國立交通大學

資訊科學與工程研究所

博士論文

機率式模型分群法之研究與其應用

Probabilistic Model-based Clustering and Its Applications

研究生：鄭士賢

指導教授：傅心家　教授

王新民　博士

中華民國九十八年五月

機率式模型分群法之研究與其應用
Probabilistic Model-based Clustering and Its Applications

研究生：	鄭士賢	Student：	Shih-Sian Cheng
指導教授：	傅心家 教授	Advisor：	Prof. Hsin-Chia Fu
	王新民 博士		Dr. Hsin-Min Wang

國立交通大學
資訊科學與工程研究所
博士論文

A Dissertation
Submitted to Department of Computer Science
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

in

Computer Science

May 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年五月

機率式模型分群法之研究與其應用

學生：鄭士賢 指導教授：傅心家 教授

王新民 博士

國立交通大學資訊科學與工程研究所

摘 要

機率式模型分群法藉由學習一個有限混合模型(finite mixture model)而達成資料分群的目的，其已經有很多成功的應用；而最常用的混合模型乃是高斯混合模型(Gaussian mixture model, i.e., GMM)。當其目的函數的最佳化無法用分析方法來達成時，我們經常用迭代式、局部最佳(local-optimal)的演算法來學習混合模型。此外，因為來自於混合模型的資料樣本有非完全(incompleteness)的特性，因此我們通常採用 EM 形式(EM-type)的演算法，如 EM 演算法與分類 EM 演算法(classification EM, i.e., CEM)。然而，用傳統 EM 形式演算法來做模型分群有幾個缺點：一、它的效能與模型初始值有高度相關性；二、混合元件(mixture component)的個數需要事先給定；三、在完成分群之後，資料樣本與群聚之間的拓樸關係(topological relationships)無法被保留。在本論文中，針對上述前兩樣缺點，我們提出一個自我分裂之高斯混合模型學習演算法(self-splitting Gaussian mixture learning, i.e., SGML)。此法起始於單一元件，然後迭代式地，用貝氏資訊法則(Bayesian information criterion, i.e., BIC)來確認分裂的有效性，分裂某一個元件而成為兩個新元件直到最佳的元件個數被找到為止，最佳的元件個數亦是用 BIC 來決定。基於此分裂程序，SGML 可為學習不同模型複雜度的 GMM 提供不錯的模型初始值。關於拓樸保留(topology-preserving)的議題，我們將一個機率式自我組織圖(probabilistic self-organizing map, i.e., PbSOM)的學習視為一種可保留資料樣本與群聚之間拓樸關係於一個神經網路(neural network)的模型分群法。基於此概念，我們為 PbSOM 發展了一個耦合概似混合模型(coupling-likelihood mixture model)，其延伸 Kohonen SOM 的參考向量(reference vectors)至多變量高斯模型。基於最大概似法則

(maximum likelihood criterion)，我們亦發展了三個學習 PbSOM 的 EM 形式演算法，亦即 SOCEM、 SOEM、以及 SODAEM 演算法。SODAEM 是 SOCEM 與 SOEM 的決定性退火(deterministic annealing, i.e., DA)變形；此外，藉由逐漸縮小鄰域大小(neighborhood size)，SOCEM 與 SOEM 可分別被解釋成：高斯模型分群的 CEM 與 EM 演算法的 DA 變形。實驗結果顯示，我們所提出的 PbSOM 演算法與決定性 EM(DAEM)演算法有相近的分群效能，並能維持拓樸保留之特性。關於應用方面，我們用 SGML 來訓練用於語者識別(speaker identification)的語者 GMMs;實驗結果顯示，SGML 可以自動地為個別語者 GMM 決定適當的模型複雜度，而此在文獻裡通常是用經驗法則來決定的。我們將所提出的 PbSOM 學習演算法應用於資料視覺化與分析;實驗結果顯示它們可用於在一個二維的網路上探索資料樣本與群聚之間的拓樸關係。此外，我們提出幾種分割且克服(divide-and-conquer)的方法來做音訊分段(audio segmentation)，其用 BIC 來評定兩個音段之間的不相似度(dissimilarity);在廣播新聞的實驗結果顯示，相較於公認最佳的視窗成長分段法(window-growing-based segmentation)，我們的方法不僅有比較低的計算量，亦有較高的分段正確性。

Probabilistic Model-based Clustering and Its Applications

Student：Shih-Sian Cheng　　Advisors：Prof. Hsin-Chia Fu
Dr. Hsin-Min Wang

Department of Computer Science
National Chiao Tung University

ABSTRACT

Probabilistic model-based clustering is achieved by learning a finite mixture model (usually a Gaussian mixture model, i.e., GMM) and has been successfully applied to many tasks. When the objective likelihood function cannot be optimized analytically, iterative, locally-optimal learning algorithms are usually applied to learn the model. Moreover, because the data samples drawn from a mixture model have the property of incompleteness, EM-type algorithms, such as EM and classification EM (CEM), are usually employed. However, model-based clustering based on conventional EM-type algorithms suffer from the following shortcomings: 1) the performance is sensitive to the model initialization; 2) the number of mixture components (data clusters) needs to be pre-defined; and 3) the topological relationships between data samples and clusters are not preserved after the learning process. In this thesis, we propose a self-splitting Gaussian mixture learning algorithm (SGML) to address the first two issues. The algorithm starts with one single component and iteratively splits a component into two new components using Bayesian information criterion (BIC) as the validity measure for the splitting until the most appropriate component number is found, which is also achieved by BIC. Based on the splitting process, SGML provides a decent model initialization for learning Gaussian mixture models with different model complexities. For the topology-preserving issue, we consider the learning process of a probabilistic self-organizing map (PbSOM) as a model-based clustering procedure that preserves the topological relationships between data samples and clusters in a neural network. Based on this concept, we develop a coupling-likelihood mixture model for the PbSOM that extends the reference vectors in Kohonen's SOM to multivariate Gaussians distributions. We also derive three EM-type algorithms, called the SOCEM, SOEM, and SODAEM algorithms, for learning the model (PbSOM) based on the maximum likelihood criterion. SODAEM is a deterministic annealing (DA) variant of SOCEM and SOEM; moreover, by shrinking the neighborhood size, SOCEM and SOEM can be interpreted, respectively, as DA variants of the CEM and EM algorithms for Gaussian model-based clustering. The experiment results show that the proposed PbSOM learning algorithms achieve comparable data clustering performance to that of the deterministic annealing EM (DAEM) approach, while maintaining the topology-preserving property. For applications, we apply SGML to the training of speaker GMMs for the speaker identification task; the experiment results show that SGML can automatically determine an appropriate model complexity for a speaker GMM, which is usually determined empirically in the literature. We apply the proposed PbSOM algorithms to data visualization and analysis; the experiment results show that they can be used to explore the topological relationships between data samples and clusters on a two-dimensional network. In addition, we propose divide-and-conquer approaches for the audio segmentation task using BIC to evaluate the dissimilarity between two audio segments; the experiment results on the broad-cast news data demonstrate that our approaches not only have a lower computation cost but also achieve higher segmentation accuracy than the leading window-growing-based segmentation approach.

ACKNOWLEDGEMENTS

I am grateful to my advisor, Prof. Hsin-Chia Fu and Dr. Hsin-Min Wang, for their intensive suggestions, patient guidance, and enthusiasm of research. Moreover, I would like to thank all the members of the Neural Network Multimedia Laboratory at Department of Computer Science, National Chiao Tung University, and the Spoken Language Group, Chinese Information Processing Laboratory at Institute of Information Science, Academia Sinica, for their valuable discussions. Finally, I would like to express my appreciation to my family for their supporting and encouragement. I dedicate this dissertation to my parents.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The goal of data clustering is to group data samples into clusters such that samples from the same cluster are more similar to each other than samples from different clusters. It is also known as unsupervised learning, numerical taxonomy, typological analysis and vector quantization [1, 2]. Among the various clustering approaches, probabilistic model-based clustering is the one derived from statistical learning; and it has been successfully applied to many tasks, for example, speaker recognition [3, 4], speech recognition [5], handwritten recognition [6], image segmentation [7], and clustering of microarray expression data [8, 9].

In model-based clustering, data samples are grouped by learning a finite mixture model (usually a Gaussian mixture model, i.e., GMM), in which each mixture component represents a cluster. There are two major learning methods for model-based clustering: the *mixture likelihood approach*, where the likelihood of each data sample is a mixture of all the component likelihoods of the data sample; and the *classification likelihood approach*, where the likelihood of each data sample is generated by its winning component only [10, 11, 12, 13, 14, 15, 16, 17, 18]. In both approaches, when the globally optimal estimation of the model parameters cannot be obtained analytically, iterative learning algorithms that only guarantee obtaining locally optimal solutions are usually employed. The expectation-maximization (EM) algorithm for mixture likelihood learning [19, 20, 21, 22] and the classification EM (CEM) algorithm for classification likelihood learning [17] are two such algorithms and have been the dominant approaches in this task. The conventional EM or CEM-based model-based clustering has three critical aspects, namely the initialization of model parameters, the model complexity[1], and the topology-preserving ability. These aspects are discussed as follows.

- **For model initialization:** The learning performance of EM and CEM are very sensitive to the initial conditions of the model's parameters. To address this issue, the

[1] In this thesis, it denotes the number of mixture components for a mixture model.

authors in [17] proposed a simulated annealing implementation for CEM, which reduces the initial-position dependence based on random perturbations. Rather than applying the randomization power of simulated annealing, the authors in [23] proposed a *deterministic annealing* EM (DAEM) algorithm that tackles the initialization issue via a deterministic annealing process. DAEM was originally derived as a DA variant of EM; however, as shown in Section 2.4.3, it is also a DA variant of CEM. Although DAEM has been reported achieving decent performance, there is still no guarantee that it finds the globally optimal model parameters because, like EM, it is a iterative, single-token search scheme. In order to search the parameter space in multiple pathes, the authors in [24, 25] and [26, 27] proposed multi-thread search strategies when employing EM and DAEM, respectively, where the search pathes are adapted in the search process according to the eigen-decomposition of the Hessian matrix of the target log-likelihood function. As another kind of multiple-path search, a GA-EM algorithm was proposed in [28], where EM learning is integrated into a Genetic search procedure. It is less sensitive to the initialization because of the stochastic search nature of the Genetic Algorithms (GA). Some heuristic-like learning algorithms have also been proposed. For example, an initialization approach for EM which is based on subsampling was presented in [29]. The authors in [30] proposed an SMEM algorithm that finds the appropriate initial conditions for EM learning by using split and merge operations. Similarly, Young and Woodland [31] proposed a component-splitting approach to learn a GMM, which iteratively splits the mean vector of the Gaussian component with the largest weight into two new ones, and then performs EM to update all the Gaussian components. Another method based on component splitting is presented in [32]. In [33], the authors suggested a simple way that one can perform several short runs of EM (by early stopping) first, then select the best model from the results and use them as the initial model for the long run (standard) EM learning. As a common and simple way, one can apply K-means clustering or hierarchical agglomerative clustering (HAC) to locate the initial mean vectors of Gaussian components for the EM learning [1, 34, 35].

Note that all the approaches mentioned above are performed with a given target number of mixture components.

- **For model complexity:** Assessment of the number of mixture components (data clusters) is an important issue in model-based clustering. The mixture model would over-fit the data if it contains too many mixture components; in contrast, it would not be flexible enough to describe the structure of the data if the number of components is too small. Various approaches have been proposed to address this issue. In [36], Furman and Lindsay developed two hypothesis test procedures based on the moment estimators to assess the number of components for a GMM. As another hypothesis test-based approach, McLachlan and Khan estimated the number of components by likelihood ratio test, where the re-sampling process is applied to assess its null hypothesis [37]. In [38], the authors estimated the mixture complexity by comparing an information theoretic-derived nonparametric estimator with the best parametric fit of a given complexity. As another information theoretic-based approach, a maximum entropy-based approach with a modified EM algorithm was proposed to assess the model complexity of GMM in [39]. Moreover, model selection criteria, also known as penalized-likelihood criteria, have been proposed to assess the model complexity; for example, Akaike's Information Criterion (AIC) [40], Bayesian Information Criterion (BIC) [41, 10, 42], Integrated Completed Likelihood (ICL) [43], Approximate Weight Evidence (AWE) [18], Minimum Description Length (MDL) [44] (which is formally identical to BIC), and Minimum Message Length (MML) [45]. BIC is derived on the basis of mixture likelihood, ICL and AWE are derived on the basis of classification likelihood, and AIC, MDL and MML are information-theoretic-derived criteria. A common way to applying these criteria to assess model complexity is that defining the upper bound, G_{max}, of the component number of candidate models first, and then choosing the one with the best score calculated using the employed criterion as the best model. For example, when using BIC, the best component number is

$$\hat{G} = \arg\max_{G}\{2\log p(\mathcal{X};\hat{\mathbf{\Theta}}_G) - Penalty(G) | G = 1, 2, \ldots, G_{max}\}, \tag{1.1}$$

where $\hat{\mathbf{\Theta}}_G$ is the maximum likelihood estimate of parameters of the mixture model with G components, $Penalty(G)$ is a monotonically increasing function of G that penalizes more for a more complex model [10]. However, there are two potential drawbacks with this model-selection-based approach. First, it needs to define the upper bound of the component number, G_{max}, beforehand. On the one hand, if G_{max} is too large (much larger than the best component number determined by the model selection criterion), the learning process will waste a lot of computation time. On the other hand, if G_{max} is too small, the selected model may be not flexible enough to describe the structure of the data. Second, $\hat{\mathbf{\Theta}}_G$ is usually obtained by EM, whose performance is highly dependent on the model initialization.

Rather than assessing the model complexity by incrementally adding mixture components during the learning process, as discussed above, the variational Bayesian framework in [46, 47] automatically determines the number of components by setting a larger component number initially and then suppressing unwanted components.

- **For topology-preserving ability**: Conventional model-based clustering cannot preserve the topological relationships among data samples and clusters after the clustering procedure. To overcome this shortcoming, the clustering task can be performed by using Kohonen's self-organizing map (SOM) [48, 49]. The SOM, rather than being a supervised neural network model for pattern recognition, is an unsupervised model for data clustering and visualization. After SOM's clustering procedure, the topological relationships among data samples and clusters can be preserved (or visualized) on the network, which is usually a two dimensional lattice. Kohonen's sequential and batch SOM learning algorithms have proved successful in many practical applications [48, 49]. However, they also suffer from some shortcomings, such as the lack of an objective (cost) function, a general proof of convergence, and a probabilistic framework [50]. Some related works that have addressed these issues are as follows. In [51, 52], the behavior of Kohonen's sequential learning algorithm was studied in terms of energy functions, based on which, Cheng [53] proposed an energy function for SOM whose parameters can be learned by a K-means type algorithm. Luttrell [54, 55] proposed a noisy vector quantization model called the topographic vector quantizer (TVQ), whose training process coincides with the learning process of SOM. The cost function of TVQ represents the topographic distortion between the input data and the output code vectors in terms of Euclidean distance. Graepel *et al.* [56, 57] derived a soft topographic vector quantization (STVQ) algorithm by applying a deterministic annealing process to the optimization of TVQ's cost function. Based on the topographic distortion concept, Heskes [58] applied a different DA implementation from that of STVQ, and obtained an algorithm identical to STVQ when the quantization error is expressed in terms of Euclidean distance. In [59], Chow and Wu proposed an on-line algorithm for STVQ; later, motivated by STVQ, they proposed a data visualization method that integrates SOM and multi-dimensional scaling [60]. Based on the Bayesian analysis of SOMs in [61], Anouar *et al.* [62] proposed a probabilistic formalism for SOM, where the parameters are learned by a K-means type algorithm. To help users select the correct model complexity for SOM by probabilistic assessment, Lampinen and Kostiainen [63] developed a generative model in which the SOM is trained by Kohonen's algorithm. Meanwhile, Van Hulle [64] developed a kernel-based topographic formation in which the parameters are adjusted to maximize the joint entropy of the kernel outputs. He subsequently developed a new algorithm with heteroscedastic Gaussian mixtures that allows for a unified account of vector quantization, log-likelihood, and Kullback-Leibler divergence [65]. Another probabilistic formulation is proposed in [66], whereby a normalized neighborhood function of SOM is used as the posterior distribution in the *E-step* of the EM algorithm for a mixture model to enforce the self-organizing of the mixture components. Sum *et al.* [67] interpreted Kohonen's sequential learning algorithm in terms of maximizing the local correlations (coupling energies) between neurons and their neighborhoods for the given input data.

They then proposed an energy function for SOM that reveals the correlations, and a gradient ascent learning algorithm for the energy function.

1.2 Contributions of this dissertation

1.2.1 On model initialization and complexity

This thesis proposes a BIC-based self-splitting Gaussian mixture learning (SGML) algorithm that starts with a single component and iteratively splits a component into two new components until the most appropriate component number is found. The proposed algorithm has several advantages as follows. 1) SGML provides a better initialization for EM. For the Gaussian mixture learning process whose initialization is based on a data clustering process like K-mens or HAC, the clustering phase may results in an poor initial mixture model with too many components in one part of the space and too few in another widely-separated part of the space; in this case, the following EM phase may fail to escape from this configuration and fall into an poor local maximum. In the splitting process of SGML, however, BIC is used to determine which part of the space should be divided into two parts. In this way, the ill initialization situation can be avoided to some extent and, thus, a better estimation for model parameters can be obtained. 2) SGML automatically determines the appropriate component number without the need to define the upper bound of the component number beforehand. It stops when a "significant maximum" in the learning curve (i.e., the BIC plot) is found, and then outputs the model yielding the maximum BIC valve. 3) SGML is deterministic; its output is always the same in different runs on the same data set since it does not contain any randomization procedure in its learning rules.

1.2.2 On topology-preserving ability

In Kohonen's SOM architecture, neurons in the network associate with reference vectors in the data space. This contrasts with a SOM whose neurons associate with reference models that present probability distributions, such as the isotropic Gaussians used in [66] and the heteroscedastic Gaussians used in [62, 65]. In this thesis, the latter is called a probabilistic SOM (PbSOM). Motivated by the coupling energy concept in Sum *et al.*'s work [67], a coupling-likelihood mixture model for the PbSOM that uses multivariate Gaussian distributions as the reference models is developed. In the proposed model, local coupling energies between neurons and their neighborhoods are expressed in terms of probabilistic likelihoods; and each mixture component expresses the local coupling-likelihood between one neuron and its neighborhood. Based on this model, we develop CEM, EM, and DAEM algorithms for learning PbSOMs, namely the SOCEM, SOEM, and SODAEM algorithms, respectively. SODAEM is a DA variant of SOCEM and SOEM.

Moreover, we show that SOCEM and SOEM can be interpreted, respectively, as DA variants of the CEM and EM algorithms for Gaussian model-based clustering, where the neighborhood shrinking is interpreted as an annealing process. The experiment results show that the proposed PbSOM learning algorithms achieve comparable data clustering performance to the DAEM algorithm, while maintaining the topology-preserving property.

1.2.3 On applications

In this thesis, we apply SGML to the training of speaker GMMs for the speaker identification task. The experiment results on NIST 2001 speaker recognition evaluation [68] show that SGML can automatically determine the appropriate model complexity for a speaker GMM, which is usually empirically determined in the literature [4, 69]. The proposed PbSOM algorithms are applied to data visualization and analysis. The experiment results on UCI data sets show that we can explore the topological relationships between data samples and clusters on a two-dimensional network. Moreover, we propose divide-and-conquer approaches for the audio segmentation task using BIC to evaluate the dissimilarity between two audio segments. The experiment results on the broadcast news data demonstrate that our approaches not only have a lower computation cost but also achieve a higher segmentation accuracy than the leading window-growing-based segmentation approach [70, 71].

1.3 Organization of this dissertation

To help readers understand the content of this thesis, we briefly review K-means clustering, hierarchical agglomerative clustering, Kohonen's SOM, and model-based clustering in Chapter 2. In Chapter 3, we describe the SGML algorithm for learning a Gaussian mixture model and its application to speaker identification (based on [72]). In Chapter 4, we describe the SOCEM, SOEM, and SODAEM algorithms for model-based clustering where the topological relationships between data samples and clusters can be preserved on a network and their application to data visualization and analysis (based on [73]). We present the application of BIC to audio segmentation (based on [74]) in Chapter 5. Finally, we give the conclusion and discuss our future works in Chapter 6.

Chapter 2

Preliminaries and related works

2.1 K-means clustering

Given a data set $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\} \subset \Re^d$, data clustering can be achieved based on learning the vector prototypes $\{\mathbf{m}_1, \mathbf{m}_2, \cdots, \mathbf{m}_G\} \subset \Re^d$; after the learning process, $\mathcal{X}$ is partitioned into clusters $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_G\}$ by classifying each data sample to the cluster associated with its nearest prototype. For the learning, the goal is to find the partition and the prototypes that minimize the distance function

$$D(\mathcal{P}, \{\mathbf{m}_1, \mathbf{m}_2, \cdots, \mathbf{m}_G\}; \mathcal{X}) = \sum_{k=1}^{G} \sum_{\mathbf{x}_i \in \mathcal{P}_k} \|\mathbf{x}_i - \mathbf{m}_k\|^2, \tag{2.1}$$

which can not be minimized analytically.

K-means clustering is a popular approach to learning the prototypes in order to minimize Eq. (2.1). Given the initial prototypes, it iteratively and alternatively applies a classification step and a distance minimization step on the data samples and prototypes as follows.

- *Nearest-neighbor classification*: Given the current prototypes, $\{\mathbf{m}_1^{(t)}, \mathbf{m}_2^{(t)}, \cdots, \mathbf{m}_G^{(t)}\}$, assign each data sample to the cluster associated with its nearest prototype, i.e., $\mathbf{x}_i \in \hat{\mathcal{P}}_j^{(t)}$ if $j = \arg\min_k \|\mathbf{x}_i - \mathbf{m}_k^{(t)}\|^2$.

- *Distance minimization*: Suppose we have the clusters $\{\hat{\mathcal{P}}_1^{(t)}, \hat{\mathcal{P}}_2^{(t)}, \cdots, \hat{\mathcal{P}}_G^{(t)}\}$ after the classification step; then, by minimizing $\sum_{\mathbf{x}_i \in \hat{\mathcal{P}}_k^{(t)}} \|\mathbf{x}_i - \mathbf{m}_k\|^2$ with respect to $\mathbf{m}_k$, we obtain the update rules for the prototypes as: $\mathbf{m}_k^{(t+1)} = \frac{1}{|\hat{\mathcal{P}}_k^{(t)}|} \sum_{\mathbf{x}_i \in \hat{\mathcal{P}}_k^{(t)}} \mathbf{x}_i$, for $k = 1, 2, \cdots, G$.

Although the K-means clustering algorithm is simple and efficient, it only guarantees converging to a local minimum of the distance function in Eq. (2.1). Besides, it has two more shortcomings that the performance highly depends on the initialization of

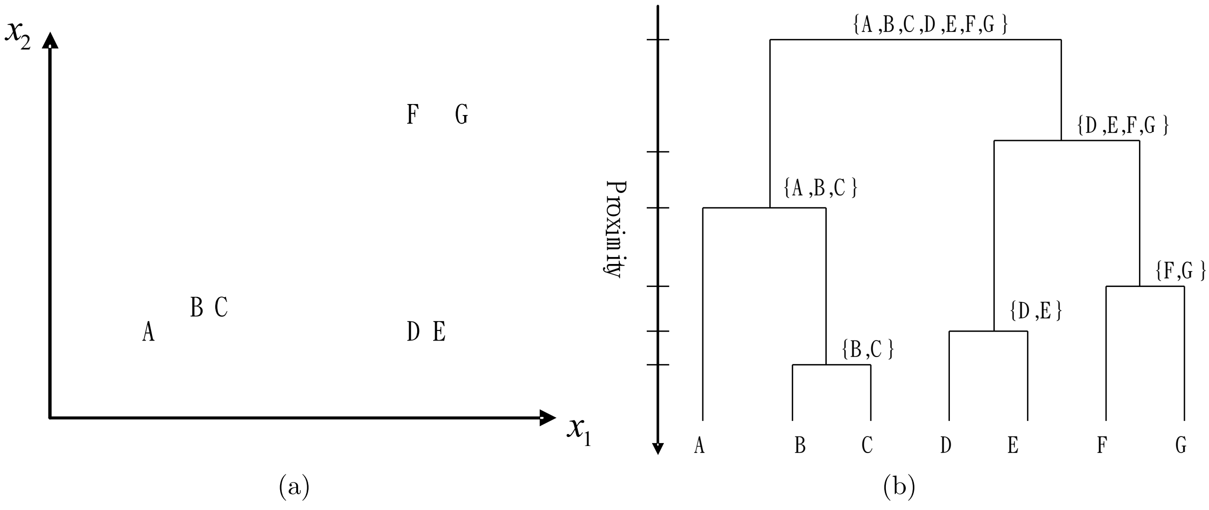


Figure 2.1: (a) {A, B, $\cdots$, G} are data samples in $\Re^2$. (b) An illustrative dendrogram of the data samples in (a) obtained by hierarchical agglomerative clustering.

prototypes and the number of prototypes need to be defined beforehand. To overcome the initialization issue, the authors in [75] proposed to iteratively split each mean vector into two new ones until the desired number of clusters is reached. Similarly, in [76], the authors proposed iteratively splitting the mean vector of the cluster with the largest accumulated distance between data samples and its prototype until the desired number of clusters is reached. As a simple way, one may conduct the K-means algorithm with random initialization for many runs, say 20, and then select the best one.

2.2 Hierarchical agglomerative clustering

Different from K-means clustering that performs data clustering based on prototype learning, hierarchical agglomerative clustering (HAC) performs the clustering according to the proximity between clusters. When performing HAC, each data sample is considered as a cluster initially; then, the algorithm iteratively merges the two clusters with the smallest distance into a new cluster. For example, if HAC is applied to clustering {A, B,$\cdots$,G} in Figure 2.1 (a) with Euclidean distance, we may obtain a dendrogram (tree) that represents the hierarchy of the clusters in Figure 2.1 (b). As shown in the figure, B and C are merged into a cluster first, then D and E are merged, and so on. And we can obtain different configurations of clusters by cutting the dendrogram according to the proximity. As for the inter-cluster distance measure, the most commonly used ones are the so-called single-linkage, average-linkage, and complete-linkage approaches defined in Table 2.1. For more discussions of these distance measure, reader can refer to [77].

Table 2.1: Common inter-cluster distance measures for hierarchical agglomerative clustering. $d(\mathbf{x}_i, \mathbf{x}_j)$ denotes the distance measure for $\mathbf{x}_i$ and $\mathbf{x}_j$.

Approach	distance between clusters $\mathcal{P}_m$ and $\mathcal{P}_n$
Single-linkage	$\min_{\mathbf{x}_i \in \mathcal{P}_m, \mathbf{x}_j \in \mathcal{P}_n} d(\mathbf{x}_i, \mathbf{x}_j)$
Average-linkage	$\frac{1}{\lvert\mathcal{P}_m\rvert\lvert\mathcal{P}_n\rvert} \sum_{\mathbf{x}_i \in \mathcal{P}_m} \sum_{\mathbf{x}_j \in \mathcal{P}_n} d(\mathbf{x}_i, \mathbf{x}_j)$
Complete-linkage	$\max_{\mathbf{x}_i \in \mathcal{P}_m, \mathbf{x}_j \in \mathcal{P}_n} d(\mathbf{x}_i, \mathbf{x}_j)$

2.3 Self-organizing map

Self-organizing map (SOM) is a neural network model for data clustering that preserves the topological relationships between data samples and clusters in a network [48, 49]. Like K-means clustering, the training of SOM is a prototype-learning process. However, in addition to the interconnections between the data samples and the prototypes as in K-means clustering, SOM involves lateral interactions between prototypes via a neighborhood definition on the network to learn the topological relationships between the prototypes. Kohonen's sequential (incremental) and batch learning algorithms have been well recognized and successfully applied to many tasks, such as data visualization [48], document processing [78, 79], image processing [80], and speech precessing [81, 82, 83].

Kohonen's SOM model consists of G neurons $\mathcal{R}=\{r_1, r_2, \cdots, r_G\}$ on a network with a neighborhood function h_{kl} that defines the strength of lateral interaction between two neurons, r_k and r_l, for $k, l \in \{1, 2, \cdots, G\}$; and each neuron r_k associates with a vector prototype $\mathbf{m}_k$ (a reference vector) in the input data space [48, 49]. In Kohonen's sequential learning algorithm, the training data samples $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N$ are applied sequentially. For each data sample, say $\mathbf{x}_i$, its winning prototype $\mathbf{m}_{win_i^{(t)}}$ such that

$$win_i^{(t)} = \arg\min_j \|\mathbf{x}_i - \mathbf{m}_j^{(t)}\|, \tag{2.2}$$

is retrieved first, then the prototypes are adapted by the following rules:

$$\mathbf{m}_k^{(t+1)} = \mathbf{m}_k^{(t)} + \alpha(t) h_{win_i^{(t)} k} (\mathbf{x}_i - \mathbf{m}_k^{(t)}), \tag{2.3}$$

for $k = 1, 2, \cdots, G$. $\alpha(t)$ is the learning-rate factor such that $0 < \alpha(t) < 1$; it decreases monotonically with the increasing of the learning iterations. The neighborhood function h_{kl} is usually a rectangular function, i.e., $h_{kl} = 1$ for r_l that is in the neighborhood of r_k and $h_{kl} = 0$ for r_l that is not in the neighborhood of r_k; or a monotonically decreasing function of the Euclidean distance $\|r_k - r_l\|$ between r_k and r_l on the SOM network, for example, the widely applied unnormalized Gaussian neighborhood function

$$h_{kl} = \exp(-\frac{\|r_k - r_l\|^2}{2\sigma^2}). \tag{2.4}$$

Kohonen's batch learning algorithm applies all the data samples as a whole to update the prototypes, rather than one by one as in the sequential learning algorithm. In each learning iteration, the winning prototype for each sample is found first; then, the prototypes are updated by

$$\mathbf{m}_k^{(t+1)} = \frac{\sum_{i=1}^{N} h_{win_i^{(t)}k}\mathbf{x}_i}{\sum_{i=1}^{N} h_{win_i^{(t)}k}}, \tag{2.5}$$

for $k = 1, 2, \cdots, G$.

Kohonen's learning algorithms need to be applied with two stages. First, a large neighborhood is applied to the algorithm to learn an ordered map near the center of the data samples. Then, the prototypes are adapted to fit the distribution of the data samples by gradually shrinking the neighborhood. When the neighborhood is reduce to zero-neighborhood, i.e., $h_{kl} = \delta_{kl}$ where $\delta_{kl} = 1$ if $k = l$ and otherwise $\delta_{kl} = 0$, Kohonen's batch learning algorithm becomes the K-means clustering algorithm.

2.4 Probabilistic model-based clustering

2.4.1 The mixture likelihood approach

In the mixture likelihood approach for model-based clustering, it is assumed that the given data set $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\} \subset \Re^d$ is generated by a set of independently and identically distributed (i.i.d.) random vectors from a mixture model:

$$p(\mathbf{x}_i; \mathbf{\Theta}) = \sum_{k=1}^{G} w(k) p(\mathbf{x}_i; \boldsymbol{\theta}_k), \tag{2.6}$$

where $w(k)$ is the mixing weight of the mixture component $p(\mathbf{x}_i; \boldsymbol{\theta}_k)$, subject to $0 \leq w(k) \leq 1$ for $k = 1, 2, \cdots, G$; $\sum_{k=1}^{G} w(k) = 1$; and $\boldsymbol{\theta}_k$ denotes the parameter set of $p(\mathbf{x}_i; \boldsymbol{\theta}_k)$.

