KTH Computer Science
and Communication

A Speaker Verification System Under The Scope: Alize

Ett system för talarverifiering under luppen: Alize

TOMMIE GANNERT
TOMMIE@KTH.SE

Master's Thesis in Speech Technology at TMH
Advisor: Håkan Melin
Examiner: Björn Granström

March 24, 2007

Abstract

Title: A Speaker Verification System Under The Scope: Alize

Automatic Speaker Recognition is the collective name for problems that involve identifying a person or verifying the identity of a person using her voice. The French speaker recognition toolkit Alize from University of Avignon has been explored from the viewpoint of a toolkit already in use at KTH, called GIVES. The exploration has aimed to study the usability of the toolkit, its features and performance. Alize was found to be comparable with GIVES in terms of performance, and there are only a few differences in functionality between the two. The most notable difference between them is that, for now, the Alize system only uses one type of model (Gaussian Mixture Models), while GIVES can use several types of models (Hidden Markov Models, Gaussian Mixture Models, Artificial Neural Networks).

The paper describes the theoretical background of automatic speaker recognition, Alize as a starting point for the project, the experiments, and results. The results verify that Alize and GIVES are equal in performance, while the evaluation of Alize as a »toolkit» shows that it is not as easy to use as it could have been. It has very little documentation, and for some parts, the documentation does not reflect the current implementation. All-in-all Alize works, but to be used by anyone but the authors as a general system for speaker recognition, it needs to become more user friendly and become more structurally simple in its design.

Sammanfattning

Titel: Ett system för talarverifiering under luppen: Alize

Automatisk talarigenkänning är det övergripande namnet för problem som kan lösas genom att identifiera en person eller bekräfta en persons identitet genom hennes röst. Det franska talarverifieringssystemet Alize har undersökts med utgångspunkt från ett system som redan används på KTH, kallat GIVES. Målet med undersökningen har varit att undersöka systemets användbarhet, funktionalitet och prestanda. I fråga om prestanda är Alize och GIVES mycket lika och litet skiljer i deras funktionalitet. Den största skillnaden mellan dem är att Alize just nu bara använder en typ av modeller (Gaussian Mixture Models), medan GIVES kan använda flera typer (Hidden Markov Models, Gaussian Mixture Models, Artifical Neural Networks).

Rapporten beskriver den teoretiska bakgrunden inom automatisk talarigenkänning, Alize som utgångspunkt för projektet, experiment och resultat. Resultaten styrker att Alize och GIVES är mycket lika sett till prestanda. Utvärderingen av Alize som en »verktygslåda» pekar mot att Alize kunde vara enklare att arbeta med. Det finns lite dokumentation och delar av den stämmer inte med den nuvarande implementationen. Som helhet fungerar Alize, men för att fungera som ett generellt system för talarigenkänning behöver det göras mer användarvänligt och bli strukturellt enklare.

Contents

Chapter 1

Introduction

This is the final report of a project focused on exploring a software toolkit called Alize and its companion LIA_RAL[1] [1]. Alize is a toolkit for automatic speaker recognition. As such, it contains algorithms which can identify a person from her voice. The LIA_RAL package uses Alize and it was the interface used during the project. Both packages are currently being developed by the *Laboratoire Informatique d'Avignon* (Computer Science Laboratory at the University of Avignon).

The paper begins with a short introduction to the mathematical statistics used in Alize, and an introduction to the problem domain of automatic speaker recognition, then continues with a brief exploration of the Alize and LIA_RAL implementations.

Using these building blocks, a working setup of Alize is then introduced, with the aim of comparing Alize and GIVES in terms of performance. GIVES (General Identity Verification System) is another system for automatic speaker recognition, developed at KTH. This second part also includes a short description of the databases used during the project. Results from running the system is presented before conclusions and comments.

Frederic Wils was the author of the initial version of Alize. Alize is now maintained together with Jean-Francois Bonastre, the author of LIA_RAL. The toolkits are presented in many papers, most notably [2] and [3].

GIVES was written primarily by Håkan Melin, with additional contributions by various others [4]. In particular, the GMM (Gaussian Mixture Model) part used during this project was written by Daniel Neiberg. Also, two major efforts to create Swedish speaker recognition databases resulted in Gandalf and PER, both of which are attributed to Håkan Melin.

1.1 Project Statement

This project was proposed by Håkan Melin on the website of the Department of Speech, Music and Hearing (Sw. »Tal, musik och hörsel», or TMH). Alize had been presented on various occasions by its authors, and the question was how this relatively new toolkit performed compared to an already existing system on KTH.

The focus of the project was to build a working speaker recognition system using the tools found in Alize and LIA_RAL, and compare the results and algorithms to those of GIVES. Some specific goals were defined in the project prospect, handed in during the first month of the project. The goals were

[1] Despite major efforts to reveal the meaning of »RAL» in »LIA_RAL», none has been found. The predecessor to Alize/LIA_RAL was named »AMIRAL», and is described as »A Block-Segmental Multirecognizer Architecture for Automatic Speaker Recognition». Hence, it would be logical to assume »RAL» meaning »Recognizer Architecture Library», or similar.

chosen with the available time and previous work in mind:

- Compare the results of an Alize setup with that of Melin's GIVES setup.
- Evaluate the Alize system on some of the databases available at the department (which GIVES had been evaluated with).
- Describe Alize and LIA_RAL in relation to GIVES.
- Identify what parts of Alize perform better than the equivalent GIVES parts.

1.2 The Art of Automatic Speaker Recognition

In general, speech contains large amounts of information, including gender, feelings, a message, an identity, and physical health. It is often easy for a human to hear these »dimensions» of information independently of each other. A 90-year old woman will »sound like an old woman», and a tired man will »sound like a tired man», despite they may have the same message. This extra information, which is not really needed for the transmission of the message, is used for estimating the credibility, the relevance and many other properties of the message.

When moving the analysis from the ear and brain to microphones and computers, these dimensions are separated early. Two fields of research have received extra attention in later years, much due to the advances in computational speed: Voice recognition and speaker recognition. Voice recognition is the art of identifying *what* a person is saying. This identification should, of course, be tolerant to dialects, illness, and aging. The important information is the *message* being delivered. If two people say the same sentence, the computer should deliver the same answer.

Speaker recognition, on the other hand, is the art of identifying *who* is speaking. What is being said is not of importance, and is discarded. In this area of research, dialects are good since it helps distinguish between people. However, as in the case of voice recognition, illness, aging and temporal environmental noise can make it harder for the computer to deliver the identity. For this reason, systems must be constructed in a robust fashion.

There are two different approaches to speaker recognition; text-dependent, and text-independent. A text-dependent system requires that the text spoken when using the system must be the same as, or a combination of, the text used during training. For simple authentication, where the user starts by stating her identity, and uses a simple pass-phrase or code, a text-dependent system is enough. However, other situations require other solutions. For instance, when tracking the identity of speakers in a conference room, it is often necessary to use a text-independent system. The area of most current research is the more generic text-independent type, where no assumptions can be made of what the text contains.

Over the years, researchers have also divided the speaker recognition problem into two sets describing two different questions to be answered:

Speaker Verification where the input is an audio segment and a speaker model. The question to answer is »Did the speaker utter this segment?» and the output is »yes» or »no».

Speaker Identification where the input is an audio segment and any number of speaker models. The question to answer is »Who uttered this segment?» and the output is an identifier describing which model best fitted the segment.

There is, at least theoretically, a big difference in computational complexity between the two classes. In the verification case, there is no requirement to check other

speaker models beside the input model. In practical applications, however, simply looking at one model will not give enough discrimination between models for the decision, since the system would not know anything about speech by looking at this one model alone. In other words, it would not be possible for the system to classify the audio segment if no classes have been defined. Defining classes can be accomplished either by using a pre-calculated composite of models (with, or without the given speaker model), or by running tests against other (so called *impostor*) models.

Speaker identification requires the algorithm to check a number of models to find the one most likely match. This is usually done by some kind of scoring, for instance by applying the audio segment at hand on all speaker models and getting an absolute score. The output of the speaker identification would be, for instance, the model with the highest score.

Since the LIA_RAL toolkit is targeted at Automatic Speaker Verification (ASV), this paper will not delve deeper into speaker identification.

1.3 The Science of Automatic Speaker Verification

During the last ten years, two implementation strategies have been in focus for research:

Hidden Markov Model (HMM) is the oldest approach and models each speaker as a Markov model.

Gaussian Mixture Model (GMM) where a sum of Gaussian probability distributions is used to model each speaker.

The Hidden Markov Model approach is often used on the phoneme level, where one HMM is used to model a single phoneme with a fixed set of states. In essence, it is a state machine using the input audio frames to determine the next state.

According to Auckenthaler et al. [5], the GMM approach is more efficient than phoneme-based HMMs on text-independent speech. LIA_RAL currently only uses GMM, which is what this paper will discuss. Other techniques, like artificial neural networks and support vector machines have also been used in various research projects.

1.3.1 Gaussian Mixture Models in Theory

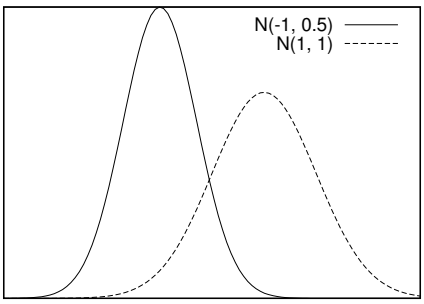
The GMM approach to speaker recognition is purely statistical. Given a number of speaker audio features (transformed audio samples), a model is created. The creation of a model includes deciding upon fixed model parameters (such as the number of terms, defined later in this section) and *training* the model on speech data. Other names for Gaussian Mixture Models include »Weighted Normal Distribution Sums», and »Radial Basis Function Approximations».

Two example Gaussian *probability density functions* (in one dimension) can be seen in Figure 1.1(a). Mathematically they are called $f_N(x, \mu, \sigma^2)$. When $\mu = 0$ and $\sigma = 1$, it is referred to as a *Standard Gaussian* density. The term *normal distribution* is a more common name for the Gaussian distribution. However, since GMM contains the word »Gaussian», this paper will use the name »Gaussian distribution».

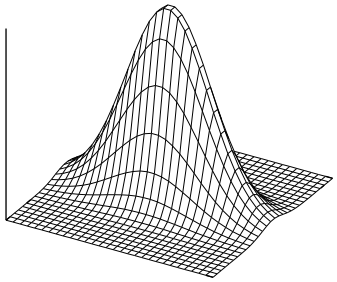
The parameter μ is the mean of the distribution, and σ^2 is its variance. Given these parameters and a stochastic variable X, $X \sim N(\mu, \sigma^2)$ is equivalent to

$$p(x) = \lim_{\delta \to 0} \int_{x-\delta}^{x+\delta} f_N(z, \mu, \sigma^2)\, dz$$

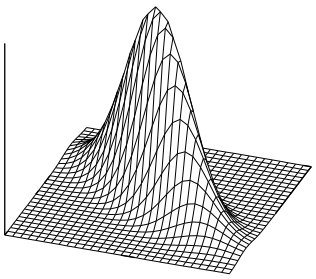
if x is an outcome of X. In essence, $p(x)$ is the probability of seeing an infinitesimal interval around x in an infinite stream of outcomes of X. In this paper, $p(x)$ will be used to denote »the probability of seeing x». This, however, implies »seeing an infinites-



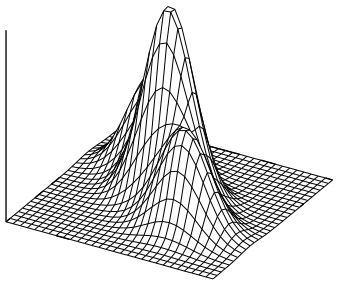
(a) Two distributions in $\mathbb{R}$.



(b) An independent distribution in $\mathbb{R}^2$.



(c) A dependent (rotated) distribution in $\mathbb{R}^2$.



(d) A mixture with three components, approximating the dependent distribution in Figure (c).

Figure 1.1. Three different Gauss distributions and one mixture.

imal hypervolume around x», with multidimensional x.

A Gaussian distribution can also be extended to $\mathbb{R}^n$ (known as the *multi-variate* Gaussian distribution), where

$$p(\mathbf{x}) = \lim_{\delta \to 0} \int_{x_n-\delta}^{x_n+\delta} \cdots \int_{x_1-\delta}^{x_1+\delta} f_N(\mathbf{z}, \mu, \Sigma^2)\, dz$$

is the probability of encountering an infinitesimal hypervolume around the vector $\mathbf{x}$. A two-dimensional distribution can be seen in Figure 1.1(b). Note that the mean is now a vector, and the variances are described using the $n \times n$ *covariance matrix* Σ^2. The use of a full covariance matrix allows the distribution to be rotated around the mean, like in Figure 1.1(c). Restricting Σ to a diagonal matrix[2] removes this possibility and makes calculations both easier and faster. Alize is capable of both full matrices (dependent dimensions), and diagonal matrices (independent dimensions). However, as we shall see, full matrices are rarely used with GMMs.

A mixture is a (finite) sum of N distributions, or

$$p_M(\mathbf{x}) = \sum_i^N w_i p_i(\mathbf{x})$$

where p_i are individual distributions, and in the case of GMMs, Gaussian distributions. The weights w_i determine how much influence each distribution has. Note that since $0 \leq p_M, p_i \leq 1$, the sum of the weights must

[2]Thus renaming it a *variance matrix*.

be 1, or p_M will not be a probability. It is easily seen that a distribution with a large variance and a large weight will have a great influence on the mixture, illustrated in Figure 1.2. Figure 1.1(d) shows an approximation of Figure 1.1(c) using a GMM with three independent (diagonal variance matrix) distributions.

The application of GMMs to speaker verification requires at least two algorithms. First, some way of determining the means, variances and weights is needed, to make the GMM model a speaker's voice characteristics. Second, a method of »evaluating» audio segments using the model must be found, to verify the identity of the claimed speaker. Since the output of the GMM is a single probability, it has to be transformed into a »yes» or »no». This is most easily accomplished by thresholding the $p_M(\mathbf{x})$. This raises two related questions:

1. How are the parameters of the mixture determined?
2. How is the threshold determined?

Assuming diagonal variance matrix, each mixture component has three parameters: The mean μ_i, the variance σ_i^2, and the weight w_i. In a mixture of N distributions, these $3N$ parameters are usually trained using the Expectation Maximization (EM) algorithm, described later in this chapter.

