MIXTURES OF INVERSE COVARIANCES: COVARIANCE MODELING FOR GAUSSIAN MIXTURES WITH APPLICATIONS TO AUTOMATIC SPEECH RECOGNITION

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Vincent Vanhoucke
July 30, 2003

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Robert M. Gray
(Principal Adviser)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Ananth Sankar

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Vaughan Pratt

Approved for the University Committee on Graduate Studies:

Preface

Gaussian mixture models (GMM) are widely used in statistical pattern recognition for a variety of tasks ranging from image classification to automatic speech recognition. Because of the large number of parameters devoted to representing Gaussian covariances in these models, their scalability to problems involving a large number of dimensions and a large number of Gaussian components is limited. In particular, this shortcoming of Gaussian mixture models affects the accuracy of real-time speech recognition systems by limiting the complexity of the mixtures used for acoustic modeling.

This thesis addresses the scalability problems of Gaussian mixtures through a class of models, collectively called "mixtures of inverse covariances" or MIC, which approximate the inverse covariances in a Gaussian mixture while significantly reducing both the number of parameters to be estimated, and the computations required to evaluate the Gaussian likelihoods. The MIC model scales well to problems involving large number of Gaussians and large dimensionalities, opening up new possibilities in the design of efficient and accurate statistical models. In particular, when applying these models to acoustic modeling for real-world automatic speech recognition tasks, they significantly improve both the speed and accuracy of a state-of-the-art speech recognition system.

Acknowledgments

This thesis would not have been possible without the leadership of Dr. Ananth Sankar, who initiated and drove this project. Ananth provided numerous insights and advices which have been central to the success of this work. I am also very much indebted to the entire Speech R&D group at Nuance for their support, both technically and financially. The extremely enjoyable work environment and the quality of the people I had the chance to work with at Nuance made this experience unique. For the quality of the software infrastructure that enabled this work, I am very much indebted to Remco Teunen and Michael Schuster. I would also like to thank Su-Lin Wu and Brian Strope for the experimental infrastructure which I used throughout this work.

I am grateful to Prof. Robert Gray for giving me the freedom to work alongside his group, which, through the exchange of ideas between the "speech world" and the "compression world" has been decisive to the success of this research. Many thanks to the Compression and Classification group, in particular Maya Gupta and Deirdre O'Brien for the enjoyable work environment. Thanks to Prof. Richard Olshen and Prof. Vaughan Pratt for their supportive feedback. Thanks to Prof. Renée Veysseyre for having me fail my undergraduate statistics exam, which got me interested in the subject in the first place.

Finally, for everything else, I want to express my infinite gratitude to my parents, Guy and Jacqueline, who have always backed me in any insane endeavor I got myself to partake in. May they rest assured that this thesis is not the last. Special thanks to my brother Olivier for his friendship, and to my grandmother Claire for her loving affection.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The field of statistical pattern recognition has undergone a profound change in the last few years as more and more people recognized that the combined advances in storage capabilities, network speed and computational power made large datasets and complex pattern recognition tasks within reach of a much broader community of researchers. Extremely large datasets of complex processes are now commonplace in a variety of areas.

As an illustration, in the biomedical field, PhysioNet [67] is a large and growing archive of well-characterized digital recordings of physiologic signals and related data for use by the biomedical research community. The field of bioinformatics has developed mostly around the exploitation of such large datasets. The National Center for Biotechnology Information (NCBI) [64] is one such source of public databases on computational biology, genome data, and biomedical information. In the same vein, the field of genomics has grown around the collection of DNA sequences such as those available through GenBank [39], on which large scale statistical analyses can be performed.

The area of automatic speech recognition (ASR) is one of the fields which underwent the transition from using small corpora of very constrained data (e.g. speech recorded by a single speaker, using a given microphone, in quiet conditions), to large all-encompassing tasks. Speech databases which are widely used nowadays cover large variations in speakers, acoustic conditions, recording equipment and languages

spoken. The Linguistic Data Consortium [58] is one of the primary sources for ASR datasets, and more and more languages now have speech corpora made publicly available by a variety of sources for training ASR systems. What came out of this transformation is the availability of large scale, "user-friendly" ASR systems which are now beginning to spread on the market. These systems are meant to be usable in any situation, without training from the user, in challenging acoustic environments such as a cellular phone line or a hands-free system used in cars.

It has also been recognized that such large tasks were challenging the tools available to statisticians as a simple result of their scale. Statistical methods developed to be robust on very sparse, high-dimensional data lose their edge on complex datasets containing large amounts of training data. On such datasets, limiting factors have more often to do with the scalability of the model, which encompasses computational efficiency during training and evaluation, compactness of the representation, and precision of the modeling.

A direct consequence of this exponential growth in the difficulty of the tasks addressed by ASR systems is that it has become a very good benchmark for the large-scale, challenging tasks that are now becoming increasingly available for researchers to study. Answers to modeling issues faced in ASR are widely applicable to tasks that broadly exhibit the same features, including:

- High dimensional input features,
- A large set of confusable, ill-defined classes,
- A broad range of varying background conditions which affect both the distribution of input features (i.e. the acoustic context) as well as the nature of the classes (the linguistic context)

In this environment, Gaussian mixture models have been very successful, allowing ASR software to exist as a viable technology product for more than a decade. Still, there is strong evidence that more can be done at the modeling level to improve the accuracy of ASR systems. There are mainly two ways the modeling can progress, and both seem to be equally promising:

1. increasing the amount of relevant information extracted from the acoustics,

2. increasing the precision of the modeling of the acoustic features.

Both avenues call for increasing the scale of the statistical model used, either by an increase of the input dimensionality, or of the number of significant parameters used in the model. In both cases, the otherwise very successful GMM model, in its present formulation, is beginning to show its limits.

This thesis addresses these shortcomings while attempting to preserve the good properties that made GMM popular in the first place. The following sections (1.3 and 1.4) introduce GMM in detail, discuss the various issues related to the model, and describe various existing solutions addressing them. Chapter 2 introduces the mixture of inverse covariances (MIC) model, which is a general covariance modeling framework which lays the foundations for the rest of this thesis. Chapter 3 introduces a variable length extension to the MIC model. Chapter 4 combines the ideas of subspace factorization and the MIC model into a modeling framework which takes explicit advantage of the specific structure of covariance matrices when such structure exists. Chapter 5 is devoted to ASR in general, and how GMM modeling fits into the broader problem of automatic speech recognition. Several GMM modeling techniques developed in the context of ASR are also discussed. Chapter 6 shows how the different models introduced apply to acoustic modeling and improve the performance of ASR systems. Chapter 7 concludes this study and proposes several potential avenues for future research.

1.1 Statistical Pattern Recognition

Statistical pattern recognition [28] is a general framework for performing classification of data using a probabilistic model. The assumption is that the data which needs to be classified can be summarized using a *feature vector* $\boldsymbol{x}$ which represents the information relevant to the classification. Determining which features of the data to use for a given classification task is in itself a difficult problem, which requires using prior knowledge or automated methods [78] to determine the relevance of individual

feature components.

Given this input feature vector $\boldsymbol{x}$ summarizing the data, the classification task can be formulated as: "find the class $\tilde{c}$, in the set of all classes $\mathcal{C}$, which is most likely to match $\boldsymbol{x}$". In mathematical terms, this can be written formally as the maximum a posteriori (MAP) principle

$$\tilde{c} = \underset{c \in \mathcal{C}}{\operatorname{argmax}}\, p(c|\boldsymbol{x}),$$

where $p(c|\boldsymbol{x})$ represent the conditional probability of the class c to match the input $\boldsymbol{x}$. Bayesian decision theory [28, Chapter 2] generalizes this principle to incorporate the notion of a *risk* incurred when misclassifying a data point. Here we will only consider a uniform risk for simplicity. Using Bayes rule, the maximization can be rewritten as:

$$\tilde{c} = \underset{c \in \mathcal{C}}{\operatorname{argmax}}\, \underbrace{p(\boldsymbol{x}|c)}_{\text{likelihood}} \cdot \underbrace{p(c)}_{\text{prior}}.$$

In this formulation, the classification problem become one of estimating two types of probability distributions:

- the *prior* $p(c)$ does not depend on the input data point considered, which is why in general coming up with a reasonable prior for a set of classes is not difficult. The simple supervised approach consists of labeling some training data with their known class labels, and simply accumulating the frequency histogram of these labeled training samples. Typically one would also smooth the prior distribution to account for events which are unseen or rarely seen in the training data but might occur in the test data.

- the likelihood $p(\boldsymbol{x}|c)$ is also a generally unknown probability density. It involves the input data point $\boldsymbol{x}$, which means that since in general the data the system is tested on differs from the training data available, the specific test point $\boldsymbol{x}$ is almost never observed during the design of the classifier, and thus $p(\boldsymbol{x}|c)$ needs to be estimated by generalizing from the training data.

The main difficulty of statistical pattern recognition is to design an *efficient* model of this likelihood. The efficiency of the model can be measured using several criteria:

- *accuracy:* the most obvious metric by which to evaluate a classification algorithm, which measures what proportion of the classification decisions are correct,
- *interpretability:* the ability for the model to expose features of the classification task that can be examined and analyzed by the designer of the classifier, which often lead to infer meaningful relationships between the input data and the classes,
- *generality:* the ability to adapt itself to unseen situations by extrapolating them from observed data,
- *robustness:* the resistance of the model to noise and interferences in the input data,
- *computational efficiency:* the speed at which the likelihood can be evaluated, and thus the classification task performed,
- *compactness:* the size of the model considered, which impacts not only the amount of memory required to run a classification task, but also often predicates both the robustness and the computational efficiency of the model.

Parametric statistical models typically define a generic template p_θ specifying the functional form of the likelihood model, while leaving a number of model parameters $\theta \in \Theta$ free. These parameters then need to be estimated using some data to determine the complete likelihood model. The simplest method used for training these parameters is maximum likelihood estimation (MLE): using training data $x_t, t \in \mathcal{T}$ corresponding to class c, the MLE estimate of the parameters is the one which maximizes the joint probability of the training data to be represented by this model

$$\tilde{\theta} = \underset{\theta \in \Theta}{\operatorname{argmax}}\, p_\theta(x_1, \ldots, x_T).$$

Under the assumption that the observations were drawn independently, this can be written as

$$\begin{aligned}\tilde{\theta} &= \operatorname*{argmax}_{\theta\in\Theta} \prod_{t\in\mathcal{T}} p_\theta(x_t) \\ &= \operatorname*{argmax}_{\theta\in\Theta} \sum_{t\in\mathcal{T}} \log p_\theta(x_t).\end{aligned}$$

Under this formulation, the training of a model using maximum likelihood can be translated into the maximization of a function of the parameters θ. For complex likelihood densities, this maximization can be a very complex optimization problem.

The following sections will introduce several popular parametric models for representing the likelihood in a statistical pattern recognition task. Section 1.2 describes the Gaussian model, which has been used in popular classification methods such as linear discriminant analysis (LDA) and quadratic discriminant analysis (QDA) [45, Chapter 4]. Section 1.3 describes the Gaussian mixture model (GMM), which is a model commonly used to approximate arbitrary smooth densities.

1.2 The Gaussian Distribution

For a D-dimensional input vector $\boldsymbol{o}$, the Gaussian distribution with mean $\boldsymbol{\mu}$ and positive definite[1] covariance Σ can be expressed as

$$\mathcal{N}(\boldsymbol{o}, \boldsymbol{\mu}, \Sigma) = \sqrt{\frac{|\Sigma^{-1}|}{(2\pi)^D}} e^{-\frac{1}{2}(\boldsymbol{o}-\boldsymbol{\mu})^\top \Sigma^{-1} (\boldsymbol{o}-\boldsymbol{\mu})}.$$

The distribution is completely described by the D parameters representing $\boldsymbol{\mu}$ and the $\frac{D(D+1)}{2}$ parameters representing the symmetric covariance matrix Σ. Because of the positive definiteness constraint on Σ, the covariance parameters are not completely independent, meaning that any collection of real numbers of size $\frac{D(D+1)}{2}$ does not necessarily represent an admissible covariance matrix.

[1] In the following, we will often denote by $\Sigma \succ 0$ the statement "Σ is positive definite"

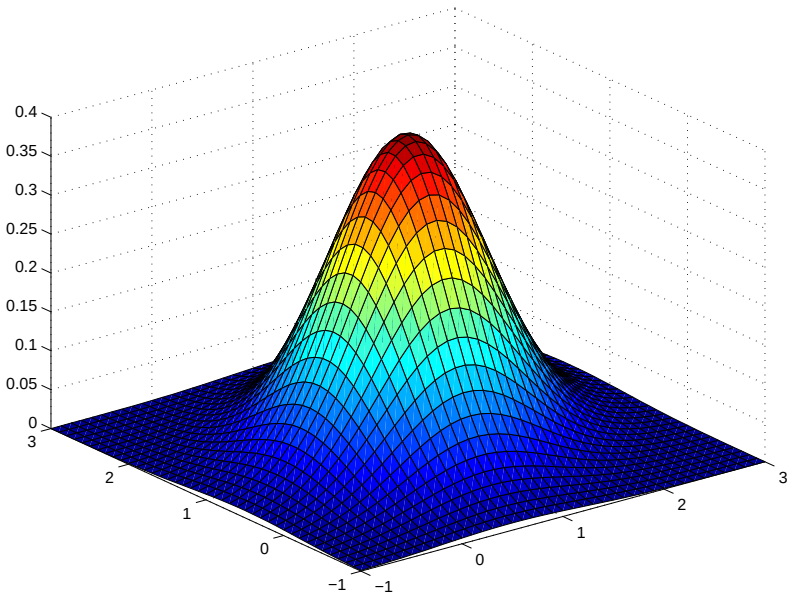

Figure 1.1: Example of a Gaussian distribution in two dimensions

1.2.1 Gaussian Estimation

Maximum likelihood estimation

The log-likelihood of a Gaussian density given a collection of data points $\boldsymbol{o}_t, t \in [1, T]$, drawn with respective frequencies $\gamma_t \in [0, 1]$ can be written:

$$\mathcal{L}_i(\boldsymbol{\mu}_i, \Sigma_i) = \sum_{t=1}^{T} \gamma_{i,t} \log \mathcal{N}(\boldsymbol{o}_t, \boldsymbol{\mu}_i, \Sigma_i). \tag{1.1}$$

The parameters maximizing $\mathcal{L}_i$ have a simple closed form solution [10]:

$$\begin{aligned} \boldsymbol{\mu} &= \frac{1}{\sum_u \gamma_u} \sum_t \gamma_t \boldsymbol{o}_t, \\ \Sigma &= \frac{1}{\sum_u \gamma_u} \sum_t \gamma_t (\boldsymbol{o}_t - \boldsymbol{\mu})(\boldsymbol{o}_t - \boldsymbol{\mu})^\top. \end{aligned}$$

In general, the training data is assumed to have been drawn from independent observations, which is why one usually assumes $\gamma_t \equiv 1$ for all training samples. However,

as we will see in 1.3, when training Gaussians within GMM, it is not the case. When implementing the maximum likelihood estimation of a Gaussians, it is usually more efficient to accumulate a set of independent sufficient statistics over the data:

$$\begin{aligned} c &= \sum_t \gamma_t, \\ \boldsymbol{f} &= \sum_t \gamma_t \boldsymbol{o}_t, \\ S &= \sum_t \gamma_t \boldsymbol{o}_t \boldsymbol{o}_t^\top, \end{aligned}$$

and in a second step use these sufficient statistics to compute the parameters of the distribution:

$$\boldsymbol{\mu} = \frac{1}{c}\boldsymbol{f}, \tag{1.2}$$

$$\Sigma = \frac{1}{c}S - \boldsymbol{\mu}\boldsymbol{\mu}^\top. \tag{1.3}$$

1.2.2 Gaussian Evaluation

The log-likelihood of a Gaussian can be written as a simple quadratic form

$$\mathcal{L}(\boldsymbol{o}) = \alpha - \frac{1}{2}(\boldsymbol{o}-\boldsymbol{\mu})^\top \Sigma^{-1}(\boldsymbol{o}-\boldsymbol{\mu}), \tag{1.4}$$

with

$$\alpha = \frac{1}{2}(\log|\Sigma^{-1}| - D\log 2\pi).$$

In practice, one would store only α and the twice the inverse covariance matrix in its Cholesky form L [40, Section 3.2.2]. With L such that

$$2\Sigma^{-1} = LL^\top,$$

the computation reduces to

$$\begin{aligned} \boldsymbol{\delta} &= \boldsymbol{o} - \boldsymbol{\mu}, \\ \boldsymbol{\xi} &= L^\top \boldsymbol{\delta}, \\ \mathcal{L} &= \alpha - \boldsymbol{\xi}^\top \boldsymbol{\xi}. \end{aligned} \tag{1.5}$$

The computational cost, measured in number of multiplies, is $\frac{D(D+1)}{2}$, and the storage requirement in memory is: $\frac{D(D+1)}{2} + D + 1$. This means that the global complexity of the Gaussian model is in $\mathcal{O}(D^2)$. This non-linear growth of the complexity of the Gaussian model makes it vastly more costly in high dimensions than in low dimensions. For systems using a large number of Gaussians, this cost can become a limiting factor with respect to the scalability of the pattern recognition system. In particular, systems using Gaussian mixture models, which are introduced in the next section, are susceptible to these limitations by involving a large number of Gaussians to represent the density of each class.

1.3 Gaussian Mixture Models

A GMM for a D-dimensional input vector $\boldsymbol{o}$, composed of M Gaussians with priors w_i, means $\boldsymbol{\mu}_i$ and covariances Σ_i can be expressed as

$$g(\boldsymbol{o}) = \sum_{i=1}^{M} w_i \mathcal{N}(\boldsymbol{o}, \boldsymbol{\mu}_i, \Sigma_i),$$

where the $w_i \in [0, 1]$ satisfy

$$\sum_{i=1}^{M} w_i = 1. \tag{1.6}$$

One common way to look at a GMM is to consider M independent Gaussian sources with each a probability w_i of generating an input. The model is thus *doubly stochastic*: at each time index, a source is selected with some probability w_i, and

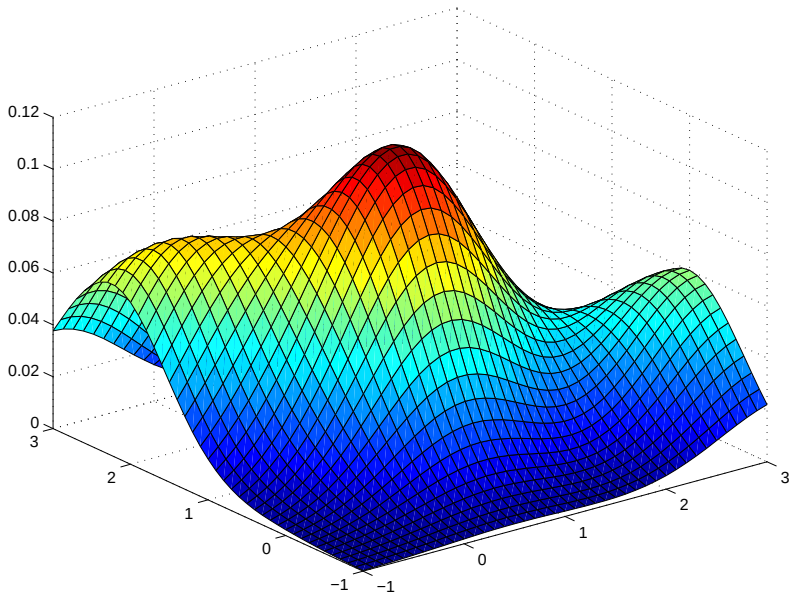

Figure 1.2: Example of a Gaussian mixture model in two dimensions

that source then generates a Gaussian signal. There is by structure no assumption in this model of any relationship between the distinct Gaussian sources: given the probability distribution $w_i, i \in [1, M]$ of the input samples across components, the parameters of Gaussian i will be assumed independent from the parameters of any other Gaussian j in the mixture. In actuality, GMM are however more often used to model a single source with a complicated density than separate sources. In this case, the assumption that the distinct mixture components are unrelated does not necessarily hold. In particular, global statistical features of the signal such as its global correlation structure will be reflected at the component level. The models introduced in Chapters 2, 3 and 4 will remove this assumption by explicitly modeling the relationship between distinct components of a GMM.

GMM have been used in a variety of pattern recognition contexts. In ASR, they have been made popular by their very simple integration into the hidden Markov model (HMM) framework [50]. In image compression [1] and classification [85], GMM are beginning to show a lot of promise. Recent theoretical considerations [43] showed

that GMM also play an interesting role in the design of robust quantization techniques.

1.3.1 GMM Estimation

There are several approaches to the design of GMM [10, 42]. We will briefly describe here the most popular one based on the expectation maximization (EM) algorithm [21].

The EM algorithm is an iterative method to perform maximum likelihood estimation of a model based on incomplete data. In the case of a mixture model, we will assume that the data as a being generated by a set of M distinct sources, but that we only observe the input observation $\boldsymbol{o}_t$ without knowing from which source it comes. What would be qualified as a *complete* observation would be the joint observation of $\boldsymbol{o}_t$ and an indicator variable $\gamma_{t,i}, i \in [1, M]$ which would be 1 for the source the output was generated from, and zero for the others. Where the complete data known, the estimation problem would amount to estimating each Gaussian component using ML. The EM algorithm uses an expected value of the likelihood of the complete data derived from the incomplete information to estimate the GMM parameters.

In its general formulation, the EM algorithm maximizes the likelihood of incomplete data by choosing initial parameters, and then alternating between two steps:

1. the E (Expectation) step: find the expected value of the log-likelihood of the *complete* data, given the observed data and the current parameter estimates,

2. the M (Maximization) step: maximizes this expected value with respect to the parameters to estimate.

Each iteration of the algorithm is guaranteed to increase the likelihood of the incomplete data.

The "E" step is straightforward for GMM: given all the parameters, the probability of a given input sample $\boldsymbol{o}_t$ to have been drawn from component i is

$$\gamma_{i,t} = \frac{w_i \mathcal{N}(\boldsymbol{o}_t, \boldsymbol{\mu}_i, \Sigma_i)}{\sum_{j=1}^{M} w_j \mathcal{N}(\boldsymbol{o}_t, \boldsymbol{\mu}_j, \Sigma_j)}. \tag{1.7}$$

The expected log-likelihood of the complete data can be derived to be

$$Q(w_1, \ldots, w_M, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_M, \Sigma_1, \ldots, \Sigma_M) = \sum_{i=1}^{M} \log w_i \sum_{t=1}^{T} \gamma_{i,t} + \sum_{i=1}^{M} \sum_{t=1}^{T} \gamma_{i,t} \log \mathcal{N}(\boldsymbol{o}_t, \boldsymbol{\mu}_i, \Sigma_i).$$

The maximization of Q, often called the *auxiliary function* of the EM algorithm, is now much simplified. Using a Lagrange multiplier to enforce the constraint in Equation 1.6, the weights can be shown to be [10]

$$w_i = \frac{1}{T} \sum_{t=1}^{T} \gamma_{i,t}$$

which matches what we would expect intuitively: the probability of an input to have been generated by source i is the sample average of all the probabilities for each data point.