The maximum likelihood estimate of the parameter set of the mixture model $\hat{\mathbf{\Theta}} = \{\hat{w}(1), \hat{w}(2)\ , \cdots, \hat{w}(G), \hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_2, \cdots, \hat{\boldsymbol{\theta}}_G\}$ can be obtained by maximizing the following log-likelihood function:

$$\begin{aligned} L(\mathbf{\Theta}; \mathcal{X}) &= \log \prod_{i=1}^{N} p(\mathbf{x}_i; \mathbf{\Theta}) \\ &= \sum_{i=1}^{N} \log(\sum_{k=1}^{G} w(k) p(\mathbf{x}_i; \boldsymbol{\theta}_k)). \end{aligned} \tag{2.7}$$

This is usually achieved by using the expectation-maximization (EM) algorithm [20, 22]. After learning the mixture model, we derive a partition of $\mathcal{X}$, $\hat{\mathcal{P}} = \{\hat{\mathcal{P}}_1, \hat{\mathcal{P}}_2, \cdots, \hat{\mathcal{P}}_G\}$, by assigning each $\mathbf{x}_i \in \mathcal{X}$ to the mixture component that has the largest posterior probability for $\mathbf{x}_i$, i.e., $\mathbf{x}_i \in \hat{\mathcal{P}}_j$ if $j = \arg\max_k p(k \mid \mathbf{x}_i; \hat{\boldsymbol{\Theta}})$.

2.4.1.1 The EM algorithm for mixture models

If the maximum likelihood estimation of the parameters cannot be accomplished analytically, the EM algorithm is normally used as an alternative approach when the given data is incomplete or contains hidden information.

In the case of the mixture model, suppose that $\boldsymbol{\Theta}^{(t)}$ denotes the current estimate of the parameter set, and k is the hidden variable that indicates the mixture component from which the data sample is generated. The *E-step* of EM algorithm then computes the following so-called *auxiliary function*:

$$Q(\boldsymbol{\Theta}; \boldsymbol{\Theta}^{(t)}) = \sum_{i=1}^{N} \sum_{k=1}^{G} p(k \mid \mathbf{x}_i; \boldsymbol{\Theta}^{(t)}) \log p(\mathbf{x}_i, k; \boldsymbol{\Theta}), \tag{2.8}$$

where

$$p(\mathbf{x}_i, k; \boldsymbol{\Theta}) = w(k) p(\mathbf{x}_i; \boldsymbol{\theta}_k), \tag{2.9}$$

and

$$p(k \mid \mathbf{x}_i; \boldsymbol{\Theta}^{(t)}) = \frac{w(k)^{(t)} p(\mathbf{x}_i; \boldsymbol{\theta}_k^{(t)})}{\sum_{j=1}^{G} w(j)^{(t)} p(\mathbf{x}_i; \boldsymbol{\theta}_j^{(t)})} \tag{2.10}$$

denotes the posterior probability of the kth mixture component for $\mathbf{x}_i$ with the given $\boldsymbol{\Theta}^{(t)}$. Then, in the following *M-step*, the $\boldsymbol{\Theta}^{(t+1)}$ that satisfies

$$Q(\boldsymbol{\Theta}^{(t+1)}; \boldsymbol{\Theta}^{(t)}) = \max_{\boldsymbol{\Theta}} Q(\boldsymbol{\Theta}; \boldsymbol{\Theta}^{(t)}) \tag{2.11}$$

is chosen as the new estimate of the parameter set. By iteratively creating the auxiliary function in Eq. (2.8) and performing the subsequent maximization step, the EM algorithm guarantees to converge to a local maximum of the log-likelihood function in Eq. (2.7).

When $Q(\boldsymbol{\Theta}; \boldsymbol{\Theta}^{(t)})$ can not be maximized analytically, the *M-step* is modified to find some $\boldsymbol{\Theta}^{(t+1)}$ such that $Q(\boldsymbol{\Theta}^{(t+1)}; \boldsymbol{\Theta}^{(t)}) > Q(\boldsymbol{\Theta}^{(t)}; \boldsymbol{\Theta}^{(t)})$. This type of algorithm, called Generalized EM (GEM), is also guaranteed to converge to a local maximum [20, 22].

2.4.1.2 The EM algorithm for Gaussian mixture models

Suppose Eq. (2.6) is a Gaussian mixture model where

$$p(\mathbf{x}_i; \boldsymbol{\theta}_k) = \frac{1}{(2\pi)^{d/2} |\boldsymbol{\Sigma}_k|^{1/2}} \exp(-\frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_k)), \tag{2.12}$$

is the multivariate Gaussian distribution and $\boldsymbol{\theta}_k = \{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}$ are its mean vector and covariance matrix. Then, each iteration of the EM algorithm for learning GMMs is summarized as follows.

- *E-step*: Given the current model, $\mathbf{\Theta}^{(t)}$, compute the posterior probability of each mixture component of $p(\mathbf{x}_i; \mathbf{\Theta}^{(t)})$ for each $\mathbf{x}_i$ using Eq. (2.10).

- *M-step*: Substituting Eq. (2.12) into Eq. (2.8) and then setting its derivative to zero, we obtain the re-estimation formulae of the parameters as follows.

$$
\begin{aligned}
w(k)^{(t+1)} &= \frac{1}{N}\sum_{i=1}^{N} p(k \mid \mathbf{x}_i; \mathbf{\Theta}^{(t)}), && (2.13)\\
\boldsymbol{\mu}_k^{(t+1)} &= \frac{\sum_{i=1}^{N} p(k \mid \mathbf{x}_i; \mathbf{\Theta}^{(t)})\mathbf{x}_i}{\sum_{i=1}^{N} p(k \mid \mathbf{x}_i; \mathbf{\Theta}^{(t)})}, && (2.14)\\
\mathbf{\Sigma}_k^{(t+1)} &= \frac{\sum_{i=1}^{N} p(k \mid \mathbf{x}_i; \mathbf{\Theta}^{(t)})(\mathbf{x}_i - \boldsymbol{\mu}_k^{(t+1)})(\mathbf{x}_i - \boldsymbol{\mu}_k^{(t+1)})^T}{\sum_{i=1}^{N} p(k \mid \mathbf{x}_i; \mathbf{\Theta}^{(t)})}. && (2.15)
\end{aligned}
$$

2.4.1.3 The Bayesian approach for model selection and BIC

Given the data set $\mathcal{X}$ and a set of candidate probability models $\mathcal{M} = \{M_G \mid G = 1, \ldots, G_{max}\}$ where model M_G associates with a parameter set $\mathbf{\Theta}_G$, the posterior probability $p(M_G \mid \mathcal{X})$ can be used to select the appropriate model from $\mathcal{M}$ to represent the distribution of $\mathcal{X}$. According to Bayes' theorem, $p(M_G \mid \mathcal{X})$ can be expressed as

$$
p(M_G \mid \mathcal{X}) = \frac{p(M_G)p(\mathcal{X}; M_G)}{p(\mathcal{X})}, \qquad (2.16)
$$

where $p(M_G)$ is the prior probability of model M_G. $p(\mathcal{X})$ can be ignored because it is identical for all models and does not affect the selection. Moreover, it is assumed that each model is equally likely (i.e., $p(M_G) = 1/G_{max}$); then, $p(M_G \mid \mathcal{X})$ is proportional to the probability that the data conforms to the model M_G, $p(\mathcal{X}; M_G)$, which can be computed by

$$
\begin{aligned}
p(\mathcal{X}; M_G) &= \int p(\mathcal{X}; \mathbf{\Theta}_G, M_G)p(\mathbf{\Theta}_G; M_G)d\mathbf{\Theta}_G,\\
&= \int p(\mathcal{X}; \mathbf{\Theta}_G)p(\mathbf{\Theta}_G)d\mathbf{\Theta}_G. && (2.17)
\end{aligned}
$$

The calculation of $\log p(\mathcal{X}; M_G)$ can be achieved by the Laplace approximation [42, 47, 84], which gives

$$
\log p(\mathcal{X}; M_G) \approx \log p(\mathcal{X}; \hat{\mathbf{\Theta}}_G) - \frac{1}{2}d(\mathbf{\Theta}_G)\log N, \qquad (2.18)
$$

where $\hat{\mathbf{\Theta}}_G$ is the maximum likelihood estimate of $\mathbf{\Theta}_G$, $d(\mathbf{\Theta}_G)$ is the number of free parameters in model M_G, and N is the number of data samples. In [10], the BIC value of model M_G over the data set $\mathcal{X}$, $BIC(M_G, \mathcal{X})$, is defined as[1]

$$BIC(M_G, \mathcal{X}) \equiv 2 \log p(\mathcal{X}; \hat{\boldsymbol{\Theta}}_G) - d(\boldsymbol{\Theta}_G) \log N. \tag{2.19}$$

Accordingly, the larger the BIC value, the stronger the evidence for the model is. In other words, the model with the maximum BIC value will be selected. The BIC can be used to compare models with different parameterizations, different numbers of components, or both.

2.4.2 The classification likelihood approach

In the classification likelihood approach for model-based clustering [15, 16, 17], instead of maximizing the log-likelihood function of the mixture model in Eq. (2.7), the objective is to find the partition $\hat{\mathcal{P}} = \{\hat{\mathcal{P}}_1, \hat{\mathcal{P}}_2, \cdots, \hat{\mathcal{P}}_G\}$ of $\mathcal{X}$ and the model parameters that maximize

$$C_1(\mathcal{P}, \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \cdots, \boldsymbol{\theta}_G\}; \mathcal{X}) = \sum_{k=1}^{G} \sum_{\mathbf{x}_i \in \mathcal{P}_k} \log p(\mathbf{x}_i; \boldsymbol{\theta}_k), \tag{2.20}$$

or

$$C_2(\mathcal{P}, \boldsymbol{\Theta}; \mathcal{X}) = \sum_{k=1}^{G} \sum_{\mathbf{x}_i \in \mathcal{P}_k} \log(w(k) p(\mathbf{x}_i; \boldsymbol{\theta}_k)). \tag{2.21}$$

The relation between C_1 and C_2 is

$$C_2(\mathcal{P}, \boldsymbol{\Theta}; \mathcal{X}) = C_1(\mathcal{P}, \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \cdots, \boldsymbol{\theta}_G\}; \mathcal{X}) + \sum_{k=1}^{G} |\mathcal{P}_k| \log w(k), \tag{2.22}$$

If all the mixture components are equally weighted, $\sum_{k=1}^{G} |\mathcal{P}_k| \log w(k)$ becomes a constant, such that C_1 and C_2 are equivalent.

2.4.2.1 The CEM algorithm for mixture models

Celeux and Govaert [17] proposed the Classification EM (CEM) algorithm for estimating the partition $\hat{\mathcal{P}}$ and model parameters. Like the EM algorithm, CEM is also an iterative learning approach. In each iteration, it inserts a *classification* step (*C-step*) between the *E-step* and *M-step* of the EM algorithm. In the *E-step*, the posterior probability of each mixture component is calculated for each data sample. In the *C-step*, to obtain the partition $\hat{\mathcal{P}}$ of the data samples, each sample is assigned to the mixture component that yields the largest posterior probability for that sample. In the *M-step*, the maximization process is applied to $\hat{\mathcal{P}}_k$ individually for $k = 1, 2, \cdots, G$. From a practical point of view, CEM

[1] Kass and Raftery [85] defined BIC as minus the value given in Eq. (2.19); Fraley and Raftery [10] used the BIC defined in Eq. (2.19) to make it easier to interpret the plot of BIC values. Here, we follow Fraley and Raftery's BIC definition.

is a K-means type algorithm where a cluster is associated with a probability distribution rather than a vector prototype [17].

2.4.2.2 The CEM algorithm for Gaussian mixture models

Suppose the multivariate Gaussian is used as the mixture component, each iteration of the CEM algorithm for learning GMMs is summarized as follows.

- *E-step*: Given the current model, $\mathbf{\Theta}^{(t)}$, compute the posterior probability of each mixture component of $p(\mathbf{x}_i; \mathbf{\Theta}^{(t)})$ for each $\mathbf{x}_i$ using Eq. (2.10).
- *C-step*: Assign each $\mathbf{x}_i$ to the cluster whose corresponding mixture component has the largest posterior probability for $\mathbf{x}_i$, i.e., $\mathbf{x}_i \in \hat{\mathcal{P}}_j^{(t)}$ if $j = \arg\max_k p(k|\mathbf{x}_i; \mathbf{\Theta}^{(t)})$.
- *M-step*: After the *C-step*, the partition of $\mathcal{X}$ (i.e., $\hat{\mathcal{P}}^{(t)}$) is formed, and the objective function C_2 defined in Eq. (2.21) becomes

$$C_2(\mathbf{\Theta}; \hat{\mathcal{P}}^{(t)}, \mathcal{X}) = \sum_{k=1}^{G} \sum_{\mathbf{x}_i \in \hat{\mathcal{P}}_k^{(t)}} \log(w(k) p(\mathbf{x}_i; \boldsymbol{\theta}_k)). \tag{2.23}$$

Substituting Eq. (2.12) into Eq. (2.23) and then setting the derivative of C_2 with respect to individual parameters to zero, we obtain the re-estimation formulae of the parameters as follows.

$$w(k)^{(t+1)} = \frac{|\hat{\mathcal{P}}_k^{(t)}|}{N}, \tag{2.24}$$

$$\boldsymbol{\mu}_k^{(t+1)} = \frac{\sum_{\mathbf{x}_i \in \hat{\mathcal{P}}_k^{(t)}} \mathbf{x}_i}{|\hat{\mathcal{P}}_k^{(t)}|}, \tag{2.25}$$

$$\mathbf{\Sigma}_k^{(t+1)} = \frac{\sum_{\mathbf{x}_i \in \hat{\mathcal{P}}_k^{(t)}} (\mathbf{x}_i - \boldsymbol{\mu}_k^{(t+1)})(\mathbf{x}_i - \boldsymbol{\mu}_k^{(t+1)})^T}{|\hat{\mathcal{P}}_k^{(t)}|}. \tag{2.26}$$

2.4.2.3 AWE and ICL for model selection

Based on classification likelihood, Banfield and Raftery proposed the AWE criterion [18, 86] to assess the number of Gaussian components (clusters) for a given data set. The AWE criterion is

$$AWE(G\ clusters; \mathcal{X}) = C_1(\hat{\mathcal{P}}, \{\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_2, \cdots, \hat{\boldsymbol{\theta}}_G\}; \mathcal{X}) - d(\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \cdots, \boldsymbol{\theta}_G\})(\frac{3}{2} + \log N), \tag{2.27}$$

In addition, Biernacki *et al.* proposed the ICL criterion [43],

$$ICL(G\ clusters; \mathcal{X}) = C_2(\hat{\mathcal{P}}, \hat{\mathbf{\Theta}}_G; \mathcal{X}) - \frac{1}{2} d(\mathbf{\Theta}) \log N. \tag{2.28}$$

2.4.3 The DAEM algorithm

In the DAEM algorithm for learning a mixture model [23], the objective is to minimize the following system energy function during the annealing process:

$$F_\beta(\boldsymbol{\Theta};\mathcal{X}) = -\frac{1}{\beta}\sum_{i=1}^{N}\log(\sum_{k=1}^{G}(w(k)p(\mathbf{x}_i;\boldsymbol{\theta}_k))^{\beta}), \tag{2.29}$$

where $1/\beta$ corresponds to the *temperature* that controls the annealing process. The auxiliary function in this case is

$$U_\beta(\boldsymbol{\Theta};\boldsymbol{\Theta}^{(t)}) = -\sum_{i=1}^{N}\sum_{k=1}^{G} f(k \mid \mathbf{x}_i;\boldsymbol{\Theta}^{(t)}) \log p(\mathbf{x}_i, k;\boldsymbol{\Theta}), \tag{2.30}$$

where

$$f(k \mid \mathbf{x}_i;\boldsymbol{\Theta}^{(t)}) = \frac{(w(k)^{(t)}p(\mathbf{x}_i;\boldsymbol{\theta}_k^{(t)}))^{\beta}}{\sum_{j=1}^{G}(w(j)^{(t)}p(\mathbf{x}_i;\boldsymbol{\theta}_j^{(t)}))^{\beta}} \tag{2.31}$$

is the posterior probability derived by using the maximum entropy principle.

Ueda and Nakano [23] showed that $F_\beta(\boldsymbol{\Theta};\mathcal{X})$ can be iteratively minimized by iteratively minimizing $U_\beta(\boldsymbol{\Theta};\boldsymbol{\Theta}^{(t)})$. When using DAEM to learn a mixture model, β is initialized with a small value (less then 1) such that the energy function itself is simple enough to be optimized. Then, the value of β is gradually increased to 1. During the learning process, the parameters learned in the current learning phase are used as the initial parameters of the next phase[2]. In the case of $\beta = 1$, $F_\beta(\boldsymbol{\Theta};\mathcal{X})$ and $U_\beta(\boldsymbol{\Theta};\boldsymbol{\Theta}^{(t)})$ are the negatives of the log-likelihood function in Eq. (2.7) and the Q-function in Eq. (2.8), respectively; thus, minimizing $F_\beta(\boldsymbol{\Theta};\mathcal{X})$ is equivalent to maximizing the log-likelihood function.

According to [23], Eq. (2.29) can be rewritten as

$$F_\beta(\boldsymbol{\Theta};\mathcal{X}) = U_\beta(\boldsymbol{\Theta}) - \frac{1}{\beta}S_\beta(\boldsymbol{\Theta}), \tag{2.32}$$

where

$$U_\beta(\boldsymbol{\Theta}) = -\sum_{i=1}^{N}\sum_{k=1}^{G} f(k \mid \mathbf{x}_i;\boldsymbol{\Theta}) \log p(\mathbf{x}_i, k;\boldsymbol{\Theta}), \tag{2.33}$$

and

$$S_\beta(\boldsymbol{\Theta}) = -\sum_{i=1}^{N}\sum_{k=1}^{G} f(k \mid \mathbf{x}_i;\boldsymbol{\Theta}) \log f(k \mid \mathbf{x}_i;\boldsymbol{\Theta}) \tag{2.34}$$

is the entropy of the posterior distribution. When $\beta\rightarrow\infty$, the rational function $f(k \mid \mathbf{x}_i;\boldsymbol{\Theta})$ approximates to a zero-one function; thus, the entropy term $S_\beta(\boldsymbol{\Theta})\rightarrow 0$. In this case, $F_\beta(\boldsymbol{\Theta};\mathcal{X})$ is equivalent to the negative of the objective function for CEM in Eq. (2.21).

[2] Each β value corresponds to a learning phase. The algorithm proceeds to the next phase after it converges in the current phase.

Therefore, DAEM can be viewed as a DA variant of CEM.

2.4.3.1 The DAEM algorithm for Gaussian mixture models

For the given β value, each iteration of the DAEM algorithm for learning GMMs is summarized as follows.

- *E-step*: Given the current model, $\mathbf{\Theta}^{(t)}$, compute the posterior probability of each mixture component of $p(\mathbf{x}_i; \mathbf{\Theta}^{(t)})$ for each $\mathbf{x}_i$ using Eq. (2.31).
- *M-step*: Substituting Eq. (2.12) into Eq. (2.30) and then setting its derivative to zero, we obtain the re-estimation formulae of the parameters as follows.

$$\begin{aligned} w(k)^{(t+1)} &= \frac{1}{N}\sum_{i=1}^{N} f(k \mid \mathbf{x}_i; \mathbf{\Theta}^{(t)}), && (2.35)\\ \boldsymbol{\mu}_k^{(t+1)} &= \frac{\sum_{i=1}^{N} f(k \mid \mathbf{x}_i; \mathbf{\Theta}^{(t)})\mathbf{x}_i}{\sum_{i=1}^{N} f(k \mid \mathbf{x}_i; \mathbf{\Theta}^{(t)})}, && (2.36)\\ \mathbf{\Sigma}_k^{(t+1)} &= \frac{\sum_{i=1}^{N} f(k \mid \mathbf{x}_i; \mathbf{\Theta}^{(t)})(\mathbf{x}_i - \boldsymbol{\mu}_k^{(t+1)})(\mathbf{x}_i - \boldsymbol{\mu}_k^{(t+1)})^T}{\sum_{i=1}^{N} f(k \mid \mathbf{x}_i; \mathbf{\Theta}^{(t)})}. && (2.37) \end{aligned}$$

Chapter 3

BIC-based self-splitting Gaussian mixture learning

3.1 The SGML algorithm

Given the data set $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$, we can partition it into G clusters after applying the EM algorithm to learn a G-component Gaussian mixture model (GMM_G). Here, to help explain the scenario of the proposed approach, we denote a cluster obtained by EM learning a $EM\_cluster$. The self-splitting Gaussian mixture learning algorithm (SGML) starts with a single component (i.e., a single $EM\_cluster$) in the data space (i.e., $\mathcal{X}$) and splits the selected component adaptively during the learning process until the most appropriate number of components ($EM\_clusters$) is found. We describe the details of SGML in Algorithm 1. There are three main aspects with respect to the self-splitting learning rules:

$I1$: *How to group the data samples into clusters?*

$I2$: *Which component should be split into two new components?*

$I3$: *How many components are enough for fitting the distribution of the data samples?*

On issue $I1$: After EM, each $\mathbf{x}_i \in \mathcal{X}$ is assigned to the $EM\_cluster$ whose Gaussian component has the largest posterior probability for $\mathbf{x}_i$, i.e., assign $\mathbf{x}_i$ to $EM\_cluster_j$ for $j = \arg\max_k p(k \mid \mathbf{x}_i; \hat{\boldsymbol{\Theta}})$.

On issue $I2$: For each $EM\_cluster$, we calculate the value of $\Delta BIC_{21}(EM\_cluster)$:

$$\Delta BIC_{21}(EM\_cluster) = BIC(GMM_2, EM\_cluster) - BIC(GMM_1, EM\_cluster), \tag{3.1}$$

where GMM_1 fits the $EM\_cluster$ with a single Gaussian component while GMM_2 fits the $EM\_cluster$ with a mixture of two Gaussian components. The component corresponding to the $EM\_cluster$ with the largest ΔBIC_{21} value is split into two new components because, according to the BIC theory, the larger the ΔBIC_{21} is, the higher the confidence that GMM_2 fits the corresponding $EM\_cluster$ better than GMM_1 does. The mean vector and covariance matrix of GMM_1 can be optimally estimated in an analytic way (which are respectively the sample mean vector and sample covariance matrix of $EM\_cluster$), whereas those of GMM_2 need to be estimated by the EM algorithm. For the learning of GMM_2, we use K-means clustering (here, K=2) to find two initial mean vectors of GMM_2, and then apply EM to fine-tune the parameters. Suppose μ is the mean vector of GMM_1, we use $\mu - \varepsilon$ and $\mu + \varepsilon$ as the initial centroids of the K-means clustering, where ε is a constant vector.

On issue $I3$: After the jth split operation, the number of components grows to $j+1$ and these Gaussian components are used as the initialization of EM to learn the GMM_{j+1} for $\mathcal{X}$; and the value of $BIC(GMM_{j+1}, \mathcal{X})$ is computed after the EM learning. During the learning process the BIC values of the learned GMMs with different numbers of mixture components are collected to form a learning curve (i.e., the BIC plot); and the learning process stops when a "significant maximum" in the learning curve is found. Given that SGML has generated $\{GMM_1, GMM_2, \cdots, GMM_{cNum}\}$, if $BIC(GMM_{cNum-SRange}, \mathcal{X})$ is the maximum among $\{BIC(GMM_1, \mathcal{X}), BIC(GMM_2, \mathcal{X}), \cdots, BIC(GMM_{cNum}, \mathcal{X})\}$, it is called the "significant maximum" of the learning curve and SGML stops and returns the model $GMM_{cNum-SRange}$; otherwise, SGML continues to learn GMM_{cNum+1}. Here, $SRange$ is a positive integer; it is appropriate to set $SRange$ around 5 according to our experience.

3.1.1 Computational complexity of SGML

Without loss of generality, we assume the splitting process of SGML stops at $GMM_{G_{max}}$. With the data set $\mathcal{X}$, we use $T_{KM}(k, \mathcal{X})$ to denote the computational cost of applying K-means clustering to initialize GMM_k for EM and use $T_{EM}(k, \mathcal{X})$ to denote the cost of using EM for learning GMM_k. For SGML, the computational cost of the *Splitting* step that splits GMM_M to GMM_{M+1} consists of the cost of computing the sample mean vectors and sample covariance matrices for all clusters, i.e., $\sum_{j=1}^{M} T_{EM}(1, EM\_cluster_j)$, plus the cost of applying K-means (K=2 here) followed by EM to learn GMM_2 for all clusters, i.e., $\sum_{j=1}^{M}[T_{KM}(2, EM\_cluster_j) + T_{EM}(2, EM\_cluster_j)]$. $\sum_{j=1}^{M} T_{EM}(1, EM\_cluster_j)$ is almost equal to $T_{EM}(1, \mathcal{X})$. Moreover, from Eqs. (2.13)-(2.15), it is obvious that $T_{EM}(k, \mathcal{X})$ is proportional to the component number k and the size of data set $\mathcal{X}$. In fact, $T_{KM}(k, \mathcal{X})$ also has this property. Therefore, $\sum_{j=1}^{M} T_{EM}(2, EM\_cluster_j)$ can be approximated by $T_{EM}(2, \mathcal{X})$, while $\sum_{j=1}^{M} T_{KM}(2, EM\_cluster_j)$ can be approximated by $T_{KM}(2, \mathcal{X})$. In addition, the cost of the *Global EM learning* step is $T_{EM}(M+1, \mathcal{X})$. Thus, the total cost of SGML is about

$$\sum_{k=1}^{G_{max}} [T_{EM}(1,\mathcal{X}) + T_{KM}(2,\mathcal{X}) + T_{EM}(2,\mathcal{X}) + T_{EM}(k,\mathcal{X})]. \tag{3.2}$$

For the common approach that uses K-means (with random initialization) followed by EM to learn the candidate models for the model selection criterion, the computational cost is

$$\sum_{k=1}^{G_{max}} [T_{KM}(k,\mathcal{X}) + T_{EM}(k,\mathcal{X})]. \tag{3.3}$$

Comparing Eq. (3.2) to Eq. (3.3), it is clear that SGML is more efficient when $k \gg 2$ because for which case, $T_{KM}(k,\mathcal{X})$ is larger than $T_{EM}(1,\mathcal{X})+$ $T_{KM}(2,\mathcal{X})+T_{EM}(2,\mathcal{X})$.

Comparing to the method proposed by Young and Woodland [31], SGML has an overhead of $T_{EM}(1,\mathcal{X}) + T_{KM}(2,\mathcal{X}) + T_{EM}(2,\mathcal{X})$ in each splitting step. However, $T_{EM}(1,\mathcal{X})$ is much small than $\frac{1}{2}T_{EM}(2,\mathcal{X})$ because it does not involve likelihood calculations when estimating GMM_1. Moreover, $T_{KM}(2,\mathcal{X})$ is much smaller than $T_{EM}(2,\mathcal{X})$. Thus, the overhead is just slightly higher than $T_{EM}(2,\mathcal{X})$. When $M \gg 2$, the overhead for splitting GMM_M to GMM_{M+1} is negligible since $T_{EM}(2,\mathcal{X})$ is much smaller than $T_{EM}(M+1,\mathcal{X})$.

3.2 The fastSGML algorithm

The computation cost becomes an important issue when the data set is very large; a fast learning procedure is therefore desirable. Since we can validate if GMM_2 fits a cluster better than GMM_1 using BIC, a straightforward idea to speedup SGML is that split all the clusters whose ΔBIC_{21} values are larger than a threshold in each splitting step. Here, we call the threshold the *splitting confidence*. Based on the concept of multiple splitting, we proposed a fastSGML approach in Algorithm 2. To avoid the over-fitting condition caused when using a too small splitting confidence, the fastSGML stops not only when the ΔBIC_{21} value for each cluster is smaller than the splitting confidence (in Step 3) but also when the learning curve starts to go down (in Step 4). Clearly, fastSGML needs a much lower computation requirement than SGML because it allows multiple splits in the splitting step. fasSGML resembles the LBG algorithm [75], but the difference is that the later iteratively splits each of the clusters into two clusters until a given cluster number is reached and no validation measure is applied to the splitting.

3.3 Experiments on Gaussian mixture learning

We evaluated the proposed SGML and fastSGML algorithms on one synthetic data set which consists of six Gaussian clusters in $\Re^2$ and one real-world data set which consists of speech feature vectors from speaker #5007 of the NIST 2001 cellular speaker recognition evaluation data (NIST01SpkEval) [68]. We evaluated the algorithms using the performance of speaker GMM training because we shall apply SGML to the GMM-based speaker identification task, as shown in Section 3.4. In each task, the performance of the proposed algorithms were compared to that of other EM-based learning approaches whose initial mean vectors of GMMs are located by hierarchical agglomerative clustering (HAC) or K-means clustering. The baseline approaches are as follows.

1. Hier-ComLink method: The initial Gaussian mean vectors in EM were obtained by the complete-linkage HAC [35].

2. Hier-SLink method: The initial Gaussian mean vectors in EM were obtained by the single-linkage HAC [35].

3. Hier-CenLink method: The initial Gaussian mean vectors in EM were obtained by the centroid-linkage HAC [35].

4. K-means-random method: The initial Gaussian mean vectors in EM were obtained by K-means clustering, in which the initial centroids are randomly selected from the data set.

5. K-means-BinSplitting method [75]: The initial Gaussian mean vectors in EM were determined by the LBG algorithm, in which each mean vector was split into two new ones in each splitting step until the desired number of clusters was reached.

6. K-means-IncSplitting method [76]: The initial Gaussian mean vectors in EM were obtained by the incremental splitting K-means algorithm, in which only the mean vector of the cluster with the largest accumulated distance was split into two new ones in each splitting step until the desired number of clusters was reached.

7. EMSplitByMaxWeight method [31]: This method splits the mean vector of the Gaussian component with the largest mixture weight into two new ones in each splitting step, and then performs EM to update all Gaussian components. The number of mixture component is incremented from one to the pre-defined number.

For each baseline approach, the initial covariance matrix of GMM_k was set as $\rho_k\mathbf{I}$, where ρ_k=$\min_{j\neq k}\{\|\mu_j^{(0)} - \mu_k^{(0)}\|\}$ and $\mu_j^{(0)}$ denotes the initial mean vector of the jth Gaussian component.

3.3.1 Results on the synthetic data

Figure 3.1 (a) shows the synthetic data that contains six Gaussian clusters, each with 100 samples. First, we conducted experiments using full covariance Gaussian components. Figure 3.1 schematically depicts the learning process of the SGML algorithm. For this example, SGML stops at GMM_{11} and obtains a “significant maximum” at GMM_6, as shown in Figure 3.2. The results show that SGML performs well on automatic clustering the synthetic data. When evaluating the baseline methods, the maximal number of mixture components was limited to 13. From Figure 3.2, we see that all the baseline methods except K-means-random estimate the parameters of $GMM_k (k = 1, 2, \cdots, 13)$ as well as SGML.

Then, we evaluated SGML with diagonal covariance Gaussians. In this case, as shown in Figure 3.3, SGML groups the synthetic data into ten clusters. From the perspective of “Gaussian mixture modeling” rather than “data clustering”, we divide the learning process into three phases:

1. The cluster-capturing phase ($GMM_1 - GMM_6$): In this phase, SGML roughly captures the locations of all the clusters of the data by the self-splitting rules.