Thresholding can be thought of as classifying a real value in one of two classes. In cases of probabilities this is often done by using a *less-than* relation. The threshold can be either floating or fixed. A floating threshold requires the evaluation of audio segments using more than one speaker model and compares the output probabilities. This is a slow approach, however, and it is avoided if possible as it requires multiple evaluations of the same audio segment.

A fixed threshold is determined during training and only requires evaluation of one or two models during deployment. However, this method requires the models to be robust against noise, audio volume and other variations which could bias the probability calculations. The bias could make impostors likely to succeed for some models, while making it nearly impossible for even the true speaker to succeed on others. Some kind of *normalization* is needed to ensure the audio quality does not affect the results. Normalization and thresholding will be discussed further in Sections 2.4 and 2.8, respectively.

1.3.2 Likelihood and A Posteriori Probability

Two types of probabilities are used in the training and evaluation of GMMs, representing different properties of the data. The first type is the *likelihood*, $p(x|H_M)$, or the probability that the data, x, was generated under hypothesis H_M. The hypothesis being that the claimed speaker is the true speaker, and assuming that the speaker in question is represented by model M. Given the data from a microphone and a GMM, it is trivial to calculate this probability. In itself, though, this likelihood is of minor importance in speaker recognition, as the interesting problem is the reverse: $p(H_M|x)$, or the *a posteriori probability*. This is the probability that the claimed identity is the true identity given a speech segment. Fortunately, Bayes' theorem defines a relationship between the likelihood and the *a posteriori* probability, namely

$$p(H_M|x) = \frac{p(x|H_M)p(H_M)}{p(x)}$$

where two other probabilities are introduced.

The first one, $p(H_M)$ is the *a priori* probability that the claimed identity is the true identity. This means the system should be biased with information on which identities users most often uses, i.e. the system will tend to give frequent users a higher probability than infrequent users. Since most systems are designed for a large speaker database, it is usually assumed that this probability is equal for all speakers, thus merely introducing a scaling factor. Hence, $p(H_M)$ is often disregarded.

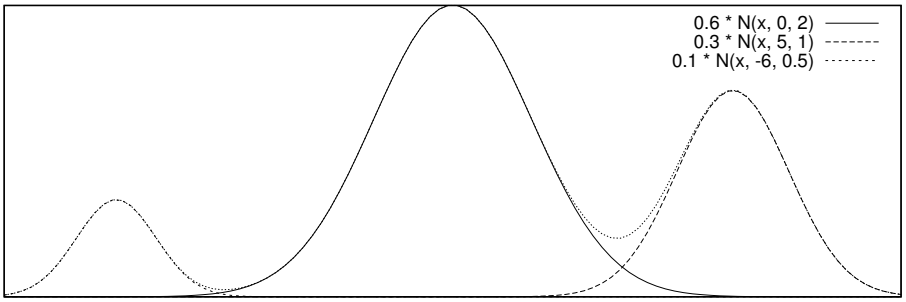


Figure 1.2. A Gaussian Mixture Model in $\mathbb{R}$ with three terms.

The second probability, $p(x)$, is the likelihood of seeing feature x at all. Thus, a very common feature will have less impact than a rare one. Most often an approximation to $p(x)$, discussed in Section 2.6, is used.

1.3.3 Expectation Maximization

As declared earlier, the most common technique for training the models is using an Expectation Maximization (EM) algorithm [6]. The problem of training involves using a limited set of samples to determine (at least local) optima for parameters of a model. In the case of speaker recognition the samples are usually segments of audio where the speaker utters various sentences. Training of the model is accomplished by iterating over the segments and finding better and better parameters for the model (weights, means and variances). By itself, the EM-algorithm is only a pseudo-algorithm and to be a complete algorithm, the object of maximization must also be given. In the area of speaker recognition it is usually either a likelihood or an *a posteriori* probability. Thus, in statistical terms, the EM-algorithm is one way of finding maximum likelihood (ML) or maximum *a posteriori* probability (MAP) estimates to data under an hypothesis.

For the rest of this paper, θ will denote model parameters, θ^* denotes optimal parameters and $\theta^{'}$ is used to denote the new value of θ after a single optimization iteration. Thus, the EM-algorithm iteratively finds $\theta^{'}$ which maximizes the probability of seeing the observed data, given the current parameters, θ:

$$\theta^{'} = \underset{\Theta}{\operatorname{argmax}}\, E(p(\mathbf{x}, Y|\Theta)|\mathbf{x}, \theta)$$

where $\mathbf{x}$ is the available data and Y is a stochastic variable denoting some unknown (hidden) data. The expected value is usually called $Q(\theta)$ and the calculation of this value is the first part of each iteration.

1.3.4 Expectation Maximization on GMMs

For the purpose of this paper, both MAP and ML variants of the EM-algorithm for GMMs are needed. However, the derivation of the equations is very intricate, and the reader is referred to Bilmes [7] for the ML-version of the algorithm, and to the references provided by Reynolds et al. [8] for derivations of the MAP-version.

Calculating the E- and M-steps for Gaussian mixtures (ML) results in three equations for updating the mixture parameters, $\theta = (w_1, \ldots, w_n, \mu_1, \ldots, \mu_n, \sigma_1, \ldots, \sigma_n)$ [7],

$$\begin{aligned} w_i^{'} &= \frac{1}{n}\sum_j^n p(i|x_j, \theta) \\ \mu_i^{'} &= \frac{\sum_j^n x_j p(i|x_j, \theta)}{\sum_j^n p(i|x_j, \theta)} \\ \sigma_i^{'2} &= \frac{\sum_j^n (x_j - \mu_i^{'})^2 p(i|x_j, \theta)}{\sum_j^n p(i|x_j, \theta)} \end{aligned}$$

where w_i is the weight for the i:th distribution and $p(i|x_j, \theta_i)$ is the likelihood that x_j

was generated by distribution i, and is usually the weighted average

$$p(i|\mathbf{x}_j, \theta) = \frac{w_i p_i(\mathbf{x}_t|\theta_i)}{\sum_k^M w_k p_k(\mathbf{x}_t|, \theta_k)}$$

The $(\cdot)^2$ operator is defined as $(\cdot)(\cdot)^{\mathrm{T}}$ for vectors.

In the case of MAP-training, the equations become [8]

$$\begin{aligned} w_i^{'} &= \frac{r^w + N_i}{n + N r^w} \\ \mu_i^{'} &= \alpha_i^{\mu} E_i(\mathbf{x}) + (1 - \alpha_i^{\mu})\mu_i \\ \sigma_i^{2'} &= \alpha_i^{\sigma} E_i(\mathbf{x}^2) + \\ &\quad +(1 - \alpha_i^{\sigma})(\sigma_i^2 + \mu_i^2) - \mu_i^{'2} \end{aligned}$$

where

$$\begin{aligned} N_i &= \sum_j^n p(i|\mathbf{x}_j, \theta) \\ E_i(\mathbf{x}) &= \frac{1}{N_i} \sum_j^n p(i|\mathbf{x}_j, \theta)\mathbf{x}_j \\ E_i(\mathbf{x}^2) &= \frac{1}{N_i} \sum_j^n p(i|\mathbf{x}_j, \theta)\mathbf{x}_j^2 \end{aligned}$$

with N as the number of mixture distributions and r^w and α are constants described in Section 2.5.

The algorithm is straight forward: Simply repeat the three equations under a convergence criterion, or a given number of times, over the same sample set. Care must be taken not to over-train the model, i.e. adapt it too tightly to the training data. This would otherwise remove the generalization property of the model, the reason to create a model in the first place.

Chapter 2

The Method of Speaker Verification

This chapter will give the reader a brief introduction to the steps of performing automatic speaker recognition. It is assumed the reader is familiar with the basic statistical concepts introduced in the previous chapter.

During speaker verification using GMMs, a few steps are common in nearly all systems. They include

Audio sampling usually accomplished by means of a microphone and an AD-converter. The audio waveforms are converted into discrete data called *samples*.

Labeling where training data is manually labeled with the speaker identity.

Feature Extraction where the audio is transformed from time domain to a better representation of voice characteristics.

Energy Filtering where silence and noise is filtered out.

Normalization where feature vectors are normalized based on energy content.

Training the speaker models so they are able to represent one speaker each.

Testing (or evaluating) unknown audio samples using the trained models.

Score normalization where testing scores are compared and normalized in order to avoid environmental differences between training and evaluation.

Computing the decision on whether the claimed speaker is the true speaker or an impostor.

The »Labeling» step is needed during training, since the GMM approach uses a supervised training style, where the speaker is known for each training segment. This could be semi-automatic in that the speech segments could be fed into some automatic classifier, thus only requiring the operator to identify one segment per speaker instead of all of them. See Figures 2.1 and 2.2.

2.1 Audio Sampling

Most ASV (Automatic Speaker Verification) systems use 8 bits and around 8 kHz sampling frequency for their input. The largest reason for this is probably that most research is performed on telephone lines, where the bandwidth is even more limited, so sampling at more than 8 kHz would give minor improvements. However, as noted in Grassi et al. [9], some systems would perform better using a higher sampling rate. This indicates that better performance still can be achieved with higher quality of data. Data stored in *A-law* format uses a non-linear scale, and 8 bits can be used to represent a 12-bit linear dynamic range, with acceptable impact on detail resolution. To accommodate for

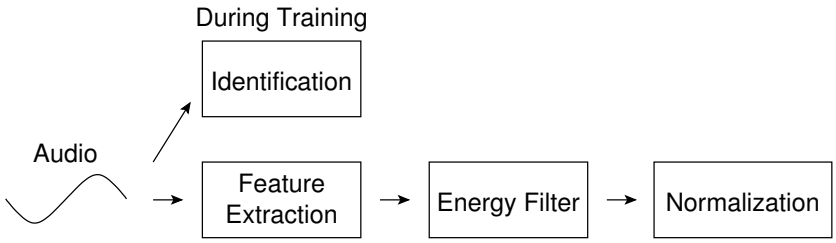


Figure 2.1. The first steps of transforming audio data.

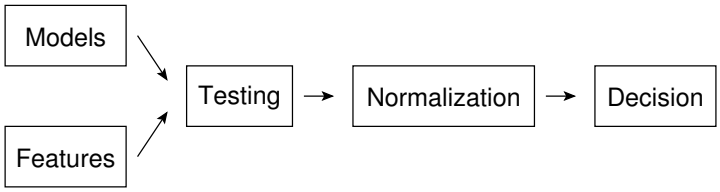


Figure 2.2. The procedure followed during evaluation of unknown audio.

the higher dynamic range, higher values are more widely spaced than are the lower values. This results in the preserving of both fine details and a large dynamic range. It is a common representation for speech.

2.2 Feature Extraction

The plain audio samples retrieved in the previous step are not directly suitable for speaker recognition. Environmental noise is the most obvious problem which totally changes the audio waveform. Other things, like the speaker having a cold, usually adds noise to the speech, but rarely changes the entire spectrum. A more robust representation is needed and a variation on Fourier transforms, called Cepstrum coefficients [10] currently dominate the field. It is defined to be $\mathrm{FT}\log\mathrm{FT}\ f$, where f is a windowed frame of audio. The length of the frame is usually in the order of 10–30 ms. Normally, the extraction is further improved by using the *mel scale* (from the word »melody»[1]), a frequency scale thought to overcome the difference between the physical frequency and the perceived pitch.

The result from a Cepstrum transform is *Cepstral Coefficients* [11, p. 162], and in the case of a Mel scale, they are called Mel-Frequency Cepstral Coefficients, or *MFCC*. The low coefficients describe the relationship between formants. The higher describe the source (glottis) [11, p. 163]. A selection of around 10 values are taken from the lower coefficients. They can be used together with other audio information, like the fundamental frequency to form *features*.

In summary, the audio samples are divided into frames. Each frame is transformed into »Cepstrum domain» to obtain simple key figures used as a representation of characteristics of the speech signal.

2.3 Energy Filtering

In the real world, it is often not possible to assume each feature file to be packed with speech data. Many times, there are pauses, hesitation and environmental noise that causes the speech data to be blurred. To avoid training and testing on feature vectors that are not speech, most systems do energy detection. This detection segments the file into one or more segments, each labeled with *speech* or *not speech*. During all subsequent steps, the non-speech segments are ignored.

[1] According to `http://en.wikipedia.org/wiki/Mel_scale`.

2.4 Feature Normalization

The effects of the recording environment can be devastating to the results of speaker recognition. To reduce the negative effects of having different characteristics from the microphone, the transmission channel and such, the features are normalized for mean and variance. A simple algorithm to do this is to calculate the mean, μ, and variance, σ^2, of feature vectors in the entire file. This is under the assumption that the environmental characteristics are constant during the entire recording. Alize also has the option of normalizing individual segments of each file. Each feature vector is then normalized using

$$x' = \frac{x - \mu}{\sigma}$$

Some systems only normalize on the mean, μ. When using Cepstral coefficients, this is called *Cepstral Mean Subtraction (CMS)*. Other systems (like the LIA_RAL) normalize on both mean and variance.

2.5 Background Model and Training

One of the basic ideas of an ASV system is that, given a few training samples for a client, it should be able to classify samples in general. This means the training samples has to be transformed into a more general model. It is called training or adaptation to training samples. For simple tasks, it can be as simple as just storing the training data and do a pattern matching. However, for the purpose of this project, a GMM is used.

Generally, it would be possible to generate one model per speaker by initializing it with random values and then train it into a usable model for the speaker. However, according to recent research [8], this leads to problems with under-training. If all models were generated independently, some distributions of the mixture will probably not be trained (due to the properties of the common training algorithm used). This leads to models with stochastic properties where a previously unseen feature vector may be classified according to a randomly generated distribution. Unfortunately, this opens the possibility of generating nearly random answers if the model is heavily under-trained.

To avoid this, a *background model* is often used. The background model is trained using the EM algorithm (usually on ML criterion) for 5 – 10 iterations, normally. Training data is collected from a special set of speakers with the only purpose of representing speech in general. This model is then used as a common starting point for training the various speaker models. By adapting from a single model, the element of chance still exists if the background model was not trained enough, but it will be evenly distributed among all of the speaker models.