The reestimated values for the means and covariances can be derived by setting the corresponding partial derivative of Q to zero, or by simply remarking that since the parameters for each component are independent of each other, maximizing Q with respect to parameters of component i amounts to maximizing

$$Q(\boldsymbol{\mu}_i, \Sigma_i | \ldots) = \sum_{t=1}^{T} \gamma_{i,t} \log \mathcal{N}(\boldsymbol{o}_t, \boldsymbol{\mu}_i, \Sigma_i) = \mathcal{L}_i(\boldsymbol{\mu}_i, \Sigma_i).$$

This is the log-likelihood of a Gaussian with parameters $\boldsymbol{\mu}_i$ and Σ_i, for the training data $\boldsymbol{o}_t, t \in [1, T]$ drawn with prior probability $\gamma_{i,t}$ (Equation 1.1). Thus, the maximization corresponds to solving the maximum likelihood estimation of a Gaussian as described in Section 1.2.

In summary, the simple EM training algorithm proceeds as described in Table 1.1. Several variants of the algorithm exist. A notable simplification consists of assuming that each training data sample is generated by the most likely component only, as opposed to being generated with each component with some probability $\gamma_{i,t}$. In effect, this amounts to replacing the loop around steps 3, 4, and 5 by:

- 3': find $i = \text{argmax}_j \gamma_{j,t}$,
- 4': update the sufficient statistics of Gaussian i.

This simplification makes the EM algorithm look more like the Lloyd algorithm used in vector quantization (VQ) [41]. One of the main advantages is that it tends to converge faster if the initial Gaussians in the mixture are initially very close to each other: by performing a hard assignment of the data to one Gaussian or the other, the data is partitioned more quickly into clusters modeled by individual Gaussians. A drawback of this method is that a Gaussian is assigned a subset of the whole training data, as opposed to the entire set weighted by the priors. This can cause the Gaussian covariances to become ill-conditioned or even singular much more easily [84].

```
1   generate initial parameters
2   for EM iteration = 1 to N
      for data = 1 to T
        for Gaussian = 1 to M
3         compute prior γ_{i,t} (Equation 1.7)
          accumulate sufficient statistics:
4         c_i ← c_i + γ_{i,t}
5         f_i ← f_i + γ_{i,t} o_t
6         S_i ← S_i + γ_{i,t} o_t o_t^⊤
        end for
      end for
      for Gaussian = 1 to M
7       reestimate parameters (Equations 1.2 and 1.3)
      end for
    end for
```

Table 1.1: Overview of the EM algorithm

Several variations on the EM algorithm are meant to speed up the rate of convergence [61]. However, the convergence properties of the algorithm appear to depend most significantly on the determination of good initial parameters. Several schemes exist for the purpose of determining good initial parameters. A very simple and effective method [69] consists of starting with a single Gaussian, split it into two, and perturbate the means by a small amount. The "VQ-like" version of the EM algorithm

described above is then used to quickly separate the split Gaussian into distinct regions of the space, followed by iterations of the full EM algorithm to come up with the final parameters. The splitting is then performed further, with an eventual remerging of the components with low priors until some stopping criterion (e.g. number of Gaussians, or likelihood on held-out dataset) is reached.

1.3.2 GMM Evaluation

The memory usage of a GMM with M components is M times the size of a single Gaussian, with the single addition of the M mixture weights. In terms of computational load, computing the log-likelihood of a GMM looks much more complex than evaluating a single Gaussian, but with some minimal approximation, we will see that it actually also scales almost linearly in the number of components in the mixture.

The log-likelihood can be written as

$$\mathcal{L}(\boldsymbol{o}) = \log \left[\sum_{i=1}^{M} w_i \mathcal{N}(\boldsymbol{o}, \boldsymbol{\mu}_i, \Sigma_i) \right].$$

The log-likelihood of a single component i is

$$\mathcal{L}_i(\boldsymbol{o}) = \log w_i + \log \mathcal{N}(\boldsymbol{o}, \boldsymbol{\mu}_i, \Sigma_i).$$

Thus the complete likelihood can be expressed as:

$$\mathcal{L}(\boldsymbol{o}) = \log \left[\sum_{i=1}^{M} \exp \mathcal{L}_i(\boldsymbol{o}) \right].$$

Because of the wide dynamic range of Gaussian likelihoods, this is very often simplified by taking the maximum likelihood over all components instead of the sum:

$$\mathcal{L}(\boldsymbol{o}) \sim \log \left[\max_i \exp \mathcal{L}_i(\boldsymbol{o}) \right] = \max_i \mathcal{L}_i(\boldsymbol{o}).$$

This approximation makes the evaluation of a GMM about M times as costly as the evaluation of a single Gaussian. However, without resorting to such extreme

simplification, the global mixture log-likelihood can be evaluated with approximately the same complexity by sorting the component log-likelihood in decreasing order. Assume that the sorting maps index i to $\sigma(i)$. The log-likelihood can be obtained by computing

$$\begin{aligned}
\mathcal{G}_{M-1}(\boldsymbol{o}) &= \log\left[\exp\mathcal{L}_{\sigma(M-1)}(\boldsymbol{o}) + \exp\mathcal{L}_{\sigma(M)}(\boldsymbol{o})\right] \\
&= \mathcal{L}_{\sigma(M-1)} + \log\left[1 + \exp\left(\mathcal{L}_{\sigma(M)}(\boldsymbol{o}) - \mathcal{L}_{\sigma(M-1)}(\boldsymbol{o})\right)\right] \\
\mathcal{G}_{M-2}(\boldsymbol{o}) &= \log\left[\exp\mathcal{L}_{\sigma(M-2)}(\boldsymbol{o}) + \exp\mathcal{G}_{M-1}(\boldsymbol{o})\right] \\
&= \mathcal{L}_{\sigma(M-2)} + \log\left[1 + \exp\left(\mathcal{G}_{M-1}(\boldsymbol{o}) - \mathcal{L}_{\sigma(M-2)}(\boldsymbol{o})\right)\right] \\
&\vdots \\
\mathcal{G}_{1}(\boldsymbol{o}) &= \log\left[\exp\mathcal{L}_{\sigma(1)}(\boldsymbol{o}) + \exp\mathcal{G}_{2}(\boldsymbol{o})\right] \\
&= \mathcal{L}_{\sigma(1)} + \log\left[1 + \exp\left(\mathcal{G}_{2}(\boldsymbol{o}) - \mathcal{L}_{\sigma(1)}(\boldsymbol{o})\right)\right].
\end{aligned}$$

It is easy to see by induction that

$$\mathcal{G}_1(\boldsymbol{o}) = \mathcal{L}(\boldsymbol{o}),$$

which means that the global log-likelihood can be computed from the individual Gaussian log-likelihood using $2M$ sums and M evaluations of the function

$$f(x) = \log(1 + e^x).$$

Because the Gaussians were ordered in order of likelihood, $x \leq 0$, and $f(x) \in [0, \log 2]$, which makes the function easy to tabulate to a reasonable degree of precision. As a result, the complete log-likelihood can be approximated to an arbitrary precision without any significant computational overhead.

These computational savings can be further improved by subsetting the Gaussians that are considered in the evaluation. The Gaussians that are the most "distant" to the data point considered will not contribute to the log-likelihood by any significant amount and can thus been pruned based on an inexpensive assessment of that distance. Several schemes have been proposed, such as the BBI algorithm based on

decision trees [33], or the shortlist method based on tree-structured vector quantization (TSVQ) [12, 62].

1.4 Covariance Modeling

It is interesting to look at how parameters get "allocated" when using a Gaussian density. Table 1.2 illustrates the following point: while in low dimensions the parameters describing a Gaussian – i.e. its degrees of freedom – are within the same order of magnitude for both the mean and the covariance, in high dimensions, the covariance matrix becomes overwhelmingly large compared to the mean vector.

D	number of mean parameters	number of covariance parameters	proportion of covariance parameters
1	1	1	50%
20	20	210	91%
100	100	5050	98%

Table 1.2: Parameter Allocation in a Gaussian in various dimensions

This explosion of the number of parameters is obviously due to the quadratic nature of the covariance matrix. It translates into several problems when using high-dimensional Gaussians for modeling:

1. the storage requirements for a single Gaussian are large,

2. the log-likelihood computation (Equation 1.4), which is dominated by the matrix product in Equation 1.5, is very expensive,

3. the number of training samples required to robustly estimate the covariance parameters is large.

The maximum likelihood estimator of a covariance matrix is not well conditioned when the ratio of the number of training samples to the dimensionality is small: in the limit, while a mean vector is still well defined if a single input vector is assigned to the component, the covariance matrix is singular if there are fewer input vectors than

the dimensionality of the space. A good introduction to these issues can be found in [56, Chapter 1].

For these reasons, several covariance modeling strategies have been proposed. A broad class of strategies can be grouped under the term of *regularizing methods*. These try to address the shortcomings of the ML estimator, without looking at the other scalability issues. Overall, these techniques tend to shrink the ML estimator of the covariance matrix towards the identity I by some controlled amount ϵ in order to improve its conditioning:

$$\Sigma \leftarrow (1 - \epsilon)\Sigma + \epsilon I.$$

More can be found on this vast subject in [44, 54, 56, 71]. These methods are corrective measures in situations where the data is inadequate for the number of parameters to be estimated. Moreover, they do not fit into the maximum likelihood framework, which can be a stability issue when the covariance estimates are used in conjuction with the EM algorithm. In addition, they do not address the other issues (storage size and computational cost) which are plaguing covariance matrices in large dimensions.

For these reasons, another class of covariance models, which can be broadly referred as *tying methods* are much more interesting in the context of large GMM. The general idea of these models stems from an observation made in Section 1.3: the parametric form of a GMM assume that the parameter sets of individual Gaussian components are independent of each other, whereas it is in general the case, if the GMM is modeling a single complex input signal for example, that the various components are intimately related.

As an example, let us assume that the input signal is globally decorrelated across feature components. It is then reasonable to consider a mixture model in which all covariance matrices are diagonal. The benefits of a diagonal GMM model, when applicable, are that the number of parameters in the covariance matrix is now equal to the dimensionality. As a consequence, the storage requirements are linear in the

dimensionality as well, and that the computational cost is $2D$ products per Gaussian:

$$\mathcal{L}(\boldsymbol{o}) = c - \sum_{d=1}^{D} \frac{1}{2\sigma_d^2}(o_d - \mu_d)^2. \tag{1.8}$$

This simple approximation has been pivotal to the popularization of GMM as acoustic models for real-time ASR systems. To accommodate the constraint of decorrelation of the feature components, it is necessary to process the input features using a decorrelating transform. In speech processing, this has been mostly achieved through a discrete cosine transform, which, under Markov conditions, approximates the Karhunen-Loève transform [18], or through the use of linear discriminant analysis [29] on a higher-dimensional feature space in order to extract orthogonal features.

Maximum likelihood methods estimating a decorrelating transform have also been developed [16]. In [11], a unit-triangular matrix with a sparse structure is used. In the *semi-tied* covariance model [35], an unconstrained real matrix transform is used, and an efficient ML algorithm to estimate the transform is proposed. In all these techniques, the resulting covariance model can be interpreted as a shared (or *tied*) transform U, combined with a Gaussian dependent diagonal covariance Δ_i, such that the full model of the covariance is

$$\Sigma_i = U^\top \Delta_i U.$$

Since any covariance matrix can be expressed as: $\Sigma_i = U_i^\top \Delta_i U_i$, the use of a decorrelating transform can be interpreted as a tying of the transform U_i across Gaussians in the mixture. The assumption that underlies this type of model is thus that there exist a rotation and scaling of the feature space that globally decorrelates the Gaussians, which means that the principal axes of the Gaussians are all aligned in the feature space. This is a very strong assumption which limits the validity of these tying techniques.

These techniques and some others will be further discussed in Chapter 2. The model introduced in that chapter is in fact one tying technique which makes much weaker assumptions on the joint structure of the covariances in the mixture.

Chapter 2

Mixtures of Inverse Covariances

Since there is much redundancy in the parameters of the covariance of a typical GMM, it is natural to consider explicitly representing this information using fewer parameters that can be estimated robustly, and which will result in a more compact representation of the probability density. By treating the covariance parameters as a highly redundant input signal, techniques of lossy compression such as vector quantization can be applied to the problem. The mixture of inverse covariances (MIC) model [75, 76] is the result of such *parametric compression*. In contrast with a "hard" clustering technique such as VQ, the model uses a linear combination of cluster codewords as an encoding of the parametric model. In that respect, this model is similar to mixture models such as generalized additive models [45, Chapter 9] or fuzzy clustering [8, Chapter 8], which make a soft decision when associating a data sample to a mixture component.

2.1 The MIC Model

The MIC model represents the inverse covariances in a GMM as

$$\Sigma_i^{-1} = \sum_{k=1}^{K} \lambda_{k,i} \Psi_k. \tag{2.1}$$

- $\Psi_k, k \in [1, K]$ is small size codebook of prototype symmetric matrices.

- $\lambda_{k,i} \in \mathbb{R}$ are the mixture weights which represent the encoding of a given inverse covariance Σ_i^{-1} using that codebook. Note that unlike the case of mixtures of densities, the $\lambda_{k,i}$ are not constrained to sum to one or even to be positive.

Inverse covariance matrices are sometimes referred to as "precision matrices" or "concentration matrices". The choice of modeling inverse covariances as opposed to covariances is driven by the log-likelihood of a Gaussian, which has a simple expression as a function of the inverse covariance (Equation 1.4). There are several arguments in favor of a soft clustering scheme as opposed to a hard one in the current context. In particular, the overall scale of typical covariance matrices can vary dramatically across components of a GMM, and this feature is well captured in the Gaussian-dependent weights of the MIC, whereas the codewords in a hard clustering scheme would have a fixed scaling. In addition, the soft clustering model retains a number of Gaussian-specific parameters — the K weights, K being anywhere between 1 and $\frac{D(D+1)}{2}$, which makes it much more expressive at a controlled level of complexity. In Section 2.4, we will see that the computational complexity of this model is also directly proportional to K.

2.2 Class-based Prototype Allocation

As the number of matrices to be modeled grows, the number of prototypes required to model all the covariances accurately might grow to the point of making the joint estimation of all the prototypes as well as the Gaussian likelihood evaluation computationally expensive. (See Section 2.4 for a detailed analysis of the computational cost of these models).

For more efficient modeling, one might consider using a class-based decomposition of the Gaussian mixture and allocate a distinct pool of prototypes to each class:

$$\forall i \in \mathcal{C},\ \Sigma_i^{-1} = \sum_{k=1}^{K_\mathcal{C}} \lambda_{k,i} \Psi_{\mathcal{C},k}.$$

This limits the number of Gaussian-specific parameters to $K_\mathcal{C}$, while allowing the pool

of prototypes to grow much larger. The determination of appropriate classes can be dictated by the problem at hand, or derived in a principled way in the same vein as classified VQ approaches [41, Section 12.5].

2.3 Related Work

By imposing some additional structure onto the prototypes, many different covariance models can be expressed in the form of MIC. Consider the symmetric canonical basis of matrices $E_{i,j}$, whose elements are 0 everywhere except at locations (i, j) and (j, i) where they are 1. The unconstrained full covariance model can be expressed by having

$$\Psi_k, k \in [1, D(D+1)/2] \equiv E_{i,j}, 1 \leq j \leq i \leq D,$$

and the diagonal covariance model by having

$$\Psi_k \equiv E_{k,k}, k \in [1, D].$$

It is clear that by relaxing these strong constraints on the structure of the prototypes, a better model can be achieved with the same number (K) of Gaussian-specific parameters. Several well-known covariance models fall under this general class. Semi-tied covariances [35] express each inverse covariance matrix Σ_i^{-1} using a diagonal inverse covariance matrix D_i and an unconstrained real transform A shared across Gaussians:

$$\Sigma_i^{-1} = AD_iA^\top.$$

The computational benefits of this model are obvious, since it differs from a plain diagonal model by a simple transform of the feature vector. As was remarked in [65], by considering the rows $\boldsymbol{\eta}_k$ of the transform matrix A, and $d_{k,i}$ the diagonal terms of matrix D, this can be rewritten as

$$\Sigma_i^{-1} = \sum_{k=1}^{D} d_{k,i}\boldsymbol{\eta}_k\boldsymbol{\eta}_k^\top.$$

Since any rank-one matrix can be expressed uniquely as the product of a vector and its transpose, the semi-tied covariance model is an instance of the MIC model in Equation 2.1 with $K = D$, and with sole constraint: $\text{Rank}(\Psi_k) \equiv 1$.

Factored sparse inverse covariance matrices [11] are a more constrained model in which the transform A is an upper triangular matrix with ones along the diagonal, and possibly a sparse structure. The main benefit of this model is that the optimization of the transform matrix in the EM framework is now linear, owing to the fact that $|\Sigma_i^{-1}| = |D_i|$ is independent of the transform. In this case, the class of prototype matrices that correspond to this model is a collection of rank-one block-diagonal matrices generated by the family of vectors:

$$\boldsymbol{\eta}'_k = [0 \dots 0 \underbrace{1}_{k} \eta'_{k,k+1} \dots \eta'_{k,D}]^\top.$$

In [65], the extended maximum likelihood linear transform (EMLLT) model was introduced, generalizing the semi-tied approach to $K > D$. In [22], an alternative approach using rank-1 matrices was also proposed. Finally, a recent series of publications presented a model analogous to MIC called subspace of precisions and means (SPAM) [3, 4, 79], which independently generalized the mixture approach to matrices of any rank. See [27] for a comparison of some of these methods.

2.4 Gaussian Evaluation

The log-likelihood of Gaussian i for observation vector $\boldsymbol{o}$ can be written as

$$\mathcal{L}_i(\boldsymbol{o}) = c_i - \frac{1}{2}(\boldsymbol{o} - \boldsymbol{\mu}_i)^\top \Sigma_i^{-1} (\boldsymbol{o} - \boldsymbol{\mu}_i),$$

where:

$$c_i = \frac{1}{2}(\log |\Sigma_i^{-1}| - D \log 2\pi).$$

When $\Sigma_i^{-1} = \sum_{k=1}^{K} \lambda_{k,i} \Psi_k$,

$$\mathcal{L}_i(\boldsymbol{o}) = \underbrace{c_i - \frac{1}{2}\boldsymbol{\mu}_i^\top \Sigma_i^{-1} \boldsymbol{\mu}_i}_{c_i'} - \sum_{k=1}^{K} \lambda_{k,i} \underbrace{\frac{1}{2}\boldsymbol{o}^\top \Psi_k \boldsymbol{o}}_{\omega_k} - \underbrace{\left[-\Sigma_i^{-1}\boldsymbol{\mu}_i\right]}_{\boldsymbol{\nu}_i}^\top \boldsymbol{o}.$$

The term $\frac{1}{2}\boldsymbol{\mu}_i^\top \Sigma_i^{-1} \boldsymbol{\mu}_i$ can be absorbed into the constant c_i'. The vector $\boldsymbol{\omega} : [\omega_1 \dots \omega_K]^\top$ is independent of the Gaussian and can be computed as an additional K-dimensional feature vector appended to $\boldsymbol{o}$. $\boldsymbol{\nu}_i$ is a D-dimensional Gaussian-specific vector, which leads to expressing the Gaussian computation in terms of:

- An extended feature vector: $\boldsymbol{o}' = \begin{bmatrix} \boldsymbol{o} \\ \boldsymbol{\omega} \end{bmatrix}$,
- A Gaussian-specific parameter vector: $\boldsymbol{\nu}_i' = \begin{bmatrix} \boldsymbol{\nu}_i \\ \boldsymbol{\Lambda}_i \end{bmatrix}$, with $\boldsymbol{\Lambda}_i = \begin{bmatrix} \lambda_{1,i} \\ \vdots \\ \lambda_{K,i} \end{bmatrix}$.

Using this notation, the likelihood can be expressed as a scalar product between these two $K + D$ dimensional vectors:

$$\mathcal{L}_i(\boldsymbol{o}) = c_i' - \boldsymbol{\nu}_i'^\top \boldsymbol{o}'.$$

This computation requires $D + K$ sums and products, to be compared with $2D$ for a diagonal Gaussian. Note that K can be smaller than D, in which case the Gaussians are less expensive to evaluate than in the diagonal case.

The front-end overhead is limited to the computation of $\boldsymbol{\omega}$. When the prototypes are positive definite, the quadratic form can be decomposed using a Cholesky factorization:

$$\frac{1}{2}\Psi_k = L_k L_k^\top.$$

The resulting computation

$$\boldsymbol{\xi}_k(\boldsymbol{o}) = L_k^\top \boldsymbol{o} \;\Rightarrow\; \omega_k = \boldsymbol{\xi}_k(\boldsymbol{o})^\top \boldsymbol{\xi}_k(\boldsymbol{o})$$

uses on the order of $\frac{1}{2}KD^2$ multiplications.

When using a class-based approach with C classes, this overhead grows as $\frac{1}{2}CKD^2$, unless the classes to which the significant Gaussians belong can be predicted, in which case only the prototypes belonging to those classes need to be evaluated, and thus computations can be saved.

Note that, using this formulation, it is possible to perform partial evaluation of the Gaussian for the purposes of quickly pruning insignificant Gaussians in the mixture. Since

$$\boldsymbol{\omega}^\top \boldsymbol{\Lambda}_i = \boldsymbol{o}^\top \Sigma_i^{-1} \boldsymbol{o} \geq 0,$$

we have the inequality:

$$\begin{aligned} \mathcal{L}_i(\boldsymbol{o}) &= c_i' - \boldsymbol{\nu}_i{}^\top \boldsymbol{o} - \boldsymbol{\omega}^\top \boldsymbol{\Lambda}_i \\ &\leq c_i' - \boldsymbol{\nu}_i{}^\top \boldsymbol{o}. \end{aligned}$$

This upper bound on the likelihood can be tested without any additional computation, prior to a full evaluation of the Gaussian, in order to determine if the Gaussian is significant or not.

It is also common [24] to weight the Gaussian log-likelihood in a mixture by a factor $\alpha \leq 1$ such that

$$f(\boldsymbol{o}) = \sum_{i=1}^{M} w_i [\mathcal{N}(\boldsymbol{o}, \boldsymbol{\mu}_i, \Sigma_i)]^\alpha.$$

This exponent typically improves the performance by reducing the dynamic range of the Gaussian scores when diagonal covariances are used. When a MIC model is used, α needs to be tuned for the particular model used. In the limit, the model approximates closely enough the full covariance, the value $\alpha = 1$ is optimal.