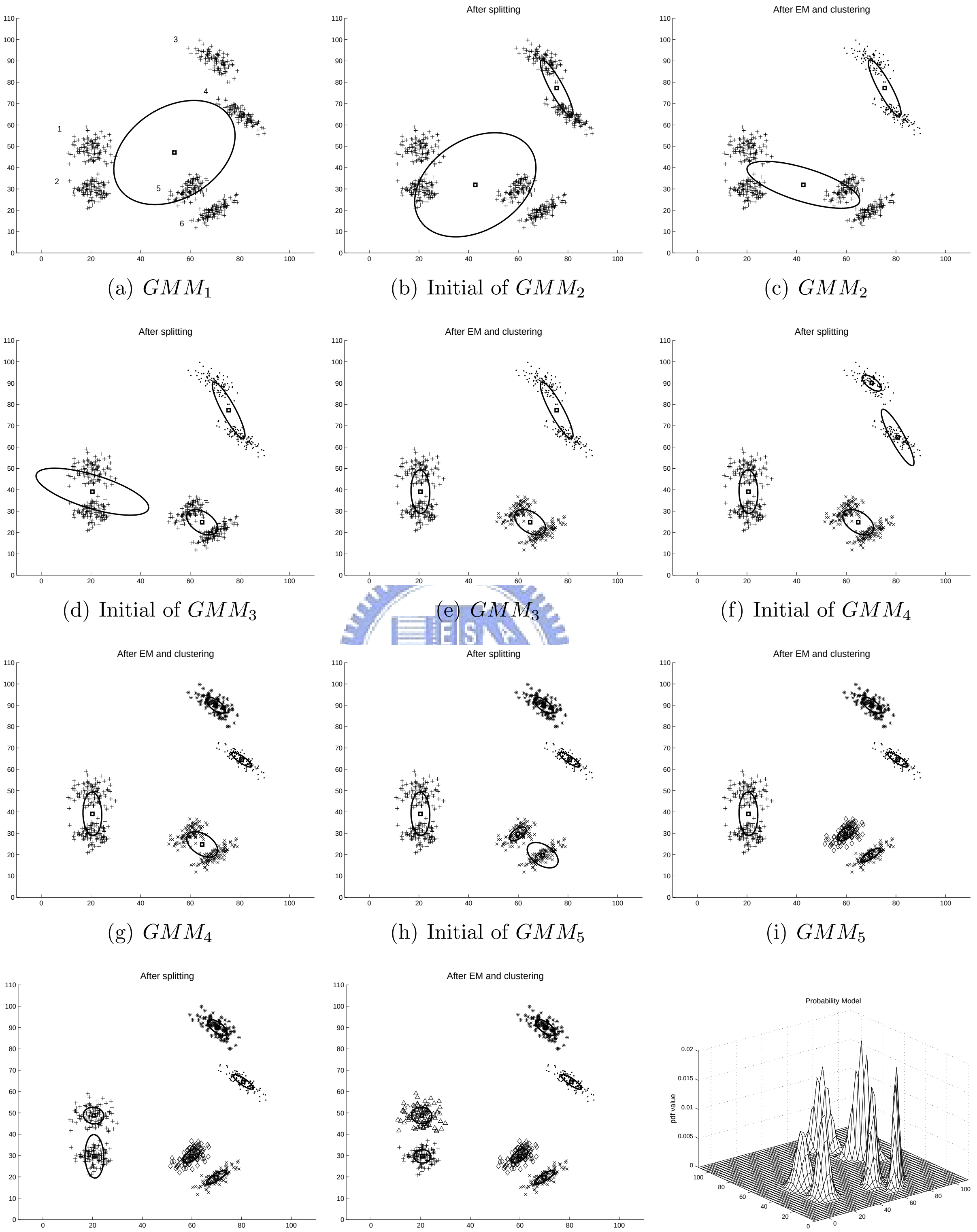
2. The shape-smoothing phase ($GMM_7 - GMM_{10}$): A diagonal covariance Gaussian component is unable to model a cluster which has a complex shape, and this kind of cluster needs to be presented by a mixture of Gaussians. For example, as shown in Figure 3.1 (a), clusters 3, 4, 5, and 6 are all oblique ellipses and each of them needs to be presented by a mixture of two Gaussians. As a result, the learning curve in Figure 3.3 has the largest BIC value at GMM_{10}. It is obvious that the increase of BIC value in the shape-smoothing phase is much smaller than that in the cluster-capturing phase.

3. The over-fitting phase ($GMM_{11}-$): After reaching the component number with the largest BIC value, SGML starts to over-fit the data in each splitting step and thus, the learning curve goes down progressively.

3.3.2 Results on the real-world data

In this section, we compared the performance of SGML and baseline methods by evaluating their performance on speaker GMM training with speech feature vectors from speaker #5007 in NIST01SpkEval. For feature extraction, 24 mel-frequency cepstral coefficients (MFCCs) were extracted using a 32-ms analysis window with a 10-ms shifting [87].

Figure 3.4 shows the learning curves obtained by the various methods on 15-second speech data (corresponding to 1500 24-dimensional MFCCs). Figures 3.4 (a)-(b) show the learning curves of the full covariance case, while (c)-(d) show the results of the diagonal covariance case. For these two cases, the upper bounds of component number were limited to 15 and 40, respectively (in fact, SGML stopped at GMM_9 and GMM_{21}, respectively). The corresponding “significant maximum” of the learning curves of SGML are at GMM_4 and GMM_{16}, respectively, which are also the global maximum in their corresponding



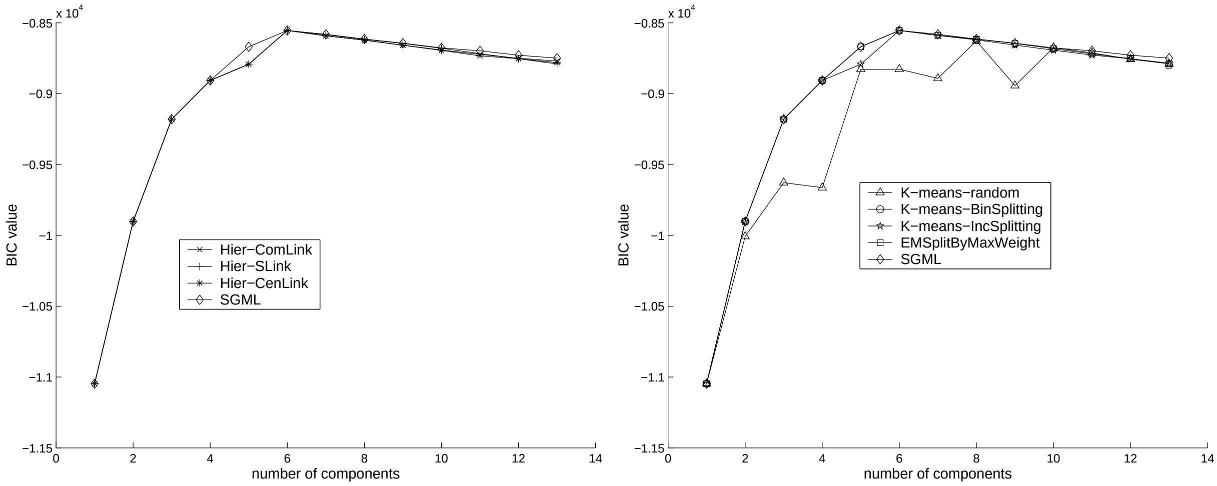
(a) GMM_1 (b) Initial of GMM_2 (c) GMM_2

(d) Initial of GMM_3 (e) GMM_3 (f) Initial of GMM_4

(g) GMM_4 (h) Initial of GMM_5 (i) GMM_5

(j) Initial of GMM_6 (k) GMM_6 (l) *3D plot of* GMM_6

Figure 3.1: The learning process of SGML on the synthetic data, where full covariance GMMs are used.



(a) (b)

Figure 3.2: The learning curves of SGML and the baseline approaches on the synthetic data, where full covariance GMMs are used.

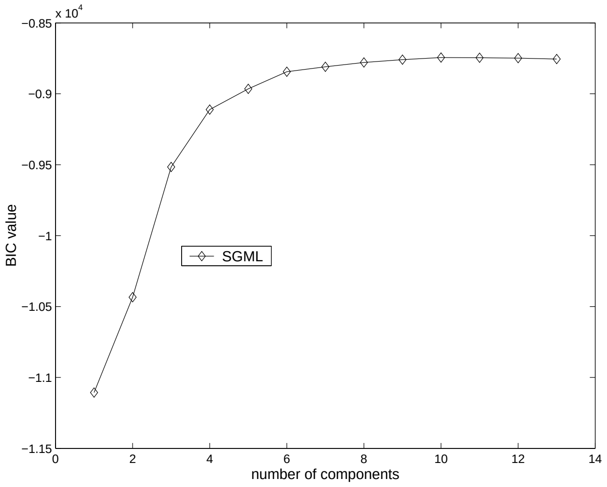


Figure 3.3: The learning curve of SGML on the synthetic data, where diagonal covariance GMMs are used.

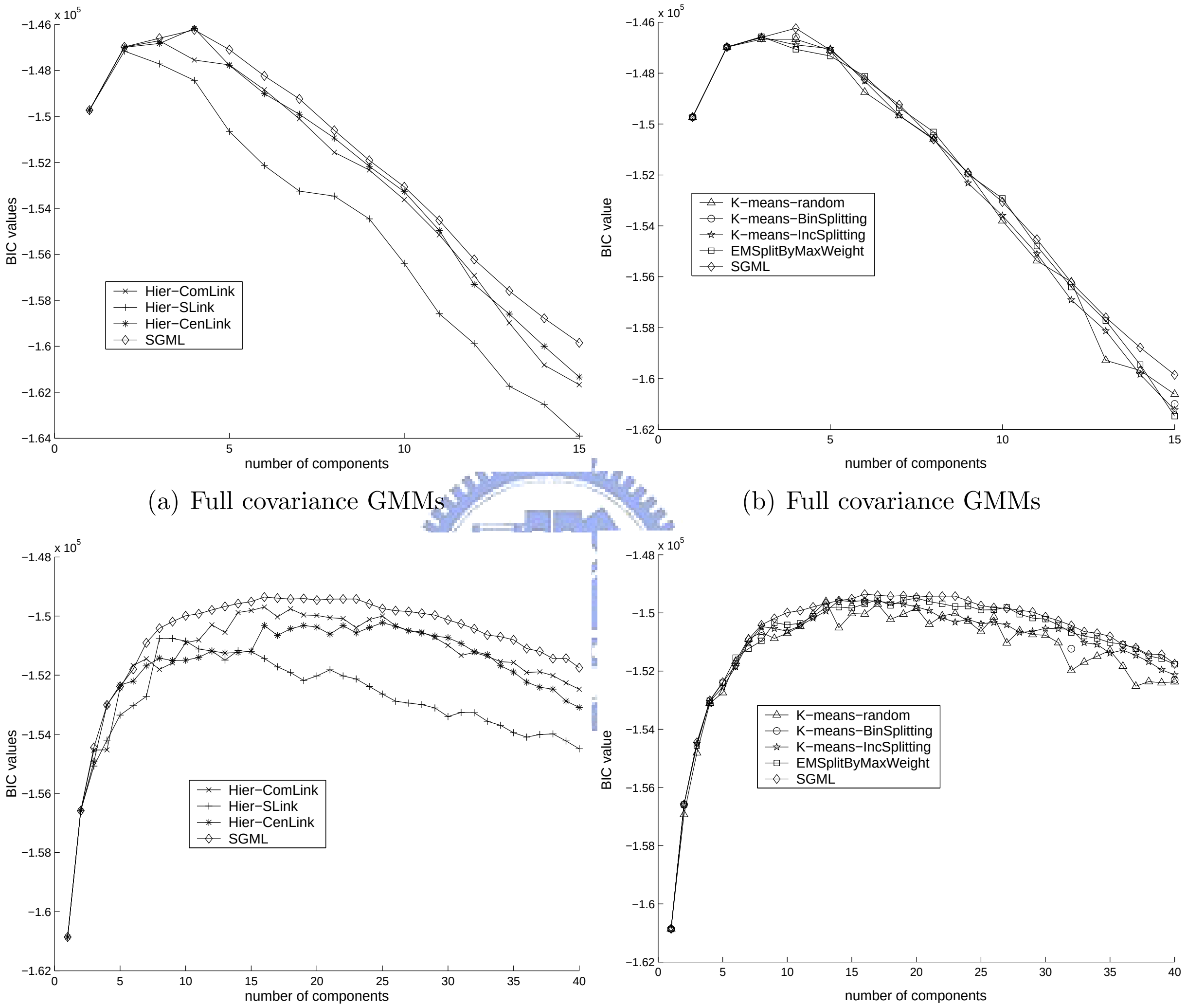
learning curves. From the figures, we see that the learning curves of SGML are smoother than those of the other methods. The self-splitting learning process splits the cluster with the largest ΔBIC_{21} value in each splitting step, and makes the learning curve go up steadily before reaching the most appropriate component number. After reaching the best component number, the learning process tends to split a well-modeled cluster in each splitting step, and makes the learning curve go down progressively. We also see that the BIC values of the GMMs trained by SGML are almost always higher than those of the GMMs trained by the other methods at any component number. As discussed in Section 2.4.1.3, with the same model complexity, the higher BIC value indicates the higher log-likelihood value. Therefore, SGML also outperforms the other methods in learning GMM with a given component number.

Next, we investigated the learning performance of fastSGML using diagonal covariance GMMs and 60-second speech data; and compared its performance to that of SGML and K-means-BinSplitting. Figure 3.5 shows the learning curves of fastSGML, K-means-BinSplitting and SGML. The *splitting confidence* of fastSGML are set at 150 (fastSGML-150), 100 (fastSGML-100), and 50 (fastSGML-50), respectively. The best component numbers obtained by fastSGML-150, fastSGML-100, fastSGML-50 and SGML were 27, 37, 43 and 40, respectively; and, hence, K-means-BinSplitting was forced to stop at GMM_{43}. From the figure, it seems that fastSGML-150 and fastSGML-100 under-fit the training data while fastSGML-50 over-fit it. From the learning curve of SGML, we see that the GMM whose component number is within the range of 30 to 50 has a BIC value similar to that of GMM_{40} (the best model selected by the SGML). fastSGML obtains a GMM with a very close BIC value and a very close component number to that of the best GMM obtained by SGML when *splitting confidence* is defined appropriately. Moreover, for a given component number, K-means-BinSplitting always obtains a smaller BIC value than that of fastSGML. We evaluated the speed of SGML and fastSGML with a Intel 3.2 GHz CPU. The run time of SGML, fastSGML-50, fastSGML-100, and fastSGML-150 were 268.96 sec., 48.01 sec., 97.18 sec., and 43.59 sec., respectively, from which we see the efficiency gain of fastSGML over SGML.

3.4 Application of SGML to GMM-based speaker identification

3.4.1 The GMM-based speaker identification

In the last decade, GMM-based approaches have been the leading ones for speaker identification and verification [4, 69, 3, 88, 89, 90, 91]. Suppose that the GMM-based speaker identification system enrolls M target speakers $S_1, S_2, \cdots, S_M$, where their GMMs are parameterized with $\mathbf{\Theta}_s^1, \mathbf{\Theta}_s^2, \cdots, \mathbf{\Theta}_s^M$, respectively. Then, for the given test speech repre-



(a) Full covariance GMMs

(b) Full covariance GMMs

(c) Diagonal covariance GMMs

(d) Diagonal covariance GMMs

Figure 3.4: The learning curves of applying the various GMM learning methods to the 15-second speech data of #5007 in NIST01SpkEval. (a) and (b) depicts the full covariance case, while (c) and (d) depicts the diagonal covariance case.

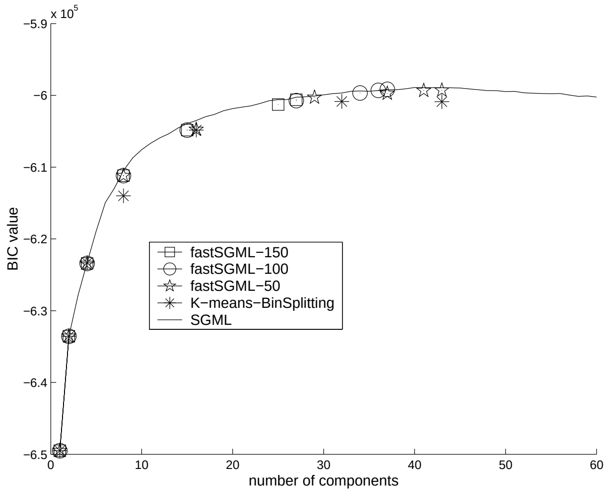


Figure 3.5: The learning curves of applying fastSGML, K-means-BinSplitting and SGML to learn diagonal covariance GMMs with 60-second speech data of #5007 in NIST01SpkEval. The *splitting confidence* of the fastSGML was set at 150, 100, and 50, respectively, and K-means-BinSplitting was forced to stop at GMM_{43}.

sented by feature vectors $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$, the goal is to find the speaker model $\hat{S}$ such that

$$\hat{S} = \arg\max_k \log p(\mathcal{X}; \mathbf{\Theta}_s^k). \tag{3.4}$$

Assuming that $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N$ are i.i.d., Eq. (3.4) can be rewritten as

$$\hat{S} = \arg\max_k \sum_{i=1}^{N} \log p(\mathbf{x}_i; \mathbf{\Theta}_s^k). \tag{3.5}$$

Typically, all the speaker GMMs are assigned the same component number which is decided empirically [4, 69, 3]. However, with the same component number, it may result in that some of the speaker GMMs are over-fitted and some are too simple to capture the distribution of the training data. Thus, it may be better that the model complexity of a speaker GMM is determined according to the distribution of the training data, which is the theme of the proposed SGML algorithm. In the following, we evaluate SGML's ability to automatically determining the model complexity for speaker GMMs using the identification accuracy.

3.4.2 Experiments

3.4.2.1 Database description and feature extraction

We conducted speaker identification experiments on NIST 2001 cellular speaker recognition evaluation data (NIST01SpkEval) [68]. This database consists of 74 male speakers and 100 female speakers; each speaker contains around two minutes speech for model training and ten test segments on average. The duration of the test segments lies within the range of 2 to 54 seconds. Both the training data and test data were first processed by a voice activity detector (VAD) to discard silence-noise frames [69, 3, 92]. After VAD, only 50 male speakers and 76 female speakers contain training speech more than 90 seconds. Thus, we used the 50 male speakers and the 50 female speakers whose training speech size are the top 50 among the 76 female speakers for experiments. There are 615 test segments for the selected 50 male speakers, and 596 test segments for the selected 50 female speakers. The duration of the test segments ranges from 3 to 54 seconds after VAD.

We used 24-dimensional MFCCs as the speech features, as described in Section 3.3.2. In addition, cepstral mean substraction (CMS) was applied in both training and test data for channel normalization.

3.4.2.2 Configuration of model training and test utterance

We conducted speaker identification experiments on each gender independently. For each gender, 30-, 60-, and 90-second speech data were used for training speaker GMMs with diagonal covariance matrices, respectively. We run the identification using variable-length test utterances and the fixed-length test utterances on the three model training configurations independently. In the former case, there are 615 test segments from male speakers and 596 test segments from female speakers. These test segments ranges from 3 to 54 seconds as described in Section 3.4.2.1. In the later case, each test segment was divided into utterances of 3, 5, and 8 seconds, which yielded a total of 4714, 2706, and 1577 test utterances for male speakers, and 5583, 3234, and 1903 test utterances for female speakers. Note that dividing utterances into fixed-length, short pieces might result in non-independent identification trials. However, the experimental results can still show the performance of the identification system being tested with short utterances.

3.4.2.3 Results

Tables 3.1 and 3.2 summarize the mean and standard deviation of the component number of the male speaker GMMs and female speaker GMMs, respectively, obtained by SGML and fastSGML on different amounts of training data. From the tables, we see that the *splitting confidence* within the range of 100 to 150 is appropriate for these two speaker identification tasks since in this case fastSGML yields a average model complexity similar to that of SGML. Furthermore, we see that, on average, a female speaker GMM needs more Gaussian components than a male speaker GMM for the same amount of training data. This reveals that, on average, the distribution of MFCC feature vectors of a female speaker is more diverse than that of a male speaker. We evaluated the CPU time cost of SGML and fastSGML with a Intel 3.2 GHz CPU. The results for male and female speakers are shown in Tables 3.3 and 3.4, respectively. These two tables show the efficiency gains of fastSGML over SGML. Moreover, the average CPU time of SGML for female speakers is significantly larger than that of the male speakers; this is due to the larger average component number in the former case. For fastSGML, however, the average CPU time for the female case is close to that of the male case; this shows the CPU time cost of fastSGML is much less sensitive to the model complexity than that of SGML does.

For the identification experiments, we used the K-means-BinSplitting algorithm described in Section 3.3 as the baseline approach since, like SGML and fastSGML, it learns the model based on component splitting. Tables 3.5 and 3.6 show, respectively, the identification accuracy of the male and female speakers, which are obtained by K-means-BinSplitting, SGML, and fastSGML on different amounts of training speech. From the male speaker case with 30-second training speech in Table 3.5, we see that the identification accuracy of K-means-BinSplitting is first improved by increasing the component number from 8 to 16, and then degraded by further increasing the component number to 32 and 64 which have over-fitted the training data. In this case, SGML yields 23.92 Gaussian components on average for each male speaker, as shown in Table 3.1.

For the female speaker case with 30-second training speech, a similar trend is observed, and the baseline system achieves the best identification accuracy with 32 mixture components. This conforms to the observation from Tables 3.1 and 3.2 that a female speaker GMM generally needs more Gaussian components than a male speaker GMM. In this case, SGML yielded 29.44 components on average for each female speaker, as shown in Table 3.2. In the cases of 60-second and 90-second training speech in Tables 3.5 and 3.6, we also observe that SGML can automatically determine the adequate model complexity for speaker GMMs according to the amount and characteristics of training data, though no significant difference is found between the identification accuracies of SGML and the best accuracies of K-means-BinSplitting under the different training and test conditions. We also observe that there is a huge identification performance gap between the female case and the male case. This gap is obviously due to the diversity of feature vectors of a female speaker. For the female case, more training data are needed to cover the diverse feature space.

From Tables 3.5 and 3.6, we see that fastSGML generally yields as good identification performance as SGML. Though the fastSGML with different splitting confidences might result in GMMs with different numbers of components, as shown in Tables 3.1 and 3.2, there is no significant difference in identification accuracy between these two approaches. The splitting confidence within the range of 100 to 150 seems appropriate for these two speaker identification tasks since for this case fastSGML yields similar model complexity for speaker GMMs and similar identification accuracy to those of SGML.

In summary, the speaker identification experimental results show that the proposed SGML and fastSGML algorithms can automatically find the appropriate component num-

Table 3.1: The mean and standard deviation of the component number of the diagonal covariance male speaker GMMs obtained by SGML and fastSGML on different amounts of training data. The first number in parentheses is the mean value, while the second number after '/' is the standard deviation.

Amount of training speech	SGML	*splitting confidence* (fastSGML)		
		150	100	50
30 sec	(23.92/5.07)	(20.58/5.31)	(25.32/6.63)	(27.16/5.99)
60 sec	(35.96/6.90)	(31.72/7.68)	(38.22/9.26)	(41.50/9.42)
90 sec	(46.70/9.23)	(41.30/9.48)	(48.58/11.13)	(53.00/10.65)

Table 3.2: The mean and standard deviation of the component number of the diagonal covariance female speaker GMMs obtained by SGML and fastSGML on different amounts of training data. The first number in parentheses is the mean value, while the second number after '/' is the standard deviation.

Amount of training speech	SGML	*splitting confidence* (fastSGML)		
		150	100	50
30 sec	(29.44/4.59)	(26.32/5.25)	(31.70/5.97)	(33.34/5.97)
60 sec	(45.06/7.18)	(42.22/6.93)	(50.26/7.85)	(52.60/10.40)
90 sec	(58.26/7.63)	(55.82/9.15)	(65.16/10.16)	(70.18/11.56)

ber for the speaker GMMs, though the identification accuracy is not significantly improved, compared to the best accuracies of the baseline system. More over, fastSGML is almost as effective as the SGML in the training of GMMs, but at a much lower computation cost.

Table 3.3: The average CPU time (in second) of the diagonal covariance male speaker GMMs obtained by SGML and fastSGML on different amounts of training data.

Amount of training speech	SGML	splitting confidence (fastSGML)		
		150	100	50
30 sec	90.41	14.23	19.09	21.83
60 sec	345.12	49.14	63.61	60.20
90 sec	752.54	97.03	110.60	109.27

Table 3.4: The average CPU time (in second) of the diagonal covariance female speaker GMMs obtained by SGML and fastSGML on different amounts of training data.

Amount of training speech	SGML	splitting confidence (fastSGML)		
		150	100	50
30 sec	116.81	14.42	20.21	19.62
60 sec	493.45	51.55	62.04	54.36
90 sec	1113.15	101.52	127.64	108.01

Table 3.5: Speaker identification accuracy (in %) for the male speakers.

Amount of training speech	Length of test utterance	number of components for K-means-BinSplitting				SGML	splitting confidence for fastSGML		
		8	16	32	64		150	100	50
30 sec	variable-length	61.46	62.28	61.46	59.35	61.95	61.46	62.76	61.79
	3 sec	51.68	54.41	54.41	52.38	54.09	53.67	53.86	54.07
	5 sec	56.91	58.39	57.02	55.99	57.98	57.58	57.80	57.91
	8 sec	59.80	60.68	60.24	58.47	61.88	60.11	61.19	60.49
60 sec	variable-length	63.25	66.02	66.34	65.85	66.67	66.67	67.15	66.83
	3 sec	54.29	58.38	58.91	59.29	59.10	59.65	59.80	59.16
	5 sec	58.94	61.90	62.82	62.12	63.45	62.75	63.19	62.75
	8 sec	61.95	65.25	66.14	65.88	66.39	66.65	67.22	66.77
90 sec	variable-length	65.53	68.78	69.11	68.62	70.08	70.57	71.06	70.24
	3 sec	55.28	59.10	61.03	61.99	61.69	61.96	61.79	61.75
	5 sec	60.01	63.34	65.59	65.67	65.78	64.56	65.37	65.78
	8 sec	63.86	66.01	68.55	68.36	69.93	69.75	69.37	68.29

Table 3.6: Speaker identification accuracy (in %) for the female speakers.

Amount of training speech	Length of test utterance	number of components for K-means-BinSplitting				SGML	splitting confidence for fastSGML		
		8	16	32	64		150	100	50
30 sec	variable-length	29.87	32.21	35.07	30.37	30.54	33.05	29.87	30.70
	3 sec	27.48	28.46	30.41	29.50	29.09	29.30	27.66	29.23
	5 sec	29.87	30.55	32.25	30.49	30.67	31.29	28.76	31.08
	8 sec	31.11	31.90	33.95	32.00	33.00	33.00	30.74	32.79
60 sec	variable-length	40.77	42.45	45.30	45.13	44.97	45.13	45.47	45.47
	3 sec	32.08	34.03	37.20	37.95	37.79	38.06	37.33	37.78
	5 sec	35.65	37.97	40.66	40.63	40.11	40.91	40.60	41.00
	8 sec	38.83	40.20	42.62	43.72	43.14	43.77	42.35	43.08
90 sec	variable-length	42.95	43.62	48.83	48.49	47.32	47.48	47.65	48.49
	3 sec	32.12	35.41	39.03	40.66	41.78	40.78	40.64	41.00
	5 sec	35.62	38.74	42.92	44.47	44.34	44.53	44.59	45.24
	8 sec	38.52	42.35	46.19	47.08	47.50	47.08	46.82	47.24

Algorithm 1 Self-splitting Gaussian Mixture Learning Algorithm (SGML)

Require: The input data set $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$

Ensure: The estimated parameter set $\hat{\boldsymbol{\Theta}} = \{\hat{w}(1), \hat{w}(2)\ , \cdots, \hat{w}(bestNum), \hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_2, \cdots, \hat{\boldsymbol{\theta}}_{bestNum}\}$, where $\hat{w}(k)$ and $\hat{\boldsymbol{\theta}}_k = \{\hat{\boldsymbol{\mu}}_k, \hat{\boldsymbol{\Sigma}}_k\}$ are the maximum likelihood estimate of mixture weight, mean vector, and covariance matrix of the kth Gaussian component

```
Begin

  1. Initialization:
       $SRange \leftarrow 5$;
       $cNum \leftarrow 1$;
       $\hat{\boldsymbol{\Theta}} \leftarrow \{\hat{w}(1), \hat{\boldsymbol{\theta}}_1\}$, where $\hat{\boldsymbol{\theta}}_1 = \{\hat{\boldsymbol{\mu}}_1, \hat{\boldsymbol{\Sigma}}_1\}$ are the sample mean vector and sample covariance matrix of $\mathcal{X}$;
       $GMM\_set(1) \leftarrow \hat{\boldsymbol{\Theta}}$;
       $//GMM\_set(cNum)$ is the parameter set of $GMM_{cNum}$ over $\mathcal{X}$
       $BIC\_set(1) \leftarrow BIC(GMM_1, \mathcal{X})$;

  2. Data clustering:
       $EM\_cluter_k \leftarrow \phi,\ for\ k \leftarrow 1, 2, \cdots, cNum$;
       for each sample $\mathbf{x}_i$:
         $j \leftarrow \arg\max_k p(k \mid \mathbf{x}_i; \hat{\boldsymbol{\Theta}})$;
         $EM\_cluster_j \leftarrow EM\_cluster_j \bigcup \mathbf{x}_i$;//add $\mathbf{x}_i$ to $EM\_cluster_j$

  3. Split (split one component into two new components):
       whichSplit$\leftarrow \arg\max_k \{\Delta BIC_{21}(EM\_cluster_k)\}$;
       Suppose the parameters of $GMM_2$ corresponding to $EM\_cluster_{whichSplit}$ are
       $\bar{\lambda}_1 \leftarrow \{\bar{w}(1), \bar{\boldsymbol{\theta}}_1\}, \bar{\lambda}_2 \leftarrow \{\bar{w}(2), \bar{\boldsymbol{\theta}}_2\}$, where $\bar{\boldsymbol{\theta}}_k = \{\bar{\boldsymbol{\mu}}_k, \bar{\boldsymbol{\Sigma}}_k\}, for\ k \leftarrow 1, 2$;
       Let
         $\bar{w}(1) \leftarrow \frac{1}{2}\hat{w}(whichSplit)$;
         $\bar{w}(2) \leftarrow \frac{1}{2}\hat{w}(whichSplit)$;
         $\hat{\boldsymbol{\Theta}} \leftarrow \hat{\boldsymbol{\Theta}} \setminus \{\hat{w}(whichSplit), \hat{\boldsymbol{\theta}}_{whichSplit}\}$;//remove $\{\hat{w}(whichSplit), \hat{\boldsymbol{\theta}}_{whichSplit}\}$
         $\hat{\boldsymbol{\Theta}} \leftarrow \hat{\boldsymbol{\Theta}} \bigcup \{\bar{\lambda}_1, \bar{\lambda}_2\}$;//add $\{\bar{\lambda}_1, \bar{\lambda}_2\}$
         $cNum \leftarrow cNum + 1$;

  4. Global EM learning:
       Perform EM learning on all the clusters with $\hat{\boldsymbol{\Theta}}$ as the model initialization,
       $GMM\_set(cNum) \leftarrow \hat{\boldsymbol{\Theta}}$;
       $BIC\_set(cNum) \leftarrow BIC(GMM_{cNum}, \mathcal{X})$;
       if ($cNum > SRange$ and $BIC\_set(cNum - SRange)$ is the
         maximum in $BIC\_set$)
         $bestNum \leftarrow cNum - SRange$;
         $\hat{\boldsymbol{\Theta}} \leftarrow GMM\_set(bestNum)$;
         goto End;
       else
         goto 2;

 End
```

Algorithm 2 Fast Self-splitting Gaussian Mixture Learning Algorithm (fastSGML)

Require: The input data set $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$
splitting confidence: the threshold for component splitting

Ensure: The estimated parameter set $\hat{\boldsymbol{\Theta}} = \{\hat{w}(1), \hat{w}(2) \ , \cdots, \hat{w}(bestNum), \hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_2, \cdots, \hat{\boldsymbol{\theta}}_{bestNum}\}$, where $\hat{w}(k)$ and $\hat{\boldsymbol{\theta}}_k = \{\hat{\boldsymbol{\mu}}_k, \hat{\boldsymbol{\Sigma}}_k\}$ are the maximum likelihood estimate of mixture weight, mean vector, and covariance matrix of the kth Gaussian component

```
Begin

  1. Initialization: the same as in SGML.

  2. Data clustering: the same as in SGML.

  3. Split:
       (a) temp ← $cNum$;
           for $k \leftarrow 1, 2, \cdots, temp$
             if ($\Delta BIC_{21}(EM\_cluster_k) > splitting\ confidence$)
               split the component corresponding to $EM\_cluster_k$ into two new
               components;
               $cNum \leftarrow cNum + 1$;
       (b) if (no component is split in (a))
             $\hat{\boldsymbol{\Theta}} \leftarrow GMM\_set(cNum)$;
             goto End;

  4. Global EM learning:
       Perform EM learning on the Gaussian components obtained in Step 3 (a);
       if (the learning curve starts to go down)
         let $bestNum$ be the component number that has the maximum value in
         the learning curve;
         $\hat{\boldsymbol{\Theta}} \leftarrow GMM\_set(bestNum)$;
         goto End;
       else
         goto 2.;

End
```

Chapter 4

Model-based clustering by probabilistic self-organizing maps

4.1 Formulation of the coupling-likelihood mixture model for PbSOMs

In this thesis, we define a PbSOM as a SOM that consists of G neurons $\mathcal{R}=\{r_1, r_2, \cdots, r_G\}$ in a network with a neighborhood function h_{kl} that defines the strength of lateral interaction between two neurons, r_k and r_l, for $k, l \in \{1, 2, \cdots, G\}$; and each neuron r_k associates with a reference model $\boldsymbol{\theta}_k$ that represents some probability distribution in the data space.