The most widely used optimization criterion for speaker model training is to maximize the *a posteriori* probability that the claimed identity is the true identity given the data ($\max p(H_M|x,\theta)$). This is called *MAP* training, and is opposed to *ML* training where the criterion is to maximize the likelihood of the data being generated by the model ($\max p(x|H_M,\theta)$).

The approach of using a background model, trained with the ML criterion and later adapted to individual speakers using the MAP criterion is referred to as *GMM-UBM*, or GMM Universal Background Model, by Reynolds et al. [8]. In fact, in the GMM-UBM, a modified version of the EM algorithm is used for adapting to the speaker speech data. In this version, to update the parameters, θ, the simple expressions

$$\begin{aligned} P(i|\mathbf{x}_j) &= \frac{w_i p_i(\mathbf{x}_j)}{\sum_k^M w_k p_k(\mathbf{x}_j)} \\ N_i &= \sum_j^n P(i|\mathbf{x}_j) \\ E_i(\mathbf{x}) &= \frac{1}{N_i} \sum_j^n P(i,\mathbf{x}_j)\mathbf{x}_j \end{aligned}$$

$$E_i(\mathbf{x}^2) = \frac{1}{N_i}\sum_j^n P(i,\mathbf{x}_j)\mathbf{x}_j^2$$

are calculated. In essence, they are weights describing what distributions to assign the feature vector to. The parameter update is expressed in

$$\begin{aligned}
\lambda_i &= \frac{\alpha_i^w N_i}{n} + (1-\alpha_i^w)w_i \\
w_i^{'} &= \frac{\lambda_i}{\sum_i^N \lambda_i} \\
\mu_i^{'} &= \alpha_i^\mu E_i(\mathbf{x}) + (1-\alpha_i^\mu)\mu_i \\
\sigma_i^{2'} &= \alpha_i^\sigma E_i(\mathbf{x}^2) + \\
&\quad +(1-\alpha_i^\sigma)(\sigma_i^2+\mu_i^2) - \mu_i^{'2}
\end{aligned}$$

where N is the number of distributions in the mixture. The α_i parameters are calculated using a »relevance factor», r^ρ,

$$\alpha_i^\rho = \frac{N_i}{N_i + r^\rho}$$

This is the standard algorithm used to train speaker models in Alize, as described in Section 3.3.4.

The update equations for $\mu^{'}$ and $\sigma^{2'}$ are equal to the equations described in Section 1.3.4 on EM with a MAP criterion. The equation for $w^{'}$, however, is different and has been found to be better than the standard equation found in Section 1.3.4 [8].

It is interesting to note that adapting all of the parameters (w_i, μ_i, σ_i) is not always better than adapting only a subset. After training the background model, most systems only adapt the distribution means, or the means and weights. It has been found [5, p. 44] that adapting anything more than merely the means does not generally improve performance. Instead, the relatively sparse training data can result in over-training[2] the system. Still, the background model is adapted on all parameters. Since it is initialized with random parameter values, training only a subset of the parameters would leave random values in the model.

To avoid cases where the background model is over-trained, the variances are often clamped. Two types of clamping exist: *Variance flooring* and *variance ceiling*, clamping on the lower and upper bounds, respectively.

2.6 Testing

After training, the preparation of the system is complete. The system should now be able to verify speakers given models and feature vectors. Before defining the test algorithm, however, a decision has to be made as to what the output of the testing should be. The simplest output from a GMM is a single scalar per audio segment and model. More advanced scores can be vectors covering different aspects of the speech. *Likelihood ratio*, defined by the Neyman-Pearson Lemma [12], is commonly used to calculate the score. It is the optimum score in respect to minimizing false negatives (the risk of rejecting a truth-telling speaker). The ratio is expressed as

$$\begin{aligned}
\Lambda_M &= \frac{p(H_M|x,\theta)}{p(\neg H_M|x,\theta)} \\
\log\Lambda_M &= \log p(H_M|x,\theta) - \\
&\quad - \log p(\neg H_M|x,\theta)
\end{aligned}$$

where $p(\neg H_M|x,\theta)$ is the probability of H_M not being the right hypothesis given x (M is not the true speaker of x).

The latter is referred to as the *log likelihood ratio* (LLR). As most probability calculations in a speaker recognition system involve multiplication and division, the LLR is the type of score generally used. An approximation is often used for $p(\neg H_M|x)$, whereby instead of using a class of all models except M, the background model, W, is used. In fact, without this approximation, the LLR would require knowledge about every possible source of sound in the universe, a problematic task. However, for the sake of speaker verification, this implies that H_M

[2] Over-trained meaning »loses its generalization properties»

is part of $\neg H_M$ even though this is obviously false. This, however, eases computation as the background model was used during training:

$$\begin{aligned}\log \Lambda_M &= \log p(H_M|x,\theta) - \\ &\quad - \log p(H_W|x,\theta)\end{aligned}$$

Likewise, in Section 1.3.2 the *a posteriori* probability was introduced with the equation

$$p(H_M|x) = \frac{p(x|H_M)p(H_M)}{p(x)}$$

It was noted that $p(x)$ is rarely used in its exact form, as it would require many more samples than needed for the rest of the system. In fact, this too would require knowledge of the entire sound universe. Instead, by assuming that all speakers and channel characteristics that will use the system are modelled in the background model, the more simple likelihood $p(x|H_W)$ can be used. The assumption can be formalized as defining $p(H_W) = 1$, hence leading to $p(x) = p(x|H_W)p(H_W) = p(x|H_W)$.

2.7 Score Normalization

Even though the previously described steps (sampling, feature transformation, training and testing) are enough for a speaker recognition system, research has shown that the systems perform better with some score normalization applied. The purpose of this being to make the score as speaker-independent as possible. Allowing an absolute threshold when making the decision allows faster algorithms than using relative thresholds.

During recent years, many techniques have been proposed. Three of the most important are Z-norm, T-norm and H-norm, discussed by Auckenthaler et al. [5]. All three will be described shortly in the following sections.

2.7.1 Z-norm

In Z-norm (*Zero Normalization*), scores from the target models are normalized by using impostor speech. The idea is to find how likely impostor speech is to have come from the target model. By evaluating a set of impostor segments for each model (as part of the setup), the distribution of impostor scores can be found using the statistical equations:

$$\begin{aligned}\mu_I &= \frac{1}{n}\sum_i^n \log \Lambda_{I_i} \\ \sigma_I &= \sqrt{\frac{1}{n-1}\sum_i^n (\log \Lambda_{I_i} - \mu_I)^2}\end{aligned}$$

and the score can be normalized by post-filtering it with

$$\log \Lambda_{M,Z} = \frac{\log \Lambda_M - \mu_I}{\sigma_I}$$

where μ_I and σ_I are the statistical properties of the scores from the impostor utterance evaluated on the model M. The advantage of this algorithm is that it can be performed off-line, during training.

2.7.2 T-norm

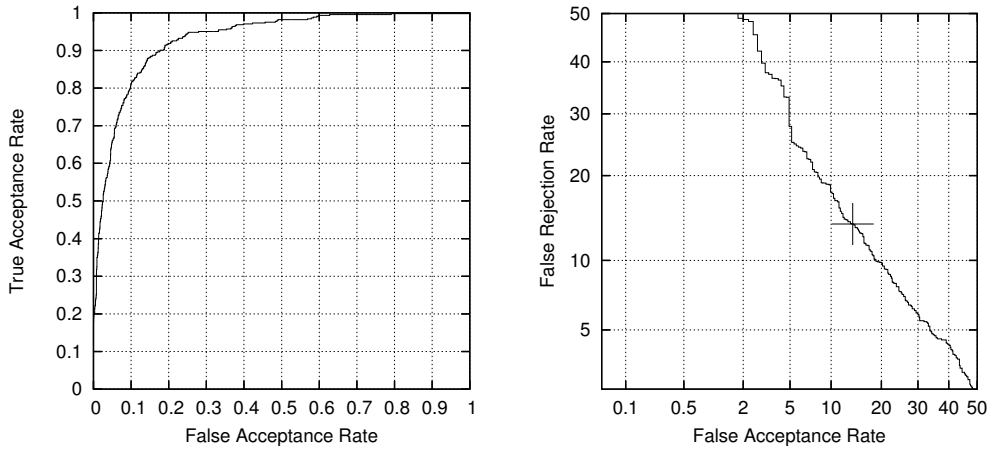
In T-norm (*Test Normalization*) [5], the idea is to compare each test utterance against possible impostor models. This results in a set of scores used for normalizing the score's mean and variance:

$$\log \Lambda_{M,T} = \frac{\log \Lambda_M - \mu_I}{\sigma_I}$$

where $\Lambda_{M,T}$ is the T-normed score for model M (for implied utterance x). Here, the μ_I and σ_I are the statistical properties of the scores from the test utterance evaluated on impostor models. Of course, this has to be done on-line for each test utterance and is thus considered a costly approach.

2.7.3 H-norm

The H-norm (*Handset Normalization*) [13] uses the same technique as Z-norm, except the segments are also divided into channel



(a) A Receiver Operating Characteristics

(b) A Detection Error Tradeoff with the point of EER.

Figure 2.3. Plots showing the graphical difference between the ROC plot and the DET plot.

characteristics before use. When used under conditions where channel characteristics are very dominant (like a database shared between cellular phones and land-line phones), its use can result in a large gain.

2.8 Making the Decision

The final step of the verification procedure is to actually decide which speaker has spoken the utterance. In the speaker verification case, the input contains a hypothesized model and the problem is to compare the output of the model to a threshold. The identity hypothesis could have, for example, been taken from a speech recognizer that can identify names from a database.

Instead of finding a threshold, researchers often use the DET-plot and EER scalar as measures of the performance of their verification system. A short introduction follows, as they will be used to describe the performance of Alize.

2.8.1 Types of Errors

In any speaker verification system, the output can be grouped into four types: True Acceptances, False Acceptances, True Rejections and False Rejections. The first type occurs when a true speaker is accepted, the second when an impostor is accepted. These are also called true or false *positives*, respectively. The third type occurs when an impostor is rejected, and the last type is the case where a true speaker is rejected.

These *rates* of error are used in various constellations to represent the performance of a speaker verification system. They are, however, rarely used alone and are often visualized in a diagram.

2.8.2 Detection Error Tradeoff

The DET was first defined by Martin et al. [14] in 1998[3]. It replaced the ROC (Receiver Operating Characteristic) which had been used for visualizing the relationship between the rates of *true positives* and *false positives*. The ROC-curve is not easily understood and is in general not a very good representation

[3]Though it had been used informally before that.

of the performance of a speaker verification system regarding ease of reading and the ability to draw conclusions from it. It is, however, easy to plot and has therefore been around since the 1940s.

In DET-plots, two things have changed. Instead of plotting a failure (false acceptance) against a success (true acceptance) the two possible types of error are plotted: False acceptances and false rejections. The second difference is the scale on the axes. While the ROC-curve uses a linear percentage scale on each axis, the DET-plot uses a transformation of the axes. The input is still a percentage, but by applying a transform, the plot can be made easier to read. Sample ROC- and DET-plots can be seen in Figure 2.3.

The score distribution for true speakers and impostors usually approximate two Gaussians, and it is therefore not surprising that a *Gaussian deviate scale*[4] results in a straight line in the DET-plot. This, in turn, makes it easy to spot deviations from the straight line (i.e., where one of the distributions are not Gaussians) and to find differences in variances (where the inclination of the line is not $-45°$). The use of a DET-plot with Gaussian deviate transformed axes are the *de facto* standard to visualize the performance of speaker verification systems.

2.8.3 Equal Error Rate

The Equal Error Rate (EER) is the standard scalar measure of the performance of a biometric verification system, be it based on speech or fingerprints. In essence, it is the point on the DET curve where the False Acceptance Rate and the False Rejection Rate are equal. For current GMM systems presented during the annual NIST SRE[5] evaluations, this is often in the area of 1%–5%.

Note that this does not necessarily mean that all systems with an EER of 5% allow 5% of the impostors in. For high security tasks, it may be required to set the threshold higher, thus allowing fewer impostors in, with the side effect of producing more false rejections. This is why the DET-plot is called Detection Error *Tradeoff* Plot, not just Detection Error Plot.

2.8.4 Defining the Threshold

The threshold is decided upon during training, and can, for instance, be set to the EER point (determined during training) as a starting point. In research, the thresholding step is often ignored, and instead the EER and DET-plot are used to show the overall system performance. During the studies of Alize, no threshold has been deployed as all results will be presented either as DET plots, or as EER values.

[4]See for instance `http://www.amex.com/servlet/AmexFnDictionary?pageid=display&titleid=4288`
[5]National Institute of Standards and Technology Speaker Recognition Evaluation

Chapter 3

Alize and LIA_RAL

The following chapter will focus on the two software packages to be evaluated, where both implementation and usage information will be described. Alize version 1.06 and LIA_RAL version 1.3 [1] have been used during the evaluation.

The relationships between, and borders of, Alize and the LIA_RAL are not easily defined. Primarily, Alize is the framework of mathematics, statistics, and I/O on which the LIA_RAL is built. The major categories of functionality that exist in Alize are:

- Command line and configuration file parser
- File I/O abstraction
- Feature file reading and writing
- Statistical functions (including histograms, mean, and variance)
- Gaussian distributions (full and diagonal (co-)variance matrices)
- Gaussian mixtures (including reading and writing to file)
- Feature file labeling and segmentation
- Line-based file I/O
- Vectors and matrices
- Managers for storing named mixtures, feature files and segments

Each of the modules will be discussed further in the next sections. Note that there are no algorithms specific to speaker (or speech) recognition in Alize. Rather, it can be viewed as a generic statistics framework highly focused on data structures and algorithms used in speech research. Key observations to support this is the lack of training algorithms and a specialization on Gaussian distributions.

3.1 Overview

LIA_RAL is the package containing all code specific to speaker recognition, although most code would also be useful in speech recognition. As the LIA_RAL package was the primary toolbox during the project, much of the following sections will be targeted to the use of Alize in the LIA_RAL.