2.5 Model Estimation

The sample covariance estimated from the observations $\boldsymbol{o}_t$ and priors $\gamma_{i,t}$ is:

$$\bar{\Sigma}_i = \sum_t \gamma_{i,t}(\boldsymbol{o}_t - \boldsymbol{\mu}_i)(\boldsymbol{o}_t - \boldsymbol{\mu}_i)^\top. \tag{2.2}$$

Given the independent parameters $w_i, \boldsymbol{\mu}_i$, and the sample covariance $\bar{\Sigma}_i$, the parameters of the model (Ψ, Λ), with

$$\begin{aligned} \Psi &= \{\Psi_1, \ldots, \Psi_K\}, \\ \Lambda &= \{\boldsymbol{\Lambda}_1, \ldots, \boldsymbol{\Lambda}_M\}, \end{aligned}$$

can be estimated jointly using the EM algorithm [21]. Using

$$\boldsymbol{x}^\top A \boldsymbol{x} = \mathrm{Tr}(A \boldsymbol{x}\boldsymbol{x}^\top),$$

the auxiliary function can be written as

$$\begin{aligned} Q(\Psi, \Lambda) &= \sum_{i=1}^{M} \sum_t \gamma_{i,t} \left[\log |\Sigma_i^{-1}| - (\boldsymbol{o}_t - \boldsymbol{\mu}_i)^\top \Sigma_i^{-1} (\boldsymbol{o}_t - \boldsymbol{\mu}_i) \right] \\ &= \sum_{i=1}^{M} w_i \left[\log |\Sigma_i^{-1}| - \mathrm{Tr}\left(\Sigma_i^{-1} \bar{\Sigma}_i \right) \right], \end{aligned} \tag{2.3}$$

with the constraint that

$$\Sigma_i^{-1} = \sum_k \lambda_{k,i} \Psi_k.$$

2.5.1 Casting the Problem in Terms of Convex Optimization

Maximum-likelihood estimation of the parameters (Ψ, Λ) of the model can not be performed by a direct method. However, owing to the concavity of $\log |A|$ when A is positive definite [15], and to the linearity of the trace, both the functions $Q(\Psi|\Lambda)$ and $Q(\Lambda|\Psi)$ are concave on the domain $\Sigma_i \succ 0$ (read "the domain in which all the

covariances Σ_i are positive definite"). Moreover, the domains

- $\mathcal{L} : \Lambda \ / \ \{\forall i, \sum \lambda_{k,i} \Psi_k \succ 0\}$,
- $\mathcal{P} : \Psi \ / \ \{\forall i, \sum \lambda_{k,i} \Psi_k \succ 0\}$

are both convex.

Thus, the problem of jointly estimating Ψ and Λ can be decomposed into two convex optimization problems to be solved iteratively:

Maximize $Q(\Lambda\|\Psi)$	Maximize $Q(\Psi\|\Lambda)$
Subject to $\Lambda \in \mathcal{L}$	Subject to $\Psi \in \mathcal{P}$

This iterative optimization of the marginals of Q is an instance of alternating optimization [9], and, as a biconvex optimization problem, is known to be locally convergent. It is interesting to relate this approach to the classic Lloyd clustering [41, Section 6.2] and EM algorithms. The maximization of $Q(\Lambda|\Psi)$ is similar to the nearest neighbor partitioning step of Lloyd, except that the partitioning performed here is a "soft" allocation of the covariance to the various prototypes. In that respect, it is similar to the E step of the EM algorithm applied to GMM, which computes the class allocation weights for each mixture component. The maximization of $Q(\Psi|\Lambda)$, on the other hand, is akin to the centroid computation of the Lloyd algorithm or the M step of the EM algorithm, which both attempt to come up with a better set of component-dependent parameters given the fixed component allocation scheme.

Here, the distortion criterion used for both the "partitioning" and the "centroid computation" is the Q function. Up to constant terms, it is identical to the MDI criterion [55], which has already been used as a criterion to cluster Gaussians [42] in the design of Gaussian mixtures. In the sections that follow, we describe a succession of algorithms for reestimating the weights, initializing the weights, reestimating the prototypes and initializing the prototypes of a MIC.

2.5.2 Reestimation of the Weights

The weight estimation given the prototype covariances can be performed efficiently using a Newton algorithm [17]. The gradient of the auxiliary function can be computed using (see e.g. [14])

$$\frac{\partial \log |A(x)|}{\partial x} = \mathrm{Tr}\left[A^{-1}(x)\frac{\partial A(x)}{\partial x}\right].$$

Thus

$$\frac{\partial}{\partial \lambda_{k,i}} \log |\Sigma_i^{-1}| = \mathrm{Tr}\left(\Sigma_i \Psi_k\right).$$

Since

$$\frac{\partial}{\partial \lambda_{k,i}} \mathrm{Tr}\left(\Sigma_i^{-1}\bar{\Sigma}_i\right) = \mathrm{Tr}\left(\Psi_k \bar{\Sigma}_i\right),$$

the gradient is

$$\frac{\partial Q}{\partial \lambda_{k,i}} = \mathrm{Tr}\left[\Psi_k(\Sigma_i - \bar{\Sigma}_i)\right]. \tag{2.4}$$

In the following, we will sometimes represent a symmetric matrix A in vector form — noted $\boldsymbol{A}^\star$, constructed by stacking together the diagonal $\boldsymbol{a}_0$ and the super-diagonals $\boldsymbol{a}_i, i \in [1, D-1]$ multiplied by $\sqrt{2}$:

$$\boldsymbol{A}^\star = [\boldsymbol{a}_0^\top \sqrt{2}\boldsymbol{a}_1^\top \ldots \sqrt{2}\boldsymbol{a}_{D-1}^\top]^\top.$$

The $\sqrt{2}$ factor ensures that

$$\mathrm{Tr}(AB) = \boldsymbol{A}^{\star\top}\boldsymbol{B}^\star.$$

This identity maps a symmetric matrix representation and its associated Frobenius norm into a vector representation with minimal dimensionality ($\frac{D(D+1)}{2}$) and the more familiar L_2 norm. It is also a memory-efficient way of representing symmetric matrices

which is well suited to the implementation of the reestimation algorithms. Using this convention, and denoting $P = [\mathbf{\Psi}_1^\star \dots \mathbf{\Psi}_k^\star]$, we can write Equation 2.4 as

$$\frac{\partial Q}{\partial \Lambda_i} = P^\top (\mathbf{\Sigma}_i^\star - \bar{\mathbf{\Sigma}}_i^\star). \tag{2.5}$$

The components of the Hessian H can be computed using the identity

$$\frac{\partial A^{-1}(x)}{\partial x} = -A^{-1}(x) \frac{\partial A(x)}{\partial x} A^{-1}(x),$$

which results in

$$\begin{aligned} \frac{\partial^2 Q}{\partial \lambda_{k,i} \partial \lambda_{l,i}} &= \mathrm{Tr}\left[\Psi_k \frac{\partial \Sigma_i}{\partial \lambda_{l,i}}\right] \\ &= -\mathrm{Tr}\left[\Psi_k \Sigma_i \Psi_l \Sigma_i\right]. \end{aligned}$$

Under the mild assumption of linear independence between the Ψ_k, the Hessian is invertible.

Proof. If $\{\Psi_k, k \in [1, K]\}$ is an independent family, since Σ_i is full rank, so is $\{\Psi_k \Sigma_i, k \in [1, K]\}$. Consider the $K \times \frac{D(D+1)}{2}$ matrix Ω_i whose k^{th} column lists all the entries of $\Psi_k \Sigma_i$ in any consistent order. The matrix Ω_i is nonsingular, and we have:

$$H_i = -\Omega_i^\top \Omega_i.$$

Thus for any $\boldsymbol{X} \neq 0$,

$$\boldsymbol{X}^\top H_i \boldsymbol{X} = -(\Omega_i \boldsymbol{X})^\top (\Omega_i \boldsymbol{X}) < 0,$$

and consequently H_i is negative definite, thus invertible. □

In the unlikely case of some prototypes being linearly dependent on others, all the inverse covariances expressed as a linear combination of the prototypes can always be expressed as a function of a smaller set of independent ones, for which the Hessian

will be invertible.

The optimization can be noticeably simplified by remarking that for any covariance Σ,

$$\boldsymbol{\Sigma}^{\star\top}\boldsymbol{\Sigma}^{-1\star} = \mathrm{Tr}(\Sigma\Sigma^{-1}) = D \ \ (= \text{dimensionality})$$

For $\boldsymbol{\Lambda}$ to be a maximum-likelihood weight vector, the gradient in Equation 2.5 is necessarily zero, and

$$P^\top\boldsymbol{\Sigma}^\star = P^\top\bar{\boldsymbol{\Sigma}}^\star,$$

thus, using

$$\boldsymbol{\Sigma}^{-1\star} = P\boldsymbol{\Lambda},$$

we have

$$\boldsymbol{\Sigma}^{\star\top}P\boldsymbol{\Lambda} = (P^\top\bar{\boldsymbol{\Sigma}}^\star)^\top\boldsymbol{\Lambda} = D.$$

This relationship defines an affine hyperplane orthogonal to

$$\boldsymbol{\Lambda}_0 = D\frac{P^\top\bar{\boldsymbol{\Sigma}}^\star}{\|P^\top\bar{\boldsymbol{\Sigma}}^\star\|^2}, \tag{2.6}$$

in which $\boldsymbol{\Lambda}$ is constrained to live. Denoting U a basis of the orthogonal of $P^\top\bar{\boldsymbol{\Sigma}}^\star$, we have

$$\boldsymbol{\Lambda} = \boldsymbol{\Lambda}_0 + U\boldsymbol{\Lambda}'.$$

The gradient ascent algorithm can now be performed on $\boldsymbol{\Lambda}' \in \mathrm{Span}(U)$, which is of dimension $K-1$, by projecting the Newton update onto $\mathrm{Span}(U)$. The Hessian can be computed easily using Equation 2.6 at each step of the iteration, leading to

an update step of

$$\tilde{\mathbf{\Delta}} = H^{-1}P^{\top}(\mathbf{\Sigma}^{\star} - \bar{\mathbf{\Sigma}}^{\star}), \tag{2.7}$$

which, projected onto Span(U), becomes

$$\mathbf{\Delta} = \tilde{\mathbf{\Delta}} - \frac{\mathbf{\Lambda}_0^{\top}\tilde{\mathbf{\Delta}}}{\|\mathbf{\Lambda}_0\|^2}\tilde{\mathbf{\Delta}}. \tag{2.8}$$

By concavity of $Q(\Lambda|\Psi)$, the algorithm will converge to a global maximum. The Newton update

$$\mathbf{\Lambda} \leftarrow \mathbf{\Lambda} + \gamma\mathbf{\Delta} \tag{2.9}$$

converges after a few iterations. In general $\gamma = 1$, although in the first steps of the iteration it sometimes needs to be reduced to prevent intermediate estimates of $\mathbf{\Lambda}$ to step out of $\mathcal{L}$. Since all the covariances have to be recomputed before the next update step (Equation 2.7) can be performed, this check for positive-definiteness can be made implicitly while inverting the matrices.

2.5.3 Weight Initialization

If the prototypes are positive definite, then $\mathbf{\Lambda}_0 \in \mathcal{L}$.

Proof. The k^{th} element of $\mathbf{\Lambda} = P^{\top}S^{\star}$ is

$$\lambda_k = {\Psi_k^{\star}}^{\top}S^{\star} = \mathrm{Tr}(\Psi_k S).$$

Since Ψ_k and S are positive definite symmetric matrices, $\mathrm{Tr}\,(\Psi_k S) = \lambda_k > 0$. As a consequence, $P\mathbf{\Lambda}_0 = \sum_k \lambda_k \Psi_k^{\star}$ is a linear combination with positive weights of positive definite matrices. From the definition of positive-definiteness,

$$\forall k,\ \boldsymbol{x}^{\top}\Psi_k\boldsymbol{x} > 0,\ \lambda_k > 0 \ \Rightarrow\ \sum_k \lambda_k \boldsymbol{x}^{\top}\Psi_k\boldsymbol{x} > 0,$$

and $P\mathbf{\Lambda}_0$ is positive definite, which implies $\mathbf{\Lambda}_0 \in \mathcal{L}$. □

We will see in Section 2.5.6 that the method that we use for generating the initial prototypes guarantees positive-definiteness, and thus $\mathbf{\Lambda}_0$ can be used to initialize the algorithm.

2.5.4 Simplified Progressive Reestimation Algorithms

An alternative scheme for reestimating the weight parameters is to consider each component of the mixture separately and optimize them progressively: consider a model Φ of Σ^{-1}, which can be for example a MIC model using K prototypes. The idea is to compute a refined model with $K+1$ prototypes by maximizing the likelihood of

$$\Sigma^{-1} = \Phi + \lambda\Psi$$

with respect to the weight λ. The potential advantages of this approach are multiple:

- for any $K' \leq K$, the decomposition which uses K' prototypes is still an admissible MIC model, in the sense that it is positive definite, and is to some extent a ML representation of the inverse covariance. This means that the model can be used with any number of prototypes without retraining,
- because it is a scalar optimization, the progressive method of weight estimation is much simpler and faster than the complete optimization,
- the model can be seen as a series of successive refinements of the covariance estimate, stating from a model of the "average" inverse covariance as prototype Ψ_0, and successive models $\Psi_1, \ldots, \Psi_N$ of the residuals. This means that the decrease in the relative magnitude of the weights, as the decomposition grows larger, will indicate how good an estimate of the covariance is.

However, this approach leads to a suboptimal model for several reasons:

- the model is not globally maximum likelihood,

- when the prototypes are retrained jointly with the weights, Ψ_0 captures the average behavior and is positive definite. However the next "residuals" $\Psi_1, \ldots, \Psi_N$ are not in general positive definite because they model departures from the average behavior. This is an issue with the front-end overhead: the number of multiplications used to evaluate a general matrix as opposed to a positive definite one is $\frac{2D}{D+1} \sim 2$.

The optimization algorithm in this case is extremely simple. We know that if

$$\Sigma^{-1} = \lambda_0 \Psi_0,$$

then from Equation 2.6

$$\lambda_0 = \frac{D}{\mathrm{Tr}(\Psi_0 \bar{\Sigma})}.$$

The weights can then be estimated successively. If

$$\Sigma^{-1} = \Phi + \lambda \Psi,$$

then to maximize the likelihood, the gradient

$$f(\lambda) = \mathrm{Tr}[\Psi(\bar{\Sigma} - \Sigma)]$$

needs to be set to zero. Its derivative is

$$\frac{df}{d\lambda} = \mathrm{Tr}(\Sigma \Psi \Sigma \Psi).$$

Starting with $\lambda = 0$, one can iterate a simple gradient descent

$$\lambda \leftarrow \lambda - \gamma \frac{f(\lambda)}{\frac{\partial f}{\partial \lambda}}$$

until f reaches 0. The convergence of the algorithm is extremely fast, which makes this algorithm a good candidate for an approximate estimation of the model.

2.5.5 Reestimation of the Prototypes

In order to reestimate the prototypes given the weights, the Q function in Equation 2.3 has to be maximized with respect to each prototype Ψ_k. With A a symmetric matrix, using the cofactor decomposition of the determinant, we have (see e.g. [10])

$$\frac{\partial |A|}{\partial A\rfloor_{i,j}} = \begin{cases} \mathcal{A}_{i,j} & \text{if } i = j \\ 2\mathcal{A}_{i,j} & \text{if } i \neq j, \end{cases}$$

where $\mathcal{A}_{i,j}$ are the cofactors of A, and $A\rfloor_{i,j}$ denotes the (i,j) entry of matrix A.

Similarly, if $A = \sum_k \lambda_k A_k$,

$$\frac{\partial |A|}{\partial A_k\rfloor_{i,j}} = \begin{cases} \lambda_k\mathcal{A}_{i,j} & \text{if } i = j \\ 2\lambda_k\mathcal{A}_{i,j} & \text{if } i \neq j. \end{cases}$$

Thus,

$$\begin{aligned} \frac{\partial \log |A|}{\partial A_k} &= \begin{cases} \lambda_k\mathcal{A}_{i,j}/|A| & \text{if } i = j \\ 2\lambda_k\mathcal{A}_{i,j}/|A| & \text{if } i \neq j \end{cases} \\ &= \lambda_k \left[2A^{-1} - \mathrm{Diag}(A^{-1})\right]. \end{aligned}$$

Consequently,

$$\frac{\partial}{\partial \Psi_k} \sum_{i=1}^{M} w_i \log |\Sigma_i^{-1}| = \sum_{i=1}^{M} w_i \lambda_{k,i} [2\Sigma_i - \mathrm{Diag}(\Sigma_i)].$$

With A a symmetric matrix, we also have

$$\frac{\partial \mathrm{Tr}(AB)}{\partial A\rfloor_{i,j}} = \begin{cases} B\rfloor_{i,i} & \text{if } i = j \\ B\rfloor_{j,i} + B\rfloor_{i,j} & \text{if } i \neq j, \end{cases}$$

and thus

$$\frac{\partial \mathrm{Tr}(AB)}{\partial A} = B + B^\top - \mathrm{Diag}(B).$$

We can see that if $A = \sum_k \lambda_k A_k$ and B is symmetric, then

$$\frac{\partial \mathrm{Tr}(AB)}{\partial A_k} = \lambda_k \left[2B - \mathrm{Diag}(B).\right]$$

Thus

$$\frac{\partial}{\partial \Psi_k} \sum_{i=1}^{M} w_i \mathrm{Tr}\left(\Sigma_i^{-1} \bar{\Sigma}_i\right) = \sum_{i=1}^{M} w_i \lambda_{k,i} [2\bar{\Sigma}_i - \mathrm{Diag}(\bar{\Sigma}_i)],$$

Consequently,

$$\frac{\partial Q}{\partial \Psi_k} = \sum_{i=1}^{M} w_i \lambda_{k,i} \left[2\left(\Sigma_i - \bar{\Sigma}_i\right) - \mathrm{Diag}\left(\Sigma_i - \bar{\Sigma}_i\right)\right].$$

For A symmetric,

$$2A - \mathrm{Diag}\left(A\right) \equiv 0 \Leftrightarrow A \equiv 0.$$

As a consequence, we can replace the likelihood gradient by

$$\frac{\partial Q'}{\partial \Psi_k} = \sum_{i=1}^{M} w_i \lambda_{k,i} \left(\Sigma_i - \bar{\Sigma}_i\right). \tag{2.10}$$

The Hessian of the auxiliary function Q' is

$$\begin{aligned} \frac{\partial^2 Q'}{\partial \Psi_k \partial \Psi_k \rfloor_{p,q}} &= \sum_{i=1}^{M} w_i \lambda_{k,i} \frac{\partial \Sigma_i}{\partial \Psi_k \rfloor_{p,q}} \\ &= -\sum_{i=1}^{M} w_i \lambda_{k,i}^2 \Sigma_i \frac{\partial \Psi_k}{\partial \Psi_k \rfloor_{p,q}} \Sigma_i. \end{aligned}$$

Let us denote by $E_{i,j}$ the matrix containing all zeros except ones at locations (i, j)

and (j,i), and by $\boldsymbol{\sigma}_i^j$ the j^{th} column of Σ_i:

$$\begin{aligned}\frac{\partial^2 Q'}{\partial\Psi_k\partial\Psi_k\rfloor_{p,q}} &= -\sum_{i=1}^{M} w_i\lambda_{k,i}^2\Sigma_i E_{p,q}\Sigma_i \\ &= \frac{-1}{1+\delta_{p,q}}\sum_{i=1}^{M} w_i\lambda_{k,i}^2[\boldsymbol{\sigma}_i^p\boldsymbol{\sigma}_i^{q\top}+\boldsymbol{\sigma}_i^q\boldsymbol{\sigma}_i^{p\top}].\end{aligned} \tag{2.11}$$

Since there are only $\frac{D(D+1)}{2}$ of the D^2 entries of the prototype matrix that are independent, we need to represent the matrix in minimal form for the Hessian to be invertible. Using the notation defined in Section 2.5.2, we can write the Newton iteration as

$$\boldsymbol{\Psi}_k^\star \leftarrow \boldsymbol{\Psi}_k^\star + \gamma H^{-1}\sum_{i=1}^{M} w_i\lambda_{k,i}\left(\boldsymbol{\Sigma}_i^\star - \bar{\boldsymbol{\Sigma}}_i^\star\right). \tag{2.12}$$

Note that because of the $\sqrt{2}$ scaling factor of the off-diagonal terms of Ψ_k (represented as $\boldsymbol{\Psi}_k^\star$), and of Σ_i (represented as $\boldsymbol{\Sigma}_i^\star$), the entries of the Hessian matrix need to be scaled accordingly.

The Hessian, however, is not guaranteed to be invertible. In particular, if $2M < D$, it is always singular:

Proof. Each column of the Hessian contains in vector form the entries of the matrix

$$C_{p,q} = -\sum_{i=1}^{M} w_i\lambda_{k,i}^2\Sigma_i E_{p,q}\Sigma_i \quad \text{for } 1 \le q \le p \le D.$$

Let us assume that $2M < D$. Since $\text{Rank}(E_{p,q}) \le 2$, then $\text{Rank}(C_{p,q}) \le 2M$. Thus, the family $\Xi = \{C_{p,q}\ 1 \le q \le p \le D\}$ is contained in the space of symmetric matrices of rank smaller or equal to $2M$, which is a strict subspace of the vector space of symmetric matrices. The vector space of symmetric matrices is of dimensionality $\frac{D(D+1)}{2}$ ($\{E_{p,q}, 1 \le q \le p \le D\}$ is a canonical basis for it), and thus Ξ lives in a space of dimensionality strictly smaller than $\frac{D(D+1)}{2}$. Since the number of vectors in Ξ is $\frac{D(D+1)}{2}$, the family is not linearly independent, and consequently the Hessian is

singular. □

This condition is not necessary, and in most cases the number of covariances in the GMM is large enough for this bound not to be reached. A simple regularization method such as flooring of the eigenvalues will guarantee that a singularity of the Hessian matrix never causes the Newton iteration to abort.

The exact gradient and Hessian could be expensive to compute using these equations because of the potentially large number of covariances in the GMM. However both can be well estimated by adding up the contributions of a small subset of significant Gaussians. A principled way of selecting the Gaussians is to sort them by the magnitude of their relative weight in Equations 2.10 and 2.11, which are $|w_i\lambda_{k,i}|$ for the gradient and $w_i\lambda_{k,i}^2$ for the Hessian, and only accumulate the contributions of the Gaussians with the highest weight. However, it is beneficial for the overall speed of the algorithm not to have to compute the weights for all the Gaussians at each iteration before being able to make the decision whether or not to use them to reestimate the prototypes. It is thus more efficient to select at the beginning of the iterative process a set of significant Gaussians based only on w_i and only run both the weight and prototype reestimation algorithm on those. In the following experiments, fewer than 10% of the Gaussians were used to estimate the gradient, and fewer than 1% were incorporated into the Hessian. As previously, the step size γ has sometimes to be reduced to a smaller value in the first iterations to avoid stepping out of the domain $\mathcal{P}$. Because the update equation (Equation 2.12) involves the covariance matrices, the check for positive-definiteness can be performed at no additional cost during the matrix inversion.