Sum *et al.* [67] interpreted Kohonen's sequential SOM learning algorithm in terms of maximizing the local correlations (coupling energies) between the neurons and their neighborhoods with the given input data. Given a data sample $\mathbf{x}_i \in \mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$, the local coupling energy between r_k and its neighborhood is defined as

$$\begin{aligned} E_{\mathbf{x}_i|k} &= \sum_{l=1}^{G} h_{kl} r_k(\mathbf{x}_i; \boldsymbol{\theta}_k) r_l(\mathbf{x}_i; \boldsymbol{\theta}_l) \\ &= r_k(\mathbf{x}_i; \boldsymbol{\theta}_k) \sum_{l=1}^{G} h_{kl} r_l(\mathbf{x}_i; \boldsymbol{\theta}_l), \end{aligned} \tag{4.1}$$

where $r_k(\mathbf{x}_i; \boldsymbol{\theta}_k)$ denotes the response of neuron r_k to $\mathbf{x}_i$, which is modeled by an isotropic Gaussian density. Then, the coupling energy over the network for $\mathbf{x}_i$ is defined as

$$E_{\mathbf{x}_i} = \sum_{k=1}^{G} E_{\mathbf{x}_i|k}, \tag{4.2}$$

and the energy function to be maximized is

$$C = \sum_{i=1}^{N} \log E_{\mathbf{x}_i}. \tag{4.3}$$

In Eq. (4.1), the term $\sum_{l=1}^{G} h_{kl} r_l(\mathbf{x}_i; \boldsymbol{\theta}_l)$ can be considered as the neighborhood response of r_k, where the conjunction between the neuron responses is implemented using the *summing* operation.

Here, we express the neuron response $r_l(\mathbf{x}_i; \boldsymbol{\theta}_l)$ as a multivariate Gaussian distribution as in Eq. (2.12) and formulate the neighborhood response of r_k as

$$\prod_{l \neq k} r_l(\mathbf{x}_i; \boldsymbol{\theta}_l)^{h_{kl}}, \tag{4.4}$$

where the conjunction between the neuron responses in the neighborhood of r_k is implemented using the *multiplicative* operation. Then, for a given $\mathbf{x}_i$, we define the local coupling energy between r_k and its neighborhood as the following coupling-likelihood:

$$\begin{aligned} p_s(\mathbf{x}_i | k; \boldsymbol{\Theta}, h) &= r_k(\mathbf{x}_i; \boldsymbol{\theta}_k)^{h_{kk}} \prod_{l \neq k} r_l(\mathbf{x}_i; \boldsymbol{\theta}_l)^{h_{kl}} \\ &= \prod_{l=1}^{G} r_l(\mathbf{x}_i; \boldsymbol{\theta}_l)^{h_{kl}} \\ &= \exp\Big(\sum_{l=1}^{G} h_{kl} \log r_l(\mathbf{x}_i; \boldsymbol{\theta}_l)\Big), \end{aligned} \tag{4.5}$$

where $\boldsymbol{\Theta}$ is the set of reference models, and h denotes the given neighborhood function[1]. Then, we define the coupling-likelihood of $\mathbf{x}_i$ over the network as the following (unnormalized) mixture likelihood:

$$p_s(\mathbf{x}_i; \boldsymbol{\Theta}, h) = \sum_{k=1}^{G} w_s(k) p_s(\mathbf{x}_i | k; \boldsymbol{\Theta}, h), \tag{4.6}$$

where $w_s(k)$ for $k = 1, 2, \cdots, G$ is fixed at $1/G$. Note that, theoretically, the mixture weights can be learned automatically. When maximizing the local coupling-likelihood $p_s(\mathbf{x}_i | k; \boldsymbol{\Theta}, h)$ for each neuron r_k, $k = 1, 2, \cdots, G$, the topological order between neuron r_k and its neighborhood for the given data sample $\mathbf{x}_i$ is learned in the learning process; therefore, we use equal mixture weights in the mixture model to take account of the topological order learning induced by the neurons faithfully (with equal prior importance).

[1] From Eq. (4.5), it is obvious that, in our formulation, the coupling between r_k and its neighboring neurons is considered jointly, whereas Sum *et al.*'s formulation considers it in a pairwise manner, as shown in Eq. (4.1). Note that we use the term "coupling-likelihood" instead of "coupling energy" for two reasons: 1) Eq. (4.5) is a coupling of Gaussian likelihoods; and 2) using "coupling-likelihood" can help describe the link between our proposed approaches and model-based clustering.

In fact, this is important for learning an ordered map. From our experimental analysis, if the mixture weights are updated in the learning process, the learning of topological order is frequently dominated by some particular mixture components, which makes it difficult to obtain an ordered map. For details, one can refer to Appendix A.1 after reading this chapter.

Comparing the network structure of the proposed coupling-likelihood mixture model in Eq. (4.6) with that of the Gaussian mixture model (GMM), as shown in Figure 4.1, the proposed model inserts a coupling-likelihood layer between the Gaussian likelihood layer and the mixture likelihood layer to take account of the coupling between the neurons and their neighborhoods. When the neighborhood size is reduced to zero (i.e., $h_{kl}=\delta_{kl}$), the coupling-likelihood mixture model becomes a GMM with equal mixture weights.

Note that other probability distributions are possible for $r_l(\mathbf{x}_i;\boldsymbol{\theta}_l)$ in the formulation of the coupling-likelihood mixture model, although we use the multivariate Gaussian distribution here.

4.2 The SOCEM algorithm for learning PbSOMs

The self-organizing process of PbSOM can be described as a model-based data clustering procedure that preserves the spatial relationships between the data samples and clusters in a network. Based on the classification likelihood criterion for data clustering [17], the computation of the coupling-likelihood of a data sample is restricted to its winning neuron. Thus, the goal is to estimate the partition of $\mathcal{X}$, $\hat{\mathcal{P}} = \{\hat{\mathcal{P}}_1, \hat{\mathcal{P}}_2, \cdots, \hat{\mathcal{P}}_G\}$, and the set of reference models, $\hat{\boldsymbol{\Theta}}$, so as to maximize the accumulated classification log-likelihood over all the data samples as follows:

$$\begin{aligned} C_s(\mathcal{P},\boldsymbol{\Theta};\mathcal{X},h) &= \sum_{k=1}^{G}\sum_{\mathbf{x}_i\in\mathcal{P}_k}\log(w_s(k)p_s(\mathbf{x}_i|k;\boldsymbol{\Theta},h)) \\ &= \sum_{k=1}^{G}\sum_{\mathbf{x}_i\in\mathcal{P}_k}\log(w_s(k)\exp(\sum_{l=1}^{G}h_{kl}\log r_l(\mathbf{x}_i;\boldsymbol{\theta}_l))). \end{aligned} \tag{4.7}$$

As $w_s(k)$ for $k=1,2,\cdots,G$ is fixed at $1/G$, the objective function can be rewritten as

$$C_s(\mathcal{P},\boldsymbol{\Theta};\mathcal{X},h) = \sum_{k=1}^{G}\sum_{\mathbf{x}_i\in\mathcal{P}_k}\sum_{l=1}^{G}h_{kl}\log r_l(\mathbf{x}_i;\boldsymbol{\theta}_l) + Const. \tag{4.8}$$

Similar to the derivation of the classification EM (CEM) algorithm for model-based clustering in [17], the CEM algorithm for the proposed PbSOM, i.e., the SOCEM algorithm, is derived as follows.

E-step: Given the current reference model set, $\boldsymbol{\Theta}^{(t)}$, compute the posterior probability of each mixture component of $p_s(\mathbf{x}_i; \mathbf{\Theta}^{(t)}, h)$ for each $\mathbf{x}_i$ as follows:

$$\begin{aligned} \gamma_{k|i}^{(t)} &= p_s(k|\mathbf{x}_i; \mathbf{\Theta}^{(t)}, h) \\ &= \frac{p_s(\mathbf{x}_i, k; \mathbf{\Theta}^{(t)}, h)}{p_s(\mathbf{x}_i; \mathbf{\Theta}^{(t)}, h)} \\ &= \frac{\exp(\sum_{l=1}^{G} h_{kl} \log r_l(\mathbf{x}_i; \boldsymbol{\theta}_l^{(t)}))}{\sum_{j=1}^{G} \exp(\sum_{l=1}^{G} h_{jl} \log r_l(\mathbf{x}_i; \boldsymbol{\theta}_l^{(t)}))}, \end{aligned} \tag{4.9}$$

for $k = 1, 2, \cdots, G$, and $i = 1, 2, \cdots, N$.

C-step: Assign each $\mathbf{x}_i$ to the cluster whose corresponding mixture component has the largest posterior probability for $\mathbf{x}_i$, i.e., $\mathbf{x}_i \in \hat{\mathcal{P}}_j^{(t)}$ if $j = \arg\max_k \gamma_{k|i}^{(t)}$.

M-step: After the *C-step*, the partition of $\mathcal{X}$ (i.e., $\hat{\mathcal{P}}^{(t)}$) is formed, and the objective function C_s defined in Eq. (4.8) becomes

$$C_s(\mathbf{\Theta}; \hat{\mathcal{P}}^{(t)}, \mathcal{X}, h) = \sum_{l=1}^{G} \sum_{k=1}^{G} \sum_{\mathbf{x}_i \in \hat{\mathcal{P}}_k^{(t)}} h_{kl} \log r_l(\mathbf{x}_i; \boldsymbol{\theta}_l) + Const. \tag{4.10}$$

Similar to the derivation of the *M-step* of the EM algorithm for learning a Gaussian mixture model [20], we can obtain the re-estimation formulae for the mean vectors and covariance matrices by taking the derivative of C_s with respect to individual parameters, and then setting it to zero. The re-estimation formulae are as follows:

$$\boldsymbol{\mu}_l^{(t+1)} = \frac{\sum_{k=1}^{G} \sum_{\mathbf{x}_i \in \hat{\mathcal{P}}_k^{(t)}} h_{kl} \mathbf{x}_i}{\sum_{k=1}^{G} |\hat{\mathcal{P}}_k^{(t)}| h_{kl}}, \tag{4.11}$$

$$\mathbf{\Sigma}_l^{(t+1)} = \frac{\sum_{k=1}^{G} \sum_{\mathbf{x}_i \in \hat{\mathcal{P}}_k^{(t)}} h_{kl} (\mathbf{x}_i - \boldsymbol{\mu}_l^{(t+1)})(\mathbf{x}_i - \boldsymbol{\mu}_l^{(t+1)})^T}{\sum_{k=1}^{G} |\hat{\mathcal{P}}_k^{(t)}| h_{kl}} \tag{4.12}$$

for $l = 1, 2, \cdots, G$. When the neighborhood size is reduced to zero (i.e., h_{kl}=δ_{kl}), SOCEM reduces to the CEM algorithm for learning GMMs with equal mixture weights, as in Eqs. (2.25)-(2.26).

4.2.1 SOCEM - a DA variant of CEM for GMM

Similar to Kohonen's sequential or batch algorithm, the SOCEM algorithm is applied in two stages. First, it is applied to a large neighborhood to form an ordered map near the center of the data samples. Then, the reference models are adapted to fit the distribution of the data samples by gradually shrinking the neighborhood.

Without loss of generality, we suppose the neighborhood function is the widely adopted (unnormalized) Gaussian kernel in Eq. (2.4). As shown in Algorithm 3, initially, SOCEM is applied with a large σ value, which is reduced after the algorithm converges. Then, we use the new σ value and the learned parameters as the initial condition of the next learning phase. This process is repeated until the value of σ is reduced to the predefined minimum value σ_{min}. The above shrinking of the neighborhood (reduction of the σ value) can be interpreted as an annealing process, where a large σ value corresponds to a high temperature. Table 4.1 lists the learning rules of the DAEM algorithm for learning GMMs with equal mixture weights [23] and the SOCEM algorithm. To facilitate the interpretation, we rewrite the objective function and re-estimation formulae of SOCEM in Eq. (4.8) and Eqs. (4.11)-(4.12), respectively, with the new variable win_i, which denotes the index of the winning neuron of $\mathbf{x}_i$. For simplicity, we only list the re-estimation formulae of the mean vectors of the Gaussian components.

By analyzing these two algorithms carefully, one may view $h_{win_i^{(t)}l}$ as a kind of posterior probability of $\boldsymbol{\theta}_l^{(t)}$ for $\mathbf{x}_i$ in the network domain. More precisely, $\mathbf{x}_i$ is initially projected into $r_{win_i^{(t)}}$ in the network domain; then, $r_{win_i^{(t)}}$ is applied to Eq. (2.4) as an observation of the Gaussian kernel centered at r_l to obtain the value of $h_{win_i^{(t)}l}$. In both the DAEM and SOCEM algorithms, when the temperature ($1/\beta$ or σ) is high, the posterior distribution becomes almost uniform; hence, all the reference models will be moved to locations near the center of the data samples in this learning phase. By gradually reducing the temperature, the influence of each $\mathbf{x}_i$ becomes more localized, and the reference models gradually spread out to fit the distribution of the data samples. When the temperature approaches zero, the probabilistic assignment strategy for the data samples becomes the winner-take-all strategy, and the objective functions and learning rules of DAEM and SOCEM are equivalent to those of CEM. The major difference between DAEM and SOCEM seems to be that the posterior distribution in SOCEM is constrained by the network topology, but DAEM does not have this property.

To visualize the transition of the objective function, we show a simulation on a simple one-dimension, two-component Gaussian mixture problem in Figure 4.2[2]. The training data contains 200 observations drawn from

$$p(x;\{m_1,v_1\},\{m_2,v_2\}) = \frac{0.3}{v_1\sqrt{2\pi}}\exp(\frac{-(x-m_1)^2}{2v_1^2}) + \frac{0.7}{v_2\sqrt{2\pi}}\exp(\frac{-(x-m_2)^2}{2v_2^2}), \quad (4.13)$$

where the Gaussian means are (m_1,m_2)=(-5,5); and the Gaussian variances are (v_1^2,v_2^2)=(1, 1). The PbSOM network structure is a 1 × 2 lattice in [0,1]. The two reference models are $\boldsymbol{\theta}_1 = \{\mu_1, \Sigma_1\}$ and $\boldsymbol{\theta}_2 = \{\mu_2, \Sigma_2\}$, where $\Sigma_1 = \Sigma_2 = 1$. The objective function in Eq. (4.7) is calculated with different setups for (μ_1,μ_2) to form the log-likelihood surface. From Figure 4.2, we observe that a larger σ for h_{kl} yields a simpler objective function for optimization. The log-likelihood surface is symmetric along μ_1=μ_2 because of the

[2] Visualization of how deterministic annealing EM/CEM works for function optimization is illustrated in detail in [23].

Algorithm 3 The SOCEM algorithm with a shrinking neighborhood size (σ)

```
Require: $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$: the input data set;
  $\sigma_{ini}$: the initial $\sigma$ value for $h_{kl}$ in Eq. (2.4);
  $\varepsilon$: the decreasing step for $\sigma$;
  $\sigma_{min}$: the target $\sigma$ value;
  $\mathbf{\Theta}^{(0)} = \{\boldsymbol{\theta}_1^{(0)}, \boldsymbol{\theta}_2^{(0)}, \cdots, \boldsymbol{\theta}_G^{(0)}\}$: the initial reference models, where $\boldsymbol{\theta}_l^{(0)} = \{\boldsymbol{\mu}_l^{(0)}, \mathbf{\Sigma}_l^{(0)}\}$ are
  the initial mean vector and covariance matrix of the $l$th Gaussian component
Ensure: $\hat{\mathbf{\Theta}} = \{\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_2, \cdots, \hat{\boldsymbol{\theta}}_G\}$: the estimated parameter set, where $\hat{\boldsymbol{\theta}}_l = \{\hat{\boldsymbol{\mu}}_l, \hat{\mathbf{\Sigma}}_l\}$ are
  the estimated mean vector and covariance matrix of the $l$th Gaussian component

  Begin
    1. $\hat{\mathbf{\Theta}} \leftarrow \mathbf{\Theta}^{(0)}$; $\sigma \leftarrow \sigma_{ini}$;
    2. create the lookup table for $h_{kl}$;
    3. //CEM:
       repeat
         E-step: for $i \leftarrow 1, 2, \cdots, N$ and $k \leftarrow 1, 2, \cdots, G$, compute $\gamma_{k|i}^{(t)}$ in Eq. (4.9)
                 using $\hat{\mathbf{\Theta}}$;
         C-step: assign $\mathbf{x}_i$ to $\hat{\mathcal{P}}_j^{(t)}$ if $j = \arg\max_k \gamma_{k|i}^{(t)}$;
         M-step: for $l \leftarrow 1, 2, \cdots, G$, update $\hat{\boldsymbol{\mu}}_l$ and $\hat{\mathbf{\Sigma}}_l$ with Eqs. (4.11)-(4.12);
       until the convergence condition is met
    4. if ($\sigma = \sigma_{min}$)
         goto End;
       $\sigma \leftarrow \sigma - \varepsilon$;
       if ($\sigma < \sigma_{min}$)
         $\sigma \leftarrow \sigma_{min}$;
       goto 2.;
  End
```

symmetric lattice structure and equal weighting of the reference models. For the case of $\sigma = 0.6$, the log-likelihood value is close to the global maximum of the surface when both μ_1 and μ_2 are close to the center of the data (2.39 in this case). With the reduction in the value of σ, the location of (μ_1,μ_2) for the global maximum moves toward (m_1,m_2) and (m_2,m_1).

4.2.2 Relation to Kohonen's batch algorithm

There are two differences between the SOCEM algorithm and Kohonen's batch algorithm. First, SOCEM considers the neighborhood information when selecting the winning neuron, but Kohonen's algorithm does not. Second, SOCEM extends the reference vectors in Kohonen's algorithm with multivariate Gaussians. In other words, if we set $\gamma_{k|i}^{(t)}$ in

Table 4.1: The DAEM algorithm for learning GMMs with equal mixture weights and the SOCEM algorithm.

Algorithm	DAEM	SOCEM
Objective function	$F_\beta(\mathbf{\Theta}; \mathcal{X})$ in Eq. (2.32) where $p(\mathbf{x}_i, l; \mathbf{\Theta}) = \frac{1}{G} r_l(\mathbf{x}_i; \boldsymbol{\theta}_l)$	$\sum_{i=1}^N \sum_{l=1}^G h_{win_i l} \log r_l(\mathbf{x}_i; \boldsymbol{\theta}_l) + Const$
Posterior distribution	$f(l \vert \mathbf{x}_i; \mathbf{\Theta}^{(t)}) = \frac{r_l(\mathbf{x}_i; \boldsymbol{\theta}_l^{(t)})^\beta}{\sum_{j=1}^G r_j(\mathbf{x}_i; \boldsymbol{\theta}_j^{(t)})^\beta}$ $l = 1, 2, \cdots, G$	$h_{win_i^{(t)} l} = \exp(-\frac{\Vert r_{win_i^{(t)}} - r_l \Vert^2}{2\sigma^2})$ $l = 1, 2, \cdots, G$
Temperature	$1/\beta$	σ
Re-estimation formulae	$\boldsymbol{\mu}_l^{(t+1)} = \frac{\sum_{i=1}^N f(l \vert \mathbf{x}_i; \mathbf{\Theta}^{(t)}) \mathbf{x}_i}{\sum_{i=1}^N f(l \vert \mathbf{x}_i; \mathbf{\Theta}^{(t)})}$ $l = 1, 2, \cdots, G$	$\boldsymbol{\mu}_l^{(t+1)} = \frac{\sum_{i=1}^N h_{win_i^{(t)} l} \mathbf{x}_i}{\sum_{i=1}^N h_{win_i^{(t)} l}}$ $l = 1, 2, \cdots, G$

SOCEM to $\frac{r_k(\mathbf{x}_i; \boldsymbol{\theta}_k^{(t)})}{\sum_{j=1}^G r_j(\mathbf{x}_i; \boldsymbol{\theta}_j^{(t)})}$, instead of the setting in Eq. (4.9), we obtain a probabilistic variant of Kohonen's batch algorithm (denoted as KohonenGaussian), where Kohonen's winner selection strategy is applied and the reference vectors are replaced with multivariate Gaussians. Thus, we may view KohonenGaussian as an approximate implementation of SOCEM that optimizes SOCEM's objective function. Moreover, if we set the covariance matrices in KohonenGaussian to be diagonal with small, identical variances, KohonenGaussian is equivalent to Kohonen's batch algorithm. Therefore, we can interpret the neighborhood shrinking of Kohonen's algorithms as a deterministic annealing process, and thereby explain why they need to start with a large neighborhood size.

Recently, Zhong and Ghosh [12] interpreted the neighborhood size of the SOM algorithms that apply Kohonen's winner selection strategy as a temperature parameter in a deterministic annealing process. However, their interpretations were not based on the optimization of an objective function, which is the essential part of DA-based optimization. In contrast, in SOCEM, the neighborhood shrinking leads to the transition of the objective function from a simpler one to a more complex one, as illustrated in Figure 4.2.

4.2.3 Computational cost

It is clear from Table 4.1 that the computational cost of DAEM is $O(GNM)$, where G, N, and M are the numbers of reference models, data samples, and learning iterations, respectively. Compared to DAEM, SOCEM needs additional $O(G^2N)$ multiplication and addition operations for winner selection in each iteration, while KohonenGaussian needs additional $O(GN)$ multiplications and additions.

4.3 The SOEM algorithm for learning PbSOMs

As is obvious from Eq. (4.7), in the formulation of the objective function of the SOCEM algorithm, only the local coupling-likelihoods associated with the winning neurons are considered. Alternatively, we can compute the coupling-likelihood of $\mathbf{x}_i$ using the mixture likelihood defined in Eq. (4.6) and apply the EM algorithm to maximize the objective log-likelihood function

$$L_s(\boldsymbol{\Theta}; \mathcal{X}, h) = \sum_{i=1}^{N} \log(\sum_{k=1}^{G} w_s(k) p_s(\mathbf{x}_i | k; \boldsymbol{\Theta}, h)). \tag{4.14}$$

The steps of the EM algorithm for the proposed PbSOM, i.e., the SOEM algorithm, are as follows.

E-step: With the mixture model in Eq. (4.6), we form the auxiliary function as

$$Q_s(\boldsymbol{\Theta}; \boldsymbol{\Theta}^{(t)}) = \sum_{i=1}^{N} \sum_{k=1}^{G} \gamma_{k|i}^{(t)} \log p_s(\mathbf{x}_i, k; \boldsymbol{\Theta}, h), \tag{4.15}$$

where $\gamma_{k|i}^{(t)}$ is the same as Eq. (4.9). Since $p_s(\mathbf{x}_i, k; \boldsymbol{\Theta}, h) = w_s(k) p_s(\mathbf{x}_i | k; \boldsymbol{\Theta}, h)$, Eq. (4.15) can be rewritten as

$$Q_s(\boldsymbol{\Theta}; \boldsymbol{\Theta}^{(t)}) = \sum_{i=1}^{N} \sum_{k=1}^{G} \gamma_{k|i}^{(t)} \log(w_s(k) p_s(\mathbf{x}_i | k; \boldsymbol{\Theta}, h)). \tag{4.16}$$

As $w_s(k)$ for $k = 1, 2, \cdots, G$ is fixed at $1/G$, by substituting Eq. (4.5) into Eq. (4.16), the auxiliary function can be rewritten as

$$\begin{aligned} Q_s(\boldsymbol{\Theta}; \boldsymbol{\Theta}^{(t)}) &= \sum_{i=1}^{N} \sum_{k=1}^{G} \gamma_{k|i}^{(t)} \sum_{l=1}^{G} h_{kl} \log r_l(\mathbf{x}_i; \boldsymbol{\theta}_l) + Const. \\ &= \sum_{l=1}^{G} \sum_{i=1}^{N} \sum_{k=1}^{G} \gamma_{k|i}^{(t)} h_{kl} \log r_l(\mathbf{x}_i; \boldsymbol{\theta}_l) + Const. \end{aligned} \tag{4.17}$$

M-step: By replacing the response $r_l(\mathbf{x}_i; \boldsymbol{\theta}_l)$ in Eq. (4.17) with the multivariate Gaussian density in Eq. (2.12) and setting the derivative of Q_s with respect to individual mean vectors and covariance matrices to zero, we obtain the following re-estimation formulae:

$$\boldsymbol{\mu}_l^{(t+1)} = \frac{\sum_{i=1}^{N} (\sum_{k=1}^{G} \gamma_{k|i}^{(t)} h_{kl}) \mathbf{x}_i}{\sum_{i=1}^{N} (\sum_{k=1}^{G} \gamma_{k|i}^{(t)} h_{kl})}, \tag{4.18}$$

$$\boldsymbol{\Sigma}_l^{(t+1)} = \frac{\sum_{i=1}^{N} (\sum_{k=1}^{G} \gamma_{k|i}^{(t)} h_{kl}) (\mathbf{x}_i - \boldsymbol{\mu}_l^{(t+1)}) (\mathbf{x}_i - \boldsymbol{\mu}_l^{(t+1)})^T}{\sum_{i=1}^{N} (\sum_{k=1}^{G} \gamma_{k|i}^{(t)} h_{kl})} \tag{4.19}$$

for $l = 1, 2, \cdots, G$. When the neighborhood size is reduced to zero (i.e., h_{kl}=δ_{kl}), SOEM reduces to the EM algorithm for learning GMMs with equal mixture weights, as in Eq. (2.14)-(2.15) .

There are two major differences between the SOCEM and SOEM algorithms. First, they learn maps based on the classification likelihood criterion and the mixture likelihood criterion, respectively. Second, SOEM adapts the reference models in a more global way than SOCEM. To explain this perspective, we can consider the learning of SOCEM and SOEM in the sense of sequential learning. As illustrated in Figure 4.3, in the SOCEM algorithm (*cf.* Eqs. (4.11)-(4.12)), each data sample $\mathbf{x}_i$ only contributes to the adaptation of the winning reference model and its neighborhood (i.e., $\mathbf{x}_i$ only contributes to the learning of the topological order between the winning reference model and its neighborhood). However, in the SOEM algorithm (*cf.* Eqs. (4.18)-(4.19)), each data sample $\mathbf{x}_i$ contributes proportionally to the adaptation of each reference model and its neighborhood according to the posterior probabilities $\gamma_{k|i}^{(t)}$ for $k = 1, 2, \cdots, G$.

4.3.1 SOEM - a DA variant of EM for GMM

As with the SOCEM algorithm, we can apply SOEM to a large neighborhood and obtain different map configurations by gradually reducing the neighborhood size, as shown in Algorithm 4. The term $\sum_{k=1}^{G} \gamma_{k|i}^{(t)} h_{kl}$ in Eqs. (4.18)-(4.19) can be considered as a kind of posterior probability, $\pi(l|\mathbf{x}_i; \mathbf{\Theta}^{(t)}, h)$, of the reference model $\boldsymbol{\theta}_l^{(t)}$ for $\mathbf{x}_i$, which is also constrained by the neighborhood function. With a large σ value in h_{kl} (Eq. (2.4)), $\pi(l|\mathbf{x}_i; \mathbf{\Theta}^{(t)}, h)$ for $l = 1, 2, \cdots, G$, will be nearly a uniform distribution due to the small variation in the values of $\gamma_{k|i}^{(t)}$ for $k = 1, 2, \cdots, G$, and the small variation in the values of h_{kl} for $k = 1, 2, \cdots, G$, for each case of l. Hence, all the reference models will be moved to locations near the center of the data samples. When the neighborhood size is reduced to zero (i.e., h_{kl}=δ_{kl}), the SOEM algorithm becomes the EM algorithm for learning GMMs with equal mixture weights. As with the annealing interpretation of SOCEM, SOEM can be viewed as a topology-constrained deterministic annealing variant of the EM algorithm for learning GMMs with equal mixture weights[3].

4.3.2 Computational cost

Comparing Eqs. (4.17)-(4.19) to Eqs. (4.10)-(4.12), we can see that, in each learning iteration, SOEM and SOCEM have a similar computational cost in the *E-step*, but the former needs additional $O(GN)$ multiplication and addition operations for updating the model parameters in the *M-step*.

[3] SOEM yielded a similar result on the one-dimension, two-component Gaussian mixture problem in Figure 4.2; however, we do not present it here to avoid redundancy.

