.57}$$

The parameter expression we have given are those identifying a critical point of the expectation. It can be proven [142, 143] that the critical point of the expectation is unique and that it corresponds to a maximum of the likelihood function.