Code in the LIA_RAL package can be divided into two categories. First, there is a more generic library (referred to as *SpkTools*), containing the training algorithms and some specialization of the statistical functions found in Alize, applied on feature vectors.

Second, it contains many small programs, designed to work in a pipeline methodology. There is essentially one program per recognition step, defined in Chapter 2. However, there is no skeleton to implement an entire speaker verification system. Connecting all programs of the pipeline re-

quires reading the documentation, example configuration files and the various unit-tests that exist for each program.

Both Alize and the LIA_RAL are implemented in C++, and Alize uses GNU Autotools[1] for platform independence. In this chapter, a more thorough description of the implementation of the packages will be given. It is assumed the reader has knowledge of basic object oriented development terminology.

3.2 The Alize Library

As stated earlier, Alize can be divided into smaller modules. These will be described from the viewpoint of how they are used by the LIA_RAL.

Note that some functionality of Alize and the LIA_RAL may have been left out from this paper, as not all functions were used during the experiments of this project.

3.2.1 Configuration Parser

The configuration parser and manager can handle Java-style property files[2] with a slightly stricter syntax. There is also initial support for an XML-based file format. A *configuration* in Alize is simply a *map* containing key–value pairs.

In addition to files, the configuration can also be given on the command line. A separate class handles parsing of the command line and inserts the parameters into a configuration. So, having a command line of

```
--nbTrainIt 3
```

is roughly equivalent to a configuration file of

```
nbTrainIt 3
```

This makes it very easy to define a static configuration file and still allow some dynamic parameters to be defined on the command line.

3.2.2 File I/O

Two classes handle reading from and writing to files. They are mostly a C++-ification of `FileStream`s found in Java and use the file API from C as the back-end. One notable difference from Java is that it has functions for swapping bytes, for use with different byte ordering, or *endianness*, depending on processor type. This is most useful when reading feature files created on machines with another processor type than the current.

3.2.3 Feature Files

For handling permanent storage of feature vectors, Alize supports some feature file formats (most notably HTK and SPro4[3]). It also supports the transparent concatenation of several feature files into a single stream. This is accomplished using *list files*, which are text files containing lines with names of other feature files. With this technique, many operations can be extended to support multiple input files during one run.

Another example of its usefulness is for speech databases with speaker data divided into classes. For instance, if training a model requires a single feature file, but each speaker has one recorded file for each digit 0–9, and one with their name, then instead of concatenating the feature files on disk, it can be done in memory with minimal loss in performance. This feature will be used in Section 4.3.

3.2.4 Statistical Functions

Alize can do simple, accumulating, statistics. Accumulating vectors and returning resulting mean and variance estimates is possible,

[1] The Autotools consists of Autoconf, Automake, and Libtool.

[2] See `http://java.sun.com/j2se/1.5.0/docs/api/java/util/Properties.html` on this

[3] HTK is found at `http://htk.eng.cam.ac.uk/`, and SPro is found at `http://www.irisa.fr/metiss/guig/spro/`.

as well as feeding the data into a histogram generator. Despite its lack of training algorithms, Alize does have support functions for the EM algorithm in general. All operations are performed in double precision.

3.2.5 Gaussian Distributions

Alize can handle Gaussians with both full covariance matrices (almost never used with mixtures, as it can be approximated with a group of diagonal variance matrix Gaussians), and with diagonal variance matrices.

3.2.6 Gaussian Mixtures

Although Alize is designed to support other distributions than Gaussians, only Gaussians are implemented and all code is targeted to Gaussians. This includes mixtures. There are functions to calculate basic values needed during EM MAP estimation, like variance matrices and mean vectors from feature vectors. All mixtures, like the distributions, are targeted to multivariate problems, where the input is a vector rather than a scalar.

Mixtures can be read from and written to disk, though a single distribution cannot.

3.2.7 Feature Segmentation

As noted in section 2.3, there is a need to label audio segments and to filter out segments based on labels. This functionality exists in Alize, with a module working in parallel with the feature file I/O.

3.2.8 Line-Based File I/O

Except for feature files and mixture files, most files have the structure of text files divided into lines and fields. One module of Alize is targeted to reading and writing line/field files. This includes list files, containing the names of feature files to be concatenated, and score files.

3.2.9 Vectors and Matrices

Only some basic vector and matrix algorithms are implemented. Alize does not implement a generic vector or matrix library.

3.2.10 Managers

A group of classes, called Servers in Alize, handle the storage of named entities in memory. The concept is more commonly referred to as *maps* or *managers* in other systems. All speaker models, for instance, are keyed on their speaker identity, and are stored in the Mixture Server. Features are cached in the Feature Server, and segment labels are stored in the Label Server. This makes it easy for the user to read definitions of speakers and feature files into memory as there is no need to organize the permanent storage of models manually.

3.3 The LIA_RAL Library

The main purpose of the LIA_RAL is to wrap all algorithms in programs which can be run from a shell script. Note that the LIA_RAL does not contain functionality for feature extraction. All input is assumed to be »usable» feature vectors. The authors of the LIA_RAL use the SPro programs to perform the extraction [3].

A common property of all LIA_RAL programs is that no files are both read and written. Input is read, transformed, and then output to other files. This is very useful when running the system with make-files.

3.3.1 Energy Filtering

The `EnergyDetector` program reads a list of feature files (or a single feature file) and segments it based on speech energy. The energy measure is assumed to be the first component of each feature vector (usually returned by the SPro programs). A threshold is calculated and subsequently used to discard feature vectors with lower energy than the threshold. In its simplest mode (called

MeanStd), the threshold is determined by training a GMM on the energy component and finding the *top distribution* (the distribution with the largest weight, w_i in Section 1.3.1). This distribution is used to compute the energy threshold as

$$\mathcal{T} = \mu_T - \alpha\sigma_T$$

where $\mathcal{T}$ is the threshold, (μ_T, σ_T) are the parameters for the the top distribution, and α is an empirical constant.

Another, more involved thresholding method called *Weight* is also implemented. It uses the weight of the top distribution as the percentage of frames to pick out. The actual threshold is then determined by building a histogram (with 100 bins) from the feature vectors of the file. The idea is to sum the bins vector counts from the top (highest value) until the requested percentage of frames have been searched. From this, the threshold is taken to be the higher threshold of the last bin visited. However, in the current implementation, the *Weight* method is incorrectly implemented and not used.

Input parameters include the thresholding method and α. The latter is used in both methods. As with most GMM training in the LIA_RAL, variance clamping (see Section 2.5) can be used to reduce the risk of over-training the (energy threshold) model.

3.3.2 Feature Normalization

The `NormFeat` program reads a list of feature files (or a single feature file) and normalizes them, either on segment basis (given label files), or per file. Two modes of operation can be chosen for normalization: Mean/variance normalization and Gaussianization. The first mode simply accumulates statistics of all feature vectors and normalizes using the expression in Section 2.4. The second mode performs feature warping [15] to transform the feature vector histogram onto a standard Gaussian distribution. The input parameter is mainly the mode selector.

3.3.3 Background Model

Training of the background model with background speakers is performed with `TrainWorld`. The input is a list of feature segments and the output is a single GMM. Initially, all distributions are set to the global variance and mean, and all weights are equal. Then the model is iteratively trained with all feature vectors using the EM algorithm with a ML criterion. For each iteration, the program can normalize the model using a special technique called *model warping* [2]. The most notable input parameters to `TrainWorld` are:

- The number of distributions. This figure will determine the number of GMM distributions for the rest of the pipeline.
- A feature selection ratio. Not all feature vectors need to be used during training. Feature selection is based on probability.
- The number of training iterations.

Since the purpose of this program is background training using large amounts of speech data, the problem of over-training is not of a major concern. However, code for variance clamping exist and is active in the experiments outlined in Chapter 5. If the speech data is highly incoherent, there is a risk of very high variance values, why variance ceiling should be employed.

3.3.4 Target Model

Once a background model has been created, the `TrainTarget` program is run to adapt the background model for each speaker (using a modified EM with a MAP criterion) to feature vectors. Input to the program is a list (called an index) with lines like

```
F1025 F1025/feature1
F1026 F1026/feature1 F1026/feature2
```

The first column designates the speaker identity, the remaining columns are feature file names. A prefix and a suffix can be set globally to allow for moving to different file formats and locations without having to update the indices.

Like in the `TrainWorld` program, not all feature vectors are necessarily used in the initial training iterations. A selection ratio can be set for using only a selection. The GMMs can be normalized using the model warping algorithm, just like during background model training.

Furthermore, the modified EM algorithm in use (see Section 2.5) can use one of two modes, called »MapOccDep» and »MapConst2», respectively. In the first mode, the α_i family of variables are calculated using a relevance factor, while in the second mode, α_i is fixed. Also, the second mode only allows adaptation based on means. Variances and weights are never trained.

During training, the iterations have been divided into two parts. The first part can make use of the feature vector selection ratio described previously. The second part uses all vectors. This functionality has not been explored in greater depth in this project.

When using T-norm, this program is also used to generate the impostor models. This is no different from generating the »true» speaker models, except the model features never contain speech feature vectors from the true speaker.

3.3.5 Testing

From a background model, the speaker models and a number of feature files, the `ComputeTest` program generates score files. The output score files contain one line per test (model and feature file, or segment, pair). The program is a generic probability evaluator and is also used with both Z-norm and T-norm to find scores. It uses the standard LLR measure for scores. One of the very few settings is the number of top distributions to use in the result. By pruning the mixture to, for instance, ten distributions, computational time can be reduced to be independent of mixture size. This is implemented by first calculating the LLK value for the world model, find the top distributions and then only evaluate the corresponding distributions in the target models. If only ten of 512 distributions are evaluated for each target model, performance can be greatly improved [8].

3.3.6 Score Normalization

When using a score normalization technique, like Z-norm or T-norm, the `ComputeTest` step produces two or three score files. One file is the true speaker test, and one or two contain various *impostor tests*. `ComputeNorm` then calculates the actual score normalization. The output is a single, normalized score file.

A restriction can be applied to only use a fixed number of impostor tests during normalization, the *cohort size*. For T-norm this is the number of impostor models, and for Z-norm it is the number of impostor feature vector segments or files.

3.3.7 Making the Decision

The final decision is made by the `Scoring` program, though `ComputeTest` also outputs a decision. It can be configured to always accept exactly one model per segment, for speaker identification tasks. The normal mode, however, is to input a scoring file and a threshold. One of several output modes must also be given, including »NIST», »ETF» and »MDTM». A last mode is also supported which performs a »light-version» of T-norm, excluding the model with the highest score. The NIST mode is in compliance with the NIST SRE 2004[4] evaluation,

[4]The National Institute of Standards and Technology Speech Recognition Evaluation, found at `http://www.nist.gov/speech/tests/spk/`.

ETF and MDTM are designed for ESTER 2005[5].

3.3.8 Common Parameters

A number of parameters are common to all LIA_RAL programs. These include settings for verbose output and debug output, but also prefix and suffix for various file types. All files are divided into five file types:

- Feature files
- Mixture files
- Label files
- List files
- Index files

Each type of file can be prefixed and suffixed separately. This allows for shorter references from list files and index files, and makes it easier to move to other file formats.

Alize does not automatically find the file format of a feature file, so a setting for the loading and saving types must be specified. The same is true for mixture files (which can be stored in a compact format, or as XML). Also, Alize cannot determine the endianness of files, and a configuration directive must be used for this.

All programs that manage mixtures have thresholds for log likelihoods. This threshold is used to clamp the LLK value in an interval, to avoid problems with division-by-zero.

3.4 The Sum of the Parts

Because the LIA_RAL toolkit is built on top of Alize, it is not possible to use only the LIA_RAL. On the other hand, since Alize in itself is too generic to be an ASV toolkit, the LIA_RAL is needed in order to evaluate the inner workings of Alize. The packages must be used together. Hence, evaluating Alize alone would not be useful for the purpose of ASV. The interesting ASV algorithms are located in the LIA_RAL.

Furthermore, as the LIA_RAL package consists of small programs rather than a library for use in another C++ application, the user interaction with Alize is small. In fact, nearly all user actions are wrapped in LIA_RAL structures, the only exception being the configuration file parser, which is found in Alize.

[5] A French Evaluation of Broadcast News Enriched Transcription Systems, listed at `http://www.elda.org/article140.html`

Chapter 4

Building a System

This chapter contains a description of the system developed and used during the evaluation of the Alize and LIA_RAL packages. It assumes the reader is familiar with the Alize and LIA_RAL toolkits from reading Chapter 3. For the remainder of the paper, the name »Alize» will be used to denote both packages, as they are never used separately, and the intended package should be evident from context.

4.1 Speech Databases

This section will guide the reader through the three speech databases used during this project. The databases are the foundation for proper research in speech. As they must be organized manually, building a database is probably the most time-consuming task in speech research. Since more complex partitionings and tests require more data, it is advantageous if the database can be easily extended.

The three databases are all found at the Department of Speech, Music and Hearing at KTH.

4.1.1 SpeechDat and Gandalf

The first database to be used was the SpeechDat FDB 1000 database. It was used for training background models during the initial stages of the project. FDB 1000 is a database containing 1,000 speakers, where all sessions are recorded using a land-line (fixed) telephone. Though this database is intended for speech recognition, rather than speaker recognition, it suits well for background model training. Of course, this assumes the target models will be trained with audio having roughly the same channel characteristics. Due to the fact that each speaker only has one session, there is a very limited set of data for each speaker. This, unfortunately, means that the database cannot be used for training and testing in ASV. More information on SpeechDat can be found in Elenius [16].

The Gandalf database was used for training and testing during the initial stages of the project. The database contains a total of 86 speakers, with 48 males and 38 females. The Gandalf database was created for speaker recognition research and has several sessions (up to 29) per speaker. Gandalf was used together with SpeechDat while developing the system. More information on Gandalf can be found in Melin [4, Sec. 6.2].

4.1.2 PER

The final database to be used was the PER database. It is a database of cellular phone, land-line phone, and wide-band microphone recordings. The wide-band microphone samples are collected at the gate leading into the TMH (Speech, music and hearing) offices. To support new research in ASV, all participants were divided into three groups: Back-

ground, target, and impostor. The target and impostor groups were not disjunct. For the background group, 51 male and 28 female speakers were recorded. A separate set of 54 speakers participated as target speakers (38 male and 16 female). Most of the target speakers also participated as impostor speakers together with some additional new speakers, for a total of 98 impostor speakers (76 male and 22 female). Most of the effort of this project was placed on experimenting on the gate recordings of PER.