2.5.6 Prototype Initialization

The initial set of prototypes can be generated by a hard clustering scheme: the M Gaussian covariances are clustered down to K initial prototypes by using the Lloyd algorithm. A Kullback-Liebler distance criterion is used, since it is a natural choice of a metric [42] between Gaussians. The distance between the Gaussian means can be ignored, since only the covariances are of interest. In addition, the variations in

the scale of the prototypes — i.e. their determinant – can be normalized for, since these are captured by the weights in Equation 2.1:

$$\Psi_i = |\Sigma_i|\Sigma_i^{-1}. \tag{2.13}$$

The distance measure used for clustering is thus

$$d(\Psi_k, \Psi_l) = \mathbf{\Psi}_k^{\star\top}\mathbf{\Psi}_l^{\star-1} + \mathbf{\Psi}_l^{\star\top}\mathbf{\Psi}_k^{\star-1}. \tag{2.14}$$

For simplicity, the centroid for each cluster is not computed exactly, but approximated with the average of all the covariances allocated to this cluster. As a consequence, each centroid is guaranteed to be positive definite, which allows us to use the simple weight initialization scheme described in Section 2.5.3. Experimentally, it has been observed that the speed of convergence of the global algorithm is much improved when such clustering is applied, as opposed to a more naive initialization scheme.

2.5.7 Implementation of the Algorithm

The implementation of the algorithm on top of a standard EM reestimation is fairly straightforward (Table 2.1). The iterative MIC reestimation scheme — steps 6 to 10 — needs to be implemented at each step of the EM reestimation, after the ML estimation of the sample mixture weights, means and covariances. In the first iteration, the prototypes can be initialized using the VQ scheme described in Table 2.2. Note that at each EM stage, the iteration between the weight estimation (Table 2.3) and prototype reestimation (Table 2.4) need only be carried over a small subset of all the Gaussians in the mixture, since only a fraction of the covariances are used to reestimate the prototypes. In the final iteration however, the weights for all the covariances have to be reestimated.

The only implementation detail worth noting in Table 2.4 is the two-phase approach to the prototype reestimation algorithm. In a first phase (Table 2.4, 1 to 8), the algorithm goes through each prototype and does one Newton update at each pass. In the second phase (Table 2.4, 9 to 17), the Newton iterations are repeated until the

gradient is small enough. The reason for using this approach is that in the first few iterations, all the prototypes are far from the optimum. When updating a particular prototype, the first Newton steps are large in magnitude. This means that the gradient and Hessian estimates for other prototypes, which depend on every prototype in the MIC, will change dramatically at each Newton step which is taken. As a consequence, it is not beneficial to take several consecutive Newton steps in one direction since this direction will change dramatically after one cycle through the prototypes. After a few cycles, however, some prototypes will be close to their optimal, while others will still be very far from it. Cycling through the prototypes and performing one Newton update each time becomes inefficient because the algorithm keeps on updating well estimated prototypes. For this reason, the second phase optimizes the prototypes one at a time until convergence.

Table 2.5 shows typical values for the various iteration loops. These vary somewhat with the dimensionality of the problem, but the overall number of Newton updates is well within the hundreds for both the weights and prototypes, which makes the overall algorithm computationally tractable. Figure 2.1 shows that the likelihood increase from the iterative process typically reaches a plateau in about 6 iterations.

2.6 Model Adaptation

Many popular techniques for model training and adaptation have been developed with the assumption the covariance matrices used in a GMM are diagonal. In the general case, the optimization procedures can be much more complex. In this section, a few common approaches are analyzed and alternative optimization schemes compatible with the use of MIC models are proposed.

2.6.1 Maximum Likelihood Linear Regression (MLLR)

MLLR [57] is a popular methods of adapting GMM with limited amounts of training data. The technique has also been exploited for modeling [36, 69]. In MLLR, each adapted mean vector is a linear transform of the original mean vector. Extending the

1	generate initial GMM (without MIC)
2	for EM iteration = 1 to N
3	compute sufficient statistics from data and model: $\boldsymbol{f}_i = \sum_t \gamma_{i,t}\boldsymbol{o}_t \quad S_i = \sum_t \gamma_{i,t}\boldsymbol{o}_t\boldsymbol{o}_t^\top$
4	compute mixture weights and means: $w_i = \sum_t \gamma_{i,t}/M, \quad \boldsymbol{\mu}_i = \boldsymbol{f}_i/w_i$
5	compute sample covariances (Equation 2.2)
	if N == 1
6	subset covariances based on w_i (Section 2.5.5)
7	initialize prototypes (Section 2.5.6 and Table 2.2)
	end if
	for iteration = 1 to P
8	estimate weights (Sections 2.5.2, 2.5.3, Table 2.3)
9	update prototypes (Section 2.5.5 and Table 2.4)
	end for
10	estimate weights for all covariances (same as step 8)
11	update model
	end for

Table 2.1: Overview of the EM algorithm

source mean vector $\boldsymbol{\mu}_0$ with a bias term

$$\boldsymbol{\nu} \equiv [\boldsymbol{\mu}_0^\top \; 1]^\top,$$

the adapted mean vector $\boldsymbol{\mu}$ is expressed using a transform matrix W that is estimated over a pool of Gaussians in the mixture:

$$\boldsymbol{\mu} = W\boldsymbol{\nu}. \tag{2.15}$$

The transform can be shared across all or part of the Gaussians in the mixture. In the following we will always assume, for simplicity, that the transform is tied across all the Gaussians.

Although variants of the MLLR model that include adaptation of the covariances exist [37], we will only consider here the adaptation of the mean parameters. In this case, the auxiliary function of the EM algorithm [21] to be maximized based on the

```
1   normalize covariance determinants (Equation 2.13)
2   select K initial prototypes out of the M covariances
3   for VQ iteration = 1 to Q
4     for covariance i = 1 to M
5       find closest prototype k (Equation 2.14)
6       accumulate statistics for this centroid:
        Φ_k = Φ_k + Ψ_i,   c_k = c_k + 1
      end for
7     reestimate centroids: Ψ_k = Φ_k / c_k
8     fix empty cells (see e.g. [41])
    end for
```

Table 2.2: Overview of the prototypes initialization

```
1   for covariance = 1 to M
2     initialize weights (Equation 2.6)
      for iteration = 1 to V
3       compute gradient (Equation 2.5)
4       break loop if gradient< ε
5       compute Δ (Equation 2.8)
6       do γ = 1, decreasing
7         update weight vector (Equation 2.9)
8       while covariance not positive definite
      end for
    end for
```

Table 2.3: Overview of the weights reestimation

data reduces to [10]

$$Q(\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_N) = -\sum_{i=1}^{N} w_i (\bar{\boldsymbol{\mu}}_i - \boldsymbol{\mu}_i)^\top \Sigma_i^{-1} (\bar{\boldsymbol{\mu}}_i - \boldsymbol{\mu}_i). \tag{2.16}$$

Here, w_i is the adaptation prior for Gaussian i, Σ_i is the covariance, $\boldsymbol{\mu}_i$ the mean to estimate, and $\bar{\boldsymbol{\mu}}_i$ the sample mean collected from the adaptation data.

In the standard MLLR case, by replacing $\boldsymbol{\mu}$ by its expression in Equation 2.15

```
 1   for iteration = 1 to S
 2     for prototype = 1 to K
 3       estimate gradient (Equation 2.10)
 4       estimate Hessian (Equation 2.11)
 5       do γ = 1, decreasing
 6         update prototype (Equation 2.12)
 7         reestimate gradient (Equation 2.10)
 8       until γ minimizing gradient is found
       end for
     end for
     do (outer loop)
 9     for prototype = 1 to K
         do (inner loop)
10         estimate gradient (Equation 2.10)
11         break inner loop if gradient< ε
12         estimate Hessian (Equation 2.11)
13         do γ = 1, decreasing
14           update prototype (Equation 2.12)
15           reestimate gradient (Equation 2.10)
16         until γ minimizing gradient is found
         loop
       end for
17   loop unless gradient< ε for all prototypes
```

Table 2.4: Overview of the prototypes reestimation

and differentiating, one can obtain the formulas to compute the transform W:

$$C^i = w_i \Sigma_i^{-1},$$

$$Z = \sum_{i=1}^{N} w_i \Sigma_i^{-1} \bar{\boldsymbol{\mu}}_i \boldsymbol{\nu}_i^\top,$$

$$\sum_{i=1}^{N} C^i W \boldsymbol{\nu}_i \boldsymbol{\nu}_i^\top = Z.$$

Assuming that the covariances are diagonal, this equation can be solved row-wise.

EM iterations	$N \sim 1$ or 2
weights/prototypes optimizations	$P = 6$
VQ iterations	$Q = 4$
weight optimization	$V \sim 10$
prototype optimization: initial loop	$S = 4$
prototype optimization: outer loop	~ 4
prototype optimization: inner loop	< 10

Table 2.5: Typical number of iterations

Let $C^i_{j,j}$ be the diagonal elements of C^i:

$$G_j = \sum_{i=1}^{N} C^i_{j,j} \boldsymbol{\nu}_i \boldsymbol{\nu}_i^\top$$

yields the system

$$\boldsymbol{W}_j^\top = G_j^{-1} \boldsymbol{Z}_j^\top .$$

When full covariances are used, the optimization is less simple [34]. A similar model adaptation method can be used by *preconditioning* the GMM. The preconditioning of a GMM is a reparametrization of the model into a space that transforms Equation 2.16 into a simpler "whitened" form. Consider the preconditioned mean:

$$\boldsymbol{\mu}'_i \equiv \sqrt{w_i} L_i^{-1} \boldsymbol{\mu}_i,$$

where L_i is the Cholesky factor of the covariance matrix Σ_i: $\Sigma_i = L_i L_i^\top$. By the change of variable

$$(w_i, \boldsymbol{\mu}_i, \Sigma_i) \rightarrow (w_i, \boldsymbol{\mu}'_i, \Sigma_i),$$

Equation 2.16 can be rewritten

$$Q(\boldsymbol{\mu}'_1, \ldots, \boldsymbol{\mu}'_N) = -\sum_{i=1}^{N} \|\bar{\boldsymbol{\mu}'}_i - \boldsymbol{\mu}'_i\|^2. \tag{2.17}$$

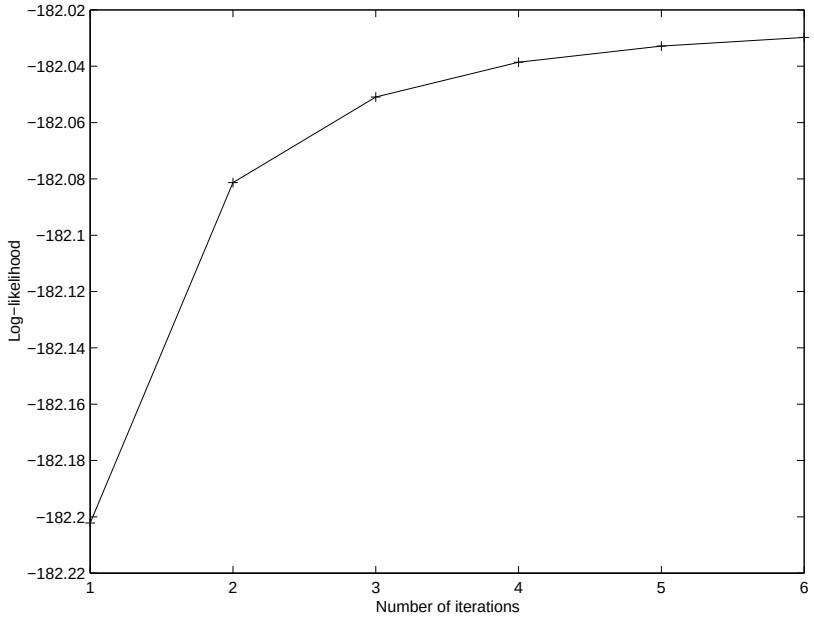

Figure 2.1: Increase in the Q function as a function of the number of iterations. One iteration corresponds to running the prototype reestimation followed by the weight reestimation algorithm once. In the first iteration, the initial prototypes are computed using VQ.

Through this reparametrization, the maximum likelihood estimation is now turned into a minimum mean-square error (MMSE) estimation for which many algebraic tools are available. Note that the mean can be recovered from the preconditioned mean using: $\boldsymbol{\mu}_i = w_i^{-1/2} L_i \boldsymbol{\mu}'_i$, which highlights one of the interesting features of this reparametrization: assume that through adaptation, the preconditioned mean $\boldsymbol{\mu}'$ is displaced by an amount $\boldsymbol{\delta}'$. The corresponding mean vector $\boldsymbol{\mu}$ gets displaced by an amount $\boldsymbol{\delta}$ such that: $\boldsymbol{\delta} \sim w_i^{-1/2} \boldsymbol{\delta}'$, which means that the amount of displacement is implicitly proportional to the amount of training data available.[1] This implies that Gaussians with little adaptation data will get altered proportionally much less than Gaussians with a large amount of adaptation data, which is not the case in standard MLLR.

[1]Note that the $\sqrt{w_i}$ factor in the preconditioning is not essential for the reestimation of a full covariance MLLR model to have a closed form solution.

The preconditioned MLLR model can be simply seen as MLLR performed in preconditioned space:

$$\nu' \equiv [{\mu'_0}^\top \; 1]^\top \text{ and } \boldsymbol{\mu}' = W'\boldsymbol{\nu}'. \tag{2.18}$$

Differentiating and setting the derivative of Equation 2.17 to zero yields

$$Z = \sum_{i=1}^{N} \bar{\boldsymbol{\mu}}'_i {\boldsymbol{\nu}'_i}^\top,$$

$$G = \sum_{i=1}^{N} \boldsymbol{\nu}'_i {\boldsymbol{\nu}'_i}^\top,$$

$$WG = Z \Rightarrow W = ZG^{-1}.$$

As opposed to the standard MLLR case, only one matrix inversion needs to be performed for the whole transform, and no assumption is made on the covariance structure. The auxiliary function has a closed form expression as a function of the sufficient statistics:

$$Q(\boldsymbol{M}) \equiv -\mathrm{Tr}[G^{-1}Z^\top Z].$$

Reestimation of the source Gaussians

In MLLR adaptation, the transform W is the only element of the model that needs to be estimated from the sufficient statistics. However, when MLLR transforms are used for modeling, the source Gaussians need as well to be reestimated from the sufficient statistics of the adapted Gaussians in order to reestimate source Gaussians and transforms iteratively.

In the MLLR case, denoting by $\boldsymbol{\beta}_i$ the bias term for the i^{th} Gaussian, and assuming that all Gaussians in the mixture are transforms of the same Gaussian, the equations

are:

$$Q = \sum_{i=1}^{N} w_i W_i^\top \bar{\Sigma}_i^{-1} W_i,$$

$$\boldsymbol{Z} = \sum_{i=1}^{N} w_i W_i^\top \bar{\Sigma}_i^{-1} (\bar{\boldsymbol{\mu}}_i - \boldsymbol{\beta}_i),$$

$$\boldsymbol{\mu} = Q^{-1} \boldsymbol{Z}.$$

In the preconditioned MLLR case, the equivalent equations are

$$Q' = \sum_{i=1}^{N} W_i^\top W_i,$$

$$\boldsymbol{Z}' = \sum_{i=1}^{N} W_i^\top (\bar{\boldsymbol{\mu}}_i' - \boldsymbol{\beta}_i),$$

$$\boldsymbol{\mu}' = Q'^{-1} \boldsymbol{Z}'.$$

This leads to an algebraic interpretation of the ML estimation. Denoting

$$\Omega = \begin{bmatrix} W_1 \\ \vdots \\ W_N \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_N \end{bmatrix}, \quad \boldsymbol{M}' = \begin{bmatrix} \bar{\mu'}_1 \\ \vdots \\ \bar{\mu'}_N \end{bmatrix},$$

then we have

$$\boldsymbol{\mu}' = (\Omega^\top \Omega)^{-1} \Omega^\top (\boldsymbol{M}' - \boldsymbol{B}).$$

This is the MMSE solution of the system

$$\boldsymbol{M}' = \Omega \boldsymbol{\mu} + \boldsymbol{B},$$

which uses the Moore-Penrose pseudo-inverse [2]:

$$\Omega^\dagger = (\Omega^\top \Omega)^{-1} \Omega^\top.$$

The pseudo-inverse minimizes the MMSE: $\|\boldsymbol{\mu}' - \bar{\boldsymbol{\mu}}'\|^2$, i.e. maximizes the likelihood of $\boldsymbol{\mu}$. This formulation leads to a simpler algorithm, since the pseudo-inverse can be computed easily using a singular value decomposition of Ω:

$$\Omega = USV^\top \;\Rightarrow\; \Omega^\dagger = VS^{-1}U^\top.$$

2.6.2 Cluster Adaptive Training

Cluster adaptive training (CAT) [36] assumes that the training data can be grouped into clusters of sources whose statistical characteristics are similar. CAT has been developed in the context of ASR, and is based on the concept of eigenvoices [30]. The principle of this method is to consider a set of models (1, ... ,K), in general obtained by clustering source-dependent (in the case of ASR, speaker-dependent) models, and to generate any new source-dependent GMM using a linear combination of the cluster parameters. Each mean vector $\boldsymbol{\mu}$ is expressed as a weighted sum $\lambda_1, \ldots, \lambda_K$ of the cluster mean vectors $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$:

$$\boldsymbol{\mu} = \sum_{k=1}^{K} \lambda_k \boldsymbol{\mu}_k.$$

In compact form, this can be written using a vector of concatenated means $\boldsymbol{M} = [\boldsymbol{\mu}_1^\top \ldots \boldsymbol{\mu}_N^\top]^\top$ for each mixture, a weight vector (which can include a bias term) $\boldsymbol{\Lambda} = [\lambda_1 \ldots \lambda_K]^\top$, and a matrix $E = [\boldsymbol{M}_1 \ldots \boldsymbol{M}_K]$ whose columns are the concatenated means for each cluster:

$$\boldsymbol{M} = E\boldsymbol{\Lambda}.$$

CAT alternatively optimizes the cluster means and the individual weights to come up with a global ML estimate of the model. A typical way to generate cluster models [30] is to perform principal component analysis (PCA) on a large collection of source-dependent models. There is no sound foundation to this, except for the fact that PCA optimizes some form of MSE on the mean parameters, and thus the initial cluster models will have a reasonable likelihood [81].

The weights $\mathbf{\Lambda}$ can be estimated by setting

$$\frac{\partial Q(E\mathbf{\Lambda})}{\partial \mathbf{\Lambda}} = 0.$$

Solving this equation leads to the following system:

$$G = \sum_{i=1}^{N} w_i E^\top \bar{\Sigma}_i^{-1} E,$$

$$\boldsymbol{K} = \sum_{i=1}^{N} w_i E^\top \bar{\Sigma}_i^{-1} \bar{\boldsymbol{M}},$$

$$\mathbf{\Lambda} = G^{-1}\boldsymbol{K}.$$

Here, $\bar{\boldsymbol{M}}$ is the ML estimate of the means using the sufficient statistics drawn from the training data. For each source, the sufficient statistics need to be accumulated and a $K \times K$ matrix needs to be inverted (K^3 operations). Maximizing the likelihood across sources with source cluster prior $\gamma_{(s)}$

$$\frac{\sum_s \gamma_{(s)} \partial Q(\boldsymbol{M}_{(s)})}{\partial E} = 0$$

yields in the diagonal covariance case:

$$G = \sum_s \gamma_{(s)} \mathbf{\Lambda}_{(s)} \mathbf{\Lambda}_{(s)}^\top,$$

$$K = \sum_s \gamma_{(s)} \boldsymbol{M}_{(s)} \mathbf{\Lambda}_{(s)}^\top,$$

$$E = KG^{-1}.$$

In order to avoid the diagonal covariance limitation, a preconditioning of the parameters identical to the one used in Section 2.6.1 can be used:

$$\boldsymbol{M}' = [\boldsymbol{\mu'}_1^\top \dots \boldsymbol{\mu'}_N^\top]^\top = E'\mathbf{\Lambda}', \tag{2.19}$$

with, as before,

$$\boldsymbol{\mu}_i' \equiv \sqrt{w_i} L_i^{-1} \boldsymbol{\mu}_i.$$

Because the preconditioning turns the ML estimation into a MMSE estimation, the PCA decomposition used is also a ML cluster estimation: by performing PCA on a collection of models, the likelihood of these models in the space of reduced dimensionality is maximized, i.e. given a collection of source-dependent models $\boldsymbol{M}_{(s)}$ and a dimensionality K, the set E_k of bases functions obtained by PCA minimizes

$$E_{(s)}\left(\|\boldsymbol{M}_{(s)} - E_k E_k^\top \boldsymbol{M}_{(s)}\|^2\right) = -E_{(s)}\left(Q(E_k E_k^\top \boldsymbol{M}_{(s)})\right).$$

Using this model, the weight optimization can be expressed as

$$\frac{\partial Q(E'\boldsymbol{\Lambda}')}{\partial \boldsymbol{\Lambda}'} = 0 \quad \Rightarrow \quad (\boldsymbol{M}' - \bar{\boldsymbol{M}}')^\top E' = 0.$$

This means that the maximum likelihood estimate of the transformed mean is the orthogonal projection of the transformed mean on the basis E:

$$\boldsymbol{\Lambda}' = E'^\top \bar{\boldsymbol{M}}'.$$

Since only the adapted means are of interest, the projection matrix can be computed *once* for all sources:

$$P = E' E'^\top \quad \boldsymbol{M}' = P \bar{\boldsymbol{M}}'.$$

The only computations left are the whitening of the means and the projection. In practice, this scheme scales much better as the number of sources and/or clusters grow. In order to reestimate the cluster, no covariance structure needs to be assumed. In addition, the priors are included into the transform. The differentiation is straightforward:

$$\frac{\sum_s \partial Q(\boldsymbol{M}_{(s)})}{\partial E'} = 0 \quad \Rightarrow \quad \sum_s (\boldsymbol{M}'_{(s)} - E' \boldsymbol{\Lambda}'_{(s)}) \boldsymbol{\Lambda}'^\top_{(s)} = 0 \ ,$$

yielding

$$G' = \sum_s \mathbf{\Lambda}'_{(s)} \mathbf{\Lambda}'^{\top}_{(s)},$$

$$K' = \sum_s \boldsymbol{M}'_{(s)} \mathbf{\Lambda}'^{\top}_{(s)},$$

$$E' = K' G'^{-1}.$$

On the surface, this result is very much identical to the standard CAT. However, one can take a slightly different look at the equation by realizing that if we organize the weights and source means in matrix form

$$\Lambda = [\mathbf{\Lambda}_{(1)} \dots \mathbf{\Lambda}_{(S)}], \quad \Gamma = [\boldsymbol{M}_{(1)} \dots \boldsymbol{M}_{(S)}].$$

Then we have

$$G' = \Lambda \Lambda^{\top}$$
$$K' = \Gamma \Lambda^{\top}$$

And thus the system can be rewritten as

$$E = \Gamma \Lambda^{\top} (\Lambda \Lambda^{\top})^{-1}.$$

Another way to show this is to consider the equation that ties the basis functions and the speaker mean vectors

$$E' \Lambda = \Gamma.$$

This is an overdetermined system, which can be solved using a Moore-Penrose pseudo-inverse [2]: $\Lambda^{\dagger} = \Lambda^{\top} (\Lambda \Lambda^{\top})^{-1}$, which minimizes $\|\Gamma - \bar{\Gamma}\|^2$, i.e. maximizes the likelihood of Γ. This yields a potentially more efficient method for computing the cluster means. The pseudo-inverse can be computed using singular value decomposition

$$\Lambda = USV^{\top} \quad \Rightarrow \quad \Lambda^{\dagger} = VS^{-1}U^{\top}.$$

Note that a bias term $\boldsymbol{b}$ that captures the average behavior of the mixtures can be included as follows:

$$\Lambda_\star = [\Lambda'^\top \; \mathbf{1}]^\top, \quad E'_\star = [E' \; \boldsymbol{b}], \quad E'_\star = \Gamma \Lambda_\star^\dagger.$$

2.6.3 Maximum a Posteriori Adaptation

The simplest form of maximum a posteriori (MAP) adaptation [38] of parameters $(\boldsymbol{\mu}, \Sigma)$ to adapted estimates $(\boldsymbol{\mu}', \Sigma')$ based on the sufficient statistics $(\bar{\boldsymbol{\mu}}, \bar{\Sigma})$ of the adaptation data, uses a smoothing parameter α, and can be written as

$$\boldsymbol{\mu}' = \boldsymbol{\mu} + \alpha(\bar{\boldsymbol{\mu}} - \boldsymbol{\mu}),$$

$$\Sigma' = (1-\alpha)\Sigma + \alpha\bar{\Sigma} + \alpha(1-\alpha)(\bar{\boldsymbol{\mu}} - \boldsymbol{\mu})(\bar{\boldsymbol{\mu}} - \boldsymbol{\mu})^\top.$$

With no constraint on the covariance parameters, the covariance adaptation is thus straightforward. However, if semi-tied covariances or a MIC model are used, the situation is complicated by the fact that reestimating the model parameters given the full covariance estimate might not be simple. In addition, one might want to avoid having to resort to storing the full covariance sufficient statistics to be able to perform the adaptation.