Algorithm 4 The SOEM algorithm with a shrinking neighborhood size (σ)

```
Require: $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$: the input data set;
  $\sigma_{ini}$: the initial $\sigma$ value for $h_{kl}$ in Eq. (2.4);
  $\varepsilon$: the decreasing step for $\sigma$;
  $\sigma_{min}$: the target $\sigma$ value;
  $\mathbf{\Theta}^{(0)} = \{\boldsymbol{\theta}_1^{(0)}, \boldsymbol{\theta}_2^{(0)}, \cdots, \boldsymbol{\theta}_G^{(0)}\}$: the initial reference models, where $\boldsymbol{\theta}_l^{(0)} = \{\boldsymbol{\mu}_l^{(0)}, \mathbf{\Sigma}_l^{(0)}\}$ are
  the initial mean vector and covariance matrix of the $l$th Gaussian component
Ensure: $\hat{\mathbf{\Theta}} = \{\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_2, \cdots, \hat{\boldsymbol{\theta}}_G\}$: the estimated parameter set, where $\hat{\boldsymbol{\theta}}_l = \{\hat{\boldsymbol{\mu}}_l, \hat{\mathbf{\Sigma}}_l\}$ are
  the estimates mean vector and covariance matrix of the $l$th Gaussian component

  Begin
    1. $\hat{\mathbf{\Theta}} \leftarrow \mathbf{\Theta}^{(0)}$; $\sigma \leftarrow \sigma_{ini}$;
    2. create the lookup table for $h_{kl}$;
    3. //EM:
       repeat
         E-step: for $i \leftarrow 1, 2, \cdots, N$ and $k \leftarrow 1, 2, \cdots, G$, compute $\gamma_{k|i}^{(t)}$ in Eq. (4.9)
                 using $\hat{\mathbf{\Theta}}$;
         M-step: for $l \leftarrow 1, 2, \cdots, G$, update $\hat{\boldsymbol{\mu}}_l$ and $\hat{\mathbf{\Sigma}}_l$ with Eqs. (4.18)-(4.19);
       until the convergence condition is met
    4. if ($\sigma = \sigma_{min}$)
         goto End;
       $\sigma \leftarrow \sigma - \varepsilon$;
       if ($\sigma < \sigma_{min}$)
         $\sigma \leftarrow \sigma_{min}$;
       goto 2.;
  End
```

4.4 The SODAEM algorithm for learning PbSOMs

Similar to the derivation of the deterministic annealing EM (DAEM) algorithm for learning GMMs [23], we developed a DAEM algorithm for the proposed PbSOM, called the SODAEM algorithm. With the mixture likelihood defined in Eq. (4.6), DAEM first derives the posterior density in the *E-step* using the principle of maximum entropy. Following the derivation of the posterior probability in [23] with the current model's parameter set $\mathbf{\Theta}^{(t)}$, we obtain the posterior probability of the kth mixture component for $\mathbf{x}_i$ as follows:

$$\begin{aligned}\tau_{k|i}^{(t)} &= \frac{p_s(\mathbf{x}_i|k; \mathbf{\Theta}^{(t)}, h)^{\beta}}{\sum_{j=1}^{G} p_s(\mathbf{x}_i|j; \mathbf{\Theta}^{(t)}, h)^{\beta}} \\ &= \frac{\exp(\beta \sum_{l=1}^{G} h_{kl} \log r_l(\mathbf{x}_i; \boldsymbol{\theta}_l^{(t)}))}{\sum_{j=1}^{G} \exp(\beta \sum_{l=1}^{G} h_{jl} \log r_l(\mathbf{x}_i; \boldsymbol{\theta}_l^{(t)}))}.\end{aligned} \tag{4.20}$$

Then, the auxiliary function to be minimized is

$$U_{s\beta}(\boldsymbol{\Theta};\boldsymbol{\Theta}^{(t)}) = -\sum_{i=1}^{N}\sum_{k=1}^{G}\tau_{k|i}^{(t)}\log p_s(\mathbf{x}_i,k;\boldsymbol{\Theta},h), \tag{4.21}$$

and the re-estimation formulae for the mean vectors and covariance matrices are

$$\boldsymbol{\mu}_l^{(t+1)} = \frac{\sum_{i=1}^{N}(\sum_{k=1}^{G}\tau_{k|i}^{(t)}h_{kl})\mathbf{x}_i}{\sum_{i=1}^{N}(\sum_{k=1}^{G}\tau_{k|i}^{(t)}h_{kl})}, \tag{4.22}$$

$$\boldsymbol{\Sigma}_l^{(t+1)} = \frac{\sum_{i=1}^{N}(\sum_{k=1}^{G}\tau_{k|i}^{(t)}h_{kl})(\mathbf{x}_i-\boldsymbol{\mu}_l^{(t+1)})(\mathbf{x}_i-\boldsymbol{\mu}_l^{(t+1)})^T}{\sum_{i=1}^{N}(\sum_{k=1}^{G}\tau_{k|i}^{(t)}h_{kl})} \tag{4.23}$$

for $l = 1, 2, \cdots, G$.

Note that the re-estimation formulae for SODAEM are the same as those for SOEM, except that $\gamma_{k|i}^{(t)}$ is replaced by $\tau_{k|i}^{(t)}$. $1/\beta$ corresponds to the temperature that controls the annealing process, in which a high temperature is applied initially. Then, the system is cooled down by gradually reducing the temperature. When $1/\beta \rightarrow 1$, the SODAEM algorithm becomes the SOEM algorithm; however, when $1/\beta \rightarrow 0$, it is equivalent to the SOCEM algorithm. In other words, SODAEM can be viewed as a deterministic annealing variant of SOEM and SOCEM. We summarize SODAEM in Algorithm 5.

By considering certain cases and approximations of SODAEM, SOEM, and SOCEM, we summarize the family of EM-based approaches for Gaussian model-based clustering discussed in this section in Figure 4.4. Both EM under the mixture-likelihood criterion and CEM under the classification-likelihood criterion are widely used model-based data clustering methods. SOEM (SOCEM) can be applied instead of EM (CEM) in model-based clustering if we want to preserve the spatial relationships between the resulting data clusters on a network. Since SODAEM is a DA variant of SOEM and SOCEM, it can be applied in model-based data clustering under both mixture-likelihood and classification-likelihood criteria.

4.4.1 Computational cost

Comparing Eqs. (4.22)-(4.23) to Eqs. (4.18)-(4.19), we can see that SODAEM and SOEM have similar computational costs in each learning iteration.

4.5 Relation to other algorithms

In this section, we explore the differences and relations between the proposed algorithms and other related algorithms.

Algorithm 5 The SODAEM algorithm

```
Require: $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$: the input data set;
  $\sigma$: the $\sigma$ value for $h_{kl}$ in Eq. (2.4);
  $\eta$: the annealing factor ($\eta > 1$);
  $\beta_{ini}$: the initial $\beta$ value;
  $\beta_{max}$: the target $\beta$ value;
  $\mathbf{\Theta}^{(0)} = \{\boldsymbol{\theta}_1^{(0)}, \boldsymbol{\theta}_2^{(0)}, \cdots, \boldsymbol{\theta}_G^{(0)}\}$: the initial reference models, where $\boldsymbol{\theta}_l^{(0)} = \{\boldsymbol{\mu}_l^{(0)}, \mathbf{\Sigma}_l^{(0)}\}$ are
  the initial mean vector and covariance matrix of the $l$th Gaussian component
Ensure: $\hat{\mathbf{\Theta}} = \{\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_2, \cdots, \hat{\boldsymbol{\theta}}_G\}$: the estimated parameter set, where $\hat{\boldsymbol{\theta}}_l = \{\hat{\boldsymbol{\mu}}_l, \hat{\mathbf{\Sigma}}_l\}$ are
  the estimates mean vector and covariance matrix of the $l$th Gaussian component

  Begin
    1. $\hat{\mathbf{\Theta}} \leftarrow \mathbf{\Theta}^{(0)}$; $\beta \leftarrow \beta_{ini}$;
    2. create the lookup table for $h_{kl}$;
    3. //EM:
       repeat
         E-step: for $i \leftarrow 1, 2, \cdots, N$ and $k \leftarrow 1, 2, \cdots, G$, compute $\tau_{k|i}^{(t)}$ in Eq. (4.20)
                 using $\hat{\mathbf{\Theta}}$;
         M-step: for $l \leftarrow 1, 2, \cdots, G$, update $\hat{\boldsymbol{\mu}}_l$ and $\hat{\mathbf{\Sigma}}_l$ with Eqs. (4.22)-(4.23);
       until the convergence condition is met
    4. if ($\beta = \beta_{max}$)
         goto End;
       $\beta \leftarrow \eta\beta$;
       if ($\beta > \beta_{max}$)
         $\beta \leftarrow \beta_{max}$;
       goto 3.;
  End
```

4.5.1 For SOCEM

In [93], Ambroise and Govaert proposed a topology preserving EM (TPEM) algorithm that introduces topological constraints in the CEM algorithm. If Kohonen's winner selection strategy is applied, SOCEM is equivalent to TPEM whose mixture weights are equally fixed. In SOCEM, the covariance matrix of a Gaussian component, $\mathbf{\Sigma}_l$, can have different parameterizations for different geometric interpretations [10]. When $\mathbf{\Sigma}_l = \lambda\mathbf{I}$ for $l = 1, 2, \cdots, G$ (where λ is a small positive constant and $\mathbf{I}$ denotes the identity matrix), the clusters are spherical and of equal volume. In this case, the SOCEM algorithm is equivalent to the TVQ algorithm in [55], which was developed for noisy vector quantization. It is also equivalent to the batch SOM learning algorithm described in [53], which employs an energy function in the learning phase of a SOM. However, SOCEM was developed from a different perspective. We consider the learning of a PbSOM as a model-based clustering process. By this perspective, a coupling-likelihood mixture model is developed first, and an objective function is then formulated based on the classification likelihood criterion. Moreover, the connection between the coupling-likelihood mixture model and the Gaussian mixture model helps interpret SOCEM as a topology-constrained DA variant of the CEM algorithm for GMM.

4.5.2 For SOEM and SODAEM

In SODAEM, when $\mathbf{\Sigma}_l = \lambda \mathbf{I}$ for $l = 1, 2, \cdots, G$, SODAEM is equivalent to the STVQ algorithm [56], which learns the parameters by maximizing their density function predicted by the maximum entropy principle. In STVQ, the inverse temperature, β, is the Lagrange multiplier introduced for the constrained optimization induced by the maximum entropy principle. Heskes [58] extends TVQ's cost function to an expected quantization error. Then, an objective function is obtained by weighting the quantization error with the inverse temperature β and pulsing it to an entropy term that introduces the annealing process. With the resulting objective function, Heskes obtained an algorithm identical to STVQ. The implementations for deterministic annealing in STVQ and Heskes' algorithm can also be found in [94, 95], where the DA is applied for vector quantization.

SODAEM differs from Graepel *et al.*'s STVQ and Heskes' algorithm in the following ways. First, the deterministic annealing processes are implemented differently. SODAEM is a DAEM algorithm developed to learn the mixture models with a deterministic annealing process, which is implemented based on predicting the posterior distribution in the *E-step* using the maximum entropy principle. Second, the case of $\beta = 1$ was not well addressed in Graepel *et al.*'s and Heskes' papers. This may be because their original goal was to develop a DA learning for TVQ. When β is fixed at 1, however, SODAEM becomes the SOEM algorithm. Moreover, the connection between the proposed coupling-likelihood mixture model and the Gaussian mixture model helps interpret SOEM as a topology-constrained DA variant of the EM algorithm for GMM.

4.6 Experiments on organizing property and data clustering

4.6.1 Experiments on organizing property

Data set description: We conducted experiments on two types of data: a synthetic data set and a real-world data set. The synthetic data set consisted of 500 points uniformly distributed in a unit square. For the real-world data set, we used the training set of class '0' in the "Pen-Based Recognition of Handwritten Digits" database (denoted as PenRecDigits_C0) in the UCI Machine Learning Database Repository [96]. The data set consists of 802 16-dimensional vectors. To demonstrate the map-learning process, we used the first two dimensions of the feature vectors as data for simulations. As a preprocessing step, we scaled down each element of the vectors in PenRecDigits_C0 to 1/100 of its original value to avoid numerical traps.

Experiment setup: In the experiments, an 8 × 8 equally spaced square lattice in a unit square was used as the structure of the SOM network. For the neighborhood function, we used the Gaussian kernel h_{kl} in Eq. (2.4).

We evaluated SOCEM, SOEM, SODAEM, and KohonenGaussian (Kohonen's batch algorithm that uses Gaussian reference models) in 20 independent random initialization trials and two setups for σ in h_{kl}. For each trial, data samples were randomly selected from the data set as the initial mean vectors, $\mu_1^{(0)}$, $\mu_2^{(0)}$, $\cdots$, $\mu_G^{(0)}$, of the reference models, which were multivariate Gaussians with full covariance matrices. The initial covariance matrix $\Sigma_l^{(0)}$ was set as $\rho_l\mathbf{I}$, where ρ_l=$\min_{k\neq l}\{\|\mu_l^{(0)} - \mu_k^{(0)}\|\}$, for $l = 1, 2, \cdots, G$. To avoid the singularity problem, we applied the variance limiting step to the covariance matrices during the learning process. If the value of any element of the covariance matrix was less than 0.001, it was set at 0.001.

4.6.1.1 Results on the synthetic data

We first demonstrate the map-learning processes of SOCEM, SOEM, and SODAEM using one of the 20 random initializations by showing the configurations of the Gaussian means on the maps, and then summarize the overall results of all the initializations.

Simulations using SOCEM: Figure 4.5 shows two simulations using the SOCEM algorithm. In the first simulation, SOCEM is run with the random initialization in Figure 4.5 (a) and a fixed σ of 0.15 in h_{kl}. As shown in Figure 4.5 (b), the algorithm's learning converges to an unordered map. In the second simulation, SOCEM starts with the same random initialization as that in Figure 4.5 (a), but with a larger σ of 0.6. When it converges at the current σ value, σ is reduced by 0.15. Then, the algorithm is applied again with the new σ value and the reference models obtained in the previous phase. This process continues until SOCEM converges at $\sigma = 0.15$. Figures 4.5 (c), (d), (e), and (f) depict the maps obtained when σ =0.6, 0.45, 0.3, and 0.15, respectively. We can explain the second simulation in terms of *annealing* (*cf.* Section 4.2.1): When using SOCEM, we start with a larger σ value (a higher temperature) so that the objective function is simple enough to be optimized. Then, we obtain the target map configuration by gradually reducing the value of σ (the temperature). Though the reduction in σ produces a more complex objective function for optimization, SOCEM can still learn well because the reference models obtained at the larger σ value provide a sound initialization for the next learning phase at the smaller σ value.

Simulations using SOEM: We conducted two similar simulations using the SOEM algorithm. In the first simulation, SOEM was run with the random initialization in Figure

4.6 (a) (the same as that in Figure 4.5 (a)) and a fixed σ of 0.15. As shown in Figure 4.6 (b), the learning of SOEM converged to an unordered map. In the second simulation, SOEM started with the random initialization in Figure 4.6 (a) and a larger σ of 0.6. Then, the value of σ was gradually reduced to 0.15 in 0.15 decrements. Figures 4.6 (c), (d), (e), and (f) depict the maps obtained when SOEM converges at σ =0.6, 0.45, 0.3, and 0.15, respectively. Similar to SOCEM, we can interpret the reduction of σ in SOEM as an annealing process (*cf.* Section 4.3.1), which overcomes the initialization issue. Comparing Figures 4.6 (c)-(d) to Figures 4.5 (c)-(d), we observe that the map obtained by SOEM is more concentrated than that obtained by SOCEM for the same σ value. This may be because SOEM learns the map in a more global manner than SOCEM, as noted in Section 4.3. In other words, each data sample contributes to all the neurons in a more global manner in SOEM than in SOCEM.

Simulations using SODAEM: Figure 4.7 depicts the simulations using the SODAEM algorithm with the same random initialization as that in Figure 4.5 (a) and Figure 4.6 (a). The value of σ is also fixed at 0.15, and the initial value of β is set to 0.16. When SODAEM converges at a β value, it is applied again with β^{new}=$\beta \times 1.6$ and the reference models obtained in the previous phase. We stop the learning process at $\beta = 17.592$. In our experience, it is appropriate to set the maximum value of β within the range 10 to 20 for practical applications. When $\beta = 0.16$, the temperature is high enough to ensure a smooth objective function. Therefore, according to the parameter update rules of SODAEM, the reference models form a compact ordered map via lateral interactions near the center of the data samples, even though the neighborhood size is small ($\sigma = 0.15$ in this case). When $\beta = 1.04$ and 17.592, SODAEM is almost equivalent to SOEM and SOCEM, respectively. In these two cases, SODAEM converges to the ordered maps in Figure 4.7 (f) and Figure 4.7 (i), respectively. However, as shown in Figures 4.5 (a)-(b) and Figures 4.6 (a)-(b), SOCEM and SOEM do not converge to an ordered map when $\sigma = 0.15$, which demonstrates that the annealing process of SODAEM overcomes the initialization problem of SOCEM and SOEM when $\sigma = 0.15$. Note that SODAEM may not be able to obtain any ordered map during the annealing process if the value of σ is too small to form an ordered map at a small β value.

Discussion: The experiment results obtained by the three proposed algorithms and KohonenGaussian for the 20 random initializations are summarized in Table 4.2. Several conclusions can be drawn from the results. First, SOEM often converges to an ordered map even at a small, fixed σ value ($\sigma = 0.15$ in the experiments); but KohonenGaussian and SOCEM seldom do so. This may be because SOEM learns the map in a more global way, as noted in Section 4.3; hence, it is less sensitive to the initialization of the parameters when σ is small. The results for KohonenGaussian and SOCEM are similar. This may be because they only differ in the winner selection strategy. Second, the initialization issue of KohonenGaussian, SOCEM and SOEM can be overcome by using a larger σ value

Table 4.2: Results of simulations using KohonenGaussian, SOCEM, SOEM, and SODAEM in 20 independent random initialization trials on the synthetic data. The algorithms were run with two setups for σ in h_{kl}. When $\sigma = 0.15$, KohonenGaussian succeeded in converging to an ordered map in one random initialization case (S:1), but failed in the remaining cases (F:19).

Setup for σ	$\sigma = 0.15$	$\sigma = 0.6$ initially, and is reduced to 0.15 in 0.15 decrements
KohonenGaussian	S:1	S:20
	F:19	F:0
SOCEM	S:1	S:20
	F:19	F:0
SOEM	S:15	S:20
	F:5	F:0
SODAEM	S:20	-
	F:0	-

(0.6 in the experiments) initially, and then gradually reducing the value to the target σ value (0.15 in the experiments). The reduction of σ can be interpreted as an annealing process (*cf.* Section 4.2.1, Section 4.2.2, and Section 4.3.1). Third, the experiment results show that SODAEM overcomes the initialization issue of SOCEM and SOEM at a small σ value (0.15 in the experiments) using the annealing process, which is controlled by the temperature parameter β.

4.6.1.2 Results on PenRecDigits_C0

We also conducted experiments on real-world data using the setups for the neighborhood function described in Section 4.6.1.1. Table 4.3 summarizes the results obtained by the four PbSOM learning algorithms. From the results, we can draw the same conclusions as those made for the experiment results on the synthetic data. Figures 4.8, 4.9, and 4.10 demonstrate, respectively, the map-learning processes of SOCEM, SOEM, and SODAEM using one of the 20 random initializations. Comparing Figures 4.8, 4.9, and 4.10 , we observe that these three algorithms obtain rather different results. SOCEM and SOEM usually obtain different maps because they learn the maps based on different clustering criteria (classification-likelihood vs. mixture-likelihood). SODAEM and SOEM (or SOCEM) usually obtain different results because SODAEM's annealing is achieved by increasing the β value, while SOEM's (or SOCEM's) annealing is achieved by decreasing the σ value. Comparing Figures 4.9 (f) and 4.10 (f), although SODAEM becomes equivalent to SOEM when the value of β is increased to 1.04, their search paths on the objective function surface are different because they have rather different seed models (Figure 4.10 (e) vs. Figure 4.9 (e)). Therefore, they converge to different local maxima of the objective function and obtain different maps. Likewise, although SODAEM becomes equivalent to SOCEM when the value of β is increased to 17.592, they converge to different local maxima of the objective function and obtain different maps (Figure 4.10 (i) vs. Figure 4.8

Table 4.3: Results of simulations using KohonenGaussian, SOCEM, SOEM, and SODAEM in 20 independent random initialization trials on PenRecDigits_C0. The algorithms were run with two setups for σ in h_{kl}. When $\sigma = 0.15$, KohonenGaussian succeeded in converging to an ordered map in one random initialization case (S:1), but failed in the remaining cases (F:19).

Setup for σ	$\sigma = 0.15$	$\sigma = 0.6$ initially, and is reduced to 0.15 in 0.15 decrements
KohonenGaussian	S:1	S:20
	F:19	F:0
SOCEM	S:2	S:20
	F:18	F:0
SOEM	S:14	S:20
	F:6	F:0
SODAEM	S:20	-
	F:0	-

(f)).

4.6.2 Experiments to evaluate the performance of data clustering

Data set description: In this section, we evaluate the data clustering performance of the proposed PbSOM algorithms on two data sets from the UCI Machine Learning Database Repository [96]: the test set of the "image segmentation" database (denoted as ImgSeg), which consists of 2,100 19-dimensional feature vectors; and the Ecoli data set (denoted as Ecoli), which consists of 336 8-dimensional feature vectors. Here, we used the full vector, rather than only two dimensions, in the experiments. As a pre-processing step, we scaled down each element of the data vectors in ImgSeg to 1/100 of its original value to avoid numerical traps.

Experiment setup: To avoid the singularity problem that often occurs when using CEM or EM to learn full covariance GMMs, we used diagonal covariance Gaussians in the experiments. We also applied the variance limiting step, in which the minimum value for a variance was set at 0.01.

For the PbSOM learning algorithms, we used five configurations for the network structure; they are 3×3, 4×4, 5×5, 6×6, and 7×7 lattices equally spaced in a unit square. We used the Gaussian kernel h_{kl} in Eq. (2.4) as the neighborhood function.

To avoid ambiguity, when the DAEM and SODAEM algorithms are applied in data clustering based on the classification-likelihood criterion, they are denoted as DAEM_C and SODAEM_C; and they are denoted as DAEM_M and SODAEM_M when applied in data clustering based on the mixture-likelihood criterion.

All the algorithms discussed here were run with random initializations generated in the same way described in Section 4.6.1.

4.6.2.1 Results on ImgSeg by using SOCEM and SODAEM_C

First, we evaluated the data clustering performance of KohonenGaussian, SOCEM, and SODAEM_C in terms of the classification log-likelihood defined in Eq. (2.20). The performance was compared with that of CEM and DAEM_C. The setting for each algorithm was as follows:

- DAEM_C: The value of β was set at 0.2 initially, and increased to 10 by the formula $\beta^{new} = \beta \times 1.2$.
- SOCEM: The value of σ in h_{kl} was set at 0.7 initially, and reduced to 0 (i.e., $h_{kl} = \delta_{kl}$) in 0.02 decrements.
- SODAEM_C: Both the values of β and σ in h_{kl} were set at 0.2 initially. To perform data clustering using the classification-likelihood criterion, the value of β was increased to 10 by the formula $\beta^{new} = \beta \times 1.2$ first; then, the value of σ was reduced to 0 in 0.02 decrements.
- KohonenGaussian: The value of σ in h_{kl} was set at 0.7 initially, and reduced to 0 in 0.02 decrements every 30 learning iterations[4].

We ran the algorithms except CEM with 20 independent trials using 9, 16, 25, 36, 49 Gaussian components. To conduct a fair comparison of CEM and the proposed approaches, we ran CEM many trials till the accumulated execution time was close to that of one SOCEM trial. The mean and standard deviations (error bars) of the classification log-likelihood values over the trials for each algorithm and the best results of CEM (denoted as CEM-best) are shown in Figure 4.11. Note that, in the figure, we slightly separate the results associated with a specific Gaussian component number in order to distinguish between them. From the figure, we observe that the clustering performance of SOCEM, SODAEM_C, and KohonenGaussian is close to that of DAEM_C. Moreover, they obtain larger and more stable classification log-likelihoods than CEM. These results are rational since SOCEM is a topology-constrained DA variant of the CEM algorithm, and SODAEM_C is an annealing variant of SOCEM with the settings for β and σ here.

4.6.2.2 Results on ImgSeg by using SOEM and SODAEM_M

First, we evaluated the performance of SOEM and SODAEM_M in learning a Gaussian mixture model with equal mixture weights. The objective function was the log mixture-likelihood function in Eq. (2.7) with equal mixture weights. We compared the performance with that of EM and DAEM_M. The setting for each algorithm was as follows:

[4] In our implementation for SOCEM, SOEM, and SODAEM, the phase transition occurs when the likelihood increase is below a threshold or the number of learning iterations exceeds 30 in the current phase. However, KohonenGaussian does not have the convergence property; thus, we ran 30 iterations for each phase of the algorithm.

- DAEM_M: The value of β was set at 0.2 initially, and increased to 1 by the formula $\beta^{new} = \beta \times 1.2$.

- SOEM: The value of σ in h_{kl} was set at 0.6 initially, and reduced to 0 (i.e., $h_{kl} = \delta_{kl}$) in 0.02 decrements.

- SODAEM_M: Both the values of β and σ in h_{kl} were set at 0.2 initially. To perform data clustering using the mixture-likelihood criterion, the value of β was increased to 1 by the formula $\beta^{new} = \beta \times 1.2$ first; then, the value of σ was reduced to 0 in 0.02 decrements.

We ran DAEM_M, SOEM, and SODAEM_M with 20 independent random initialization trials. Similar to the experiments on CEM, we ran EM many trials till the accumulated execution time was close to that of one SOEM trial. The mean and standard deviations (error bars) of the log mixture-likelihood values over the trials for each algorithm and the best results of EM (denoted as EM-best) are shown in Figure 4.12. From the figure, it is clear that DAEM_M, SOEM, and SODAEM_M achieve similar performance. Moreover, they obtain larger and more stable log mixture-likelihoods than EM. The results are rational since SOEM is a topology-constrained DA variant of the EM algorithm, and SODAEM_M is an annealing variant of SOEM with the settings for β and σ here.

4.6.2.3 Results on Ecoli

We conducted experiments on Ecoli using the algorithms applied to ImgSeg in Section 4.6.2.1 and Section 4.6.2.2. Figures 4.13 (a) and (b) show the data clustering performance of each algorithm in terms of the classification log-likelihood and the log mixture-likelihood, respectively. Similar to the results on ImgSeg, the PbSOM learning algorithms also achieve decent data clustering performance on Ecoli.

4.7 Application of SOCEM, SOEM, and SODAEM to data visualization and analysis

In this section, we apply the SOCEM, SOEM, SODAEM algorithms to data visualization and analysis. We applied the data sets described in Section 4.6.2, ImgSeg and Ecoli, for experiments. We also applied the same experiment setups for the PbSOM algorithms as those in Section 4.6.2.

4.7.1 Experiment results on ImgSeg by using SOCEM and SODAEM_C

To visualize the data samples and clusters on the network, each data sample was assigned to its winning reference model, and then randomly plotted within the neuron that associates to the reference model [35]. Here, the winner selection strategy for SODAEM_C was the same as that of SOCEM (i.e., the *C-step* of SOCEM). Figure 4.14 shows the projections of the data samples on 7 × 7 lattices obtained by different algorithms. The ImgSeg data set is comprised of seven classes, namely brickface: B, sky: S, foliage: F, cement: C, window: W, path: P, and grass: G; each class consists of 300 data samples. Figure 4.14 (a) depicts the initial mapping of the data obtained with a random initialization for the reference models. As we can see from the figure, the data clusters are randomly projected to the neurons (lattice nodes) and the network does not preserve the topological (spatial) relationships among the data samples and clusters. Figures. 4.14 (b)-(f) shows the results of the three PbSOM learning algorithms obtained with the random initialization in Figure 4.14 (a). We see that they can preserve the topological relationships among the data samples and clusters on the network. Moreover, it seems that the data samples of classes ‘S’, ‘G’, ‘P’, and ‘C’ are more distinguishable and well-grouped on the network than those of the other classes. In particular, from Figures 4.14 (b), (c) and (d), we see that only class ‘S’ is separated from the other classes with empty nodes; thus, we may infer that the separability between ‘S’ and the other classes is higher than that between the remaining classes.

For SOCEM, as shown in Figures 4.14 (c) and (d), the network contains less empty nodes at $\sigma = 0$ than at $\sigma = 0.06$. This may be because in the former case the lateral interactions have vanished, and thus the reference models are adapted to more fit the data distribution than the latter case. Comparing Figure 4.14 (b) to Figure 4.14 (d), we see that the data projection results of KohonenGaussian and SOCEM are rather different although they obtain similar classification log-likelihoods in Figure 4.11. However, we can draw similar observations from the two figures. For example, the data samples of class ‘S’ are more close to those of class ‘C’ and ‘P’ than those of class ‘G’. Figures 4.14 (e) and (f) show the results obtained by SODAEM_C. We see that the result in Figure 4.14 (f) is rather different from that in Figure 4.14 (d) although SODAEM_C has become equivalent to SOCEM when $\sigma = 0.2$. This may be because these two approaches search on the objective function surface along different paths and converge to different local maxima, as the explanation for the difference of Figures 4.9 (f) and 4.10 (f) in Section 4.6.1.2.

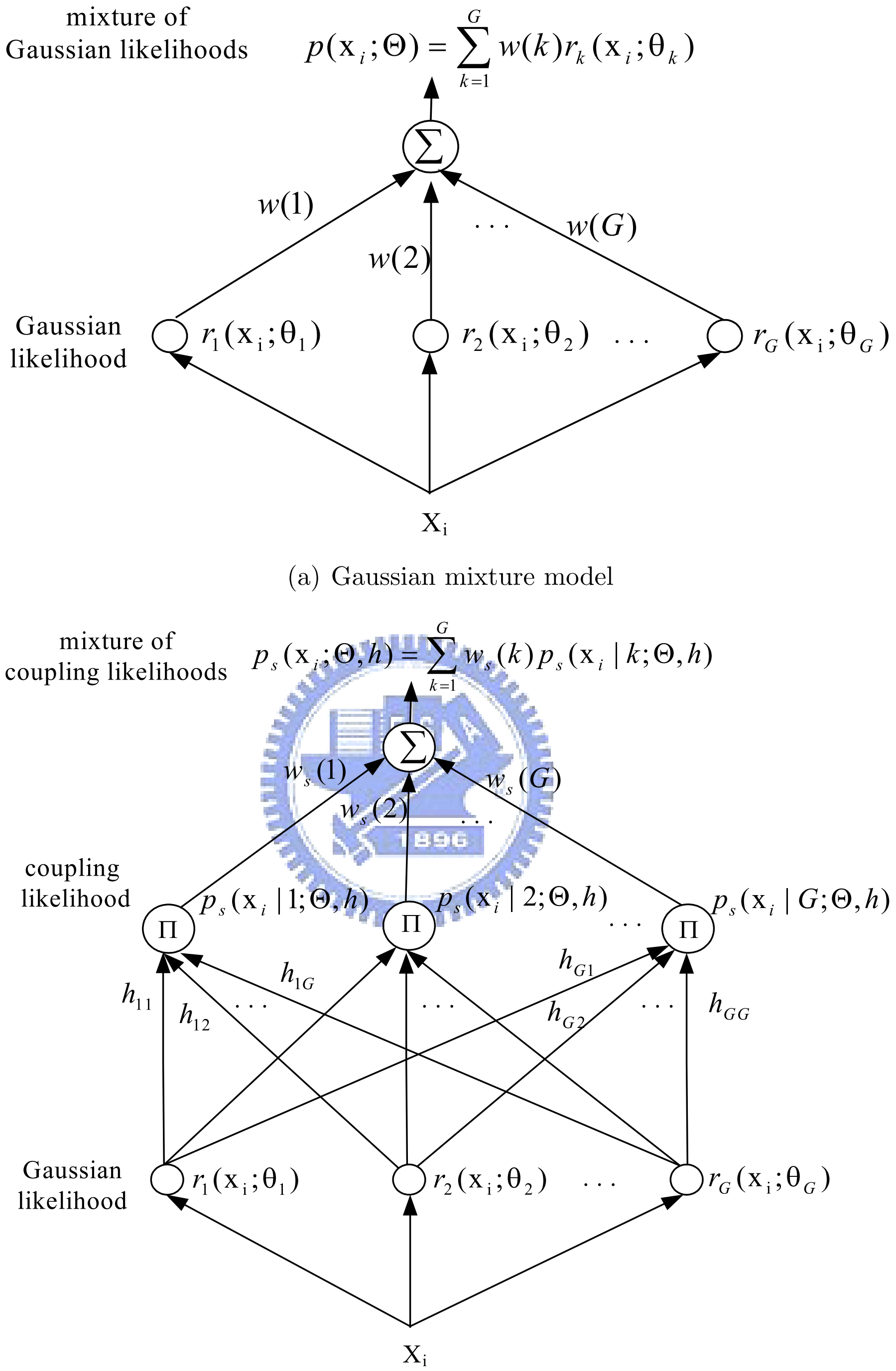
4.7.2 Experiment results on ImgSeg by using SOEM and SODAEM_M

We ran SOEM and SODAEM_M with a 7×7 lattice and the initial reference models used in Section 4.7.1 for evaluating SOCEM; therefore, the initial projection of the data was the same as that shown in Figure 4.14 (a). When clustering the data samples, each sample was assigned to its winning reference model using SOCEM's winner selection strategy. From Figure 4.15, we observe that these two algorithms can preserve topological relationships among data clusters (samples). Similar to the results revealed by Figure 4.14, data samples of classes 'S', 'G', 'P' and 'C' are more distinguishable than those of the other classes.