PER has been divided into four conditions, depending on the recording condition. These are named

- G8, gate, wide-band microphone in hallway,
- LO, land-line telephone in office,
- MO, cellular telephone in office, and
- MH, cellular telephone in hallway.

Most focus in this project was on the *G8* condition, with additional tests run on *LO*. These are presumed to be of the best quality. More information on PER can be found in Melin [4, Sec. 6.3].

The difference between the *MO* and *MH* conditions is perhaps subtle, but theoretically significant. There may be influences by echoing and envelope in a hallway which do not exist in an office environment. To study these differences, Melin [4] included both conditions in his research.

4.1.3 File Segmentation

All three databases had label files already prepared by Håkan Melin using different automatic systems. In the case of SpeechDat and Gandalf, the files were generated by the use of an end-point analyzer, which used an energy measure together with zero-crossing rate [1] to find the speech contents. The zero-crossing rate detector was setup to include an entire utterance, including any inter-word silences. For most feature files, this results in one speech label per file. Some label files contained no speech label, which had to be corrected (automatically, once the problem was found) as Alize would abort if no feature vectors could be used from a single file.

For the PER database, more detailed label files were available. These files were created using a speech recognizer. The best hypotheses from the speech recognizer was compared to an expected utterance (name and digits) and the best hypothesis was selected. The time coordinates and IPA[2] information from the speech recognizer was recorded into the label file. The files, called *tri_bst_02*, are described in detail in [4, Sec. 5.6]. To create these files, a tri-phone based speech recognizer was used, and the best hypothesis was selected based on *a priori* knowledge about the utterance.

4.1.4 GIVES Software

GIVES is the counterpart to Alize and the LIA_RAL at TMH. In an attempt to both narrow the project down, and to be as close to GIVES as possible, it was decided to use the GIVES functionality for feature transformation. Since Alize does not have support for this, but GIVES does, it was a natural choice. This would give Alize the same starting point that GIVES had.

The only program used was `Aparam`, which reads a settings file and audio files and transforms them into feature vector files. The settings file can contain a variety of filters, and is modelled as a directed acyclic graph, where the different stages are named and uses input from earlier stages.

4.2 Assembling the Parts

A fundamental design decision was to make the entire system as modular as possible,

[1] Using an algorithm by B. T. Lowerre, found at `ftp://svr-ftp.eng.cam.ac.uk/pub/comp.speech/tools/ep.1.2.tar.gz`.

[2] International Phonetic Alphabet

since it was not known at the start of the project which databases and programs had to be used later. Another aspect was that since the interface to Alize consists of separate programs, a pipeline architecture seemed like the natural choice. This led to the decision to use the Unix software build utility, `make`, to build the system.

The project was split into different *make-files* for each Alize program to be easier to work with. A master make-file is responsible for connecting the pieces. Together, this includes everything from converting GIVES job descriptions into Alize format, to creating DET plots for the score files. During development, a goal was to minimize the number of recalculations needed when updating a parameter. Changing the list of speaker models used during testing, for instance, should not require remaking the world or speaker models, unless they do not exist or are out-of-date.

Because of the assumed big differences between male and female voice, all work was split to handle them separately. This also implies that no females were tested against male models and vice versa.

4.3 GIVES Job Descriptions and Corpus Labels

The first task at hand was to transform the job descriptions (background model training, speaker model training, and testing) used in GIVES to a format Alize could read. This proved to be a challenge, as the three databases all have somewhat different directory structures, thus requiring one transformation tool each to be written. The utilities to do the transform were implemented using Bash or Perl, depending on the task.

All job descriptions are under a make-file umbrella. They are fetched from the source location on the network, transformed and stored on local disk. One of the make-files is also responsible for building Alize list files for the Gandalf database. This had to be done because the Alize test files are »transposed» versions of GIVES job descriptions. This has the unfortunate side-effect that what is single feature files in GIVES has to be concatenated to suite Alize. Hence the usefulness of being able to create list files in Alize (described in Section 3.2.3).

An early decision was made as to how energy filtering of the feature files would be performed. Since all databases already had label files equivalent to energy information, there seemed to exist no reasons to use the Alize functions for energy filtering. This was in part due to project time constraints, and in part to make the setup as close to GIVES as possible.

Because of the small size and the static nature of label files, they were not under the supervision of a make-file. Instead, they were converted for the entire database at the same time. This made it possible to do the conversion before the job descriptions were available.

4.4 Feature Transformation

All databases have its audio data stored in the time domain, mostly A-law encoded with 8 kHz sampling frequency and 8-bit resolution (corresponding to 12 bits in linear dynamic range terms). These have to be converted into features before use. The GIVES program `AParam` was used for this. All files in the databases were transformed in one pass. The main reason to do this was that the database was located on the network, and it was not desirable to stream data across the network during testing. In relation to the other tasks in speaker verification, feature transformation is fast. Each of the three databases has approximately 2.5 GB of feature data in this setup.

4.4.1 Transformation

An illustration of the over-all transformation algorithm can be seen in Figure 4.1. The first stage is a band-pass filter bank, set to 24 filters in the range 300 – 3,400 Hz. The filters are equally spaced in the mel-scale and

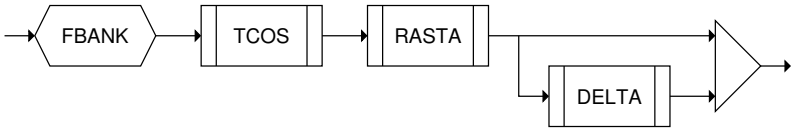


Figure 4.1. A schematic view of the feature transformation steps.

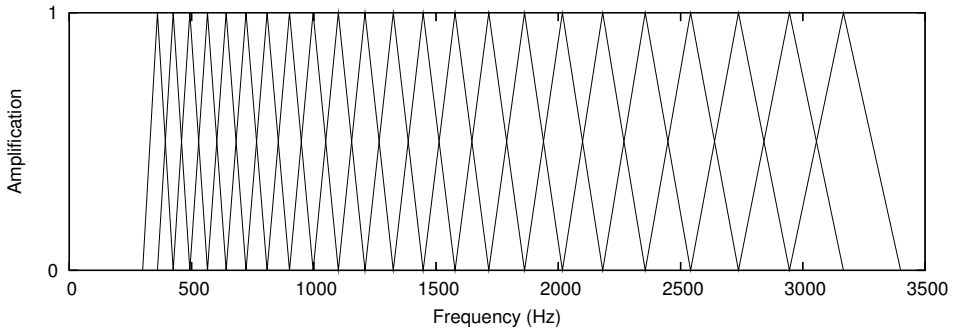


Figure 4.2. Equally spaced (in mel-scale) filters used in the FBANK during feature transformation.

are depicted in Figure 4.2. Output is in frequency domain. Then, a cosine transform is applied which reduces the 24 coefficients from the first stage to 12 Cepstrum coefficients. Finally a RASTA [17] filter is applied to reduce the effects of channel characteristics, like echoes and envelope. The output consists of 24 feature components, with the 12 components from the RASTA filter and their first-order differentials (discrete derivatives).

The entire feature transformation design was taken from previous GIVES experiments, and is designed specifically for text-independent GMM systems. All feature vectors have 24 coefficients used by the GMMs.

4.4.2 Normalization

To ensure that the features input to Alize were conditioned correctly, all feature files used were normalized using the normalizer found in Alize, `NormFeat`. In addition to preconditioning, this also served as a check to see that all feature files needed later could be read.

The `NormFeat` program is slow (in relation to `AParam`), thus an initial attempt to normalize all files was abandoned. Instead, one of the make-files generates one file per database containing all feature files that were referenced in any job description. Because of the large number of feature files, this was also run in a single pass outside the scope of make-files. This required some manual effort when using a new job description, but it helped speeding up normal operations in return.

4.5 Structure of the Make-files

When all feature files are in place, the rest of the steps are performed with make-files. The marginal time cost for trying a new configuration is therefore reduced to a minimum. The modules with their own make-file are

- List Generation
- World (Background) Training
- Target Training
- Impostor Training (duplicate of Target Training)
- Testing
- T-norm Testing (duplicate of Testing)

- Score Normalization
- Master

The List Generator has already been described in this chapter. The World Trainer takes an Alize configuration file and a list of feature files containing background speaker audio. The Target Training, in turn, uses the background model previously created, in addition to a configuration file and an index file of speaker/segment groups. The Testing module uses target models, the background model, a configuration file and an index file of segment/speaker groups. Next, the Score Normalizer uses data from Testing and, possibly, from the T-norm Testing to create an overall score file. Finally, the score file is rendered as a DET plot and an EER scalar is generated using the tools available at NIST's website and Gnuplot[3].

All of the different parts are run from the Master make-file, which also does some initial checks to see that all configuration files are in place before starting a job. The Master is also able to run a single job in parallel, with the females and males in different branches.

4.6 Run Logs

In order to be able to quickly analyze and otherwise use the results from a run, the key results were stored in a *run log*. The log used XML to be easily parseable by both humans and machines. Each log contains information on

- start time,
- end time,
- which host had run the job,
- module configuration names,
- the DET-plot points,
- EER values, and
- links to pre-rendered DET plots in EPS and PNG formats created using Gnuplot, for use in this paper and for previewing.

Additionally, links to the raw score files were saved.

Since most of the development was done on non-backuped space on a TMH server, it was important to save critical data, like test results and configurations. Subversion[4] was used for revision control, and after each run log was created, it was automatically added to the repository. In the event of a disk crash, this would at least preserve the results and make it possible to rerun the tests after the feature files had been regenerated.

[3] Available at `http://www.nist.gov/speech/tools/index.htm` and `http://www.gnuplot.info/` respectively.

[4] Found at `http://subversion.tigris.org/`.

Chapter 5

Experiments

In this chapter, the experimental setup used for evaluating the system described in the previous chapters is presented. Configuration files, experiment hypotheses and data sets will all be discussed in depth.

The single most important task during experimentation was to create a system which could compete with GIVES in terms of EER performance. Since both systems have been under active development the last years, it was expected that they would work equally well.

However, a few concepts in Alize lack a counterpart in GIVES. The most notable of these is the use of »model warping» (see Section 3.3.3), where every trained model can be *normalized* after each iteration of the training. Another interesting, albeit not unique, feature is the ability to change the MAP approximation algorithm used during training (see Section 3.3.4). These two features were used as the starting point for the experiments, together with the obvious studies of mixture sizes and the number of training iterations.

5.1 Data Sets

In this section, the data sets used in this project will be described. A *data set* is a subset of audio files and speakers from a specific database. The division into data sets can have two purposes. First, three sets (preferably disjunct) must be defined that are given to each of the background training, target training and testing. Despite that background training and target training could use the same audio files, this is often avoided when there is enough available data, to avoid over-training. The training data and the testing data must, of course, always be disjunct.

5.1.1 Development and Evaluation

Another often occurring division is into a development set and an evaluation set. These are depicted in Figure 5.1. Since the current research of speaker verification technology at the moment require much manual testing, tweaking, and analysis, most research is done iteratively:

1. Create the system
2. Define some system variables
3. Set these variables
4. Run the system using development data
5. Analyze, and refine the variables
6. Repeat from step four
7. Run the system using evaluation data
8. Use the evaluation output as a performance measure

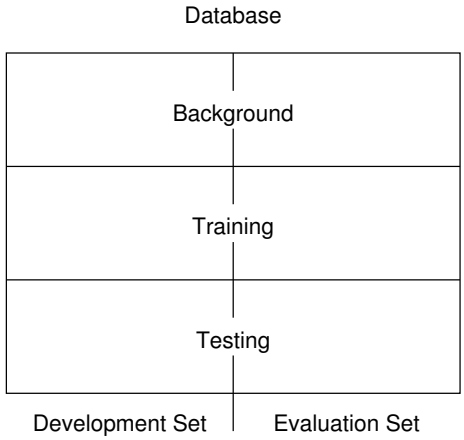


Figure 5.1. The division of a database into two separate sets.

If only development data were used, the models would be adapted to training data, and the *entire system* would be adapted to a local optimum for the testing data of the development set. This way, it may very well be that the system performance is due to an adaption of the parameters to the development data. Thus, the relevance of the experiment would be questionable.

The use of disjunct development sets and evaluation sets avoids this problem. Instead of using the output from the development set as a performance measure, a last run, on evaluation data, is made.

However, because the project was targeted at the evaluation of Alize, rather than primarily finding an optimal configuration, the division into development and evaluation has not been strictly followed. The experiments have been divided into two classes. The first was to explore the parameters of Alize, with a »what happens if» type of investigation. By far, most time was used for this. The second class involved finding an optimal configuration. For this class of experiments, SpeechDat was used together with Gandalf for the development, and PER was used for evaluation. During parameter exploration, all three databases were used to more or less extent.

It should be noted that all sets were provided by the project supervisor in form of GIVES job descriptions. This helped assure that the bases of the experiments on both GIVES and Alize are the same. The names of the data sets used in this paper are also used by Melin [4]. They have been preserved for reference.

5.1.2 SpeechDat

The set of background speech used is called *bs3w2*, where from each session, one digit sequence, three sentences and two words are included. The set contains a total of 561 unique female sessions and 399 unique male sessions, and a total of 3,303 female feature files and 2,358 male feature files. Because 40 of the speakers in the SpeechDat set are also available in Gandalf, these were removed using a list of exception sessions.

5.1.3 Gandalf

From Gandalf, a data set resembling the data set from the PER database, described later, was used. During training, the set *fs0xd5/per-fiver* was used. It contains five repetitions of a sentence (simulating the names in PER) and five repetitions of a ten digit sequence from a single session. For testing, the set *1fs+1r4-fs0x-SSOnly* was used. It contains one sentence (again, simulating a name) and four consecutive digits. Furthermore, it only tests voices of the same

gender[1].

In the end, the training set contained 18 female, and 22 male speakers, with a total of 266 female feature files and 329 male feature files. The testing set contained another 532 female feature files and 437 male feature files. Using the models and testing set, 419 female and 508 true speaker tests were performed. The testing set also contained 36 female and 46 male impostor tests, for use in T-norm score normalization.