One solution to this issue is to consider the adaptation of the Gaussian-dependent parameters exclusively, as opposed to attempting to adapt all the parameters in the model. In the semi-tied covariance case, this means keeping the global transform constant and only adapting the diagonal matrix parameters. In the MIC case, this means only adapting the weights of the mixture.

In the semi-tied case, the covariance can be written as

$$\Sigma = ADA^\top.$$

The covariance adaptation can be written

$$\begin{aligned} AD'A^\top &= (1-\alpha)ADA^\top + \alpha\bar{\Sigma} + \alpha(1-\alpha)(\bar{\boldsymbol{\mu}}-\boldsymbol{\mu})(\bar{\boldsymbol{\mu}}-\boldsymbol{\mu})^\top \\ D' &= (1-\alpha)D + \alpha A^{-1}\bar{\Sigma}A^{\top-1} + \alpha(1-\alpha)A^{-1}(\bar{\boldsymbol{\mu}}-\boldsymbol{\mu})(\bar{\boldsymbol{\mu}}-\boldsymbol{\mu})^\top A^{\top-1}. \end{aligned}$$

It is easy to show that the ML estimate of D given A and the sufficient statistics $\bar{\Sigma}$ is

$$\bar{D} = \mathrm{Diag}(A^{-1}\bar{\Sigma}A^{\top-1}).$$

Thus, denoting by d_k the diagonal components of D, the system of equations resulting from the diagonal terms of the previous equation is

$$d'_k = (1-\alpha)d_k + \alpha d_k + \alpha(1-\alpha)\nu_k,$$

with ν_k diagonal elements of

$$N = \mathrm{Diag}(A^{-1}(\bar{\boldsymbol{\mu}}-\boldsymbol{\mu})(\bar{\boldsymbol{\mu}}-\boldsymbol{\mu})^\top A^{\top-1}).$$

One can rewrite semi-tied covariances in the form of a MIC model using the rows $\boldsymbol{\eta}_k$ of A^{-1} as

$$A^{\top-1}D^{-1}A^{-1} = \sum_k \frac{1}{d_k}\eta_k\eta_k^\top = \sum_k \frac{1}{d_k}\Psi_k.$$

Using this notation,

$$\nu_k = (\bar{\boldsymbol{\mu}}-\boldsymbol{\mu})^\top \boldsymbol{\eta}_k\boldsymbol{\eta}_k^\top(\bar{\boldsymbol{\mu}}-\boldsymbol{\mu}).$$

Thus, the covariance adaptation can be rewritten in terms of the d_k and Ψ_k only:

$$d'_k = (1-\alpha)d_k + \alpha\bar{d}_k + \alpha(1-\alpha)(\bar{\boldsymbol{\mu}}-\boldsymbol{\mu})^\top\boldsymbol{\Psi}_k(\bar{\boldsymbol{\mu}}-\boldsymbol{\mu}). \tag{2.20}$$

This means that the weights can be updated using the reduced sufficient statistics $\bar{d}_k$.

Unfortunately, the situation for the general MIC model is not as simple. One can

easily show that Equation 2.20 still holds in the general MIC case if

$$d_k = \mathrm{Tr}\left(\Psi_k \Sigma\right).$$

This means that unlike the semi-tied case, there is no closed form solution for the Gaussian-dependent parameter reestimation problem. Given the sufficient statistics d'_k computed using 2.20, one still need to solve the set of non-linear equations

$$d'_k = \mathrm{Tr}\left(\Psi_k [\sum_l \lambda_l \Psi_l]^{-1}\right).$$

This can be achieved efficiently using the Newton algorithm described in Section 2.5.2.

2.7 Conclusion

The MIC model is a flexible simplified representation of the covariance matrices in a GMM, which is very suitable to a compact representation and fast computations. Through a parametric compression of the covariances, it captures redundancies across components of the GMM, and reduces greatly the parametric complexity of the model. In Chapter 6, we will show that, in the case of ASR, this parametric reduction can be very large, without any cost in the accuracy of the overall model. ML estimation of the model can be conducted using efficient convex optimization algorithms with guaranteed convergence properties. Adaptation of GMM based on a MIC model is also possible with appropriate modifications to the algorithms typically used with diagonal covariance models.

Chapter 3

Variable Length MIC

When using the MIC model, The evaluation of a Gaussian log-likelihood amounts to a scalar product between an extended feature vector and a parameter vector, both of which have dimensionality $D + K$, where D is the input feature dimensionality, and K is the number of prototypes. Given this $D + K$ complexity cost, it is natural to consider optimizing the number of prototypes used on a per-Gaussian basis, so that at a given average complexity level $D + \bar{K}$, Gaussians requiring a more detailed approximation can use a larger number of prototypes than those needing only a coarse approximation. Solving the variable length problem turns the MIC estimation into a constrained maximum likelihood estimation, which requires several notable modifications to the algorithm [77].

The variable-length MIC model can be expressed concisely as

$$\Sigma_i^{-1} = \sum_{k=1}^{K_i} \lambda_{k,i} \Psi_k.$$

Note that K_i is now an additional discrete Gaussian-dependent parameter in the model. Because adding additional prototypes can only improve the likelihood of the model, the addition of this parameter turns the ML estimation into a constrained problem.

3.1 Model Estimation

Denoting by $\boldsymbol{K} = [K_1 \dots K_M]^\top$ a vector listing for $i \in [1, M]$ the length K_i of the MIC decomposition of Gaussian i, the variable-length estimation problem can be expressed as:

Maximize $Q(\Psi, \Lambda, \boldsymbol{K})$, subject to:

- a complexity constraint for the average per-Gaussian computational cost:

$$E[K_i] = \bar{K}$$

  This constraint fixes the average per-Gaussian complexity in the model, and thus the average amount of CPU spent on computing individual Gaussians,

- a complexity constraint for the front-end overhead:

$$K_i \leq K_{\max}$$

  This constraint limits the number of additional feature components ω_k which need to be computed upfront,

- a feasibility constraint:

$$K_{\min} \leq K_i$$

  K_i should be at least 1 for the MIC decomposition to be defined, but $K_{\min} > 1$ might also be used for practical reasons discussed later.

In a manner similar to variable rate vector quantization [41, Chapter 17], the length optimization will be carried out iteratively within the MIC reestimation framework:

1. maximize $Q(\Psi|\Lambda, \boldsymbol{K})$ subject to $\forall i \Sigma_i \succ 0$,
2. maximize $Q(\Lambda|\Psi, \boldsymbol{K})$ subject to $\forall i \Sigma_i \succ 0$,

3. maximize $Q(\boldsymbol{K}|\Psi,\Lambda)$ subject to $E[K_i] = \bar{K}$ and $K_{\min} \leq K_i \leq K_{\max}$.

Steps 1,2 and 3 are iterated until the Q function reaches its maximum. The optimization becomes an instance of a tri-level alternating optimization [46]. The first two steps are not different from the fixed-length case. The last one is more difficult: once the optimal Ψ and Λ have been found for a given set of lengths $\boldsymbol{K}^\star$, then we only know $Q(\Psi,\Lambda,\boldsymbol{K}^\star)$. From this data point, deducing the rest of the function $Q(\Psi,\Lambda,\boldsymbol{K})$ for an arbitrary $\boldsymbol{K}$, in order to optimize it, requires finding an optimal set of weights for every $\boldsymbol{K}$. This is prohibitively expensive, since we need to re-run a descent algorithm akin to step 2 for each Gaussian and each value of K_i. Even a search strategy around $\boldsymbol{K}^\star$ would be complex, since there is no guarantee that the function Q_i for a given Gaussian i is convex in K_i.

We can, however, *model* Q given the information we know about it. Section 3.1.1 describes a parametric model used to represent Q for the purposes of this optimization. The model used is convex, which turns the maximization into a constrained convex optimization problem, which is solved in Section 3.1.2.

3.1.1 Parametric Model of Q

Let us assume that an initial length vector $K^\star$ is known, and that steps 1 and 2 were run to estimate

$$\Sigma_{i,K^\star}^{-1} = \sum_{k=1}^{K^\star} \lambda_{k,i}\Psi_k.$$

We know several things about $Q = \sum_i w_i Q_i(\Psi, \boldsymbol{\Lambda}_i, K_i)$:

- Since the likelihood can only be improved by adding components, Q_i is increasing with K_i,

- $Q_{i,K^\star} = Q_i(\Psi, \mathbf{\Lambda}_i)$ is known for the current length:

$$\begin{aligned} Q_{i,K^\star} &= \log|\Sigma_{i,K^\star}^{-1}| - \mathrm{Tr}\left(\Sigma_{i,K^\star}^{-1}\bar{\Sigma}_{i,K^\star}\right) \\ &= \log|\Sigma_{i,K^\star}^{-1}| - \sum_{k=1}^{K^\star}\lambda_{k,i}\mathrm{Tr}\left(\Psi_k\bar{\Sigma}_{i,K^\star}\right). \end{aligned}$$

  Since the model is estimated using maximum likelihood, the gradient in Equation 2.4 is zero, and

$$\begin{aligned} Q_{i,K^\star} &= \log|\Sigma_{i,K^\star}^{-1}| - \sum_{k=1}^{K^\star}\lambda_{k,i}\mathrm{Tr}\left(\Psi_k\Sigma_{i,K^\star}\right) \\ &= \log|\Sigma_{i,K^\star}^{-1}| - \mathrm{Tr}\left(\Sigma_{i,K^\star}^{-1}\Sigma_{i,K^\star}\right) \\ &= \log|\Sigma_{i,K^\star}^{-1}| - D. \end{aligned}$$

- $Q_{i,1} = Q(\Psi_0, \Lambda_i, 1)$ can be found analytically:

  We know from Section 2.5.2 that

$$\mathbf{\Lambda}_i = \mathbf{\Lambda}_{i,0} + U\mathbf{\Lambda}_i',$$

  where $\mathrm{Rank}(U) = K_i - 1$. Thus if $K_i = 1$, then

$$\mathbf{\Lambda}_i = \mathbf{\Lambda}_0 = \frac{D}{\mathrm{Tr}(\Psi_0\bar{\Sigma}_i)}.$$

  Thus

$$Q_{i,1} = \log|\frac{D}{\mathrm{Tr}(\Psi_0\bar{\Sigma}_i)}\Psi_0| - D.$$

- In the limit, when the number of weights reaches the number of parameters in the full covariance matrix, at $K_{\mathrm{lim}} = \frac{D(D+1)}{2}$, the ML estimate of the covariance is reached exactly:

$$Q_{i,\mathrm{lim}} = Q_i(\Psi, \Lambda_i, K_{\mathrm{lim}}) = \log|\bar{\Sigma}_i^{-1}| - D.$$

From this information, we can build a parametric model of Q_i for all length $K \in [1, K_{\lim}]$. Figure 3.1 shows how Q behaves on average across all Gaussians in a test GMM used in acoustic modeling.

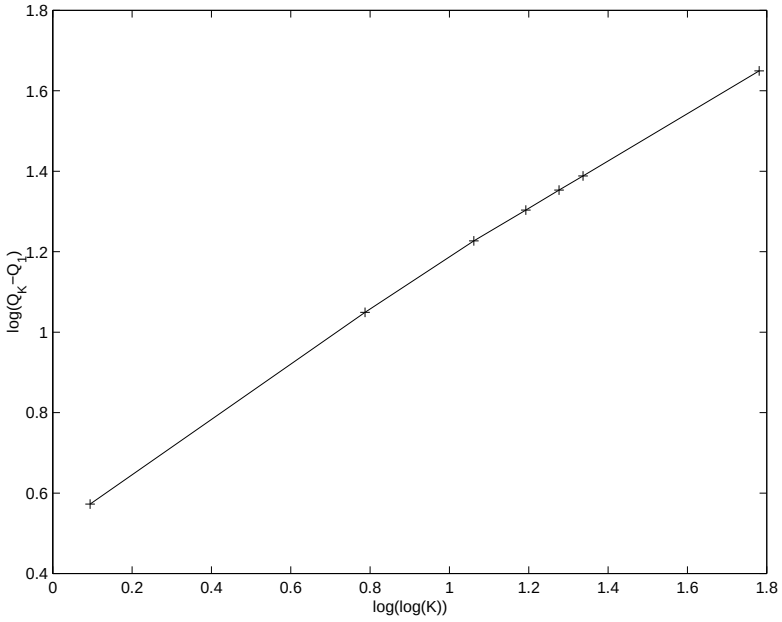

Figure 3.1: Plot of $\log(Q_K - Q_1)$ against $\log\log K$. The approximately affine relationship suggests a simple parametric model for the Gaussian likelihood as a function of K.

This suggests that a reasonable model for the likelihood would be linearly connecting $\log(Q_K - Q_1)$ with $\log\log K$. For this reason, in the following we used the parametric model

$$Q_i(K) = Q_{i,1} + \alpha_i [\log K]^{\beta_i}.$$

The two free parameters α_i and β_i can be computed for each Gaussian i using a regression on the known values of Q_i: $Q_{i,K^\star}$ and $Q_{i,\lim}$. Using $\tau_k = \log\log K_k$, the

parameters are

$$\begin{aligned} \log \alpha_i &= \frac{\tau_\infty \log(Q_{i,K^\star} - Q_{i,1}) - \tau_{K^\star} \log(Q_{i,\infty} - Q_{i,1})}{\tau_\infty - \tau_{K^\star}}, \\ \beta_i &= \frac{\log(Q_{i,\infty} - Q_{i,1}) - \log(Q_{i,K^\star} - Q_{i,1})}{\tau_\infty - \tau_{K^\star}}. \end{aligned}$$

3.1.2 Convex Optimization

The MLE process can now be formulated as:

- maximize: $Q(\boldsymbol{K}) = \sum_i w_i \alpha_i (\log K_i)^{\beta_i}$,
- subject to: $\sum_i w_i K_i = \bar{K}$ and $K_{\min} \leq K_i \leq K_{\max}$.

We will use standard convex optimization methods to solve the problem. First, let us assume that the constraints are not present. In that situation, a standard Newton algorithm can be used to optimize Q [15, Chapter 9]. For that, we compute the gradient $\boldsymbol{\nabla}$ with respect to $\boldsymbol{K}$ and the Hessian H:

$$\begin{aligned} \frac{\partial Q(\boldsymbol{K})}{\partial K_i} &= \frac{w_i \alpha_i \beta_i (\log K_i)^{\beta_i - 1}}{K_i}, \\ \frac{\partial^2 Q(\boldsymbol{K})}{\partial K_i \partial K_j} &= \delta_i^j \frac{w_i \alpha_i \beta_i [\beta_i (\log K_i)^{\beta_i - 2} - (\log K_i)^{\beta_i - 1}]}{K_i^2}. \end{aligned}$$

Note that the Hessian is diagonal here. For simplicity we will denote by $\boldsymbol{R}$ the diagonal of the inverse of the Hessian. Denoting by $\star$ the Kronecker product of two vectors, the Newton update can be written as

$$\boldsymbol{\Delta}_{\boldsymbol{K}} = -\boldsymbol{R} \star \boldsymbol{\nabla}.$$

The equality constraint $\sum_i w_i K_i = \bar{K}$ is linear. Denoting by $\boldsymbol{\Pi}$ the vector of priors, the constraint can be written as

$$\boldsymbol{\Pi}^\top \boldsymbol{K} = \bar{K}.$$

The Newton update can be modified simply to incorporate it as follows [15, Chapter 10]. Denoting $\boldsymbol{U} = \boldsymbol{R} \star \boldsymbol{\Pi}$,

$$\boldsymbol{\Delta_K} = \frac{\boldsymbol{U}^\top \boldsymbol{\nabla}}{\boldsymbol{U}^\top \boldsymbol{\Pi}} \boldsymbol{U} - \boldsymbol{R} \star \boldsymbol{\nabla}.$$

This modification still bears the same convergence properties as the unconstrained update, but preserves the equality constraint by forcing the update to happen in the hyperplane orthogonal to $\boldsymbol{\Pi}$. Indeed we have that

$$\boldsymbol{\Pi}^\top \boldsymbol{\Delta_K} = 0, \tag{3.1}$$

and thus if $\boldsymbol{\Pi}^\top \boldsymbol{K} = \bar{K}$, then $\boldsymbol{\Pi}^\top (\boldsymbol{K} + \boldsymbol{\Delta_K}) = \bar{K}$.

In order to enforce the inequality constraints, we use a barrier method [15, Chapter 11]. The idea is to augment the function to optimize with a family of *barrier functions* which satisfy the inequality constraints by design. The family $\phi(\boldsymbol{K})/t$ parameterized by a parameter t is such that when $t \to +\infty$, the function goes to 0 everywhere in the admissible space, and to $-\infty$ outside of it. Instead of optimizing $Q(\boldsymbol{K})$ directly, t is fixed to some finite value, and $Q(\boldsymbol{K}) + \phi(\boldsymbol{K})/t$ is optimized by only taking the equality constraints into account. t is then increased and the optimization iterated until convergence. This turns the overall problem into a succession of problems which only involve equality constraints, and which we know how to solve.

Here we use the simple log barrier function to ensure $K_{\min} \leq K_i \leq K_{\max}$:

$$\phi(\boldsymbol{K}) = \log(\boldsymbol{K} - K_{\min}) + \log(K_{\max} - \boldsymbol{K}).$$

The length allocation algorithm runs after each iteration of the weight reestimation. Figure 3.2 shows the likelihood increase during a given run of the length optimization on data used in the experiments described in Chapter 6. The first sharp rise in likelihood happens as the Newton algorithm is run for a fixed barrier factor t and corresponds to the initial optimization starting from a uniform length distribution. The second likelihood increase corresponds to the barrier factor being slowly increased, bringing the constrained length distribution closer to its global optimum.

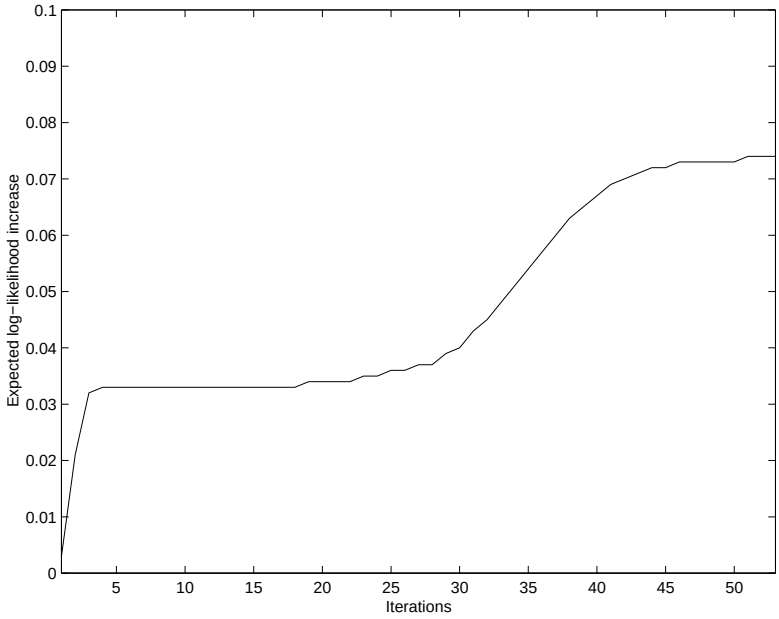

Figure 3.2: Likelihood increase as the length allocation algorithm is iterated.

Figure 3.3 shows how the allocation algorithm distributes the weights to the various covariances in the GMM in an acoustic model used in the same experiments.

Since fewer than 27 weights are used on average, the total number of prototypes that need to be evaluated at each input frame of speech might be less than 27 as well. Thus if the front-end computation is implemented in a lazy way, substantial computational savings can be obtained in addition to the reduction in per-Gaussian computations.

3.2 Conclusion

This chapter demonstrates that the MIC model can be improved by optimizing the degree of precision by which covariances are approximated on a per-covariance basis instead of globally. An efficient constrained MLE algorithm was proposed to perform this per-covariance weight allocation.

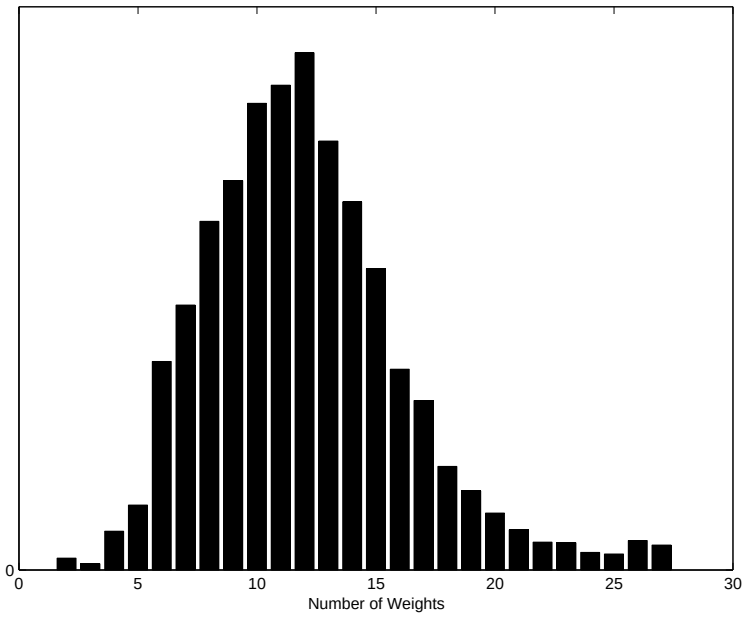

Figure 3.3: Histogram of the number of weights allocated per Gaussian by the MLE algorithm. Here, the average number of weights is set to 12, the minimum 2 and the maximum 27.