Comparing Figure 4.15 (b) to Figures 4.14 (b) and (d), it is clear that SOEM produces less empty nodes than KohonenGaussian and SOCEM when the value of σ is reduced to zero. It may be explained as follows. For KohonenGaussian and SOCEM, in the case of $\sigma = 0$, they become the CEM (K-means type) algorithm where each data sample only adapts its winner. However, when $\sigma = 0$, SOEM becomes the EM algorithm where each data sample adapts all the reference models according to their posterior probabilities; thus, the models are more adapted to fit the data than the models of the other two algorithms.

4.7.3 Experiment results on Ecoli

We conducted experiments on Ecoli using the algorithms applied to ImgSeg in Section 4.7.1 and Section 4.7.2. In Figure 4.16, for each algorithm we show the result at the σ value that the class separability can be best visualized on the network. The Ecoli data set is comprised of eight classes, namely cp: C, im: I, pp: P, imU: U, om: O, omL: M, imL: L, and imS: S. The numbers of data samples are 143, 77, 52, 35, 20, 5, 2, and 2, respectively. From the figure, we can see that topological relationships among data samples and clusters are preserved well and data classes can be roughly separated on the network.

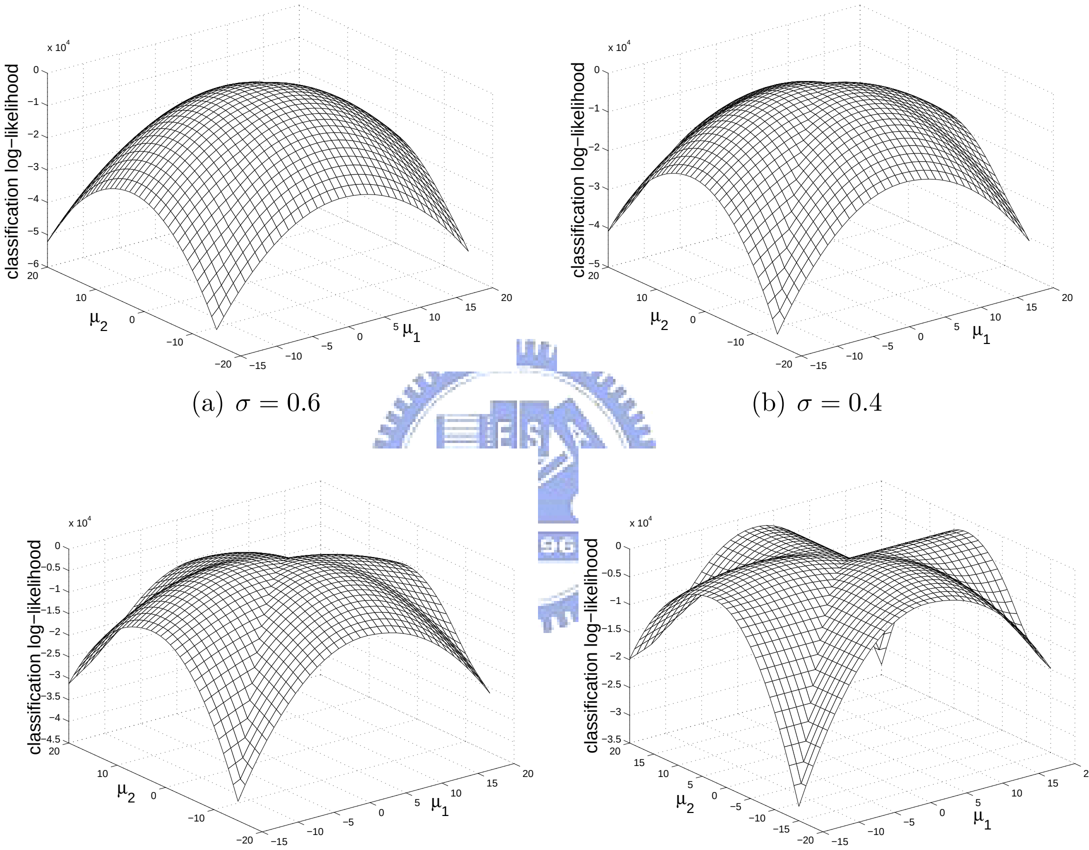

mixture of
Gaussian likelihoods
$p(\mathrm{x}_i;\Theta)=\sum_{k=1}^{G} w(k) r_k(\mathrm{x}_i;\theta_k)$
$\sum$
$w(1)$
$w(2)$
$w(G)$
. . .
Gaussian
likelihood
$r_1(\mathrm{x}_\mathrm{i};\theta_1)$
$r_2(\mathrm{x}_\mathrm{i};\theta_2)$
. . .
$r_G(\mathrm{x}_\mathrm{i};\theta_G)$
X_i

(a) Gaussian mixture model

mixture of
coupling likelihoods
$p_s(\mathrm{x}_i;\Theta,h)=\sum_{k=1}^{G} w_s(k) p_s(\mathrm{x}_i \mid k;\Theta,h)$
$\sum$
$w_s(1)$
$w_s(2)$
$w_s(G)$
. . .
coupling
likelihood
$p_s(\mathrm{x}_i \mid 1;\Theta,h)$
$p_s(\mathrm{x}_i \mid 2;\Theta,h)$
. . .
$p_s(\mathrm{x}_i \mid G;\Theta,h)$
Π
Π
Π
h_{11}
h_{12}
h_{1G}
h_{G1}
h_{G2}
h_{GG}
. . .
. . .
. . .
Gaussian
likelihood
$r_1(\mathrm{x}_\mathrm{i};\theta_1)$
$r_2(\mathrm{x}_\mathrm{i};\theta_2)$
. . .
$r_G(\mathrm{x}_\mathrm{i};\theta_G)$
X_i

(b) The proposed coupling-likelihood mixture model

Figure 4.1: (a) The network structure of a Gaussian mixture model, and (b) the proposed coupling-likelihood mixture model. Here, $r_l(\mathbf{x}_i;\boldsymbol{\theta}_l)$ denotes the multivariate Gaussian distribution described in Eq. (2.12).



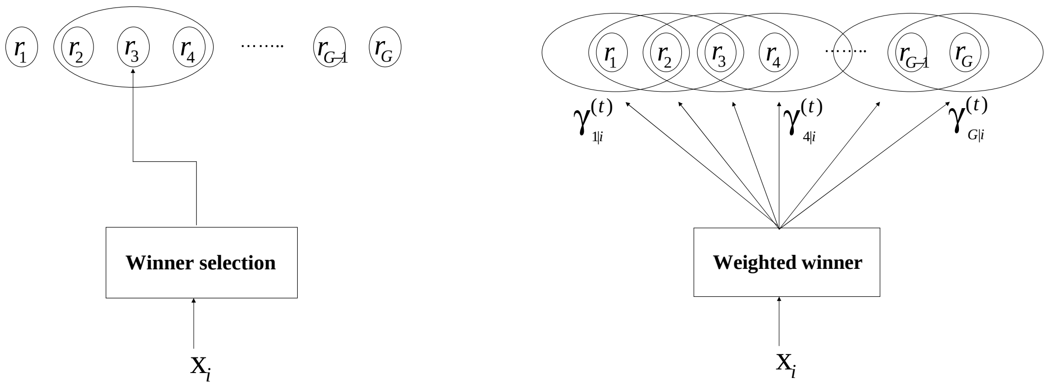
(a) $\sigma = 0.6$

(b) $\sigma = 0.4$

(c) $\sigma = 0.3$

(d) $\sigma = 0$ (i.e., $h_{kl} = \delta_{kl}$)

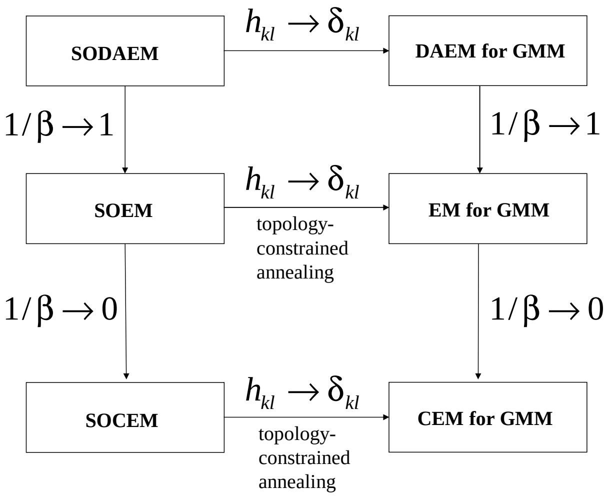
Figure 4.2: SOCEM's objective function becomes more complex with the reduction of neighborhood size (σ in h_{kl}).



(a) SOCEM

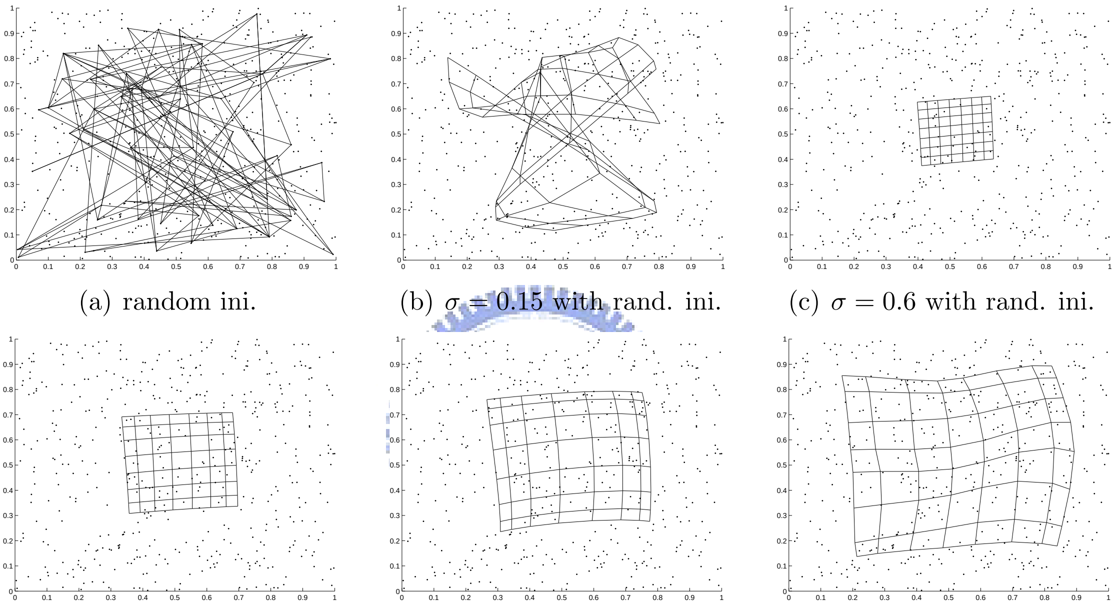
(b) SOEM

Figure 4.3: For each data sample $\mathbf{x}_i$, the adaptation of the reference models in SOCEM is restricted to the winning reference model and its neighborhood. However, in SOEM, the winner is relaxed to the weighted winners by the posterior probabilities $\gamma^{(t)}_{k|i}$, for $k = 1, 2, \cdots, G$. Each data sample $\mathbf{x}_i$ contributes proportionally to the adaptation of each reference model and its neighborhood according to the posterior probabilities.



SODAEM
$h_{kl} \rightarrow \delta_{kl}$
DAEM for GMM
$1/\beta \rightarrow 1$
$1/\beta \rightarrow 1$
SOEM
$h_{kl} \rightarrow \delta_{kl}$
topology-constrained annealing
EM for GMM
$1/\beta \rightarrow 0$
$1/\beta \rightarrow 0$
SOCEM
$h_{kl} \rightarrow \delta_{kl}$
topology-constrained annealing
CEM for GMM

Figure 4.4: The family of Gaussian model-based clustering algorithms derived from the SODAEM, SOEM and SOCEM algorithms. $\delta_{kl} = 1$ if $k = l$; otherwise, $\delta_{kl} = 0$.



(a) random ini.

(b) $\sigma = 0.15$ with rand. ini.

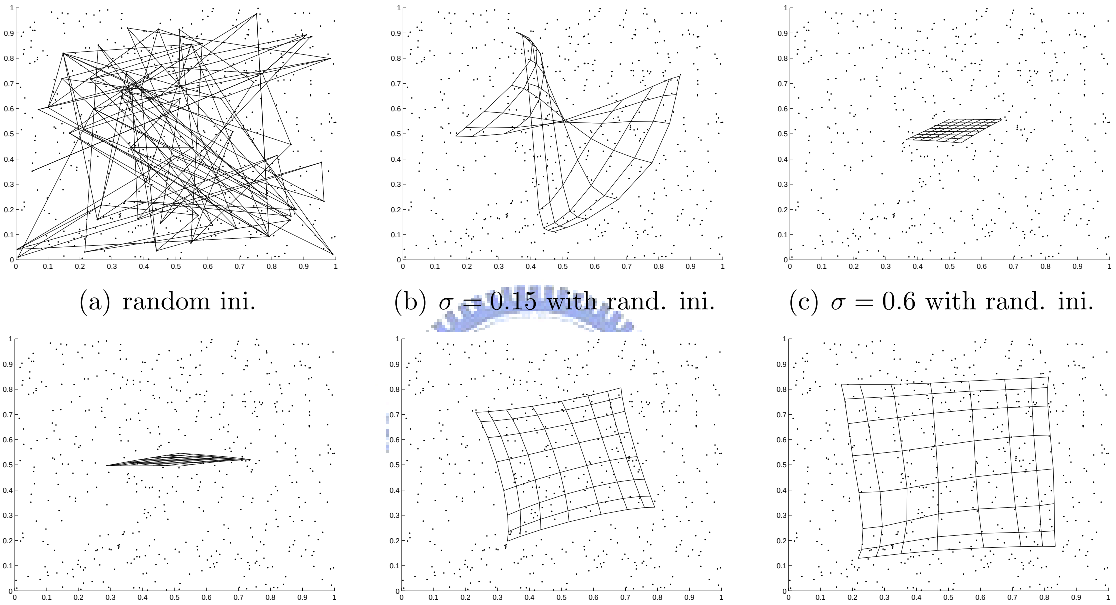
(c) $\sigma = 0.6$ with rand. ini.

(d) $\sigma = 0.45$

(e) $\sigma = 0.3$

(f) $\sigma = 0.15$

Figure 4.5: The map-learning process obtained by running the SOCEM algorithm on the synthetic data. Simulation 1 ((a)-(b)): When SOCEM is run with the random initialization in (a) and $\sigma = 0.15$, it converges to the unordered map in (b). Simulation 2 ((a) and (c)-(f)): SOCEM starts with $\sigma = 0.6$ and the random initialization in (a). Then, the value of σ is reduced to 0.15 in 0.15 decrements.



(a) random ini.

(b) $\sigma = 0.15$ with rand. ini.

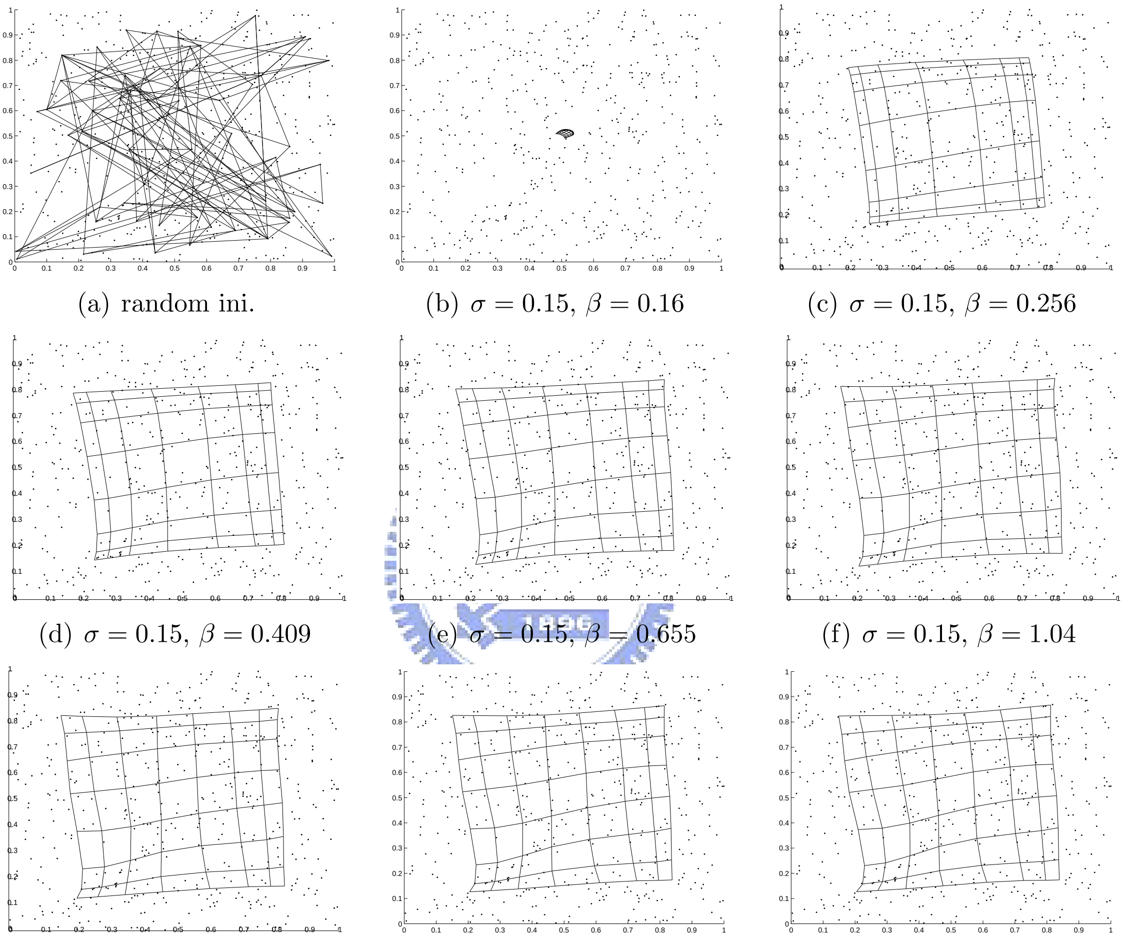
(c) $\sigma = 0.6$ with rand. ini.

(d) $\sigma = 0.45$

(e) $\sigma = 0.3$

(f) $\sigma = 0.15$

Figure 4.6: The map-learning process obtained by running the SOEM algorithm on the synthetic data. Simulation 1 ((a)-(b)): When SOEM is run with the random initialization in (a) and $\sigma = 0.15$, it converges to the unordered map in (b). Simulation 2 ((a) and (c)-(f)): SOEM starts with $\sigma = 0.6$ and the random initialization in (a). Then, the value of σ is reduced to 0.15 in 0.15 decrements.



(a) random ini.

(b) $\sigma = 0.15$, $\beta = 0.16$

(c) $\sigma = 0.15$, $\beta = 0.256$

(d) $\sigma = 0.15$, $\beta = 0.409$

(e) $\sigma = 0.15$, $\beta = 0.655$

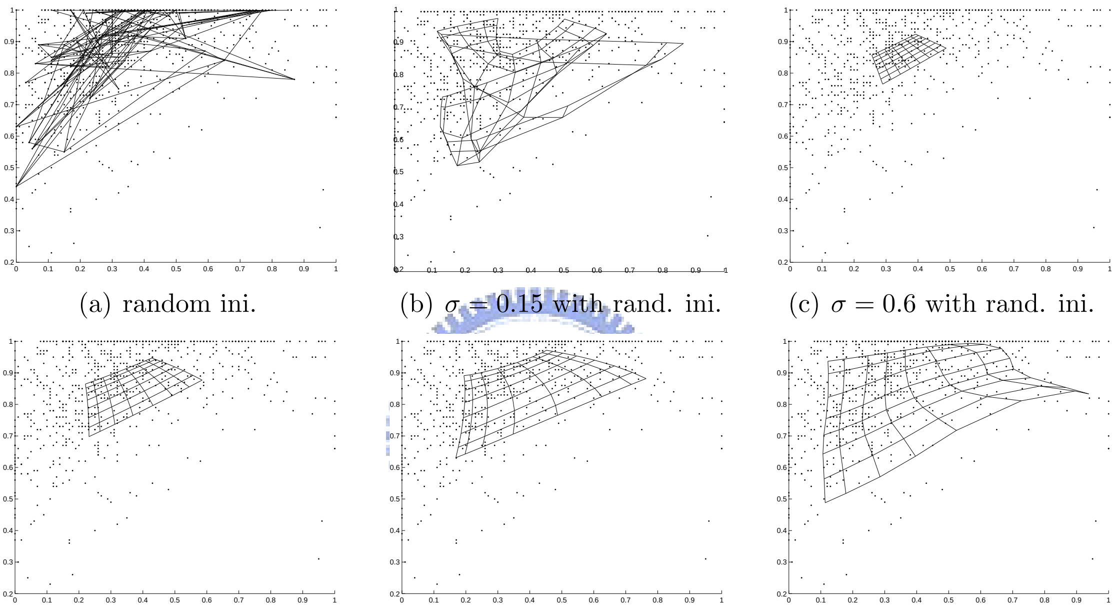
(f) $\sigma = 0.15$, $\beta = 1.04$

(g) $\sigma = 0.15$, $\beta = 2.68$

(h) $\sigma = 0.15$, $\beta = 6.871$

(i) $\sigma = 0.15$, $\beta = 17.592$

Figure 4.7: The map-learning process obtained by running the SODAEM algorithm on the synthetic data. The value of σ is fixed at 0.15, while value of β is initialized at 0.16 and increased in multiples of 1.6 up to 17.592.



(a) random ini.

(b) $\sigma = 0.15$ with rand. ini.

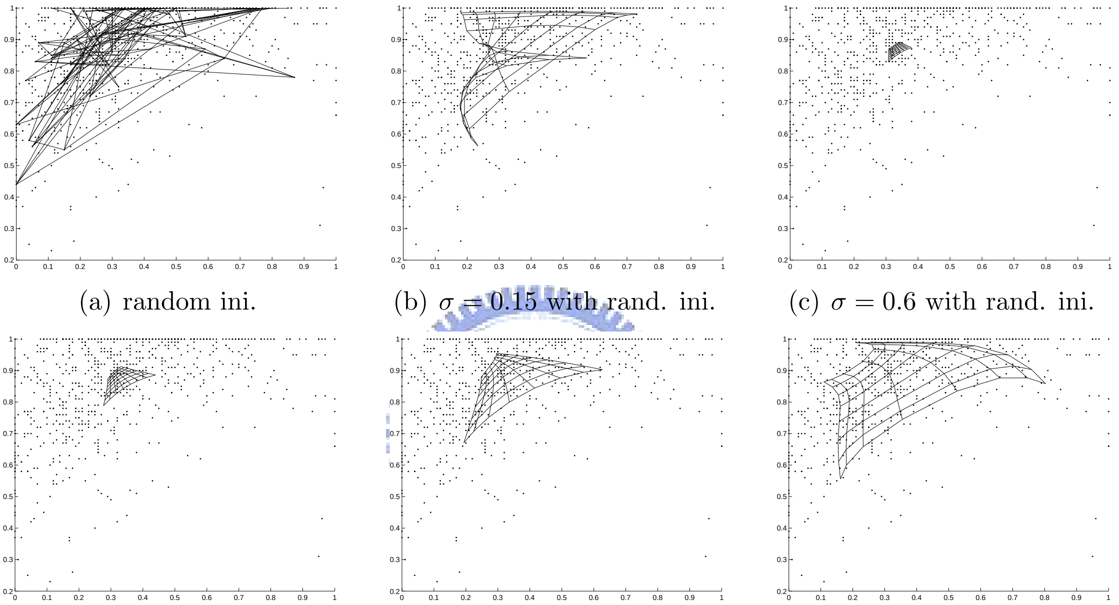
(c) $\sigma = 0.6$ with rand. ini.

(d) $\sigma = 0.45$

(e) $\sigma = 0.3$

(f) $\sigma = 0.15$

Figure 4.8: The map-learning process obtained by running the SOCEM algorithm on PenRecDigits_C0. Simulation 1 ((a)-(b)): When SOCEM is run with the random initialization in (a) and $\sigma = 0.15$, it converges to the unordered map in (b). Simulation 2 ((a) and (c)-(f)): SOCEM starts with $\sigma = 0.6$ and the random initialization in (a). Then, the value of σ is reduced to 0.15 in 0.15 decrements.



(a) random ini.

(b) $\sigma = 0.15$ with rand. ini.

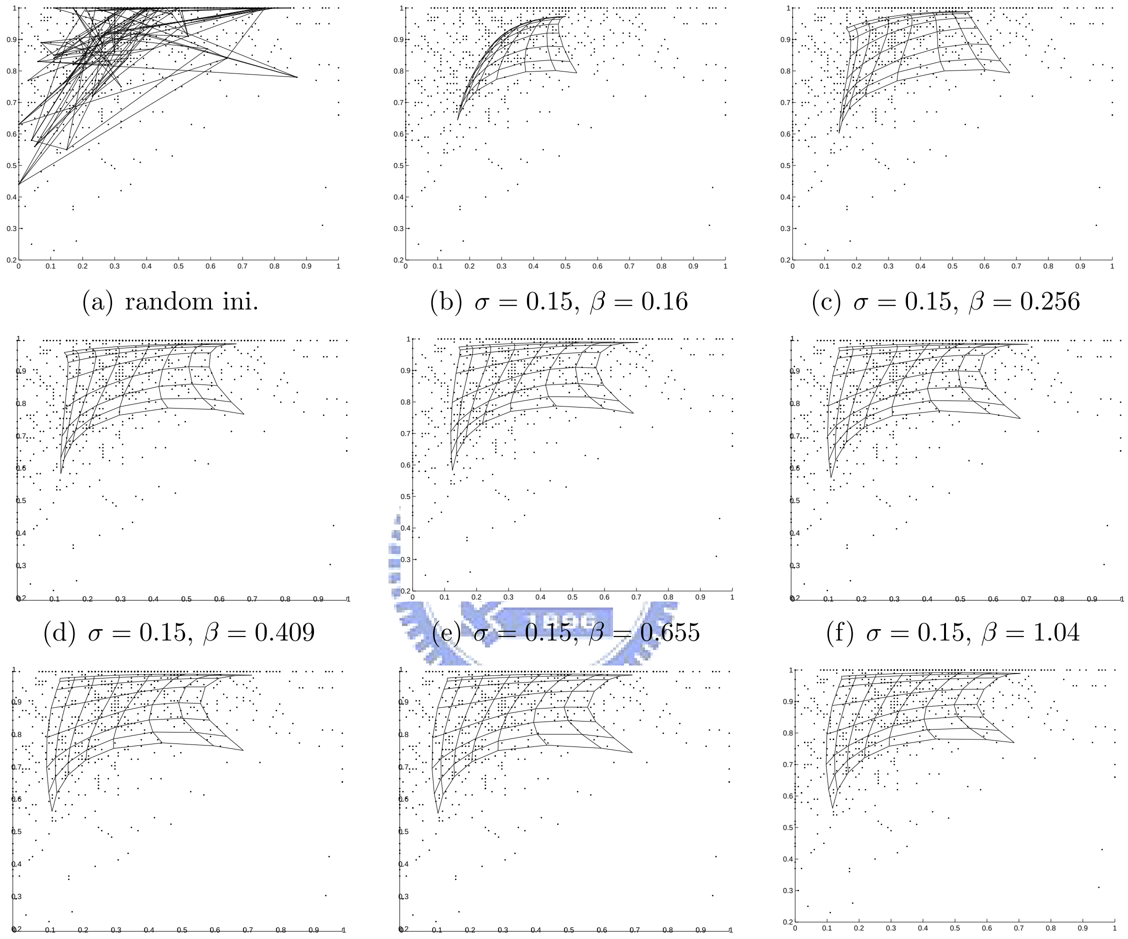
(c) $\sigma = 0.6$ with rand. ini.

(d) $\sigma = 0.45$

(e) $\sigma = 0.3$

(f) $\sigma = 0.15$

Figure 4.9: The map-learning process obtained by running the SOEM algorithm on PenRecDigits_C0. Simulation 1 ((a)-(b)): When SOEM is run with the random initialization in (a) and $\sigma = 0.15$, it converges to the unordered map in (b). Simulation 2 ((a) and (c)-(f)): SOEM starts with $\sigma = 0.6$ and the random initialization in (a). Then, the value of σ is reduced to 0.15 in 0.15 decrements.



(a) random ini. (b) $\sigma = 0.15$, $\beta = 0.16$ (c) $\sigma = 0.15$, $\beta = 0.256$

(d) $\sigma = 0.15$, $\beta = 0.409$ (e) $\sigma = 0.15$, $\beta = 0.655$ (f) $\sigma = 0.15$, $\beta = 1.04$

(g) $\sigma = 0.15$, $\beta = 2.68$ (h) $\sigma = 0.15$, $\beta = 6.871$ (i) $\sigma = 0.15$, $\beta = 17.592$

Figure 4.10: The map-learning process obtained by running the SODAEM algorithm on PenRecDigits_C0. The value of σ is fixed at 0.15, while value of β is initialized at 0.16 and increased in multiples of 1.6 up to 17.592.

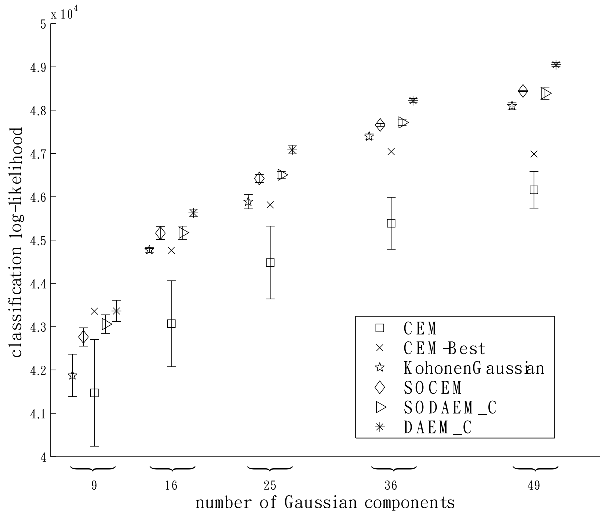


Figure 4.11: The data clustering performance of CEM, DAEM_C, SOCEM, SODAEM_C, and KohonenGaussian on ImgSeg in terms of the classification log-likelihood.