5.1.4 PER

From PER, three sets were used: Background, speaker training, and testing. The background set was called *E2a-gen*. Each background speaker contributes with approximately 30 seconds of speech. For training, the accompanying set *E2a* was used. Finally, the testing set was called *S2b*. Since these sets are divided into conditions, they all contain at least four subsets (see Section 4.1.2).

As a comparative example, the *G8* condition of the *E2a-gen* data set contained 277 female and 508 male feature files. These were used when training the female and male background models. The training set contained 16 female and 38 male true speakers, plus 28 female and 51 impostor speakers.

Tests were performed with 1,029 female and 3,614 male true speakers. Impostor tests, for T-norm, contained 208 female and 913 male tests. A complete description of these sets, including the other three conditions, can be found in Melin [4, App. C].

5.2 System Setups

To test Alize and LIA_RAL, a system had to be built (see Chapter 4), and parameters for the system had to be chosen. It was decided early in the project that the experiments should be divided into two groups: Parameter exploration and performance optimization. The first of these, the »parameter exploration» was intended as a hands-on crash course in speech research for learning how sensitive the various parameters were and what impact they had on the results.

On the other hand, the normal approach while evaluating an hypothesis (or a system) in speech research is to create a configuration with as high performance as possible. To show how Alize compares to GIVES or any other system using Swedish databases, it was therefore also important to run performance optimization tests, where the goal was to have Alize and GIVES perform equally well, under roughly equal conditions.

5.2.1 Parameter Exploration

A baseline setup was created with a configuration as close to the configuration used with Alize during NIST's SRE 2004 evaluation [3] as possible, and named *plain*. Most of the parameters were taken from the example scripts available at the LIA_RAL website [1]. However, some parameters were found to be unused in the programs, while others were lacking in the configuration file. They also had to be regrouped to support the use of make-files in place of a simple shell-script, thus supporting an iterative approach, rather than a »waterfall».

For the baseline setup, GMMs with 512 distributions were used. The background model was trained using ten iterations. Speaker model training had the model warping feature enabled and did three iterations using the »MAPOccDep» method, with a regression factor of 14 (see Section 3.3.4). The same configuration applies to impostor model training (used for T-norm). During testing, the top ten distributions were used for LLR calculations. Finally, in case of T-norm, the T-norm was applied with selection type »noSelect», i.e. all T-norm score data were used. A complete listing of the plain configuration can be found in Appendix A.

Both the G8 (gate) and LO (telephone)

[1]It is assumed for most parts of ASV research that it is easy to classify a voice as male or female.

Name	Description
plain	baseline configuration
notnorm	without T-norm
nodistnorm	without model warping
halftopdists	5 LLR distributions instead of 10
cohort	a selection of T-norm models was used
allworld	use all features during world training
dists/1024	1024 distributions instead of 512
dists/256	256 distributions
dists/128	128 distributions
dists/90	90 distributions
dists/64	64 distributions
mapreg/7	reg-factor 7 instead of 14
trainits/5	5 training iterations instead of 3
trainits/2	2 training iterations
worldits/20	20 world training iterations instead of 10
worldits/15	15 world training iterations
worldits/5	5 world training iterations

Table 5.1. Tests performed during the parameter exploration phase.

PER conditions were used, and the basic idea was to change one parameter at a time in the configuration and study the effects on the EER. Table 5.1 lists the tests performed. Of these tests, the empirically most interesting is the *nodistnorm* attempt, as it explores the model warping feature, currently not available in GIVES. Differences between baseline configuration and the various exploration configurations are shown in detail in Appendix B.

While testing GMM sizes, the 90-term GMM was added to investigate the effects of a size not a power of two. This test was set up to verify that Alize makes no assumptions on the GMM sizes.

5.2.2 Performance Optimization

To see how the system performed under development conditions similar to GIVES, a classical development–evaluation test was performed. The system was trained with *bs3w2* (SpeechDat) background models, *fs0xd5/per-fiver* (Gandalf) speaker models, and tests were performed with *1fs+1r4-fs0x-SSOnly* (Gandalf). Using the knowledge from the parameter exploration runs of how various parameters influence the end result, a local optimum was found for the development data. Finally, the PER sets *E2a-gen*, *E2a*, and *S2b* were used to run the evaluation and to gather EER values. These tests too were performed on the G8 and LO conditions separately.

Three *generations* were run on development data with three different configurations per generation. The first generation optimized the number of training iterations, and also helped decide whether to use model warping or not. The second generation was used to optimize the MAP regression factor, used in the speaker model training algorithm. The final generation was setup to see if only using a subset of feature vectors during world training would be better.

After each generation, one configuration was selected by hand. The selection was based on EER on both females and males. In the end, one configuration was left and this was used for the final run on the PER data sets. The full final configuration is documented in Appendix C.

Chapter 6

Results

This chapter will list the results from the experiments initially described in the previous chapter. In the first section, the experimental results from parameter exploration will be presented. They are to be interpreted as a guidance to the sensitivity of various parameters and served as a starting point for the performance optimization tests presented last.

6.1 Parameter Exploration

The key results for the parameter exploration on the G8 data set are visualized in Figure 6.1. A number of observations can be made from this plot and Figure 6.2, corresponding to the LO data set. The underlying numerical results are presented in Table B.1.

First of all, three runs in Figure 6.1 show improvement in all data sets relative to *plain*: *allworld*, *trainits/5*, and *worldits/15*. It seems like the baseline configuration leads to under-trained models. Increasing the number of feature vectors used during background training (*allworld*), or increasing the number of training iterations results in better matches (*worldits/20*), though the improvement in the case of more background data is marginal. There also seems to be a limit on the number of background iterations, as is shown in the LO case, with a small change in results between the use of 15 and 20 world iterations. This is further

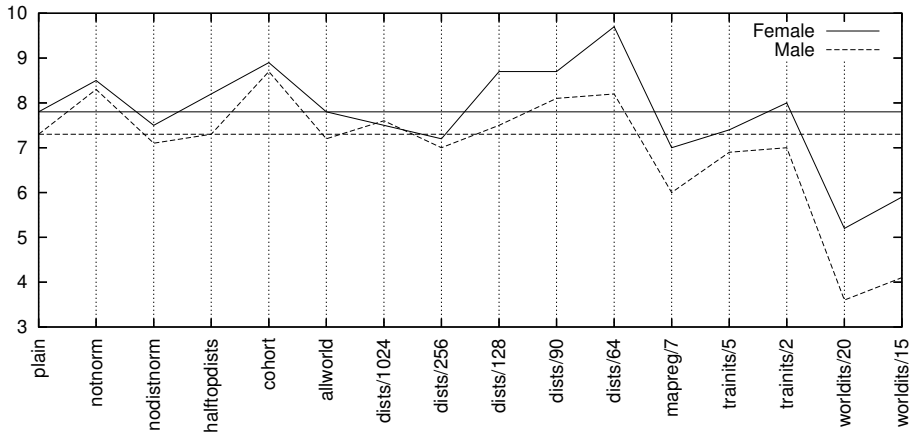


Figure 6.1. The EER values, in percent, for all runs on the G8 data set. The horizontal lines show the values for the *plain* setup.

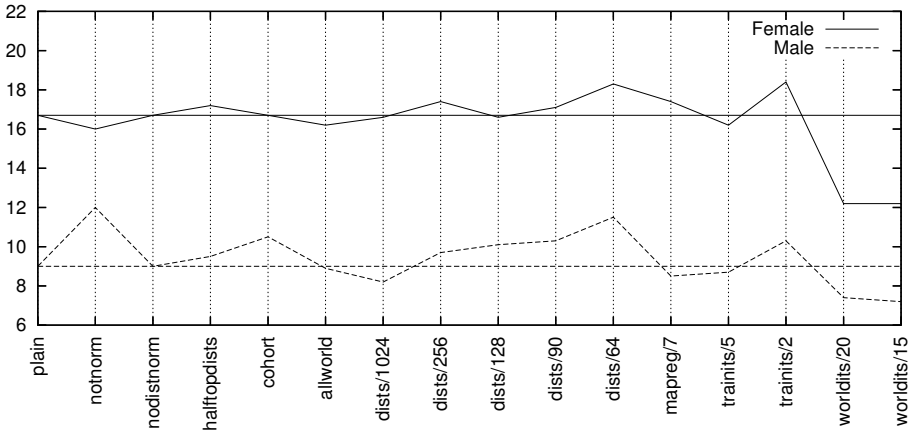


Figure 6.2. The EER values, in percent, for all runs on the LO data set. The horizontal lines show the values for the *plain* setup.

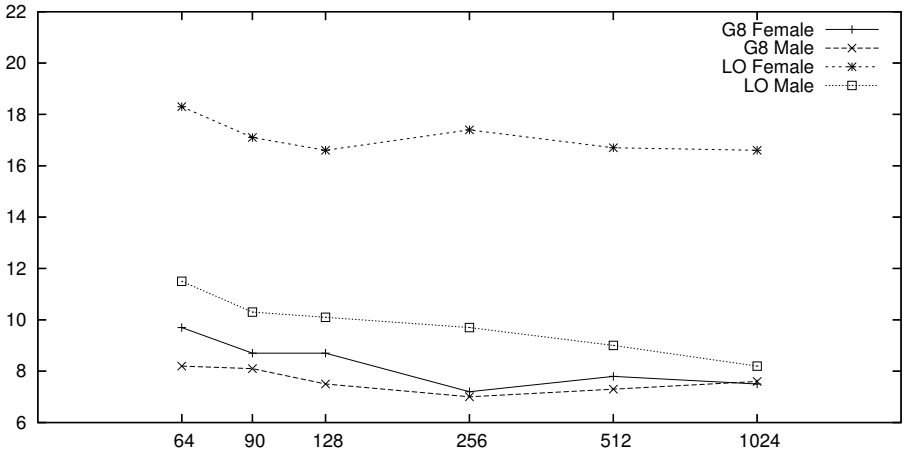


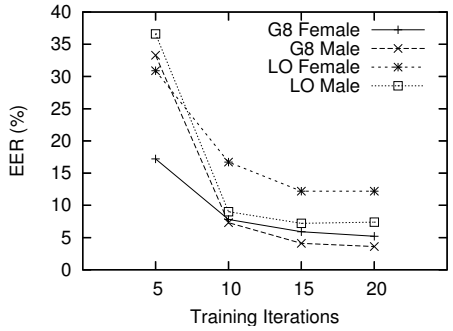
Figure 6.3. EER values, in percent, for six GMM sizes.

indicated by the increase of EER when decreasing the number of iterations.

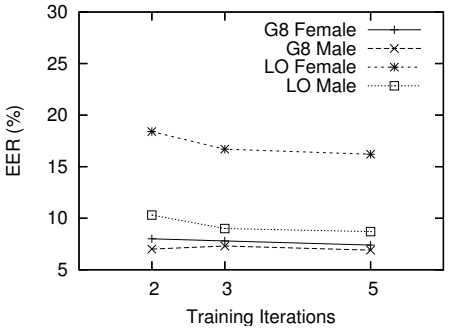
Second, removing the model warping (*nodistnorm*), gives an EER improvement. Finally, any removal of T-norm data results in a performance degradation (*notnorm*, and *cohort*).

A closer look at the number of distributions, seen in Figure 6.3, reveals that, although a greater number of distributions does improve performance, using more than 512 distributions only gives slightly better EER values. The exception to this is the LO male set, which seems to continue to improve even beyond 1,024 distributions.

Comparing the G8 and LO results, it can be seen that the relative changes over the baseline configuration are generally lower for the LO (telephone) set. In Figure 6.4, the EER as a function of various numbers of training iterations is shown. Figure 6.4(a) shows the number of world training iterations, while Figure 6.4(b) depicts the number of speaker model training iterations. For all data sets of the world training iterations tests, raising the number of iterations above 15 only results in little improvement, in relation to the major improvement for the iteration numbers below 10. Changing the number of speaker training iterations, on the

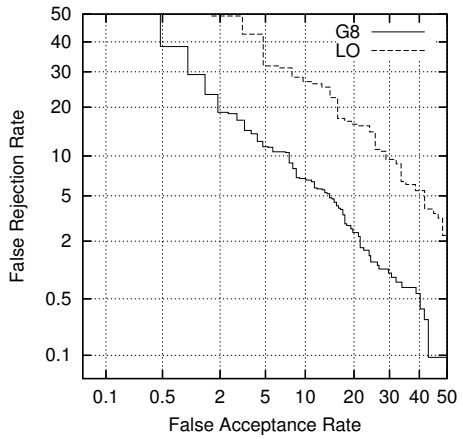


(a) EER for four different number of world training iterations.

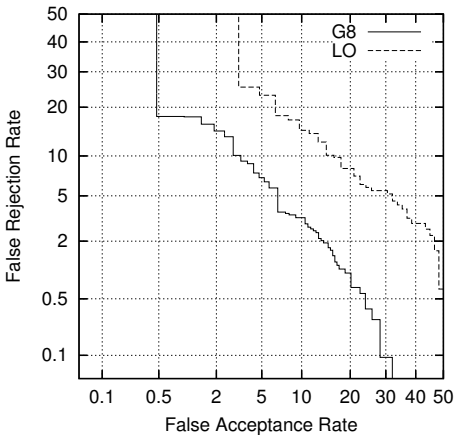


(b) EER for three different number of training iterations.

Figure 6.4. Plots of EER values versus the number of target speaker training iterations.



(a) Plot of the *plain* results.



(b) Plot of the *worldits/20* results.

Figure 6.5. DET-plots for the *Female* tests of two configurations.