Chapter 4

Subspace Factored MIC

A very powerful extension of the basic MIC model can be defined in the case of probability densities which, at the component level, can be assumed to be the product of independent or near-independent distributions. In this situation, the covariance matrices of the mixture components will have a block-diagonal structure. Note that we do not require the complete distribution to be (near-)independent, since the different mixture components can still model correlated events. However, the limit case of a distribution which is globally the product of independent distributions is useful to illustrate the following point: if the probability density in one of the independent subspaces bears no relationship with the density in distinct subspaces, then there is no modeling benefit at clustering the distinct subspaces jointly.

Let us thus consider a block-diagonal model, with an independent set of prototypes and bases for each sub-block. Considering L covariance sub-blocks of dimensionality D_l, the model is

$$\Sigma_{i,l}^{-1} = \sum_{k=1}^{K_l} \lambda_{k,i,l} \Psi_{k,l}.$$

The advantages of this model are multiple. First, the global estimation problem is decomposed into multiple, lower-dimensional problems that will be less expensive to solve. In addition, the cost of evaluating the log-likelihood of a subspace-factored model is lower. The last advantage is of combinatorial nature: a block-diagonal

system with L subspaces and K prototypes per subspace contains implicitly K^L "full" prototypes, while only requiring $K \times L$ weights per Gaussian. As a result, for a given number of Gaussian-specific parameters, the subspace-factored model can make use of a larger collection of prototypes than its single-block counterpart.

This means that if the independence of the distinct subspaces can be assumed, there is a modeling benefit in using a block diagonal model instead of a full covariance model. This subspace decomposition method is known in coding as partitioned VQ [41, Section 12.8] which is the simplest instance of a product code. In ASR, this has been used to revive the concept of VQ-based acoustic modeling using discrete mixture hidden Markov models (DMHMM) [26], which compare well with standard hidden Markov models (HMM) which use GMM as underlying density models, and allow for a more compact representation of cepstral parameters [25]. In GMM/HMM systems, the same idea has been exploited by performing subspace clustering of the Gaussians in a mixture [59].

The training of the a subspace-factored model amounts to reestimating at each iteration of the EM algorithms all the MIC parameters in each subspace independently. Since the the CPU cost of training the model is very non-linear in the dimensionality, the amount of computations required to train a subspace-factored model is much smaller that the amount required to train a full MIC model. This means that under appropriate independence assumptions, the SFMIC model training can scale to much larger input dimensionalities than the simple MIC model.

4.1 Factorization of Arbitrary Subspaces

The simple subspace-factored approach implicitly assumes that the independent subspaces to be modeled correspond to distinct groups of feature components, which cause the subspaces considered to be:

- orthogonal to each other,
- parallel to feature components.

The diagonal covariance model, which can be seen as a SFMIC model with $K_l \equiv 1$ makes similar assumptions. It has been shown that some modeling benefits could be obtained by removing both these assumptions. The factored sparse inverse covariance matrices model [11] removes the orthogonality assumption (see Section 1.4), and the semi-tied model [35] removes both while keeping the same underlying diagonal model. In this section, a similar model built around the SFMIC model is introduced. This approach allows the SFMIC method to be used on arbitrary subspaces. In addition, the subspaces which are most suitable for being modeled independently can be derived automatically using a ML estimation procedure.

4.1.1 Transformed SFMIC

In the vein of semi-tied models, the basic principle is to add a transform of the global space to the subspace-factored model. The motivation here is to capture the average correlation structure of the data, while still using a block-diagonal approach to model more finely subcomponents of the features which are meaningful. Let us assume that we have a real matrix U, and let us express the covariance as

$$\Sigma_i^{-1} = U^\top \Phi_i U,$$

where Φ_i is a MIC decomposition whose prototypes live in smaller subspaces (i.e. are zero except for a square block along the diagonal).

The Gaussian evaluation becomes, using the notations of section 2.4:

$$\begin{aligned}
\boldsymbol{\omega} \quad &: \quad \omega_k = \frac{1}{2}(U\boldsymbol{o})^\top \Psi_k (U\boldsymbol{o}), \\
\boldsymbol{\nu} \quad &= \quad -\Phi_i U \boldsymbol{\mu}, \\
\boldsymbol{o}' \quad &= \quad \begin{bmatrix} U\boldsymbol{o} \\ \boldsymbol{\omega} \end{bmatrix}, \\
\boldsymbol{\nu}' \quad &= \quad \begin{bmatrix} \boldsymbol{\nu} \\ \boldsymbol{\Lambda} \end{bmatrix}.
\end{aligned}$$

This model thus adds one matrix transform per input vector to the total computational cost of evaluating a GMM.

4.1.2 Model Estimation

The estimation of the model parameters is much simplified if one assumes as in [11] that the transform U is unit triangular. Let us consider the problem of finding U and Φ_i such that:

- U is unit triangular, implying that $|U| = 1$,
- $U^\top \Phi_i U$ is a maximum likelihood estimate of Σ_i^{-1},
- Φ_i is block diagonal.

Note that for simplicity, we do not force Φ_i to be estimated using MIC, the assumption being that whatever the matrix Φ_i becomes as a result of this joint estimation, it will be then approximated using a SFMIC model. The auxiliary function of the EM reestimation can be written as

$$\begin{aligned} Q &= \sum_i w_i \left[\log |U^\top \Phi_i U| - \mathrm{Tr}(U^\top \Phi_i U \bar{\Sigma}_i) \right] \\ &= \sum_i w_i \left[\log |\Phi_i| - \mathrm{Tr}(\Phi_i U \bar{\Sigma}_i U^\top) \right]. \end{aligned}$$

Let us denote by $\mathcal{B}$ the operator which zeroes out components outside of the subspaces modeled by Φ_i. From the Q function above, given a fixed transform U, the maximum likelihood estimate of Φ_i can be computed by replacing the sufficient statistics $\bar{\Sigma}_i$ with $U\bar{\Sigma}_i U^\top$. Thus

$$\bar{\Phi}_i = \mathcal{B}(U\bar{\Sigma}_i U^\top)^{-1}.$$

Conversely given Φ_i, the maximum likelihood estimate of U can be computed analytically. Let us denote

$$U = I - B,$$

with B strictly upper triangular:

$$\begin{aligned}\frac{\partial Q(\Psi, \Lambda, U)}{\partial U} &= -\sum_{i=1}^{M} w_i \frac{\partial \mathrm{Tr}\left(U^\top \Phi_i U \bar{\Sigma}_i\right)}{\partial U} \\ &= -2\sum_{i=1}^{M} w_i \Phi_i U \bar{\Sigma}_i,\end{aligned}$$

which translates into

$$\sum_{i=1}^{M} w_i \Phi_i B \bar{\Sigma}_i = \sum_{i=1}^{M} w_i \Phi_i \bar{\Sigma}_i, \tag{4.1}$$

which is a linear system of equations. With $k < l$, and denoting $\phi^i_{k,m}$ the entries of Φ_i and $\bar{\sigma}^i_{j,l}$ the entries of $\bar{\Sigma}_i$, the system can be written using:

$$\tau_{k,m,j,l} = \sum_{i=1}^{M} w_i \phi^i_{k,m} \bar{\sigma}^i_{j,l},$$

$$\xi_{k,l} = \sum_{i=1}^{M} w_i \sum_j \phi^i_{k,j} \bar{\sigma}^i_{j,l} = \sum_j \tau_{k,j,j,l},$$

$$\sum_j \sum_{m<j} b_{m,j} \tau_{k,m,j,l} = \xi_{k,l}.$$

As a consequence, we can proceed iteratively with the parameter estimation:

1. Set $U^0 = I$,
2. $\Phi^0_i = \mathcal{B}(\bar{\Sigma}_i)^{-1}$,

3. U^1 is estimated from the Φ_i^0 and $\bar{\Sigma}_i$ using Equation 4.1,

4. $\Phi_i^1 = \mathcal{B}(U^1 \bar{\Sigma}_i U^{1\top})^{-1}$,

5. iterate until convergence.

The case of a full matrix U transform is more complex, akin to the semi-tied reestimation process. The notable difference is that $Q(U)$ is not quadratic in the general case, which means that a convex optimization method needs to be applied to reestimate U.

4.2 Multiresolution Subspace Factorization

Weaker constraints can be put on the model by only assuming that some of the correlations require a lesser degree of precision in the modeling than others. This leads to the idea of a model which interpolates between SFMIC models with different subspace configurations. Let us assume that $\Phi_1, \ldots, \Phi_R$ are a collection of SFMIC models for a given inverse covariance matrix Σ^{-1}. An interpolated model of the inverse covariance can be expressed with weights $\pi_1, \ldots, \pi_R \in [0,1]^R$ ($\sum_r \pi_r = 1$), as

$$\Sigma^{-1} = \sum_{r=1}^{R} \pi_r \Phi_r.$$

The weights π_r can be multiplied with the weights of the corresponding SFMIC model Φ_r, leading to an expansion to the inverse covariance matrix into a linear combination of prototypes living in different, not necessarily overlapping, subspaces. Because, formally, this model can be viewed as an instance of the general MIC model with some subspace constraints on the prototypes, the training algorithm is unchanged. Depending on the actual structure of the subspaces considered, however, there might not be the possibility of training independent subspaces separately, which makes the training of a multiresolution SFMIC model not significantly faster than the training of a MIC model.

Chapter 5

Automatic Speech Recognition

5.1 Introduction

Automatic speech recognition [51, 68] has been studied for several decades, but only in the last ten years has it been able to become a real-world technology with a significant commercial market, largely through the development of large data corpora and large scale statistical modeling tools.

An ASR system comprises several distinct functional modules:

1. A speech input channel: the input channel converts the sound pressure wave into an analog waveform, which in turn is digitized. The input channel is very often critical to the robustness of the ASR system. Advanced channels also perform voice activity and endpoint detection in order to isolate speech segments from the background audio input and reduce the data rate of the digital signal sent to the recognizer. In other systems, the endpoint determination is done at a later stage in order to take advantage of the processing done by the recognizer to improve its accuracy.

2. A feature extraction module: the feature extraction converts the digital waveform into a sequence of features which are deemed representative of the speech signal and will serve as the basis for the statistical analysis. These features often use psychoacoustic [87], articulatory [53] or phonetic considerations to

isolate meaningful features of the signal. In its simplest form, the front-end can be a simple discrete time filterbank over the range of frequencies spanned by speech followed by some normalization of the extracted energy spectrum. Commonly used front-ends include Mel filter-bank cepstral coefficients (MFCC) [20], perceptual linear prediction (PLP) [47] and RASTA/PLP [48].

3. An acoustic model: the acoustic model probabilistically matches the feature sequence to a phonetic representation of the utterance. The structure of acoustic models will be examined in more detail in Section 5.2.

4. A dictionary: the dictionary links a phonetic sequence to a word. Typically dictionaries are hand-crafted by linguist experts, but more and more often (semi-)automatic methods are being developed to learn pronunciations based on linguistic rules or statistical analysis of the phone / word relationships. In the case where no transcription is available, the phonetic sequence can also be directly derived from the acoustics of a set of sample recordings [73].

5. A language model [52, 60]: the language model constrains the space of possible sequences of words through rule-based or probabilistic models of the task. In doing so, it reduces the computational complexity of the model by dynamically determining a set of possible word sequences based on what is being uttered by the user. It also improves the accuracy of the system by eliminating nonsensical candidates that the acoustic pattern matching might hypothesize. The language model is usually tightly coupled with the acoustic model and the dictionary into a combined decoder which is the core of the ASR system.

6. A natural language (NL) processing module [60]: typically takes a word sequence as an input and outputs a semantic interpretation of it. The extent of the semantic analysis is tightly linked with the actual usage that is made of the ASR system. Simple NL models only perform keyword spotting, attaching an interpretation to an utterance solely based on the words it comprises. Other models perform statistical classification of whole phrases based on their word

components, or further refine the semantic tagging using a parser which provides the grammatical structure of the spoken utterance. The NL processing module is sometimes integrated with the actual language model since performing a semantic analysis can provide valuable information as to the probability of a word sequence to have been uttered.

7. A user interface [19, 63]: typically left out of the traditional perspective on ASR, voice user interface design is becoming a science driven by an increasing number of voice activated user interface studies in human computer interaction. The interface plays an important role in the perceived success rate of an ASR system [74]. It also interacts strongly with both the language modeling part of the system and the natural language processing component by shaping the discourse of the user in ways that influence the statistics of the language and the semantics of the input words.

The core of the speech decoder can be seen a maximum a posteriori (MAP) classifier [28, 68]. Consider a sequence of input features $\boldsymbol{o}$ and a set of possible corresponding word sequences $w_1, \ldots, w_N$. The decoder needs to find the most likely word sequence $w^\star$ based on $\boldsymbol{o}$:

$$w^\star = \underset{n}{\operatorname{argmax}}\, p(w_n|\boldsymbol{o}).$$

Using Bayes rule,

$$p(w_n|\boldsymbol{o}) = \frac{p(\boldsymbol{o}|w_n)p(w_n)}{p(\boldsymbol{o})},$$

so that

$$w^\star = \underset{n}{\operatorname{argmax}}\, p(\boldsymbol{o}|w_n)p(w_n).$$

The term $p(\boldsymbol{o}|w_n)$ is usually what is referred to as the *acoustic model*, and the term $p(w_n)$ as the *language model*. This simplified view folds the dictionary into the acoustic model, although sometimes it is more useful to look at the probability of a word

sequence given its phonetic representation as a linguistic issue which resorts from language modeling.

5.2 Acoustic Modeling

The acoustic model maps the sequence of acoustic events, represented by the sequence of feature vectors extracted from the front-end, to the word sequence through a sequence of phonetic units. The phonetic units are connected but not necessarily identical to the phonemes constituting the words. The individual units can be syllables [83], acoustically-driven prototypes [73], or sub-phonetic units. Non-speech events such as silence or mouth noises also need to be represented as phonetic units in the case of continuous ASR.

For the purposes of this introduction, we will consider phonetic units as sequences of states s_i, and not concern ourselves with the nature of those states. In typical ASR systems, the recognizer can contain tens of thousand of these states, each modeling a very specific phonetic event. An example of such phonetic event could be: "the onset of phone /A/, when following phone /p/, and followed by phone /l/". Different ASR systems use a variety of definitions for a given phonetic event [70]. The essence of acoustic modeling is thus to map a sequence of observations $\boldsymbol{o}_1, \ldots, \boldsymbol{o}_T$ to a sequence of states and compute

$$p(o|w_n) = p(\boldsymbol{o}_1, \ldots, \boldsymbol{o}_T|s_1, \ldots, s_N).$$

Various models are commonly used to represent this density [66], the simplest of which being the hidden Markov model (HMM) [50]. HMMs make strong assumptions about the dependency structure of both the state sequence and the acoustic observations:

- The state sequence is assumed to be Markov:

$$p(s_1, \ldots, s_N) = p(s_N|s_{N-1}) \ldots p(s_2|s_1).$$

- The acoustic observations are assumed independent:

$$p(\boldsymbol{o}_1, \ldots, \boldsymbol{o}_T) = p(\boldsymbol{o}_1) \ldots p(\boldsymbol{o}_T).$$

It is often considered a weakness of the HMM model to consider the acoustic observations as independent of each other. However, the short-term dependencies between observations are actually explicitly represented in general in the feature vector itself. Indeed, in addition to the spectral information, the feature vector usually comprises the first and second derivatives of those features computed across a small window of time. Another weakness is the Markovian structure of the state sequence. This assumption is perceived not to reflect the dependencies between successive phonetic events such as coarticulations. However, the state sequence very often models those contextual dependencies explicitly by using context-dependent phonetic units that take into account the surrounding phones as well as the current phone to define a state.

Using these assumptions, the complete sequence likelihood can be computed easily from the knowledge of two sets of parameters:

1. the state transition probabilities: $p(s_j|s_i)$,

2. the state conditional densities: $p(\boldsymbol{o}_t|s_i)$.

The conditional probability of the full sequence can be expressed considering the set $\mathcal{A}$ of all possible monotonic mappings of the time index t onto the state sequence $a \in \mathcal{A} : t \in [1, T] \rightarrow i \in [1, N]$:

$$p(\boldsymbol{o}_1, \ldots, \boldsymbol{o}_T|s_1, \ldots, s_N) = \sum_{a \in \mathcal{A}} \prod_t p(s_{a(t-1)}|s_{a(t)}) p(\boldsymbol{o}_t|s_{a(t)}).$$

In practice, one typically makes the *Viterbi assumption* which vastly simplifies the evaluation of this likelihood: the simplification amounts to assuming that the best alignment between the state and acoustic sequences dominates the sum, which means

that

$$p(\boldsymbol{o}_1, \ldots, \boldsymbol{o}_T | s_1, \ldots, s_N) \sim \max_{a \in \mathcal{A}} \prod_t p(s_{a(t-1)} | s_{a(t)}) p(\boldsymbol{o}_t | s_{a(t)}).$$

The computation of the conditional probability is turned into a best path search problem which can be solved efficiently using the Viterbi algorithm [31], which performs a dynamic programming over the sequence of possible states. The other advantage of the Viterbi algorithm is that the search can be performed jointly over all admissible state sequences in the model by considering the space of possible utterances w_n as a lattice of states instead of each one as a distinct linear state sequence. The search for a best path within this lattice, weighted by the language model contribution $p(w_n)$, makes the decoding of the spoken utterance extremely efficient.

The state transition probabilities $p(s_j|s_i)$ are typically simple probability masses, which can be efficiently estimated using the Baum-Welch reestimation algorithm [6]. What remains to be specified given the HMM structure is a model of the state conditional densities $p(\boldsymbol{o}_t|s_i)$. The simplest way of representing those state probabilities is to use a single Gaussian, in which case we can directly compute ML estimates for each state using the equations introduced in Section 1.2. However, unless the state structure is carefully designed for each state to be as Gaussian as possible [72], a simple Gaussian density is not sufficient to accurately model the state density. More complex approaches use for example neural networks [49, 13, 32] or frequency domain HMM [80]. The most popular model, however, is to use Gaussian mixture models (GMM).

5.3 GMM for Acoustic Modeling

The simplest method for using GMM as state probability densities is to treat each state as a separate probability density:

$$p(\boldsymbol{o}_t|s_i) = \sum_{j=1}^{M_i} w_j \mathcal{N}(\boldsymbol{o}_t, \boldsymbol{\mu}_{ij}, \Sigma_{ij}).$$

In order to train such a GMM, one can remark that a single HMM state s_i modeled using a GMM density with M_i Gaussians is formally identical to M_i GMM states in parallel modeled using a single Gaussian each and with an input transition probability equal to the corresponding mixture weight. The Baum-Welch algorithm can thus be used to estimate, for each element $\boldsymbol{o}_t$ of the input sequence, the probability $\gamma_{i,j,t}$ to correspond to the j^{th} Gaussian in state s_i.

While it is sometimes useful to represent the HMM/GMM system as a network of single Gaussians, it is also possible to take the converse view and consider the whole collection of Gaussians in the system as a single large GMM, with individual states pointing at subsets of it through state-dependent mixture weights:

$$w_{k,j} = \begin{cases} \frac{1}{T}\sum_t \gamma_{i,j,t} & \text{if } k = i, \\ 0 & \text{otherwise.} \end{cases}$$

The motivation for taking this view is that now the individual Gaussians can be shared across states in a consistent manner. In general, using a dedicated GMM for each state is extremely expensive and rather inefficient. States are phonetically related to each other, making the GMM parameters very redundant across them. It is thus beneficial for the parametric complexity as well as the computational speed to share the Gaussian parameters. Several of these GMM tying schemes exist, each with different levels of granularity in the sharing [7, 23, 86]. Even though state-dependent mixture weights are used to generate the state densities, a global mixture weight representing the average prior of the Gaussian in the GMM can be computed using

$$w_j = \frac{1}{NT}\sum_i \sum_t \gamma_{i,j,t}.$$

Using this representation, the GMM/HMM model can be seen as a Markov random walk over individual Gaussians in a single large GMM. The training of the GMM part of the model is for all practical purposes identical to EM training, with the added complexity that the individual Gaussian occupancy probabilities $\gamma_{i,j,t}$ are obtained using the Baum-Welch algorithm applied to the HMM.

Chapter 6

MIC for Acoustic Modeling

In Chapter 2, we saw how the MIC model could be trained within the EM framework on large GMMs. In Chapter 5, we saw how the GMM in GMM/HMM acoustic models could be trained using EM based on the Gaussian component probabilities learned over the HMM network. In order to use MIC for acoustic modeling, one only needs to combine those two elements. When using the subspace factored approach, a choice has to be made as to which subspaces to use. Section 6.1 addresses this issue using a data driven analysis of the correlation structure of typical speech input features. Section 6.2 show performance results for the MIC, VLMIC and SFMIC models.

6.1 SFMIC and Acoustic Modeling

In order to take advantage of the subspace-factored approach, it is necessary to determine which correlation components can be discarded without any loss. The following analyzes the case of models based on MFCC [20] feature vectors, and demonstrates some non-intuitive results as to which components of the MFCC-derived covariance matrices are relevant. Section 6.2.6 will later show experimental results validating this approach.

The global structure of a covariance matrix resulting from a MFCC input vector is described in Figure 6.1.

Each component of the matrix models distinct types of correlations, some of which

Cepstrum	a	d	f
Δ	d	b	e
Δ^2	f	e	c

Figure 6.1: Structure of covariance matrices describing MFCC inputs. Sorting the MFCC feature vector into 3 blocks containing respectively the cepstra, first and second order derivative, the covariance matrix can be decomposed into 9 blocks. For example, block (d) models the correlations between the cepstral features and their derivatives

can be qualified as *structural*, and others *incidental*. Structural correlations result from the way feature components are computed from each other, leading to dependencies between them. Incidental correlations are a result of the relationships between components preexisting in the data being modeled, independently of the front-end processing.

A good example of *structural* versus *incidental* correlation occurs when building MFCC derivatives out of the cepstral coefficients. Typically, for a given input observation $\boldsymbol{o}(t)$ at time t, the derivative would be computed by applying a finite impulse response (FIR) filter onto the observation sequence such as depicted in Figure 6.2. The common features of the filters used are that they estimate the value of the signal

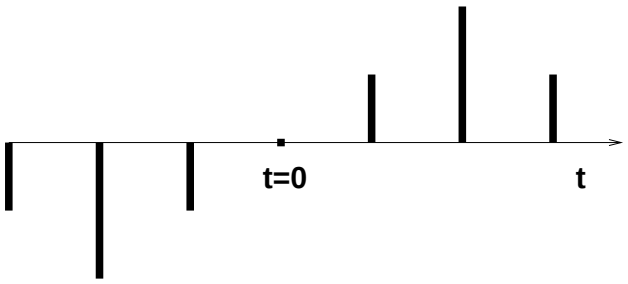

Figure 6.2: Profile of a FIR filter used to compute the cepstral derivative from a sequence of observations. Note that the value of the input at $t = 0$ is not typically used in the computation, which implies that correlations between the cepstrum and its derivative will only result from time correlations in the signal itself.

at $t < 0$ and subtract it from an estimate of the signal at $t > 0$ over a small window. Note that here the current input $\boldsymbol{o}(t)$ is not involved. As a consequence, any correlation arising between $\boldsymbol{\delta}(t)$ and $\boldsymbol{o}(t)$ would be *incidental*, i.e. would be providing

information about the relationship between consecutive frames of data.