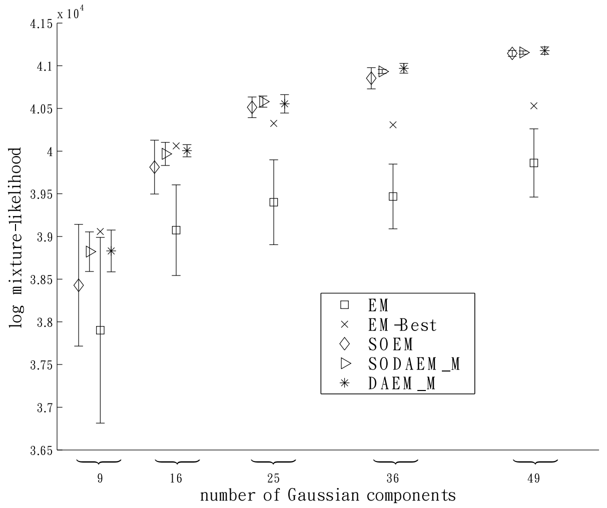
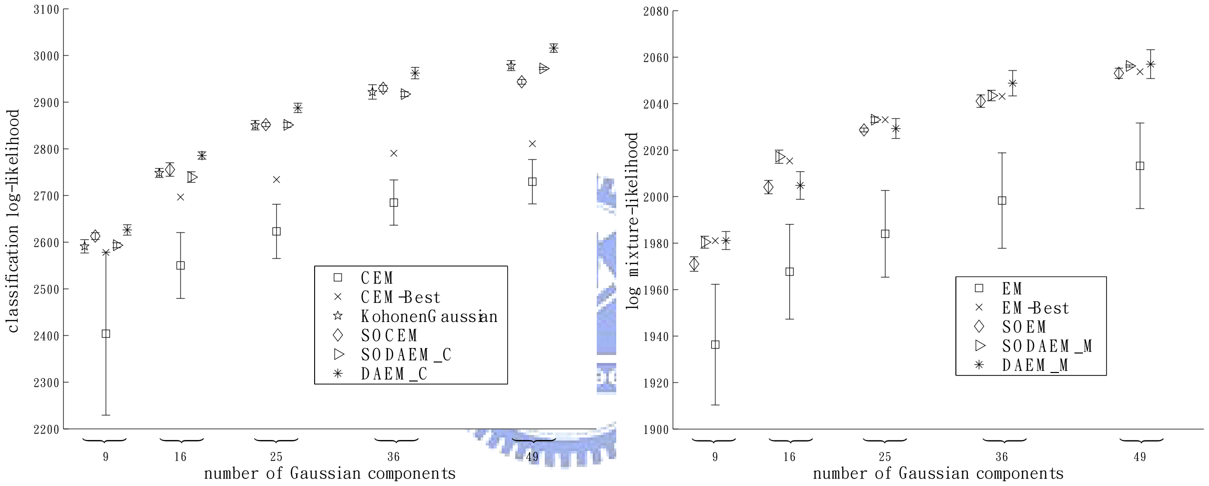
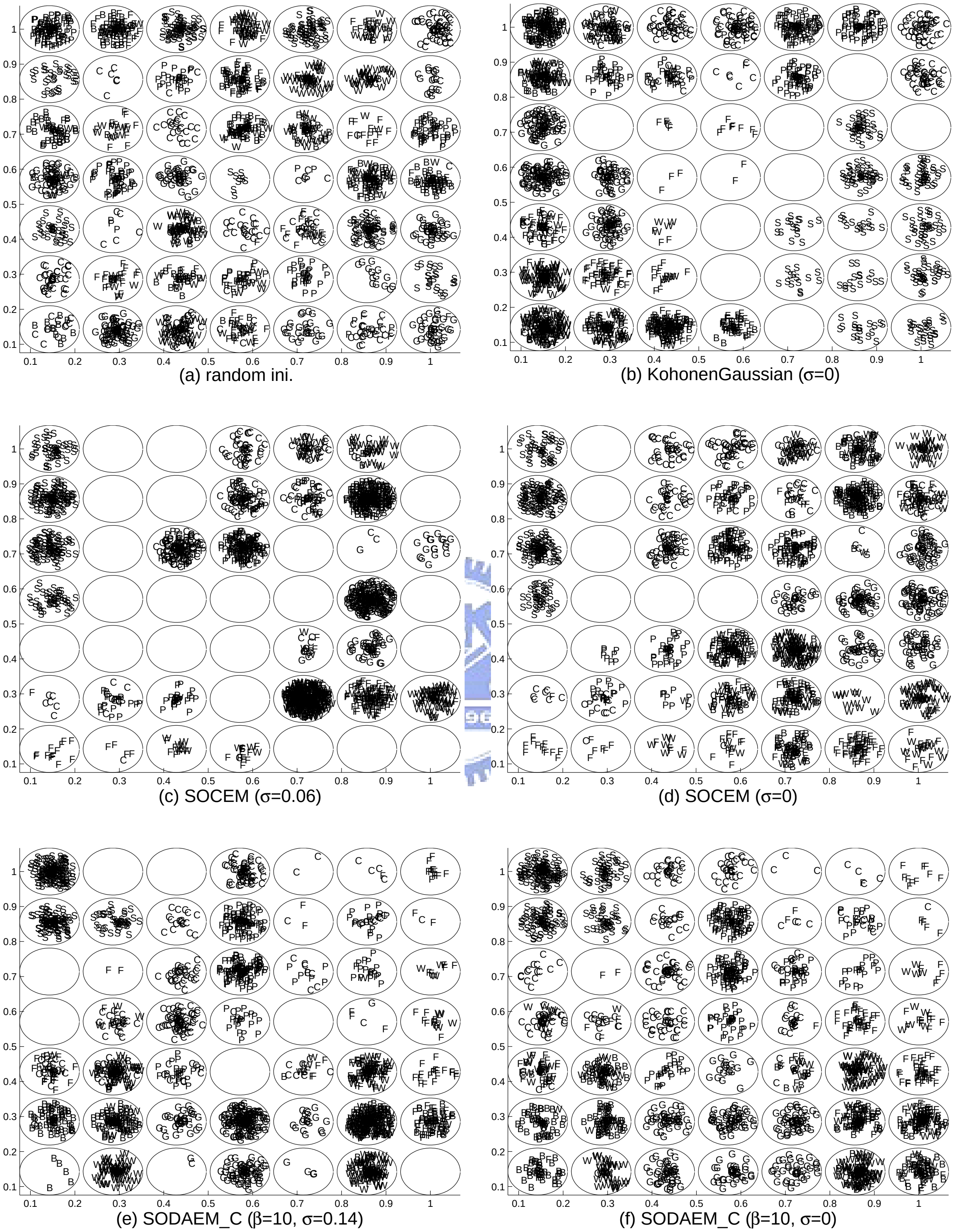


Figure 4.12: Learning a Gaussian mixture model by applying EM, DAEM_M, SOEM, and SODAEM_M to ImgSeg.



(a) (b)

Figure 4.13: The data clustering performance on Ecoli in terms of (a) the classification log-likelihood and (b) the log mixture-likelihood.



(a) random ini.

(b) KohonenGaussian (σ=0)

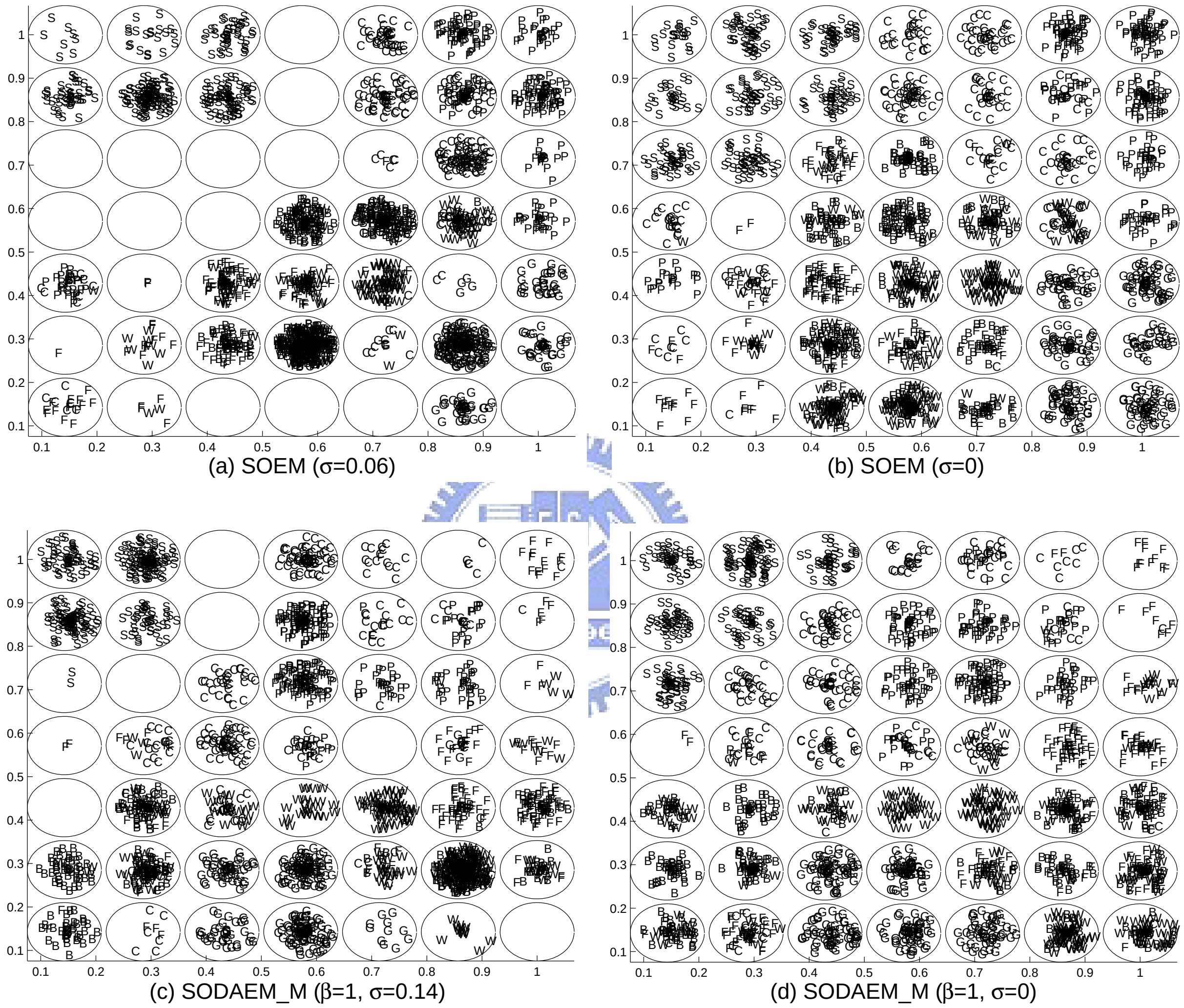
(c) SOCEM (σ=0.06)

(d) SOCEM (σ=0)

(e) SODAEM_C (β=10, σ=0.14)

(f) SODAEM_C (β=10, σ=0)

Figure 4.14: Data visualization for ImgSeg by running KohonenGaussian ((b)), SOCEM ((c), (d)), and SODAEM_C ((e), (f)) with the random initialization in (a). The network structure is a 7 × 7 equally spaced square lattice in a unit square.

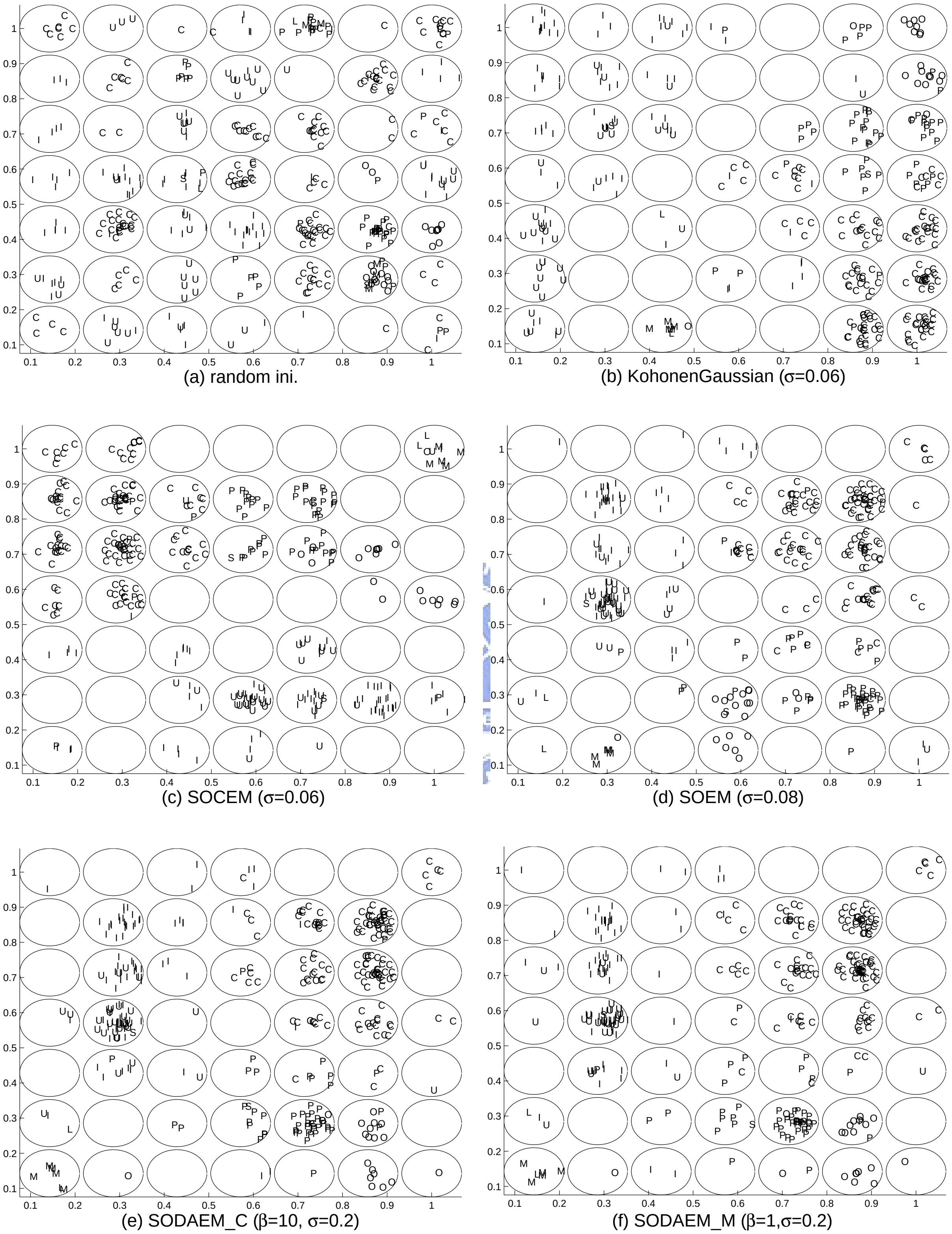

(a) SOEM (σ=0.06)

(b) SOEM (σ=0)

(c) SODAEM_M (β=1, σ=0.14)

(d) SODAEM_M (β=1, σ=0)

Figure 4.15: Data visualization for ImgSeg by running SOEM ((a), (b)) and SODAEM_M ((c), (d)) with the random initialization in Figure 4.14 (a). The network structure is a 7 × 7 equally spaced square lattice in a unit square.



(a) random ini.

(b) KohonenGaussian (σ=0.06)

(c) SOCEM (σ=0.06)

(d) SOEM (σ=0.08)

(e) SODAEM_C (β=10, σ=0.2)

(f) SODAEM_M (β=1,σ=0.2)

Figure 4.16: Data visualization for Ecoli by running (b) KohonenGaussian, (c) SOCEM, (d) SOEM, (e) SODAEM_C, and (f) SODAEM_M with the random initialization in (a). The network structure is a 7 × 7 equally spaced square lattice in a unit square.

Chapter 5

BIC-based audio segmentation using divide-and-conquer

The goal of audio segmentation is to detect acoustic changes in an audio stream, e.g., boundaries between two speakers or two environmental conditions. In the last decade, researchers in the speech processing community have put much effort on this problem for its potential applications to many speech and audio processing tasks, such as audio indexing [97], automatic transcription of audio recordings [98], speaker tracking [99], and speaker diarization [100]. Existing audio segmentation approaches generally fall into two categories, namely, distance-based segmentation [101, 70, 102, 71, 103, 104, 105] and model-decoding-based segmentation [70, 106].

In distance-based segmentation, a distance measure of two audio segments is first defined, and then an acoustic change detection strategy is designed based on the distance measure. Compared to model-decoding-based segmentation, these methods have a great advantage that they do not need *a priori* knowledge about the content of the input audio stream. It is assumed that the acoustic feature vectors in each of the two audio segments are drawn from a probability distribution (e.g., multivariate Gaussian). Then, the distance between the two segments is represented as the dissimilarity between the two distributions. Many distance measures have been investigated, e.g., Kullback-Leibler distance (KL or KL2) [101], Generalized Likelihood Ratio (GLR) [104], ΔBIC [70, 71], Mahalanobis distance, and Bhattacharyya distance [105].

Fixed-size sliding window detection [101, 104, 105] and BIC-based growing-size sliding window detection [70, 102, 71, 103, 107] are two leading approaches in distance-based segmentation. In the fixed-size sliding window detection approach, as shown in Figure 5.1, a certain distance measure is used to evaluate the dissimilarity between two adjacent windows that slide along the audio stream to produce a distance curve. This distance curve is often low-pass filtered. Then, the locations of peaks are judged if they are acoustic changes by some heuristic thresholds. This method has the advantage of low computation cost. However, in order to detect the change boundary associated with a short homogeneous segment, the size of the analysis window is usually set at a small value (e.g., two seconds). This is a dilemma because a small analysis window does not contain sufficient feature vectors to obtain a reliable distance statistic.

BIC-based growing-size sliding window detection was first proposed by Chen and Gopalakrishnan [70]. For the distance measure of two audio segments, they used Bayesian Information Criterion (BIC) [41] to evaluate the following two hypotheses: 1) The union of the feature vectors of the two segments forms a Gaussian cluster in the feature space. 2) The feature vectors of each segment form a distinct Gaussian cluster. Then, the difference of the two evaluation scores, ΔBIC, was used as the distance measure. In their acoustic change detection procedure, a small analysis window is put at the beginning of the audio stream, initially. If there is no change point detected in the analysis window, it is enlarged to have a larger search range. However, with the window size growing, this approach suffers from a heavy computation cost due to numerous ΔBIC calculations, in particular when the audio stream contains many long homogenous segments. To reduce the computation cost, Tritschler and Gopinath [102] proposed some heuristics to ignore the distance computations at the locations where the acoustic changes unlikely happen. Zhou and Hansen [107] used the low computation cost Hotelling's T^2-Statistic as the distance measure in the detection process, while ΔBIC was used only to verify the acoustic change candidates. In [71] and [103], the authors proposed more efficient implementations for the ΔBIC calculation without affecting the detection accuracy. Since the growing-size sliding window detection approach detects acoustic changes using a size-growing analysis window, we denote it as *window-growing-based segmentation* (WinGrow).

In this thesis, we propose two divide-and-conquer approaches that detect acoustic changes by recursively partitioning a large analysis window into two sub-windows using ΔBIC, rather than detecting acoustic changes with a size-growing analysis window. For the efficiency comparison, we analyzed their computational costs and reported their respective run times in the experiments. The experiment results on the broadcast news data show that the proposed recursive (top-down) multiple-change-point detection strategies are more effective and efficient than WinGrow's bottom-up multiple-change-point detection strategy.

To help explain our proposed approaches, we review the ΔBIC distance measure and the WinGrow approach in Section 5.1. We then present the proposed divide-and-conquer approaches for audio segmentation in Section 5.2. In Section 5.3, we analyze the computational costs of the baseline approaches and the proposed approaches. Section 5.4 details the experiments on audio segmentation.

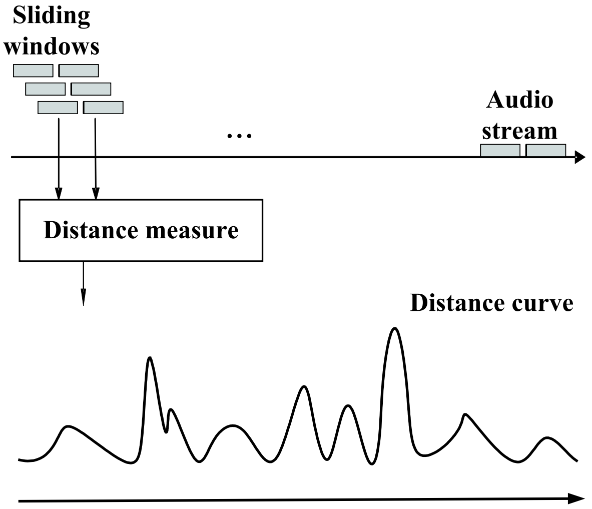


Figure 5.1: The fixed-size sliding window detection approach.

5.1 Window-growing-based segmentation

5.1.1 ΔBIC as the distance measure of two audio segments

Given two audio segments represented by feature vectors, $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_{n_x}\} \subset \Re^d$ and $\mathcal{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_{n_y}\} \subset \Re^d$, we evaluate the following two hypotheses [70]:

$$\begin{aligned} &H_0 : \mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_{n_x}, \mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_{n_y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \\ &H_1 : \mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_{n_x} \sim \mathcal{N}(\boldsymbol{\mu}_{\mathcal{X}}, \boldsymbol{\Sigma}_{\mathcal{X}});\ \mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_{n_y} \sim \mathcal{N}(\boldsymbol{\mu}_{\mathcal{Y}}, \boldsymbol{\Sigma}_{\mathcal{Y}}). \end{aligned} \tag{5.1}$$

H_0 posits that $\mathcal{X}$ and $\mathcal{Y}$ are derived from the same multivariate Gaussian, while H_1 posits that they are derived from two distinct multivariate Gaussians.

Let $\mathcal{Z} = \mathcal{X} \bigcup \mathcal{Y}$ and $n = n_x + n_y$. Then, the ΔBIC value can be computed as the difference between the BIC values of H_1 and H_0 as follows:

$$\begin{aligned} \Delta BIC_{\{\mathcal{X},\mathcal{Y}\}} &= BIC(H_1, \mathcal{Z}) - BIC(H_0, \mathcal{Z}) \\ &= \log p(\mathcal{X}; \hat{\boldsymbol{\mu}}_{\mathcal{X}}, \hat{\boldsymbol{\Sigma}}_{\mathcal{X}}) + \log p(\mathcal{Y}; \hat{\boldsymbol{\mu}}_{\mathcal{Y}}, \hat{\boldsymbol{\Sigma}}_{\mathcal{Y}}) \\ &\quad - \log p(\mathcal{Z}; \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}) - \frac{1}{2}\lambda(d + \frac{1}{2}d(d+1)) \log n \\ &= \frac{n}{2} \log |\hat{\boldsymbol{\Sigma}}| - \frac{n_x}{2} \log |\hat{\boldsymbol{\Sigma}}_{\mathcal{X}}| - \frac{n_y}{2} \log |\hat{\boldsymbol{\Sigma}}_{\mathcal{Y}}| - \frac{1}{2}\lambda(d + \frac{1}{2}d(d+1)) \log n, \end{aligned} \tag{5.2}$$

where $\hat{\boldsymbol{\mu}}$, $\hat{\boldsymbol{\mu}}_{\mathcal{X}}$, and $\hat{\boldsymbol{\mu}}_{\mathcal{Y}}$ are, respectively, the sample mean vectors of $\mathcal{Z}$, $\mathcal{X}$, and $\mathcal{Y}$; $\hat{\boldsymbol{\Sigma}}$, $\hat{\boldsymbol{\Sigma}}_{\mathcal{X}}$, and $\hat{\boldsymbol{\Sigma}}_{\mathcal{Y}}$ are, respectively, the sample covariance matrices of $\mathcal{Z}$, $\mathcal{X}$, and $\mathcal{Y}$; and d is the dimension of the feature vector [71][1]. The larger the value of ΔBIC, the less similar the two segments will be; thus, the larger the distance between the two segments will be. When $\lambda = 0$, the ΔBIC distance between two segments is equivalent to the GLR distance [70, 108].

5.1.2 One-change-point detection

Let the feature vectors of the input audio stream be $\mathcal{Z} = \{\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_n\}$. In Chen and Gopalakrishnan's one-change-point detection algorithm [70] (denoted as OCD-Chen in this paper), it is assumed that there is at most one change point in $\mathcal{Z}$. Then, the $\Delta BIC_{\{\mathcal{X}_i,\mathcal{Y}_i\}}(i)$ value for $i_{min} < i \leq n - i_{min}$ is computed as

$$\Delta BIC_{\{\mathcal{X}_i,\mathcal{Y}_i\}}(i) = \frac{n}{2}\log|\hat{\mathbf{\Sigma}}| - \frac{i}{2}\log|\hat{\mathbf{\Sigma}}_{\mathcal{X}_i}| - \frac{n-i}{2}\log|\hat{\mathbf{\Sigma}}_{\mathcal{Y}_i}| - \frac{1}{2}\lambda(d + \frac{1}{2}d(d+1))\log n, \quad (5.3)$$

where $\hat{\mathbf{\Sigma}}$, $\hat{\mathbf{\Sigma}}_{\mathcal{X}_i}$, and $\hat{\mathbf{\Sigma}}_{\mathcal{Y}_i}$ are, respectively, the sample covariance matrices of $\mathcal{Z}$, $\mathcal{X}_i$=$\{\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_i\}$, and $\mathcal{Y}_i$=$\{\mathbf{z}_{i+1}, \mathbf{z}_{i+2}, \cdots, \mathbf{z}_n\}$. If $max_{i_{min}<i\leq n-i_{min}}\Delta BIC_{\{\mathcal{X}_i,\mathcal{Y}_i\}}(i) > 0$, the time index corresponding to the maximum value is output as the change point; otherwise, there is no change point in $\mathcal{Z}$. It is not necessary to compute the ΔBIC value for time indices within the ranges 1 to i_{min} and $n - i_{min} + 1$ to n because in these cases the number of samples in $\mathcal{X}_i$ or $\mathcal{Y}_i$ is insufficient to give a reliable estimate of the parameters. Empirically, it is appropriate to set i_{min} at a value within the range 30 to 50 for practical applications. According to the BIC theory, the penalty factor λ in Eq. (5.3) is 1; however, in practical segmentation tasks, it is usually adjusted to allow a tradeoff between error types.

5.1.3 Multiple-change-point detection

OCD-Chen outputs at most one change point, even though there are multiple change points in the analysis window. To detect multiple change points in an audio stream, as shown in Figure 5.2, OCD-Chen can be applied sequentially to a sliding, size-growing analysis window whose initial size is N_{ini} samples. The window repeatedly grows by N_g samples until a change point is detected or its size exceeds a pre-defined upper bound N_{max}. Here, the upper bound ensures the search efficiency [103, 71]. If a change point is detected during the window growing step, the detection process restarts at that change point with an analysis window of N_{ini} samples. When the size of the window grows to N_{max}, it is repeatedly shifted by N_s samples until a change point is detected or the analysis

[1] In fact, Eq. (5.2) is derived based on classification likelihood. Thus, it may be more appropriate considering Eq. (5.2) a AWE-based distance measure since ΔAWE and Eq. (5.2) only differ in the penalty term which can be adjusted with λ. To avoid ambiguities, however, here we call the well-recognized distance measure in the speech processing community, Eq. (5.2), the ΔBIC measure.

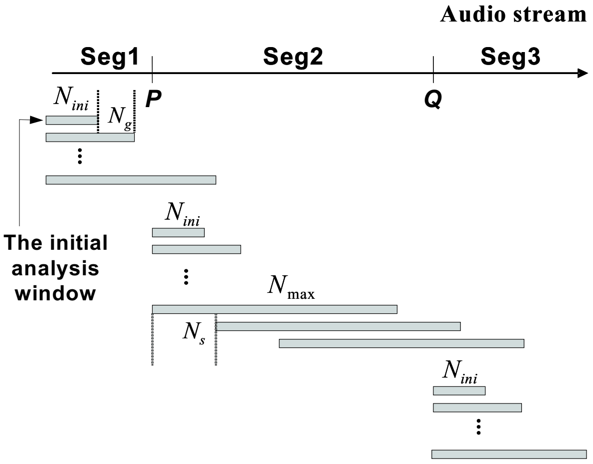


Figure 5.2: Diagram of the multiple-change-point detection in window-growing-based segmentation (WinGrow). The audio stream contains three segments, namely Seg1, Seg2, and Seg3; P and Q denote the change points.

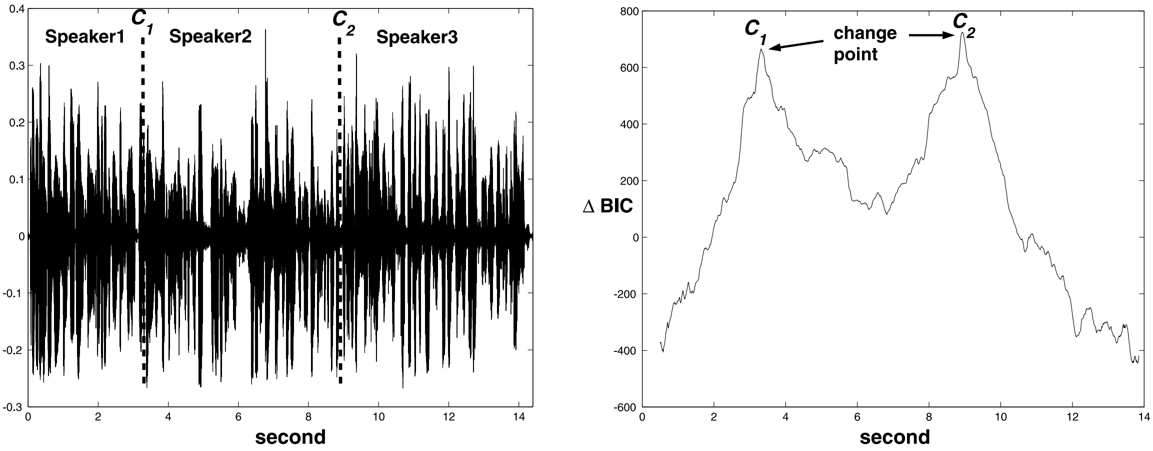
window reaches the end of the audio stream. In this way, the change points in the audio stream can be detected sequentially.

5.2 Divide-and-Conquer-based segmentation

In this section, we present two implementations of the divide-and-conquer paradigm for detecting multiple change points in an analysis window. Note that the proposed approaches are based on the same assumption as that of WinGrow, i.e., the feature vectors of audio segments from different acoustic source are derived from different Gaussian distributions.

5.2.1 The DACDec1 approach

We use the example in Figure 5.3 to explain the concept of divide-and-conquer-based segmentation. It is assumed that the audio stream in Figure 5.3 (a) consists of three homogeneous segments derived from different speakers. Initially, OCD-Chen is applied in an analysis window that covers the entire audio stream. After the change point C_2 has been detected with the ΔBIC curve in Figure 5.3 (b), the audio stream is divided into two analysis windows. Then, OCD-Chen is recursively applied in these two windows to search for the remaining change points so that C_1 can be detected. This approach, called DACDec1, allows us to detect the change points by a divide-and-conquer (DAC) strategy. As described in Algorithm 6, DACDec1 terminates (returns) if no change point is detected by OCD-Chen in the analysis window or the size of the analysis window is smaller than a pre-defined value, denoted as N_{min} samples. In the *Divide* stage, the analysis window is partitioned into two sub-windows at the change point detected by OCD-Chen. Then,



(a)

(b)

Figure 5.3: (a) An audio stream comprised of three speech segments, each derived from a distinct speaker. C_1 and C_2 are the change points. (b) The ΔBIC curve obtained by applying OCD-Chen to the audio stream in (a).

the sub-windows are input to DACDec1 in the *Solve sub-instances* stage. Finally, the *Combine* stage outputs all the change points detected in step 1) and step 4) (i.e., the *Solve sub-instances* stage).