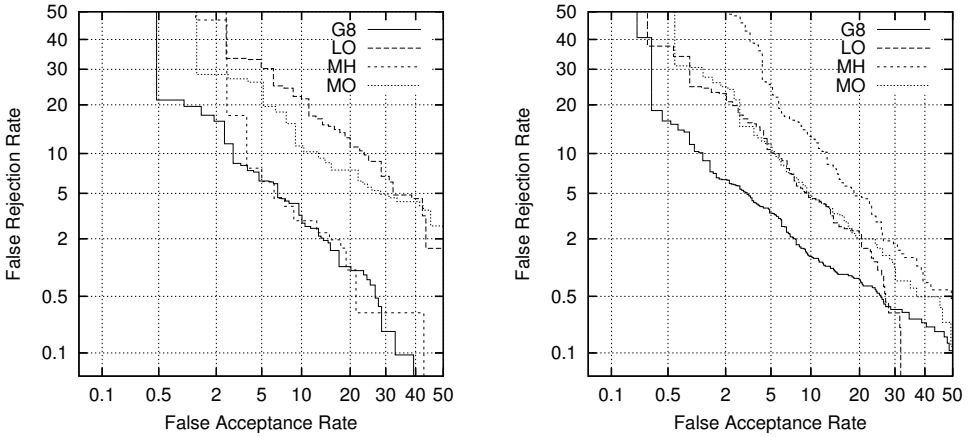
other hand, seems to give little change to the EER in all cases.

In Figure 6.5, standard DET-plots show details on the *plain* (baseline) and *worldits/20* configurations. Generally, performance is significantly better for the *G8* condition. The curves are almost straight and have a 45° inclination, as is normally expected from DET-plots. Unfortunately, the sparse data available result in low resolution in the extremes of the plots.

Note that all tests were run on the baseline configuration with only one parameter changed at a time, thus the results presented are in no way optimal. For instance, an increase in the number of world iterations would most probably result in better overall performance.

6.2 Performance Optimization

Experimental results on the four conditions are presented in Figure 6.6, with only the most successful runs of each generation plotted. The EER values of these plots, together with comparative results from the



(a) Plot of the females.

(b) Plot of the males.

Figure 6.6. DET-plots of the final run of the performance optimization experiments.

	Alize			GIVES GMM
	Female	Male	Average	Average
G8	5.6	3.9	4.8	4.2
LO	14.	6.8	11.	5.2
MO	10.	7.0	6.2	5.8
MH	5.5	11.	8.1	6.4
Average	8.9	5.9	7.4	5.4

Table 6.1. EER-values, in percent, for the final run.

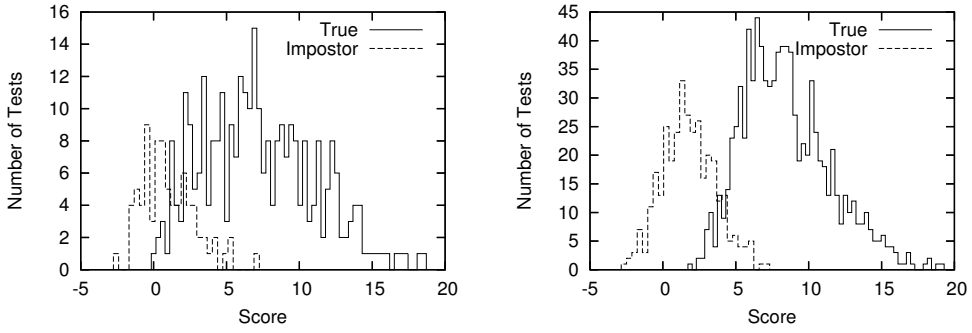
GMM subsystem of GIVES where available, are shown in Table 6.1. Note that GIVES normally runs a combination of an HMM subsystem and a GMM subsystem. Hence, GIVES tend to perform better overall. To make the comparison more fair, only the results from the GMM subsystem have been shown. Complete results from GIVES can be found in Melin [4, Sec. 10.4]. The GIVES figures for *MO* and *MH* are approximate, as they also cover the HMM subsystem.

Except for the *MH* condition, there seems to be consensus on males being easier to verify correctly. On the other hand, there are big differences in how well the two different genders of people are verified. Ordering the male tests with lower EER first results in *G8*, *LO*, *MO*, and *MH*, while an ordering based on female tests results in *MH*, *G8*, *MO*, and *LO*. Hence, any fact that cellular phones (*MH* and *MO*) are harder to verify cannot be established in these results.

It is interesting to note the close relationship between the extremes. On the one hand *G8*, with a wide-band microphone in a hallway, and on the other hand *MH* with its narrow-band microphone and transmission channel. In the case of female voice, they are almost equal, while for the male voices, they are the most extreme.

Additional research was done on the *LO* and *MH* conditions as the performance is much lower than that of GIVES. Histograms were plotted for the score distribution in the four cases (two conditions with two genders). The results are presented in Figures 6.7 and 6.8.

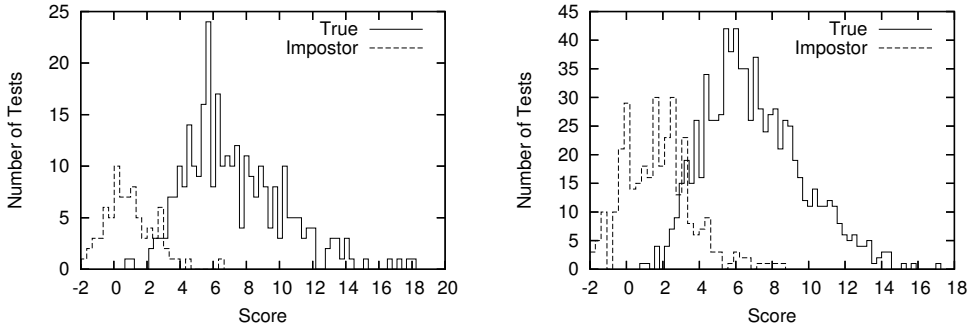
As expected, the cases with high EER



(a) Plot of the females.

(b) Plot of the males.

Figure 6.7. Score histograms of the *LO* condition.



(a) Plot of the females.

(b) Plot of the males.

Figure 6.8. Score histograms of the *MH* condition.

also have scores in a small interval. The females of *LO* and the males of *MH* both have very little separation between true and impostor speakers. Unfortunately, the explicit cause of this could not be found within the time limits of the project.

Chapter 7

Comments and Conclusions

This final chapter will focus on the conclusions from experimental results, the author's experiences and general comments on the systems used during the project. A few ideas for future work are also presented.

7.1 On the Experimental Results

It should be noted that even though the performance optimization was performed by development experiments on Gandalf, and running the evaluation on PER, the parameter exploration still used PER. This can result in a decrease of reliability in the experimental results. However, to get acquainted with the research area, the parameter exploration was necessary. The rationale for not using Gandalf was the greater amount of data available in PER, and the many types of data sets already available.

It is interesting to see that removing the model warping, a feature not available in GIVES, results in a minor decrease in EER on the system with higher audio quality (and more data), and no change in the telephone case. Since the use of Alize with model warping previously has shown little improvement of performance on NIST data [2], the author is sceptical to the importance of this algorithm.

A straight-forward conclusion from the results is that a cohort selection during score normalization, where only a few T-norm models are taken into account, is no good. This is perhaps due to the limited number of impostor models created by the data sets.

In other respects, the results of the experiments have been as expected. Differences exist between the three data sets used, but they were anticipated due to the statistical methods used. A single statistical model cannot model all data sets equally well. This is especially true for low-quality audio such as the PER LO data set.

7.2 Comments On Alize

There are many aspects of using Alize, including class hierarchy, design patterns and performance. From a computer science point of view, Alize leaves much to wish for. It is a compact statistical kernel, but with a specialization towards speaker verification. However, the design patterns used are mostly taken from Java or similar languages, and does not use the »C++ way» of solving problems. There is, for instance, no use of templates. This means that, for each new type of `vector` that needs to be used, it must also be implemented. Nothing uses the Standard Template Library, which means the kernel itself is much larger than necessary.

Further, the file format readers for feature files are highly specialized and uses an intricate system of class inheritance instead of separating the file format readers and

making them individual. This leads to problems when a new format has to be read, and this will be most obvious when the need arises outside of LIA (Laboratoire Informatique d'Avignon). The readers and writers also do not make use of the standard C++ streams for I/O, but has a reimplemented Java-like streams interface, which uses the standard C library `FILE` stream.

The feature server, imposes some unnecessary restrictions on its buffer size. During the experimentation a major hurdle was that the feature normalization program (`NormFeat`) failed to write features. After much debugging, it turned out that this was due to a too small feature buffer. As there is no real need to buffer all feature files (due to independence between files), this was not an obvious cause of error.

Since very little code is unique to speaker recognition, Alize should be transformed into a generic statistics and I/O library to better reflect its purpose. Alternatively, the LIA_RAL could be integrated into Alize to form one library. A reimplementation of Alize using STL and C++ streams would probably make the library both smaller and easier to overview. The apparent Java style is not well suited for use in C++ and results in more code and code which is harder to maintain.

7.3 Comments On LIA_RAL

As it is the LIA_RAL package that contains most of the speaker recognition algorithms, this seems to be the most natural platform for continued research. It contains unused code for Hidden Markov Models (HMM) and is probably better maintained than Alize. The current implementation does, however, have its drawbacks.

The use of a generic configuration map (key–value) used as an argument to every function imposes problems on syntax checking. It is possible that an erroneous configuration parameter is simply silently ignored because of the lack of syntactic and semantic verification. This is most apparent in the handling of the `verbose` and `debug` parameters. They are treated differently in different programs. Some programs use the *existence* of the configuration parameters, while others use the *boolean value* of the parameters. This means that, for some programs, a configuration line of `verbose false` still means »be verbose». This is most confusing, and is hard to spot in the source code as the entire free-text configuration is passed around without checking.

A problem found late in the project was the lack of valid exit codes from the programs. Normally, all programs can return an integer value between -128 and 127, with zero (by convention) meaning »success». However, it was found that none of the programs used in the make-files report errors. Though the errors are reported as text on the terminal, the exit code is always zero. This fact implies that some parsing of the output has to be done in order to determine if the run was successful or not. During experimentation the output of one run showed an EER of 0.2%, due to all tests not being run, though all programs (and thus the entire run) reported success (the real figure turned out to be 7% in a more successful rerun).

On the positive side, the use of prefix and suffix configuration parameters for file access proved to be very useful. Moving between different data sets and file formats was very easy. A straight-forward generalization of this concept would be to allow an *escape-sequence*[1] in a single free-form string, instead of separate prefixes and suffixes. This change would also reduce the number of configuration parameters.

7.4 Comments on the Project

In this section, some thoughts on the project itself has been put together.

[1] Something like a variable substitution.

7.4.1 Time Disposition

The original time estimation did not reflect actual work. Due to part-time work (50%) the project was not completed on time (due December 2005). This is unfortunate and could have, presumably, been avoided, had the project been worked on full-time. It was further delayed due to periods where the other work required more than 50% time.

In retrospect, working on the project involved seven tasks:

1. Reading and understanding the ASV field of research.
2. Understanding Alize and LIA_RAL.
3. Converting three TMH databases to Alize format.
4. Creating the make-files to create a speaker verification system.
5. Creation of a batch job manager.
6. Running tests and evaluate the systems.
7. Presenting the work.

The total duration of the project was from May 2005 to February 2007. During 2005, the first three tasks were prioritized. Understanding Alize and LIA_RAL and understanding ASV required more time than originally estimated. A large amount of time was spent on reading source code and literature during 2005, but some make-files and the conversion of SpeechDat was also performed during this time.

In Spring 2006, a batch job manager was created to assist in running tests. As each test creates two DET-plots and two EER-values, it was necessary to structure the job archive for easy searching. The manager also made it possible to queue jobs, and to detach the user interface from the server, thus being more fault tolerant than running the jobs directly at the terminal.

When the job manager was working, the make-files were completed (Summer 2006). Finally, testing and writing on the paper were the major tasks of Autumn 2006. This is revision 76 of the paper.

7.4.2 Missing Tasks

The idea of connecting parts of GIVES with parts of Alize was not pursued during the project. When it was found that the performance of Alize was comparable to GIVES, it was decided to skip this idea, originally described in the project prospect.

Because of the high EER seen in some cases in the results, more work should have been spent on drawing conclusions from the results. However, as the *MO* and *MH* conditions were only run very late in the project, there was no time to pursue the causes.

7.4.3 Prior Knowledge

Before this project, a course in speech technology (Sw. »Talteknologi») given at TMH had been completed. As this course was wide rather than deep, the author felt an urge to expand his knowledge on speech recognition. Also, some courses in machine learning (»Articiella Neuronnät» and »Maskininlärning») were completed prior to this project, and proved somewhat useful while dealing with GMMs.

7.4.4 Future Work

There are aspects of Alize still not explored in the context of Swedish and the databases available at TMH. The most notable aspect being the HMM module which, at least in the version used for this project, was not fully implemented. Furthermore, a few helper programs like the turn detector or energy detector could be of interest.

As noted in Melin [4], the performance of a system may change if the prompting behavior changes. Thus, a conversion to a real-time classification system may lead to other results. Due to restrictions in the current LIA_RAL implementation, this track was not pursued.

Another library more focused on machine learning is Torch[2] from IDIAP, Switzerland. However, at the writing of this paper, the project seems to have stalled. It contains far more types of models than does Alize, for instance, support vector machines. Exploration of this toolkit could be the basis for a new Master Thesis project. Many other toolkits are used in the NIST Speaker Recognition Evaluations (SRE), and there seems to be little comparative data on these projects, except for pure performance measures of the NIST SRE.

7.5 Conclusions

All-in-all, Alize and the LIA_RAL toolkits seem to fulfill their intended behavior in most aspects. It does not appear to include any performance critical features not already available in GIVES. However, the vague documentation available makes it hard to use, and much time must be spent reading source code to find answers not available in the documentation.

Performance-wise (EER), the system is comparable to GIVES with GMMs. This is not surprising as they use the same algorithms. When GIVES is used with its HMM subsystem, GIVES performs better. Currently, Alize does not offer any HMM support and it would be unfair to compare the two systems based on results other than from the GMM systems. The model warping feature was not found to be useful in the experiments of this project, and in some cases reduced the performance somewhat.

However, the Alize and LIA_RAL toolkits are still young and need more time and effort to be a useful alternative to GIVES. Using parts of Alize together with GIVES should be possible, given the modular nature of Alize. Intermixing is merely a matter of writing conversion routines for the representations of scores and mixture models. This project has not shown any part of Alize to perform better than the equivalent part in GIVES.

7.6 Acknowledgements

Many thanks to Håkan Melin for patiently answering many questions and also for reviewing and proofreading the paper in detail. Also thanks to Daniel Neiberg at TMH for spreading light on the GMM part of GIVES and its configuration. This project would not have been possible without their time and knowledge.