When computing the second order derivatives, a typical profile would be as depicted in Figure 6.3. In this case, the component $\boldsymbol{o}(t)$ is explicitly part of the ex-

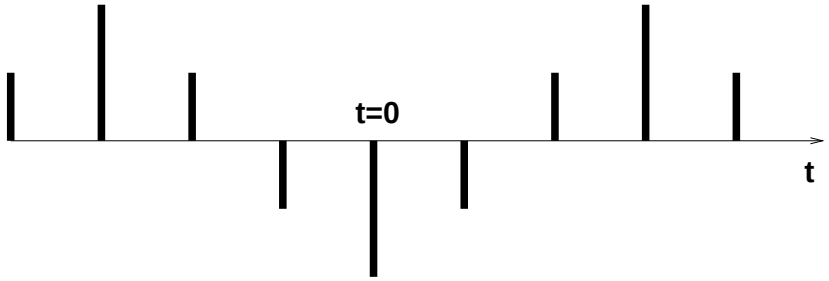

Figure 6.3: Profile of a FIR filter used to compute the cepstral second derivative from a sequence of observations. Note that the value of the input at $t = 0$ is heavily weighted by this type of filter, which implies that there will be structural correlations between the cepstrum and its second derivative.

pression of $\boldsymbol{\delta}^2(t)$, and thus there will be a structural correlation between the i^{th} MFCC component and its corresponding $\boldsymbol{\delta}^2(t)$ component. These considerations generally hold regardless of the actual implementation of the computation of the derivatives, however the exact distribution of structural correlations depends highly on the specifics of the feature extraction. Figure 6.4 illustrates which components of the inverse covariance matrix are structurally large in magnitude in the situation just described.

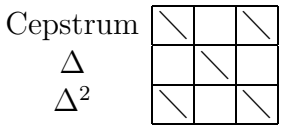

Figure 6.4: Structural correlations in a typical MFCC-derived inverse covariance matrix. The large magnitude components are the result of the way the second-order derivatives are computed from the cepstral coefficients.

The importance of this distinction lies into the following observation: while *structural* correlations are usually large in magnitude, they do not provide any real information about the data, and thus modeling those will not improve the model much. On the other hand, *incidental* correlations can be smaller in magnitude, but they

bring information about the data, and explicitly representing these will improve the model.

To illustrate this point, the following experiments were carried out. Several otherwise identical acoustic models were trained using different covariance structures. The error rates of recognition experiments run using these acoustic models are reported in Figure 6.5. The test-set is described in Section 6.2.1. Each (■) represents a block of non-zero entries in the covariance matrix, while an empty cell denotes entries that were zeroed.

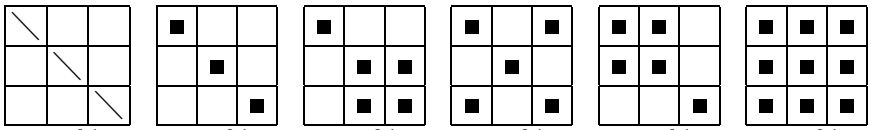

Figure 6.5: Error rates for different covariance structures, ranging from diagonal (top-left) to full (bottom-right). Note that most of the gain results from modeling within-block correlations along the diagonal. Adding the block corresponding to correlations between cepstra and Δ^2, most of which are *structural*, does not improve the accuracy significantly. Introducing correlations between cepstra and Δ improves the performance by a proportionally larger amount.

From Figure 6.5, it is clear that modeling the correlations within blocks, i.e. incorporating the 3 blocks denoted *a*, *b* and *c* in Figure 6.1 into the model, is responsible for a large part of the benefits of full covariance modeling with respect to diagonal models. It is also clear that adding the correlations between cepstra and Δ^2 (block *f*), which are large in magnitude but mostly structural, does not cause a significant decrease in error rate. On the other hand, incorporating correlations between cepstra and Δ coefficients (block *d*) brings the performance of a 2 block system close to the performance of a full covariance model.

In conclusion, it appears that three classes of models are of interest for MFCC-based system. These are the models whose error rate figures are underlined in Figure 6.5. The first model (on the lower right of the figure) is a full-covariance model, that will be referred to as a "1-block" model. The second one is a "2-block" model, one block modeling jointly the cepstra and Δ features, and the second modeling the Δ^2. The third "3-block" model uses one block per group of features: cepstra, Δ and Δ^2.

6.2 Experiments

In this section, the MIC model is applied to a GMM used for acoustic modeling in a HMM-based continuous ASR system. A comparison against semi-tied covariances is carried out in Section 6.2.2. In Section 6.2.3, the accuracy gains are reported for MIC models at various levels of parametric complexity. Section 6.2.4 shows how much CPU time a benchmark implementation of the estimation algorithm takes. Section 6.2.5 shows the performance of the approximate estimation scheme introduced in 2.5.4. Section 6.2.6 reports results for the SFMIC model, and the speed / accuracy trade-off of both models is explored on a complete real-time ASR system in Section 6.2.7. Section 6.2.8 reports results for the VLMIC model, and finally the class-based approach is explored in Section 6.2.9.

6.2.1 Experimental Setup

The recognition engine used is a context-dependent HMM system with 3358 triphones and tied-mixtures based on genones [23]: each state cluster shares a common set of Gaussians, while the mixture weights are state-dependent. The system has 1500 genones and 32 Gaussians per genone. The test-set is a collection of 10397 utterances of Italian telephone speech spanning several tasks, including digits, letters, proper names and command lists, with fixed task-dependent grammars for each test-set. The features are 9-dimensional MFCC with Δ and Δ^2.

The training data comprises 89000 utterances. Each model is trained using fixed HMM alignments for fair comparison. The GMM are initially trained using full or block-diagonal covariances — depending on the MIC structure used — using Gaussian splitting (see Section 1.3.1). After the number of Gaussian per genone is reached using splitting, the sufficient statistics are collected and the MIC model trained in one iteration. For this reason, the performance results reported here are lower bounds on the accuracy that is achievable using the MIC model. Better performance would certainly be achieved by jointly optimizing the alignments and by reiterating the MIC training a few times.

The accuracy is evaluated using a sentence understanding error rate, which measures the proportion of utterances in the test-set that were interpreted incorrectly. The Gaussian exponent α (see Section 2.4) was globally optimized for each model on the entire collection of test-sets.

6.2.2 Comparison against Semi-Tied Covariances

The semi-tied covariance model [35] is a very closely related model to the MIC as discussed in Section 2.3. To compare the two approaches, the number of Gaussian-specific parameters in the GMM was kept constant (27) for the MIC and semi-tied models. Table 6.1 shows the error rate on the test-set described previously. The error rate reduction using the MIC model is more than 3 times the error rate reduction obtained with semi-tied covariances.

Structure	Error Rate	Relative Improvement
Diagonal	9.64%	-
Semi-tied	9.24%	4.1%
MIC	8.29%	14.0%

Table 6.1: Error rates on a set of Italian tasks

6.2.3 Accuracy versus Complexity

Figure 6.6 shows how the model performs as the number of Gaussian-specific parameters change. The MIC model almost matches the performance of a full-covariance system with about 45 Gaussian-specific parameters. As few as 9 parameters are sufficient for the model to match the accuracy of the diagonal covariance system.

This demonstrates that, as expected from the fact that it is more general mathematically than both the diagonal and semi-tied model (see Section 2.3), the MIC model improves the power of a GMM model. It also adds an additional degree of freedom – the number K of prototypes – which allows the system to efficiently trade off complexity against modeling accuracy. Finally, the model reaches a modeling precision equivalent to the precision of a full covariance model with far fewer parameters, which is an improvement upon the full GMM model which comes at no cost

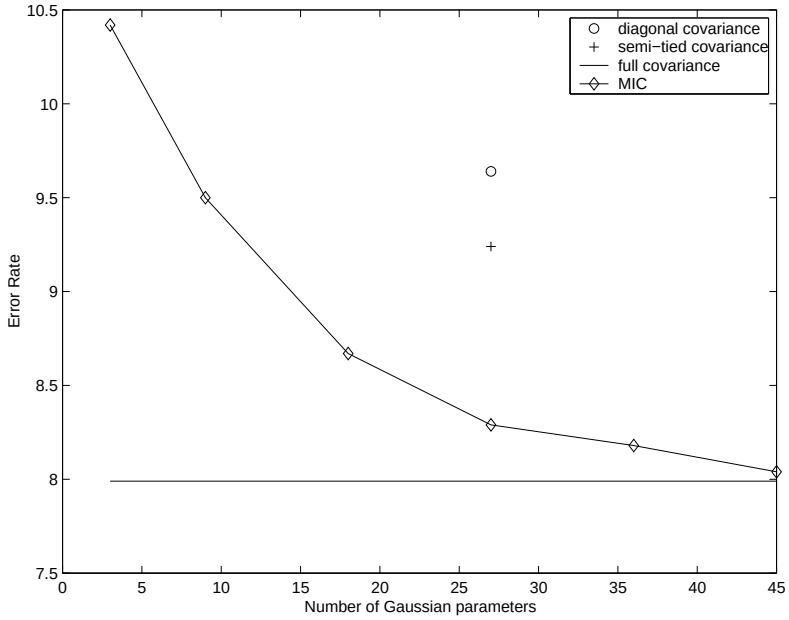

Figure 6.6: Accuracy as a function of the number of Gaussian-specific parameters. The performance of the diagonal system is around 10%. As the number of Gaussian-specific parameters grows, the accuracy of the MIC approaches the accuracy of the full covariance model.

whatsoever.

6.2.4 Complexity of the Estimation Algorithm

The theoretical complexity of the MIC estimation algorithm is difficult to assess exactly. In [15], it is suggested that for the Newton algorithm: "Once the quadratic convergence phase is reached, at most six or so iterations are required to produce a solution of very high accuracy".

This is exactly what is observed during the weights reestimation, regardless of the dimensionality. The number of iterations of the prototype reestimation algorithm, as well as the total number of alternated optimizations, also appear to be rather independent of the dimensionality of the problem. This means that the complexity of a single step of the iteration process will predicate the complexity of the whole algorithm. The dominant computation is the matrix inversion when turning inverse

covariances into covariances to compute the gradients and Hessians. This computational cost is thus roughly $\mathcal{O}(KMD^3)$, with K the number of prototypes, M the number of Gaussians, and D the dimensionality.

In practice, this rough estimate corresponds well to the observed behavior. A benchmark C++ implementation of the MIC reestimation algorithm was tested on an Intel Pentium 4 machine running at 3 GHz, under the Sun Solaris 5.8 operating system. The results below correspond to running a single iteration of steps 3 through 11 of the algorithm in Table 1.1. In all cases, $M = 48000$. Figure 6.7 shows that the complexity is indeed linear in the number of prototypes. Figure 6.8 shows the non-linear growth of the number of computations as a function of the dimensionality of the problem. The estimation in any of these models was never longer than a few hours on a single CPU, which makes the algorithm practical for any comparable setup. For models with much larger dimensionalities, the cost of estimating the prototypes can become more of an issue. In this case, the SFMIC is a good alternative to explore, since its estimation complexity is linear in the number of subspaces, while being only non-linear in the (smaller) dimensionality of each subspace.

6.2.5 Progressive Estimation of the Weights

In Section 2.5.4, a fast approximate reestimation algorithm for the MIC weights has been presented. The performance of the algorithm compared with the maximum likelihood algorithm is presented in Table 6.2.

Algorithm	# Prototypes	Error Rate
ML	9	9.50%
Progressive	9	9.54%
ML	27	8.29%
Progressive	27	8.91%

Table 6.2: Comparison between weight reestimation algorithms

It appears that the accuracy loss due to the suboptimality of the progressive reestimation algorithm is limited when a small number of prototypes is used, while being

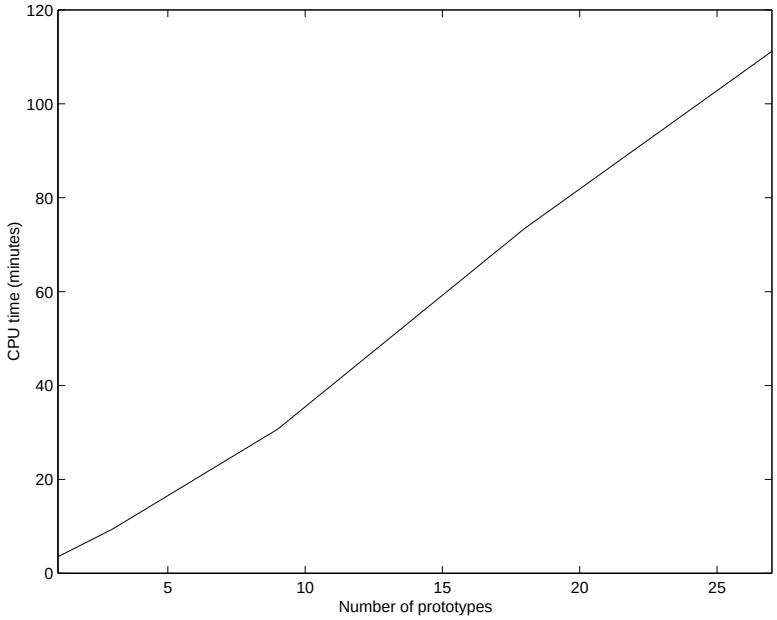

Figure 6.7: CPU time, in minutes, used during MIC estimation, on a 3 GHz machine, as a function of the number of prototypes in the system. The number of Gaussians is 48000, and the dimensionality 27.

much larger on longer decompositions. The rate of convergence of the complete progressive reestimation algorithm, including the prototype update, is also significantly slower, although the weight reestimation step is much faster.

6.2.6 SFMIC Experiments

From the analysis in Section 6.1, we would expect two things from a subspace-factored model using MIC:

1. In the limit of large number of Gaussian-specific parameters, the model should tend to the performance of a system where each Gaussian has a separate block-diagonal covariance (Figure 6.5). Thus its performance will be worse than a full covariance system.

2. In the limit of small number of Gaussian-specific parameters, the subspace-factored systems should outperform a full-covariance MIC system due to the

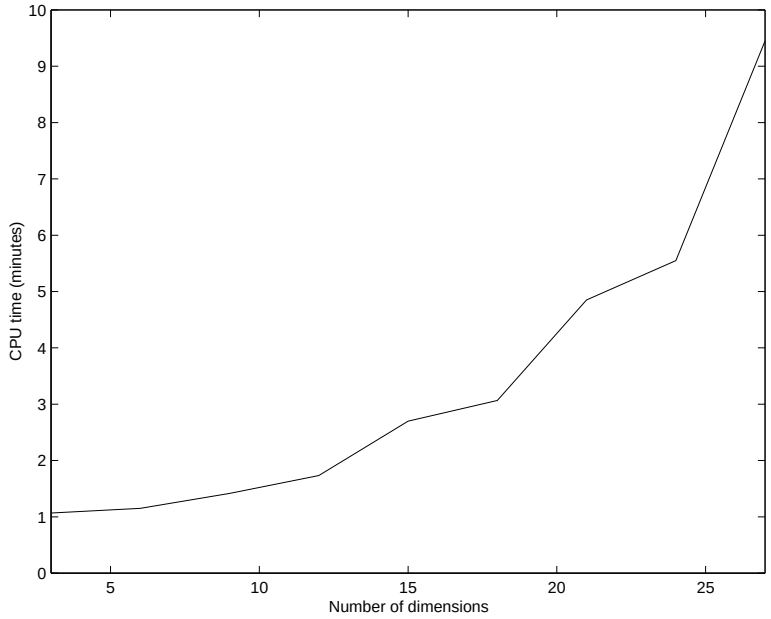

Figure 6.8: CPU time, in minutes, used during MIC estimation, on a 3 GHz machine, as a function of the dimensionality of the input vector. The number of Gaussians is 48000, and the number of prototypes 3.

effect of having a much larger number of effective prototypes in the system for a same number of weights.

Figure 6.9 shows that it is indeed the case: with 9 parameters, the 3-block system performs as well as the 1-block system, and outperforms it with only 3 parameters, while the 2-block system outperforms the 1-block system up to approximately 16 Gaussian-specific parameters. In these experiments, the number of parameters allocated to each block was kept proportional to the block size, but the allocation scheme could also be optimized.

Note that because of the front-end computations, for a given number of Gaussian-specific parameters, the computational complexity of a 3-block system will be lower than the computational complexity of a 2-block system, which in turn will be lower than the computational complexity of a 1-block system. This means that in the limit of low number of Gaussian-specific parameters, although the accuracy of a 2-block system is comparable to the accuracy of a 3-block system, the latter will be

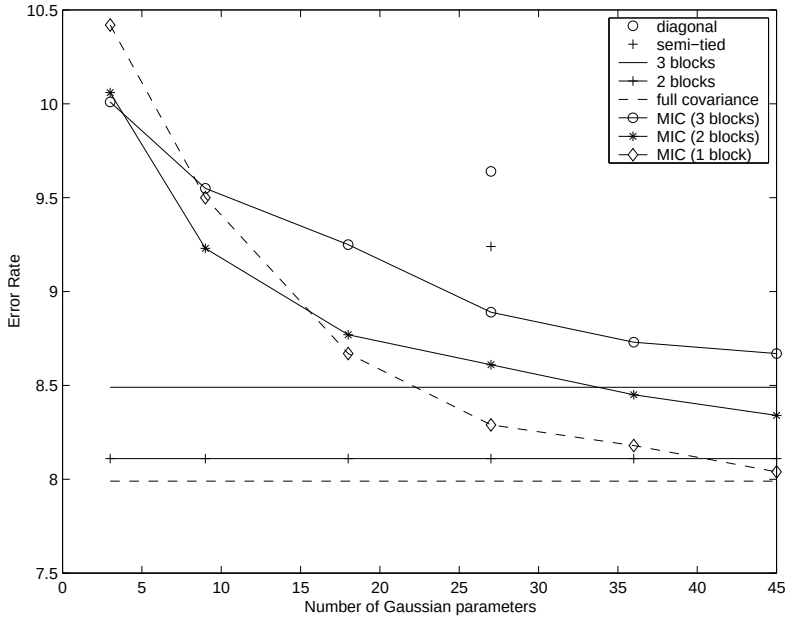

Figure 6.9: Accuracy as a function of the number of Gaussian-specific parameters for the 2-block and 3-block subspace-factored approach, compared with the 1-block full covariance system.

computationally more efficient.

6.2.7 Speed versus Accuracy

In the following experiments, tasks with small perplexity, i.e. with a small average number of possible spoken utterances such as digit strings and "yes or no" confirmations, were benchmarked separately from the large perplexity ones such as names lists or business listing queries. Figures 6.10 and 6.11 show how various configurations perform in real-time environments, respectively on small and large perplexity tasks. Each curve depicts the performance of a given system at a various operating points obtained by varying the degree of pruning in the acoustic search. By using a tight pruning, the recognizer evaluates only a few alternative recognition hypotheses at any point in time, which causes the system to operate faster while increasing the error rate due to the larger number of correct recognition hypotheses which are potentially

discarded. On the other hand, a loose pruning causes the system to evaluate many more hypotheses, improving the accuracy while slowing down the system. Thus, by trading the number of search errors against the number of active hypotheses in the search, the accuracy of the system can be traded against its speed. Both the small and large perplexity test-sets are drawn from the Italian test-set described in Section 6.2.1, and contain respectively 5098 and 4612 utterances.

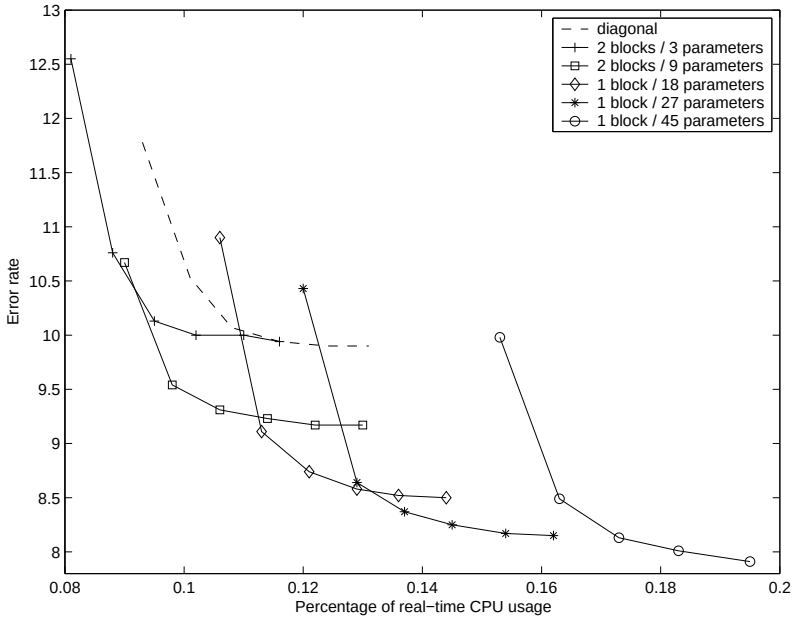

Figure 6.10: Speed / accuracy trade-off on a set of low-perplexity tasks. The error rate is plotted against the fraction of real-time CPU computations required to perform recognition.

Because of the larger front-end overhead incurred by systems using the MIC model with full covariances (1 block), the relative slowdown on test-sets with low perplexity is much larger than the slowdown on high-perplexity test-sets. Indeed, for small perplexity test-sets, the front-end computations are a much larger proportion of the total computational cost, and any additional cost in the front-end is significant. When using models with multiple blocks, this effect is much smaller and does not appear to influence the results. Thus, the faster 2-block systems scale with the perplexity of the task approximatively in the same way as the diagonal model does. The speed

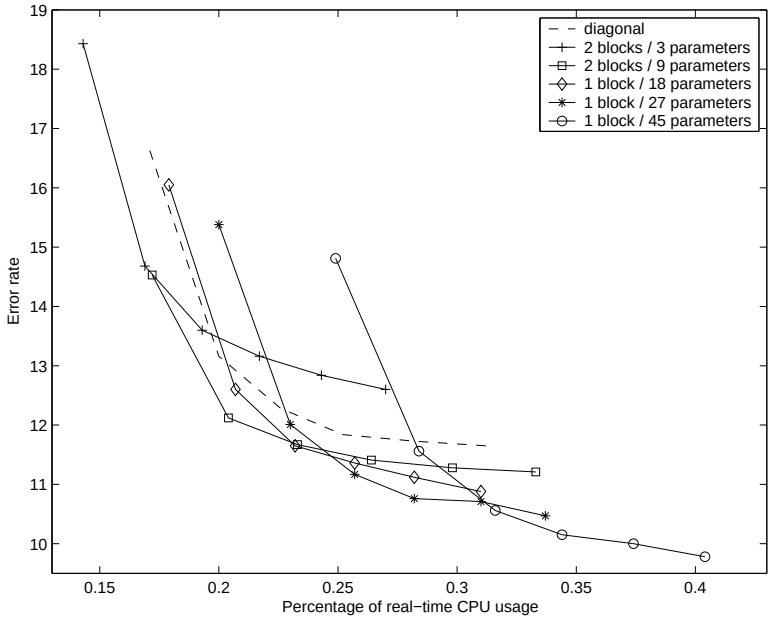

Figure 6.11: Speed / accuracy trade-off on a set of large-perplexity tasks for the same configurations as Figure 6.10.

improvement of a 3-block system (not plotted) compared to a 2-block system with similar complexity is never large enough to compensate for the loss in accuracy.