Finally, thanks go to Martin Nygren and Ibrahim Ayata for their time while proofreading the paper.

[2] Available at `http://www.torch.ch/`.

References

[1] Bonastre, J.-F., Wils, F., *Alize: a free, open tool for speaker recognition*, `http://www.lia.univ-avignon.fr/heberges/ALIZE/`.

[2] Bonastre, J.-F., Scheffer, N., Fredouille, C., Matrouf, D., *NIST'04 Speaker Recognition Evaluation Campaign: New LIA Speaker Detection Platform Based on Alize Toolkit*, NIST Speaker Recognition Evaluation 2004.

[3] Bonastre, J.F., Wils, F., Meignier, S. *Alize, A Free Toolkit for Speaker Recognition*, Proceedings of ICASSP 2005.

[4] Melin, H., *Automatic Speaker Verification On Site and by Telephone: Methods, Applications and Assessment*, Doctoral Thesis, Department of Speech, Music and Hearing, KTH, 2006.

[5] Auckenthaler, R., Carey, M., and Lloyd-Thomas, H., *Score Normalization for Text-Independent Speaker Verification Systems*, Digital Signal Processing **10** (2000), pp. 42–54.

[6] Dempster, A.P., Laird, N.M., Rubin, D.B., *Maximum-Likelihood From Incomplete Data via the EM Algorithm*, Journal of the Royal Statistical Society, Ser. B., 39, 1977.

[7] Bilmes, J., *A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models*, International Computer Science Institute, 1998.

[8] Reynolds, D. A., Quatieri, T. F., Dunn, R. B., *Speaker Verification Using Adapted Gaussian Mixture Models*, Digital Signal Processing, 2000.

[9] Grassi, S., Ansorge, M., Pellandini, F., Farine, P.-A., *Distributed Speaker Recognition Using the ETSI AURORA Standard*, Proc. 3rd COST 276 Workshop on Information and Knowledge Management for Integrated Media Communication, pp. 120–125, 2002.

[10] Bogert, B. P., Healy, M. J. R., Tukey, J. W., *The Quefrency Analysis of Time Series For Echoes: Cepstrum, Pseudo-Autocovariance, Cross-cepstrum, and Saphe Cracking*, Proceedings of the Symposium on Time Series Analysis, Ch. 15, pp. 209–243, 1963.

[11] Holmes, J., Holmes, W., *Speech Synthesis and Recognition, 2nd ed.*, Taylor and Francis, 2001.

[12] Jiang, H., *Confidence Measures For Speech Recognition; A Survey*, Speech Communication, Volume 45 No. 4, pp. 455–470, 2005.

[13] Reynolds, D.A., *Comparison of Background Normalization Methods for Text-Independent Speaker Verification*, Proceedings of Eurospeech, 1997, pp. 963–966.

[14] Martin, A., Doddington, G., Kamm, T., Ordowski, M., Przybocki, .M, *The DET Curve In Assessment of Detection Task Performance*, Proceedings of Eurospeech 1997, Volume 4, pp. 1895–1898.

[15] Pelecanos, J., Sridharan, S., *Feature Warping for Robust Speaker Verification*, Proc. ISCA Workshop on Speaker Recognition - 2001: A Speaker Odyssey, 2001.

[16] Elenius, K., *Experiences From Collecting Two Swedish Telephone Speech Databases*, International Journal of Speech Technology, Volume 3, 2000.

[17] Hermansky, H., Morgan, N., Bayya, A., Kohn, P., *RASTA-PLP Speech Analysis*, ICSI Technical Report TR-91-069, Berkley, 1991.

Appendix A

Base Line Configuration

This appendix is intended as a reference for the Experiments chapter. It is a listing of the configuration directives used for the baseline setup, *plain*, during the parameter exploration phase.

Note that some configuration directives are dynamically generated in the calling make-file. However, these are directives for search paths and gender declaration, and should not be of importance for the results. Also note that these files were taken from the PER configuration. Similar files exist for the SpeechDat/Gandalf combination. Finally, the `TrainTarget` and `ComputeTest` configurations for impostor model training are verbatim copies of the target models, hence they have been left out.

Note that the exclamation point present on the `debug` and `verbose` lines is a fix to a problem described in Section 7.3.

A.1 TrainWorld — Background Model

```
featureFilesPath ../../per/param/eval02/
mixtureFilesPath ./gmm/
labelFilesPath ../../per/labels/energy/eval02/
loadMixtureFileFormat RAW
loadMixtureFileExtension .gmm
saveMixtureFileFormat RAW
saveMixtureFileExtension .gmm
loadFeatureFileFormat SPRO4
loadFeatureFileExtension .norm.spro
bigEndian false
featureServerBufferSize ALL_FEATURES
distribType GD
frameLength 0.01
vectSize 24
labelSelectedFrames speech
!debug true
!verbose true
fileInit false
inputWorldFilename xxx
normalizeModel true
```

```
mixtureDistribCount 512
baggedFrameProbabilityInit 0.04
maxLLK 200
minLLK -200
baggedFrameProbability 0.1

nbTrainIt 6
initVarianceFlooring 0.5
initVarianceCeiling 10.0

nbTrainFinalIt 4
finalVarianceFlooring 0.5
finalVarianceCeiling 10.0
```

A.2 TrainTarget — Speaker Models

```
distribType GD
loadMixtureFileExtension .gmm
saveMixtureFileExtension .gmm
loadFeatureFileExtension .norm.spro
maxLLK 200
minLLK -200
bigEndian false
saveMixtureFileFormat RAW
loadMixtureFileFormat RAW
loadFeatureFileFormat SPRO4
featureServerBufferSize ALL_FEATURES
featureFilesPath ../../per/param/eval02/
mixtureFilesPath ./
labelFilesPath ../../per/labels/energy/eval02/
labelSelectedFrames speech
frameLength 0.01
mixtureServer /unwritable.ms Hardcoded not used in TrainTarget.cpp
!verbose true
!debug true
vectSize 24

nbTrainIt 3
nbTrainFinalIt 0
MAPAlgo MAPOccDep
MAPRegFactor 14
baggedFrameProbability 1
normalizeModel true
```

A.3 ComputeTest — Segment Testing

```
distribType GD
saveMixtureFileExtension .gmm
```

```
loadMixtureFileExtension .gmm
loadFeatureFileExtension .norm.spro
maxLLK 200
minLLK -200
bigEndian false
saveMixtureFileFormat RAW
loadMixtureFileFormat RAW
loadFeatureFileFormat SPRO4
featureServerBufferSize ALL_FEATURES
featureFilesPath ../../../data/per/param/eval02/
lstPath ./
labelFilesPath ../../../data/per/labels/energy/eval02/
!verbose true
!debug true
vectSize 24
labelSelectedFrames speech
frameLength 0.01

topDistribsCount 10
computeLLKWithTopDistribs COMPLETE
segmentalMode completeLLR
```

A.4 ComputeNorm — Score Normalization

```
verbose true
debug true
maxScoreDistribNb 500
maxIdNb 1000
maxSegNb 5000
selectType noSelect
tnormCohortNb 0
znormCohortNb 0
```

Appendix B

Parameter Exploration

This appendix contains the configurations and hard results from the parameter exploration tests. They are presented here for reference, and as a guide to the relevance of various dimensions in speaker recognition algorithms.

B.1 Configurations

All of the tests were performed by changing the value of one baseline parameter. For this reason, only the value of the changed parameter will be presented here. For the rest of the configuration, refer to Appendix A.

notnorm

The modules *impostor*, *tnorm* and *score* were completely removed. The *plain* configuration used all modules.

nodistnorm

The property `normalizeModel` was set to `false` for the modules *world*, *target*, and *impostor*. The property was set to `true` in the *plain* configuration.

halftopdists

The property `topDistribsCount` was set to `5` for the modules *test* and *tnorm* (`10` in the *plain* configuration).

cohort

The property `selectType` was set to `selectNBestByTestSegment`, and `tnormCohortNb` was set to `15` for the module *score* (all score values were used in the *plain* configuration).

allworld

The property `baggedFrameProbability` was set to `1` for the module *world*. This property was `0.1` (10%) in the *plain* configuration.

dists/

The property `mixtureDistribCount` in module *world* was set to the corresponding value. The mixture size was 512 distributions in the *plain* configuration.

mapreg/7

The property `MAPRegFactor` was set to `7` for the modules *target* and *impostor* (as opposed to `14` in the *plain* configuration).

trainits/

The property `nbTrainIt` in modules *target* and *impostor* was set to the corresponding value. The number of training iterations in the *plain* configuration was set to 3.

worldits/

The property `nbTrainFinalIt` in module *world* was set to the corresponding value. This property had the value `4` in the *plain* configuration.

B.2 Results

	EER				Change			
	G8		LO		G8		LO	
Name	**F**	**M**	**F**	**M**	**F**	**M**	**F**	**M**
plain	7.8	7.3	16.7	9.0	0.0	0.0	0.0	0.0
notnorm	8.5	8.3	16.0	12.0	9.0	13.7	-4.2	33.3
nodistnorm	7.5	7.1	16.7	9.0	-3.9	-2.7	0.0	0.0
halftopdists	8.2	7.3	17.2	9.5	5.1	0.0	3.0	5.6
cohort	8.9	8.7	16.7	10.5	14.1	19.2	0.0	16.7
allworld	7.8	7.2	16.2	8.9	0.0	-1.4	-3.0	-1.1
dists/1024	7.5	7.6	16.6	8.2	-3.9	4.1	-0.6	-8.9
dists/256	7.2	7.0	17.4	9.7	-7.7	-4.1	4.2	7.8
dists/128	8.7	7.5	16.6	10.1	11.5	2.74	-0.6	12.2
dists/90	8.7	8.1	17.1	10.3	11.5	11.0	2.4	14.4
dists/64	9.7	8.2	18.3	11.5	24.4	12.3	9.6	27.8
mapreg/7	7.0	6.0	17.4	8.5	-10.26	-17.81	4.19	-5.56
trainits/5	7.4	6.9	16.2	8.7	-5.1	-5.5	-3.-3	3.
trainits/2	8.0	7.0	18.4	10.3	2.6	-4.1	10.2	14.4
worldits/20	5.2	3.6	12.2	7.4	-33.3	-50.7	-27.0	-17.8
worldits/15	5.9	4.1	12.2	7.2	-24.4	-43.8	-27.0	-20.0
worldits/5	17.2	33.3	30.9	36.6	121.0	356.0	85.0	307.0

Table B.1. To the left, the resulting EER values, in percent, for the parameter exploration tests. To the right, relative change, in percent over *plain*, of EER for the parameter exploration tests. Lower is better in both cases.

Appendix C

Performance Optimization

This appendix is intended as a reference for the Experiments chapter. It is a listing of the configuration directives used during the final run of the performance optimization experiments.

Note that some configuration directives are dynamically generated in the calling make-file. However, these are directives for search paths and gender declaration, and should not be of importance for the results. Also note that these files were taken from the PER configuration. Similar files exist for the SpeechDat/Gandalf combination. Finally, the `TrainTarget` and `ComputeTest` configurations for impostor model training are verbatim copies of the target models, hence they have been left out.

Note that the exclamation point present on the `debug` and `verbose` lines is a fix to a problem described in Section 7.3.

C.1 TrainWorld — Background Model

```
# created by AlizeActiveJob
# Wed Nov 29 10:26:52 CET 2006
baggedFrameProbabilityInit 0.04
saveMixtureFileFormat RAW
initVarianceCeiling 10.0
nbTrainIt 5
loadFeatureFileFormat SPRO4
featureServerBufferSize ALL_FEATURES
mixtureFilesPath ./gmm/
finalVarianceFlooring 0.5
featureFilesPath ../../per/param/eval02/
nbTrainFinalIt 10
maxLLK 200
loadMixtureFileExtension .gmm
vectSize 24
saveMixtureFileExtension .gmm
mixtureDistribCount 512
baggedFrameProbability 0.2
initVarianceFlooring 0.5
loadFeatureFileExtension .norm.spro
labelFilesPath ../../per/labels/energy/eval02/
```

```
loadMixtureFileFormat RAW
bigEndian false
fileInit false
finalVarianceCeiling 10.0
labelSelectedFrames speech
minLLK -200
inputWorldFilename xxx
normalizeModel false
distribType GD
frameLength 0.01
```

C.2 TrainTarget — Speaker Models

```
# created by AlizeActiveJob
# Wed Nov 29 10:26:52 CET 2006
mixtureServer /unwritable.ms
saveMixtureFileFormat RAW
nbTrainIt 5
MAPRegFactor 7
loadFeatureFileFormat SPRO4
featureServerBufferSize ALL_FEATURES
mixtureFilesPath ./
featureFilesPath ../../per/param/eval02/
nbTrainFinalIt 0
maxLLK 200
loadMixtureFileExtension .gmm
vectSize 24
saveMixtureFileExtension .gmm
baggedFrameProbability 1
loadFeatureFileExtension .norm.spro
labelFilesPath ../../per/labels/energy/eval02/
loadMixtureFileFormat RAW
bigEndian false
labelSelectedFrames speech
MAPAlgo MAPOccDep
minLLK -200
normalizeModel false
distribType GD
frameLength 0.01
```

C.3 ComputeTest — Segment Testing

```
distribType GD
saveMixtureFileExtension .gmm
loadMixtureFileExtension .gmm
loadFeatureFileExtension .norm.spro
maxLLK 200
minLLK -200
```

```
bigEndian false
saveMixtureFileFormat RAW
loadMixtureFileFormat RAW
loadFeatureFileFormat SPRO4
featureServerBufferSize ALL_FEATURES
featureFilesPath ../../../data/per/param/eval02/
lstPath ./
labelFilesPath ../../../data/per/labels/energy/eval02/
!verbose true
!debug true
vectSize 24
labelSelectedFrames speech
frameLength 0.01

topDistribsCount 10
computeLLKWithTopDistribs COMPLETE
segmentalMode completeLLR
```

C.4 ComputeNorm — Score Normalization

```
verbose true
debug true
maxScoreDistribNb 500
maxIdNb 1000
maxSegNb 5000
selectType noSelect
tnormCohortNb 0
znormCohortNb 0
```