Typically, an optimally tuned recognizer would operate in the lower-right half of the speed/accuracy curve, close to the knee of the curve, where the efficiency of the system is maximized while not sacrificing accuracy by any significant amount. For both the small and large perplexity test-sets, the 9 parameter / 2 blocks system is the fastest model that would operate at the same level of accuracy as the baseline diagonal model at its optimal operating point. In both cases, the speed increase is about 10% at no cost in accuracy. In both cases as well, the full covariance MIC system is the most accurate at the same speed as the diagonal system at its optimal operating point. The accuracy gain without any slowdown is about 13% for the low-perplexity test-sets, and 8% for the high-perplexity test-sets.

Overall, the different model architectures allow for a wide range of operating points, and makes a system with an accuracy comparable to the accuracy of a full covariance MIC system (45 parameter / 1 block) reachable at an additional cost in

computations of approximately 50%. On the same test-set, the increase of computation incurred when using a full covariance model is approximately 1100%.

6.2.8 VLMIC Experiments

Table 6.3 shows the error rate achieved on the set of Italian tasks using several setups: the baseline model is a fixed-length model with 12 weights, which is compared with a variable-length model with the same average number of weights. In these experiments, only the genones corresponding to triphone models were trained using a variable-length optimization. The other genones in the system were trained with a fixed number of weights equal to the average number of weights. The variable-length model achieves an improved accuracy of about 5.6%. A fixed length model with the same total number of prototypes (18) achieves a 6.3% relative improvement on the same task.

Table 6.3: Error rates on a set of Italian tasks.

Type	$K_{\min}$	K	$K_{\max}$	Error Rate
Diagonal cov.				9.64%
Fixed length		12		9.25%
Variable length	2	12	15	9.04%
Variable length	2	12	18	8.73%
Fixed length		18		8.67%

This demonstrates that a better accuracy can be achieved with the same overall number of Gaussian-dependent parameters.

Figure 6.12 shows the speed / accuracy trade-offs attained by the variable-length models. Each curve displays the error rate against the speed of a given system when the level of pruning in the acoustic search is varied. The variable-length system with $K_{\max} = 15$ matches the speed of the 12 weight, fixed-length model at aggressive levels of pruning, while leading to better accuracy for larger pruning thresholds. The variable-length system with $K_{\max} = 18$ matches closely the accuracy of the 18 weight, fixed-length model at large pruning thresholds, while being faster at a given error rate at lower pruning levels. Overall, the variable length systems are capable of achieving

trade-offs that were not attained by the fixed-length models.

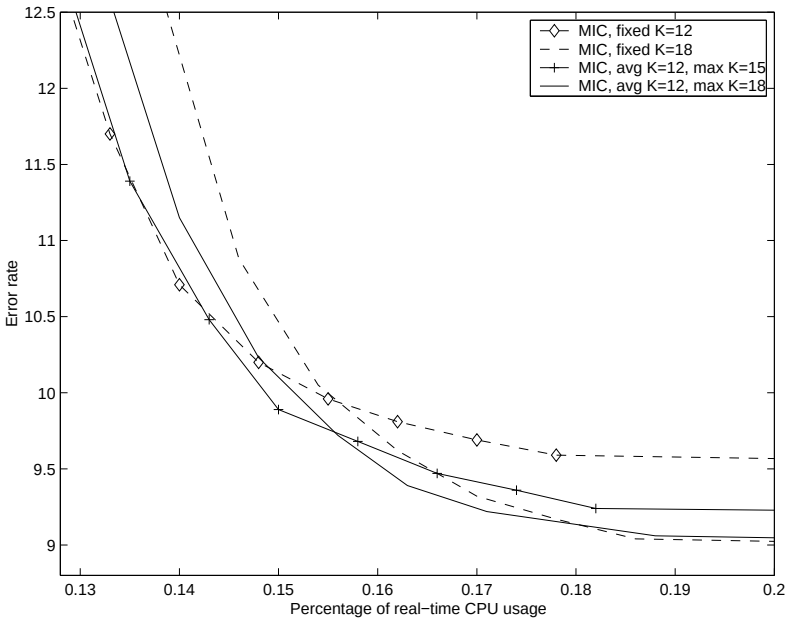

Figure 6.12: Speed / accuracy trade-off on the set of Italian tasks. The curves are generated by varying the level of pruning in the acoustic search.

6.2.9 Class-based Approach

Table 6.4 compares the performance of a 2-block system with systems for which the acoustic model is partitioned into a series of phonetically-derived classes. The gains obtained from using a class-based approach are small, and do not compare well to the gains that would be obtained by increasing the number of Gaussian-specific parameters. While it is possible that the phonetic clustering used here is sub-optimal, and that a more data-driven approach would show larger gains, it is very likely that with such a large number of Gaussians in the system, the optimal set of prototypes derived for a particular phonetic class is close to the optimal for the entire GMM, and that significant accuracy benefits will only show with a much larger set of classes, which makes the approach unappealing in this context. Nevertheless, since the front-end overhead for 2-block systems is rather small, these small accuracy gains come with

# parameters	# classes	Error Rate
9	1	9.23%
9	3	9.14%
9	11	9.10%
27	1	8.61%
27	3	8.62%
27	11	8.48%

Table 6.4: Error rates for 2-block systems for various numbers of class-based MIC models in the system. Each class is derived by clustering the HMM states using their phonetic labels.

an extremely limited computational cost and can be of interest in contexts where the Gaussian computations dominate the front-end processing.

Chapter 7

Conclusion

We presented a low-complexity approximation to full covariance Gaussian mixture models, along with robust maximum likelihood estimation algorithms to compute the parameters of the model. A low-complexity subspace-factored approach extending that model, a class-based model and a variable length model were also introduced. Each class of models were applied to acoustic-modeling for ASR. When used in the context of a GMM/HMM acoustic models, these models lead to a broad range of systems which vastly improve the tradeoff between speed and accuracy of the system.

There are several directions in which this study can be further extended:

- in [3], it is suggested that the mixture could be extended to further include the mean vector in the tying. While the gains one would expect from this development are substantially smaller, it might be an interesting avenue to pursue,

- the complexity of the MIC reestimation algorithm can also be limiting in some applications, and it could be worthwhile looking at other more efficient convex optimization methods to speed up the process. In particular, it would be interesting to compare the complexity of the proposed implementation with the techniques proposed in [79],

- amongst the unanswered questions is also the problem of discriminative training: the most popular discriminative training method for GMM, maximum mutual

information estimation (MMIE) [5, 82] has mostly been studied in the context of diagonal covariances[1]. How these techniques generalize to full covariances and further to MIC models is still open,

- another interesting study would be to carefully assess the behavior of the model in situations where the amount of training data is insufficient for a full covariance model to be usable. The parametric compression is expected to reduce significantly the amount of data required to train an accurate model,

- another interesting avenue to explore is to quantify how much the improved modeling can help reducing the number of Gaussians used per state density in ASR. Since it is usually assumed that when using diagonal Gaussians the correlations are modeled by using many Gaussians for each "mode" of the distribution, the explicit modeling of the correlations is expected to reduce the number of Gaussians required,

- it would be interesting to apply these techniques to other problems involving large GMM, outside of the realm of speech. GMM used in image classification, because their covariance structure is by nature of the problem very non-diagonal, seem like prime candidates for this study,

- efficient reestimation formulas for the full transformed SFMIC model of section 4.1.1 are still to be derived. The analogy with semi-tied covariance reestimation should be a good guide in deriving those,

- it would be interesting to extend the progressive scheme to estimate the weights to the prototypes as well, with the expectation that the suboptimality of the algorithms will not impair the accuracy of the resulting model too much. When both the prototypes and the weights are estimated progressively, the complexity of the decomposition can be varied online easily by dynamically setting the number of prototypes used. This can be extremely useful to trade accuracy

[1] See [72, Chapter 3] for an overview of MMIE which proposes a novel algorithm which significantly improves on [82]

against speed in applications which are bound to CPU with highly variable loads,

- the basis expansion which is central to the MIC model is reminiscent of many other data analysis methods which project the data onto appropriately discriminative subspaces to measure distances between data points and perform clustering, classification, principal component analysis, etc. The use of the MIC expansion framework for analyzing covariances might be very instructive as well: when prototypes are shared across Gaussians – i.e. when the projection basis is consistent across them – the weight vector $\mathbf{\Lambda}_i$ is a compact representation of the covariance which can be the basis of a distance measure between covariances, for example Euclidian: $\|\mathbf{\Lambda}_i - \mathbf{\Lambda}_j\|^2$. What meaning and what uses the topology induced by such metric over the space of covariances can have is still to be explored.

Bibliography

[1] A. Aiyer. *Robust Image Compression using Gauss Mixture Models*. PhD thesis, Stanford University, School of Engineering, 2001.

[2] A. Albert. *Regression and the Moore-Penrose Pseudoinverse*. Academic Press, New York and London, 1972.

[3] S. Axelrod, R. Gopinath, and P. Olsen. Modeling with a subspace constraint on inverse covariance matrices. In *Proceedings of the International Conference on Spoken Language Processing, ICSLP*, pages 2177–2180, 2002.

[4] S. Axelrod, R. Gopinath, P. Olsen, and K. Visweswariah. Dimensional reduction, covariance modeling, and computational complexity in ASR systems. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, volume 1, pages 912–915, 2003.

[5] L.R. Bahl, P.F. Brown, P.V. de Souza, and R.L. Mercer. Maximum mutual information estimation of hidden Markov model parameters for speech recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, volume 1, pages 49–52, May 1986.

[6] L.E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, 1:1–8, 1972.

[7] J.R. Bellegarda and D. Nahamoo. Tied mixture continuous parameter modeling for speech recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 38(12):2033–2045, December 1990.

[8] M. Berthold and D. Hand, editors. *Intelligent Data Analysis, An Introduction*. Springer-Verlag, 1999.

[9] J.C. Bezdek, R.J. Hathaway, R.E. Howard, C.A. Wilson, and M.P. Windham. Local convergence analysis of a grouped variable version of coordinate descent. *Journal of Optimization Theory and Applications*, 54(3):471–477, 1987.

[10] J.A. Bilmes. A gentle tutorial of the EM algorithm and its applications to parameter estimation for Gaussian mixture and HMM. Technical Report TR-97-021, UC Berkley, 1998. http://www.cs.ucr.edu/~stelo/cs260/bilmes98gentle.pdf.

[11] J.A. Bilmes. Factored sparse inverse covariance matrices. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, 2000.

[12] E. Bocchieri. Vector quantization for efficient computation of continuous density likelihoods. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, volume 2, pages 692–695, Minneapolis, USA, 1993.

[13] H. Bourlard and N. Morgan. *Connectionist Speech Recognition - A Hybrid Approach*. Kluwer Academic Publishers, Massachusetts, USA, 1994.

[14] S. Boyd and L. El Ghaoui. Method of centers for minimizing generalized eigenvalues. *Linear Algebra and Applications, special issue on Numerical Linear Algebra Methods in Control, Signals and Systems*, 188:63–111, 1993.

[15] S. Boyd and L. Vandenberghe. *Convex Optimization*. draft available on the web, `http://www.stanford.edu/~boyd/cvxbook.html`, 2003.

[16] S. Chen and R. A. Gopinath. Gaussianization. In *Proceedings of the Neural Information Processing Systems Conference, NIPS*, 2000.

[17] E.K.P. Chong and S.H. Zak. *An Introduction to Optimization*. Wiley Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Inc., NY, USA, second edition, 2001.

[18] R. Clarke. Relation between the Karhunen Loève and cosine transforms. *IEE Proceedings*, 128(6-F):359–360, November 1981.

[19] M. Cohen. *(in preparation).* Addison Wesley, 2003.

[20] S.B. Davis and P. Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. In A. Waibel and K-F. Lee, editors, *Readings in Speech Recognition*, volume 1, pages 65–74. Morgan Kaufmann Publishers, March 1990.

[21] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[22] S. Dharanipragada and K. Visweswariah. Covariance and precision modeling in shared multiple subspaces. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, volume 1, pages 904–907, 2003.

[23] V. Digalakis, P. Monaco, and H. Murveit. Genones: Generalized mixture tying in continuous hidden Markov model-based speech recognizers. *IEEE Transactions on Speech and Audio Processing*, 4(4):281–289, 1996.

[24] V. Digalakis and H. Murveit. Genones: Optimizing the degree of mixture-tying in a large-vocabulary HMM-based speech recognizer. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, volume 1, pages 537–540, 1994.

[25] V. Digalakis, L. Neumeyer, and M. Perakakis. Product-code vector quantization of cepstral parameters for speech recognition over the WWW. In *Proceedings of the International Conference on Spoken Language Processing, ICSLP*, 1998.

[26] V. Digalakis, S. Tsakalidis, C. Harizakis, and L. Neumeyer. Efficient speech recognition using subvector quantization and discrete-mixture HMM. *Computer Speech and Language*, 14(1):33–46, January 2000.

[27] P. Ding, S. Zhang, and B. Xu. Comparison and study of some variants of partially tied covariance modeling. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, volume 1, pages 908–911, 2003.

[28] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification.* John Wiley & Sons, Inc., NY, USA, second edition, 2001.

[29] T. Eisele, R. Haeb-Umbach, and D. Langmann. A comparative study of linear feature transformation techniques for automatic speech recognition. In *Proceedings of the International Conference on Spoken Language Processing, ICSLP*, pages 252–255, 1996.

[30] R. Kuhn et al. Eigenvoices for speaker adaptation. In *Proceedings of the International Conference on Spoken Language Processing, ICSLP*, pages 1771–1774, 1998.

[31] G. D. Forney. The Viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[32] J. Fritsch. *Hierarchical Connectionist Acoustic Modeling for Domain-Adaptive Large Vocabulary Speech Recognition.* PhD thesis, Fakultät für Informatik, Universität Karlsruhe (TH), October 1999.

[33] J. Fritsch and I. Rogina. The bucket box intersection (BBI) algorithm for fast approximative evaluation of diagonal mixture Gaussians. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, pages 837–840, Atlanta, GA, 1996.

[34] M.J.F. Gales. Adapting semi-tied full-covariance matrix HMMs. Technical Report CUED/F-INFENG/TR.298, Cambridge University, July 1997.

[35] M.J.F. Gales. Semi-tied covariance matrices for hidden Markov models. *IEEE Transactions on Speech and Audio Processing*, 7(3):272–281, 1999.

[36] M.J.F. Gales. Cluster adaptive training of hidden Markov models. *IEEE Transactions on Speech and Audio Processing*, 8:417–428, 2000.

[37] M.J.F. Gales, D. Pye, and P. Woodland. Variance compensation within the MLLR framework for robust speech recognition and speaker adaptation. In *Proceedings of the International Conference on Spoken Language Processing, ICSLP*, volume 3, pages 1832–1835, Philadelphia, PA, 1996.

[38] J.-L. Gauvain and C.H. Lee. Maximum a posteriori estimation for multivariate gaussian mixture observations of Markov chains. *IEEE Transactions on Speech and Audio Processing*, 2(2):291–298, April 1994.

[39] GenBank. `http://www.ncbi.nlm.nih.gov/Database/`.

[40] J.E. Gentle. *Numerical Linear Algebra for Applications in Statistics*. Springer-Verlag, 1998.

[41] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, Massachusetts, USA, 1992.

[42] R.M. Gray. Gauss mixtures quantization: clustering Gauss mixtures. In D.D. Denison, M.H. Hansen, C.C. Holmes, B. Mallick, and B. Yu, editors, *Proceedings of the Math Sciences Research Institute Workshop on Nonlinear Estimation and Classification, Mar. 17-29, 2002*, pages 189–212. Springer-Verlag, 2003.

[43] R.M. Gray and T. Linder. Mismatch in high rate entropy constrained vector quantization. *IEEE Transactions on Information Theory*, May 2003.

[44] L.R. Haff. Empirical Bayes estimation of multivariate normal covariance matrix. *Annals of Statistics*, 8:586–597, 1980.

[45] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer-Verlag, 2001.

[46] R.J. Hathaway, Y.K. Hu, and J.C. Bezdek. Local convergence of tri-level alternating optimization. *Neural, Parallel & Scientific Computations*, 9:19–28, 2001.

[47] H. Hermansky. Perceptual linear predictive (PLP) analysis of speech. *Journal of the Acoustical Society of America*, 87(4):1738–1752, 1990.

[48] H. Hermansky and N. Morgan. Rasta processing of speech. *IEEE Transactions on Speech and Audio Processing*, 2(4):578–589, 1994.

[49] M. M. Hochberg, S. J. Renals, A. J. Robinson, and D. J. Kershaw. Large vocabulary continuous speech recognition using a hybrid connectionist-HMM system. In *Proceedings of the International Conference on Spoken Language Processing, ICSLP*, 1994.

[50] X. D. Huang, Y. Ariki, and M. A. Jack. *Hidden Markov Models for Speech Recognition*. Edinburgh University Press, Edinburgh, 1990.

[51] F. Jelinek. *Statistical Methods for Speech Recognition*. MIT Press, 1998.

[52] D. Jurafsky and J.H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, 2000.

[53] K. Kirchhoff. Integrating articulatory features into acoustic models for speech recognition. In *Proceedings of Workshop PhonASR*, Saarbrücken, Germany, May 2000.

[54] T. Kubokawa. Shrinkage and modification techniques in estimation of variance and the related problems: A review. *Communication in Statistics - Theory and Methods*, 28:613–650, 1999.

[55] S. Kullback. *Information Theory and Statistics*. Dover, New York, 1968. (Reprint of 1959 edition published by Wiley.).

[56] O. Ledoit. *Essays on risk and return in the stock market*. PhD thesis, Massachusetts Institute of Technology, Sloan School of Management, 1995.

[57] C. J. Leggetter and P. C. Woodland. Maximum likelihood linear regression for speaker adaptation of continuous density HMMs. *Computer Speech and Language*, May 1995.

[58] Linguistic data consortium. `http://www.ldc.upenn.edu`.

[59] B. Mak and E. Bocchieri. Direct training of subspace distribution clustering hidden Markov model. *IEEE Transactions on Speech and Audio Processing*, 9(4):378–387, May 2001.

[60] C.D. Manning and H. Schutze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[61] G.J. McLachlan and T. Krishnan. *The EM algorithm and Extensions*. John Wiley & Sons, Inc., NY, USA, 1997.

[62] H. Murveit, P. Monaco, V. Digalakis, and J. Butzberger. Techniques to achieve an accurate real-time large vocabulary speech recognition system. In *Proceedings of ARPA Workshop on Human Language Technology*, pages 368–373, Princeton, New Jersey, USA, March 1994.

[63] C. Nass. *Voice Interfaces: The Psychology and Design of Interfaces that Talk and Listen*. in preparation, 2003.

[64] National center for biotechnology information. `http://www.ncbi.nlm.nih.gov`.

[65] P. Olsen and R. Gopinath. Modeling inverse covariance matrices by basis expansion. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, 2002.

[66] M. Ostendorf, V. Digalakis, and O. Kimball. From HMMs to segment models: A unified view of stochastic modeling for speech recognition. *IEEE Transactions on Speech and Audio Processing*, 4(5), September 1996.

[67] Physionet: The research resource for complex physiologic signals database. `http://www.physionet.org`.

[68] L. Rabiner and B.-H. Juang. *Fundamentals of Speech Recognition*. Prentice Hall, 1993.

[69] A. Sankar. Robust HMM estimation with Gaussian merging-splitting and tied-transform HMMs. In *Proceedings of the International Conference on Spoken Language Processing, ICSLP*, 1998.

[70] I. Shafran and M. Ostendorf. Use of higher level linguistic structure in acoustic modeling for speech recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, 2000.

[71] C. Stein. Inadmissibility of the usual estimator for the variance of a normal distribution with unknown mean. *Annals of the Institute of Statistical Mathematics*, 16:155–160, 1964.

[72] R. Teunen. *Acoustic Modeling For Automatic Speech Recognition: Deriving Discriminative Gaussian Networks*. PhD thesis, Department of Electrical Engineering, Stanford University, August 2002.

[73] V. Vanhoucke, M.M. Hochberg, and C.J. Leggetter. Speaker-trained recognition using allophonic enrollment models. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU*, 2001.

[74] V. Vanhoucke, W.L. Neeley, M. Mortati, M.J. Sloan, and C. Nass. Effects of prompt style when navigating through structured data. In *Proceedings of INTERACT 2001, Eighth IFIP TC.13 Conference on Human Computer Interaction*, pages 530–536, Tokyo, Japan, 2001. IOS Press.

[75] V. Vanhoucke and A. Sankar. Mixtures of inverse covariances. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, volume 1, pages 900–903, 2003.

[76] V. Vanhoucke and A. Sankar. Mixtures of inverse covariances (submitted). *IEEE Transactions on Speech and Audio Processing*, 2003.

[77] V. Vanhoucke and A. Sankar. Variable length mixtures of inverse covariances. In *Proceedings of the European Conference on Speech Communication and Technology, EUROSPEECH*, 2003.

[78] V. Vanhoucke and R. Silipo. Interpretability in multidimensional classification. In J. Casillas, O. Cordon, F. Herrera, and L. Magdalena, editors, *Interpretability Issues in Fuzzy Modeling*, volume 128 of *Studies in Fuzziness and Soft Computing*. Springer-Verlag, 2003.

[79] K. Visweswariah, P. Olsen, R. Gopinath, and S. Axelrod. Maximum likelihood training of subspaces for inverse covariance modeling. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, 2003.

[80] K. Weber, S. Bengio, and H. Bourlard. HMM2- a novel approach to HMM emission probability estimation. In *Proceedings of the International Conference on Spoken Language Processing, ICSLP*, 2000.

[81] R. Westwood. Speaker adaptation using eigenvoices. Master's thesis, Department of Engineering, Cambridge University, 1999.

[82] P.C. Woodland and D. Povey. Large scale discriminative training for speech recognition. In *Proceedings of ISCA ITRW ASR2000*, 2000.

[83] S.-L. Wu, B. Kingsbury, N. Morgan, and S. Greenberg. Incorporating information from syllable-length time scales into automatic speech recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, pages 721–724, 1998.

[84] S. Yoon, K. Pyun, C.S. Won, and R.M. Gray. Image classification using GMM with context information and reducing dimension for singular covariance. In *Proceedings of the Data Compression Conference, DCC*, 2003.

[85] J.C. Young. *Clustered Gauss Mixture Models for Image Retrieval*. PhD thesis, Stanford University, School of Engineering, 2003.

[86] S.J. Young. The general use of tying in phoneme-based HMM speech recognisers. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, pages 569–572, 1992.

[87] E. Zwicker, H. Fastl, and H. Frater. *Psychoacoustics: Facts and Models*, volume 22 of *Springer Series in Information Sciences*. Springer-Verlag, 1999.