Discussions: In general, when the data samples are derived from more than one Gaussian distribution, two Gaussians (the H_1 hypothesis) fit the distribution of the data better than one Gaussian (the H_0 hypothesis) if the samples belonging to the same Guassian are used together to estimate the parameters. For example, Figure 5.4 schematically illustrates a case where the three audio segments are derived from three different speakers and their feature vectors distribute as three Gaussian clusters. This case explains why the ΔBIC values at C_1 and C_2 in Figure 5.3 (b) are positive. From the above perspective, if the homogeneous segments in the analysis window of DACDec1 are always derived from different speakers during the recursive process, we can be confident that, at each change point, the H_1 hypothesis will fit the data better than the H_0 hypothesis; thus, the ΔBIC value will be positive.

However, if two or more segments in the analysis window are derived from the same speaker, the performance of DACDec1 may decline dramatically. For example, in Figure 5.5 (a), the first and third segments are derived from the same speaker (Speaker1), while the second segment is derived from another speaker (Speaker2). When applying OCD-Chen to the audio stream in Figure 5.5 (a) with the same λ value of BIC used in the example in Figure 5.3, we obtain the ΔBIC curve in Figure 5.5 (b). The curve still has two peaks at the change points C_1 and C_2 because the H_1 hypothesis models the distribution of the data samples better at change points than it does at non-change points. We use Figures 5.5 (c) and (d) to explain this perspective. Figure 5.5 (c) diagrammatically

Algorithm 6 $CP \leftarrow$DACDec1(W)

```
Require: $W$: the analysis window
Ensure: $CP$: the set of change points detected in $W$
  Begin

    1. detect whether there is a change point in $W$ by OCD-Chen;

    2. //Check termination
       if (there is no change point in $W$ or the size of $W$ is smaller than $N_{min}$)
          $CP \leftarrow \phi$; //empty set
          goto End; //return

    3. //Divide
       let $\hat{t}$ be the change point detected in 1);
       divide $W$ into two sub-windows, $W_1$ and $W_2$, at $\hat{t}$;

    4. //Solve sub-instances
       $CP_{W_1} \leftarrow DACDec1(W_1)$; $CP_{W_2} \leftarrow DACDec1(W_2)$;

    5. //Combine
       $CP \leftarrow \hat{t} \cup CP_{W_1} \cup CP_{W_2}$;

  End
```

illustrates the two hypotheses at C_2, where all the data samples of Speaker2 (the circles) are used with those of Speaker1 (the stars) to estimate one Gaussian in H_1. In contrast, at the non-change point R in Figure 5.5 (b), as shown in Figure 5.5 (d), the data samples of Speaker2 are divided into two parts, each of which is combined with the data samples of Speaker1 (one with the stars and the other with the diamonds) to estimate a distinct Gaussian in H_1. Clearly, the H_1 hypothesis in Figure 5.5 (c) fits the data better than that in Figure 5.5 (d).

In this example, we have peaks at C_1 and C_2. However, their ΔBIC values are negative, and no change point will be output by OCD-Chen because, as illustrated in Figure 5.5 (c), H_1 over-fits the data samples of Speaker1 and obtains a smaller BIC value than that of H_0. We may adjust the value of λ so that, at C_2, the ΔBIC value will be positive (i.e., the hypothesis test favors H_1). However, this may result in false alarms when the recursive process continues to detect change points in a homogeneous segment. In other words, it is difficult to determine a reliable λ value for an audio stream like the example in Figure 5.5 (a). Moreover, it is infeasible to adjust the value of λ for each specific audio stream in practical applications.

5.2.2 The DACDec2 approach

To overcome the performance limitation caused by unreliable ΔBIC measurements of the over-fitting cases in DACDec1, we developed an alternative implementation of the divide-

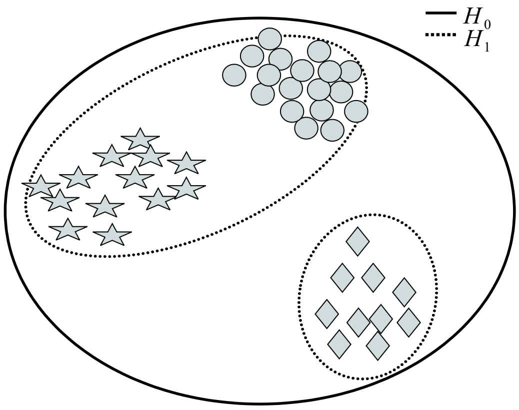
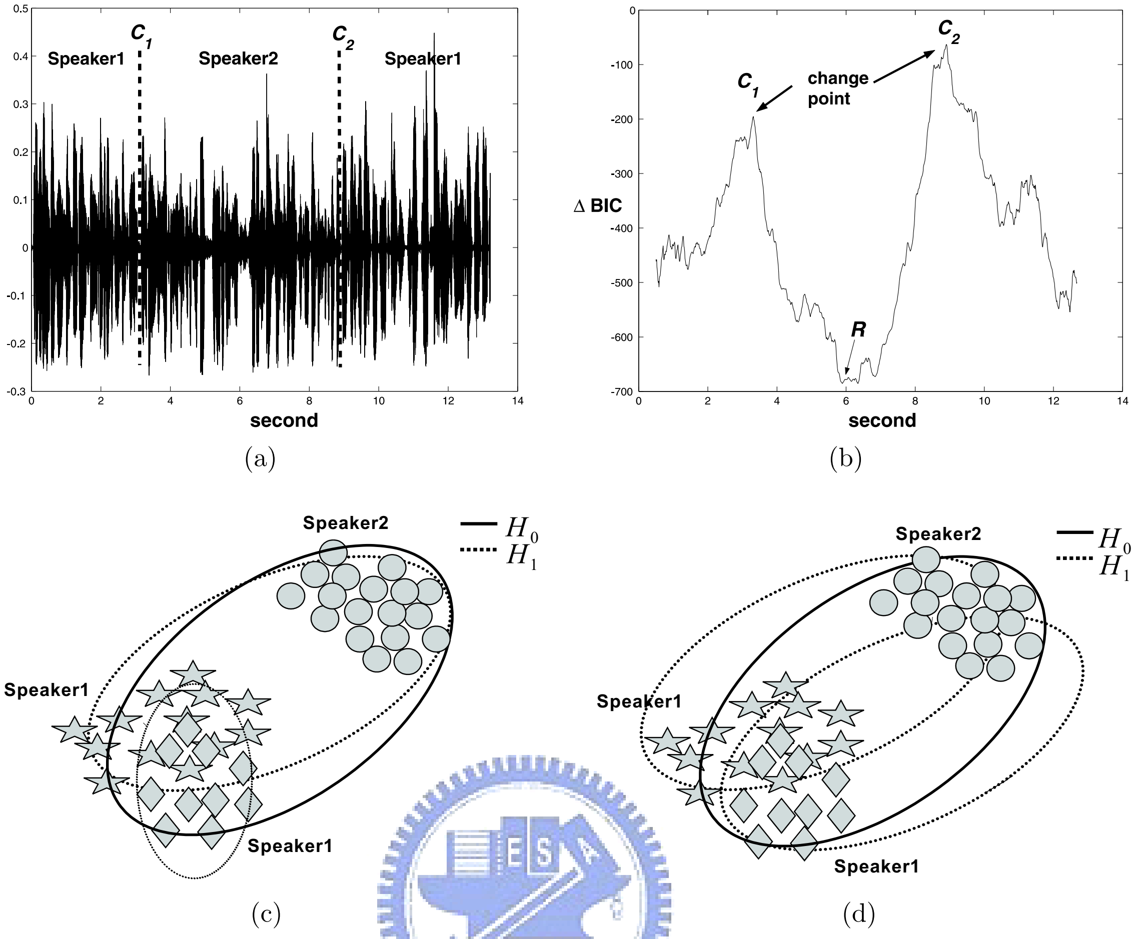


Figure 5.4: An illustration that data samples distribute as three Gaussian clusters. For this case, generally, two Gaussians (H_1) fit the distribution of the data better than one Gaussian (H_0) if the samples belonging to the same Gaussian cluster are used together to estimate the parameters.

and-conquer paradigm, called DACDec2. In this approach (Algorithm 7), the ΔBIC value is not used to check the termination in the *Check termination* stage because it may be unreliable, as illustrated in Figures 5.3 and 5.5. The recursive process terminates (returns) when the size of the analysis window is smaller than N_{min} samples. In the *Divide* stage, the analysis window is partitioned into two sub-windows at the time index $\hat{t}$ that has the largest ΔBIC value located by OCD-Chen. Then, the sub-windows are input to DACDec2 in the *Solve sub-instances* stage. In the *Combine* stage, $\hat{t}$ is labeled as a change point if the ΔBIC value at $\hat{t}$ calculated in the *Divide* stage is positive; otherwise, it needs to be verified using its two neighboring segments $\mathcal{X}$ and $\mathcal{Y}$. In the verification process, $\hat{t}$ is only labeled as a change point if $\Delta BIC_{\{\mathcal{X},\mathcal{Y}\}}(\hat{t}) > 0$.

Figure 5.6 illustrates a recursive tree that simulates the recursive process of DACDec2 on the audio stream in Figure 5.5 (a). We assume that there are no miss and false alarm errors in the detection process. In the figure, each tree node corresponds to a *divide-point* (i.e., $\hat{t}$) in the analysis window; the number inside the node indicates the order of the division, while the number below the node indicates the order in which the divide-point is verified in the *Combine* stage. In Figure 5.5 (b), Node 1 (C_2) has a negative ΔBIC value in the *Divide* stage; however, it will be labeled as a change point by the verification process with segments {c, d, e, f} and {g, h, i} in the *Combine* stage. Node 2 (C_1) has a positive ΔBIC value in the *Divide* stage; thus, it is labeled as a change point and verification is not necessary. Segments {a} and {b} will be used for verifying Node 3; segments {c, d} and {e, f} will be used for verifying Node 4, and so on.



(a) (b)

(c) (d)

Figure 5.5: (a) An audio stream comprised of three speech segments; the first and third segments are derived from the same speaker (Speaker1), while the second is derived from another speaker (Speaker2). (b) The ΔBIC curve obtained by applying OCD-Chen to the audio stream in (a). (c) The diagram of the hypothesis test at the change point C_2 in (b). (d) The diagram of the hypothesis test at the non-change point R in (b).

5.2.3 Sequential segmentation by DACDec1 and DACDec2

For a long audio stream, such as a one-hour broadcast news program, the segmentation task becomes computationally intractable when DACDec1 or DACDec2 are used to detect change points. Moreover, if the initial analysis window contains too many segments, it may be difficult for OCD-Chen to find an appropriate λ value to obtain robust ΔBIC measurements for the various hypothesis tests in the recursive process. Therefore, in practical applications, we apply DACDec1 and DACDec2 in a large analysis window of fixed-size (e.g., 20 seconds) that moves from the beginning to the end of the audio stream to detect the speaker changes sequentially. The proposed sequential segmentation algorithms, SeqDACDec1 and SeqDACDec2, are shown in Figure 5.7. In SeqDACDec1 (or SeqDACDec2), if a change point is detected in the fixed-size analysis window by DACDec1 (or DACDec2), the window is moved to the change point with the largest time

Algorithm 7 $CP \leftarrow$DACDec2(W)

```
Require: $W$: the analysis window
Ensure: $CP$: the set of change points detected in $W$
  Begin

    1. //Check termination
       if (the size of $W$ is smaller than $N_{min}$)
          $CP \leftarrow \phi$; //empty set
          goto End; //return

    2. //Divide
       apply OCD-Chen to $W$ and let $\hat{t}$ be the time index with the largest $\Delta BIC$ value;
       divide $W$ into two sub-windows, $W_1$ and $W_2$, at $\hat{t}$;

    3. //Solve sub-instances
       $CP_{W_1} \leftarrow DACDec2(W_1)$; $CP_{W_2} \leftarrow DACDec2(W_2)$;

    4. //Combine
       if ($\Delta BIC_{\{W_1,W_2\}}(\hat{t})$ calculated in 2) is positive)
          $CP \leftarrow \hat{t} \cup CP_{W_1} \cup CP_{W_2}$;
       else
          let $\mathcal{X}$ be the segment on the left of $\hat{t}$ in $W_1$ and $\mathcal{Y}$ be the segment on the right
       of $\hat{t}$ in $W_2$;
          if ($\Delta BIC_{\{\mathcal{X},\mathcal{Y}\}}(\hat{t}) > 0$) //$\hat{t}$ is a change point
             $CP \leftarrow \hat{t} \cup CP_{W_1} \cup CP_{W_2}$;
          else //$\hat{t}$ is not a change point
             merge $\mathcal{X}$ and $\mathcal{Y}$;
             $CP \leftarrow CP_{W_1} \cup CP_{W_2}$;
  End
```

index. Otherwise, it is moved forward by ηL samples, where L denotes the window size, and $\eta > 0$. Note that a small η will allow a missed change point to be checked again by DACDec1 (or DACDec2) in the subsequent fixed-size analysis window. Like WinGrow, SeqDACDec1 and SeqDACDec2 are suitable for on-line applications.

5.3 Computational cost analysis

WinGrow, DACDec1, and DACDec2 detect acoustic changes by applying the OCD-Chen process to the analysis window. From Eq. (5.3), it is clear that the computational cost of ΔBIC is mainly from the cost of calculating covariance matrices, which is proportional to the number of data samples. Let the time cost of calculating ΔBIC with m samples be $m\tau$, where τ represents the time unit; then, $m^2\tau$ denotes the time cost of applying OCD-Chen to an analysis window of m samples[2].

[2] As mentioned in Section 5.1.2, the ΔBIC value is not computed for samples at the beginning and the end of the analysis window. However, to simplify the analysis, we assume that the ΔBIC value is

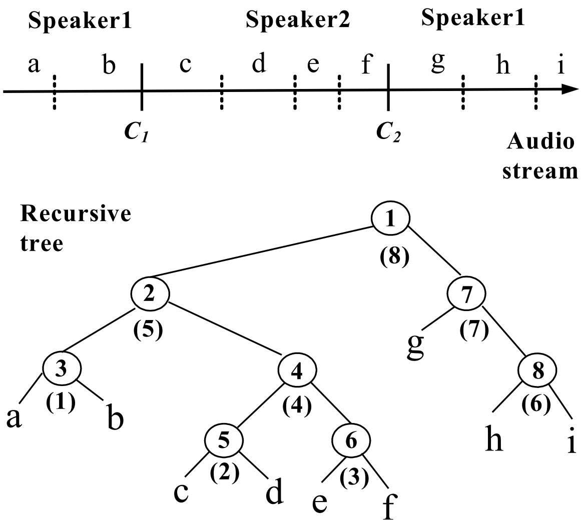


Figure 5.6: A recursive tree that simulates the recursive process of DACDec2 on the audio stream in Figure 5.5 (a).

To simplify the analysis, we assume that each homogeneous segment in the input audio stream (i.e., the initial analysis window for DACDec1 and DACDec2) contains m samples. Moreover, we assume the detection process is perfect, i.e., miss and false alarm errors never occur.

5.3.1 For DACDec1

Let $T_1(k)$ denote the time cost of applying DACDec1 to an audio stream of k change points (i.e., $k+1$ homogeneous segments). When the audio stream is divided at the i-th change point, as shown in Figure 5.8, we obtain the following recursive expression of $T_1(k)$:

$$T_1(k) = T_1(i-1) + T_1(k-i) + (k+1)^2 m^2 \tau, \tag{5.4}$$

where $(k+1)^2 m^2 \tau$ is the time cost of finding the divide-point by OCD-Chen; $T_1(i-1)$ and $T_1(k-i)$ are the time costs of applying DACDec1 in the left sub-stream and the right sub-stream, respectively. We have $T_1(0) = m^2\tau$, since it represents the time cost of applying OCD-Chen to a m-sample homogeneous segment.

We assume that the division occurs at each change point with equal probability; therefore, the average time cost of DACDec1 is

$$T_1(k) = \frac{1}{k}\sum_{i=1}^{k}(T_1(i-1) + T_1(k-i)) + (k+1)^2 m^2 \tau. \tag{5.5}$$

computed for each sample of the window.

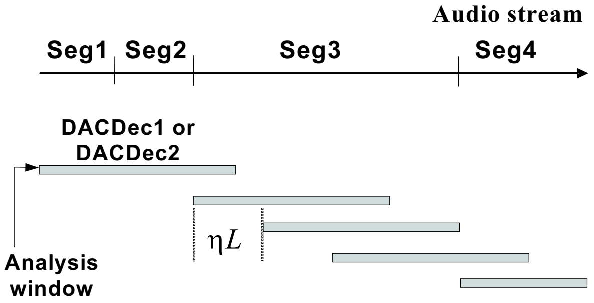


Figure 5.7: Diagram of the detection process of SeqDACDec1 and SeqDACDec2. If a change point is detected in the fixed-size analysis window by DACDec1 or DACDec2, the window is moved to the change point with the largest time index. Otherwise, it is moved forward by ηL samples, where L denotes the window size, and $\eta > 0$.

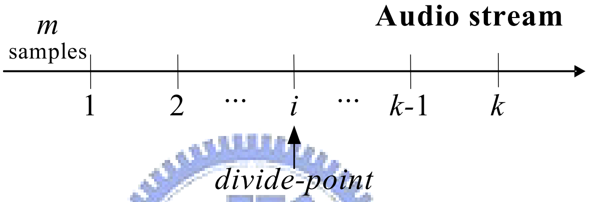


Figure 5.8: An audio stream comprised of $k+1$ homogeneous segments, each containing m samples. The stream is divided at the i-th change point.

After the algebraic manipulation detailed in Appendix B.1, we obtain

$$\begin{aligned} T_1(k) &\approx (3(k+1)^2 - 2(k+1)\ln(k+1))m^2\tau \\ &= O(k^2m^2\tau). \end{aligned} \tag{5.6}$$

5.3.2 For DACDec2

Compared to DACDec1, DACDec2 incurs an additional time cost in the *Combine* stage as it has to determine whether the divide-point with a negative ΔBIC value calculated in the *Divide* stage is a change point. The cost is $2m\tau$ because each of the divide-point's two neighboring segments contains m samples. To simplify the analysis, we assume that each divide-point must be verified, even though its ΔBIC value calculated in the *Divide* stage is positive. Hence, the average time cost of DACDec2 is

$$T_2(k) = \frac{1}{k}\sum_{i=1}^{k}(T_2(i-1) + T_2(k-i)) + (k+1)^2m^2\tau + 2m\tau. \tag{5.7}$$

Unlike DACDec1, DACDec2 recursively partitions each homogeneous segment of m samples until the analysis window is smaller than the pre-defined minimum value N_{min}. Therefore, $T_2(0)$ is equivalent to the time cost of applying DACDec2 to an m-sample stream in which each sample can be a divide-point. The cost of finding a divide-point in an m-sample stream in the *Divide* stage is $m^2\tau$. In the *Combine* stage, the cost of verifying the divide-point is at most $m\tau$ because the two segments used for verification are sub-segments of the m-sample segment. Therefore, the upper bound of $T_2(0)$ is

$$T'(m) = \frac{1}{m}\sum_{i=1}^{m}(T'(i-1) + T'(m-i)) + m^2\tau + m\tau, \tag{5.8}$$

where $T'(0) = 0$. After the algebraic manipulation detailed in Appendix B.2, we obtain

$$T_2(0) \le T'(m) \approx (3m + 4 - 4\ln(m+1))(m+1)\tau. \tag{5.9}$$

Then, we can solve the recursive equation in Eq. (5.7) with $T_2(0)$ in Eq. (5.9). After the algebraic manipulation detailed in Appendix B.3, we obtain

$$\begin{aligned} T_2(k) \le\ & (3k + 5 - 2\ln(k+1))(k+1)m^2\tau \\ & +(9 + 2\ln k - 2\ln(k+1) - 4\ln(m+1))(k+1)m\tau \\ & +(4 - 4\ln(m+1))(k+1)\tau \\ =\ & O(k^2m^2\tau). \end{aligned} \tag{5.10}$$

5.3.3 For FixSlid

Suppose FixSlid uses GLR (ΔBIC) as the distance measure and the analysis window consists of ω samples. Then, the time cost of FixSlid is

$$\begin{aligned} T_3(k) &= (2\omega\tau)(k+1)m \\ &= O(km\tau). \end{aligned} \tag{5.11}$$

5.3.4 For WinGrow

We analyze the case where the maximum window size N_{max} is large enough to ensure that the search process always restarts at a true change point[3]. In this case, the analysis

[3] Without this assumption, the time cost analysis for WinGrow might be intractable. However, this assumption is appropriate for many kinds of real-world data. For example, in our experiments on the

window W initialized with a small number of N_{ini} samples grows repeatedly by N_g samples until it contains more than m samples, so that there is at least one change point in W. Suppose W needs to grow to γm samples to detect the change point, where $\gamma > 0$; then, the time cost of sequentially detecting k change points will be

$$T_1'(k) = k[N_{ini}^2 + \sum_{i=1}^{(\gamma m - N_{ini})/N_g} (N_{ini} + iN_g)^2]\tau. \tag{5.12}$$

After the k-th change point has been detected, the detection process continues to search in the last homogeneous segment; the time cost is

$$C_s = [N_{ini}^2 + \sum_{i=1}^{(m - N_{ini})/N_g} (N_{ini} + iN_g)^2]\tau. \tag{5.13}$$

In practical applications, both N_{ini} and N_g are set at small values. To simplify the analysis, we assume $N_{ini} \approx N_g$. Then, the time cost of WinGrow is

$$\begin{aligned} T_5(k) &= T_1'(k) + C_s \\ &\approx k[\sum_{i=1}^{(\gamma m - N_g)/N_g} (iN_g)^2]\tau + [\sum_{i=1}^{(m - N_g)/N_g} (iN_g)^2]\tau \\ &= (\frac{\gamma^3 m^3}{3N_g} - \frac{\gamma^2 m^2}{2} + \frac{\gamma m N_g}{6})k\tau + (\frac{m^3}{3N_g} - \frac{m^2}{2} + \frac{mN_g}{6})\tau \\ &= O(km^3\tau). \end{aligned} \tag{5.14}$$

5.3.5 Discussion

From Eqs. (5.6), (5.10), (5.11), and (5.14), it is obvious that FixSlid is more efficient than DACDec1, DACDec2, and WinGrow.

DACDec1 and DACDec2 are more efficient than WinGrow when the input audio stream is composed of long homogeneous segments. For example, if the frame rate is 100 frames per second (i.e., there are 100 feature vectors for a one-second audio stream), it is appropriate to set the value of N_{ini} and N_g at 100. Moreover, the value of γ can be set at 1.5 generally. Then, for a 30-second audio stream (which consists of 3000 feature vectors) containing only one change point (i.e., $k = 1$ and $m = 1500$), the speedups of DACDec1 and DACDec2 over WinGrow are 2.55 and 1.78, respectively. When there is no change point in the 30-second stream, the speedups of DACDec1 and DACDec2 over Wingrow are 10.51 and 3.51, respectively. In contrast, when the audio stream is composed of short

broadcast news data described in Section 5.4, it is appropriate to set N_{max} at 20 seconds, which is longer than most of the homogeneous segments in the data set.

homogeneous segments, WinGrow is more efficient than DACDec1 and DACDec2. For example, for a 30-second stream containing five change points (i.e., $k = 5$ and $m = 500$), the speedups of DACDec1 and DACDec2 over WinGrow are 0.42 and 0.37, respectively.

5.4 Experiments

We conducted experiments on a synthetic data set using SeqDACDec1 and SeqDACDec2 to verify the unreliable ΔBIC measurement issue in DACDec1, and on two real-world broadcast news data sets to evaluate the performances of the baseline and proposed approaches.

For feature extraction, we used a 32-ms Hamming window shifted with a step of 10-ms to extract 24 mel-frequency cepstral coefficients as the acoustic features [70]. There were 100 24-dimensional feature vectors in a one-second audio stream.

For the performance evaluation, we used the Receiver Operating Characteristic (ROC) curve to show the various miss detection (MD) rates and false alarm (FA) rates yielded by adjusting the threshold parameters. A true change point t was counted as a miss detection if there was no hypothesized change point within $[t - \xi, t + \xi]$ (a 2ξ-second window centered on t); and a hypothesized change point $\hat{t}$ was counted as a false alarm if there was no true change point within $[\hat{t} - \xi, \hat{t} + \xi]$. The miss detection rate (MDR) and false alarm rate (FAR) are defined as [104]

$$\begin{aligned} \text{MDR} &= 100\% \times \frac{\text{number of MD}}{\text{number of true change points}}, \\ \text{FAR} &= 100\% \times \frac{\text{number of FA}}{\text{number of true change points} + \text{number of FA}}. \end{aligned}$$

5.4.1 Experiments on the synthetic data

Data set description: We used the training data of six speakers from the NIST 2001 speaker recognition evaluation database [68] to create three artificial audio streams of conversational speech as the synthetic data set. The speech from speaker#5077 and speaker#5232 was divided into three-second utterances and interlaced to form an audio stream of conversational speech of two speakers. In the same way, the speech from speaker#5326 and speaker#5333 was used to form the second audio stream; and the speech from speakers#5446 and speaker#5269 was used to form the third audio stream. There were 231 speaker change points in total in the three audio streams.

Results: Here, we set ξ at 0.5 seconds for the MD and FA definitions. Figure 5.9 shows the ROC curves obtained by running SeqDACDec1 and SeqDACDec2 on the synthetic data with different analysis window sizes. η was set at 0.25, N_{min} in DACDec1 and

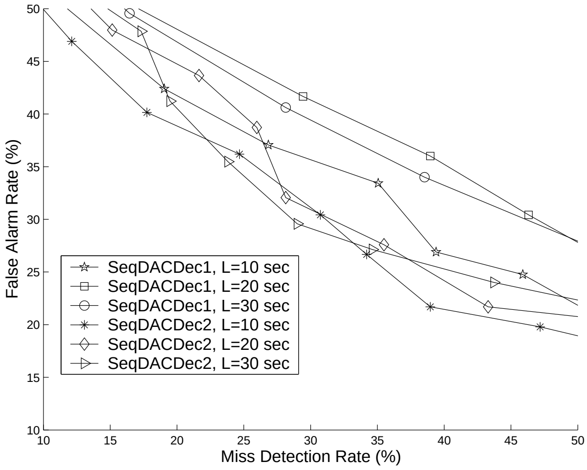


Figure 5.9: ROC curves obtained by running SeqDACDec1 and SeqDACDec2 on the synthetic data using 10-second, 20-second, and 30-second analysis windows. L denotes the size of the analysis window.

DACDec2 was set at one second (i.e., 100 samples), and the penalty factor λ in ΔBIC was set at 0.7 initially and increased to 1.7 in 0.05 increments. From the figure, we observe that SeqDACDec2 outperforms SeqDACDec1 for every window size. Moreover, SeqDACDec2 yields similar performances at different window sizes, whereas the performance of SeqDACDec1 declines significantly when the window size is increased from 10 seconds to 20 or 30 seconds. In other words, SeqDACDec2 is more robust to the size of the analysis window than SeqDACDec1. The experiment results conform to the discussion in Section 5.2.1; that is, DACDec1 might not work as well as DACDec2 if the condition that the homogeneous segments in the analysis window are derived from different acoustic sources is not met.

5.4.2 Experiments on broadcast news data

Data set description: We evaluated FixSlid, WinGrow, and the proposed methods on two broadcast news data sets. Three one-hour broadcast news programs (PTSND-20011203, PTSND-20011204, and PTSND-20011205) selected from the MATBN corpus [109] were used as the development set (denoted as MATBN3hr). We used the 1998 DARPA/NIST HUB-4 broadcast news evaluation test data set [110], which is comprised of two 1.5-hour audio streams, as the evaluation set (denoted as HUB4-98). There were 1386 and 1184 change points in MATBN3hr and HUB4-98, respectively. Figure 5.10 shows the empirical cumulative distributions of the size of homogeneous segments in the two data sets. As shown in the figure, the average length of the segments in HUB4-98 is longer than that in MATBN3hr.

Parameter setting: For FixSlid, we used the GLR distance as the distance measure

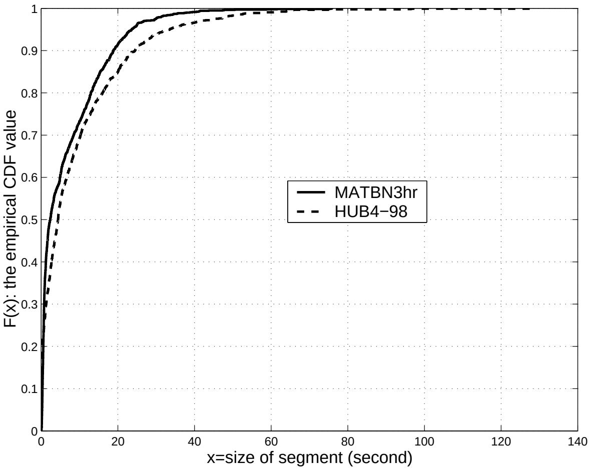


Figure 5.10: The empirical cumulative distributions of the size of homogeneous segments in MATBN3hr and HUB4-98.

of two adjacent windows. In the experiments, the window size was fixed at two seconds; and the value of α used to evaluate the "significant" local maximum, as shown in Figure 5.11, was set at 0.4 initially, and increased to 2 in 0.05 increments to obtain the ROC curve. For WinGrow, the values of N_g and N_s were set at one second and $N_{max}/4$ seconds, respectively; and the values of N_{ini} and N_{max} were tuned with the development set. For SeqDACDec1 and SeqDACDec2, η was fixed at 0.25; and L and N_{min} in DACDec1 and DACDec2 were tuned with the development set. In all the approaches except FixSlid, the penalty factor λ in the ΔBIC computation was set at 0.7 initially, and increased to 1.7 in 0.05 increments to obtain the ROC curves. The GLR or ΔBIC distance was evaluated every 0.1 seconds in all the approaches; that is, the resolution for change point detection was 0.1 seconds. However, the tolerance ξ for counting the number of miss detection or false alarm was set at one second rather than 0.5 seconds. Basically, we made this change because of the limited precision of human reference annotation.

Results: We first evaluated all the approaches on MATBN3hr. When conducting experiments, we found that it was appropriate to set N_{ini} at three seconds and N_{max} at 20 seconds for WinGrow. For both SeqDACDec1 and SeqDACDec2, it was appropriate to set N_{min} at two seconds and L at 20 seconds. Figure 5.12 (a) shows the ROC curves obtained by SeqDACDec1 with analysis windows of different size. Unlike the results for the synthetic data in Figure 5.9, the results with 10-second and 20-second analysis windows are similar. This is because, in the broadcast news data, homogeneous segments within a 10-second or 20-second analysis window are usually derived from different acoustic sources or speakers. For SeqDACDec2, the results for 10-second, 20-second, and 30-second analysis windows are similar, as shown in Figure 5.12 (b). The ROC curves obtained by all the approaches are shown in Figure 5.12 (c). We observe that the proposed approaches,

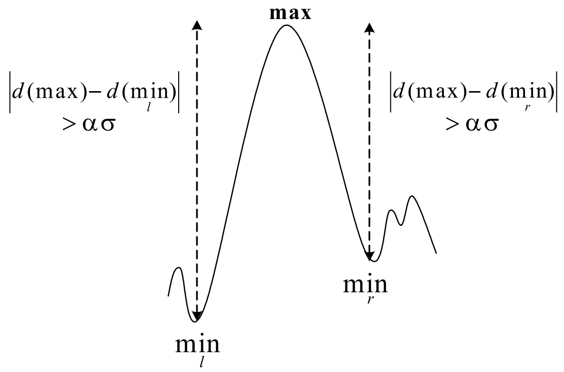


Figure 5.11: A significant local maximum on the distance curve.

Table 5.1: The CPU time of different audio segmentation approaches evaluated on MATBN3hr in the EER case and the associated EERs, where M and F denote the miss detection rate and the false alarm rate, respectively.

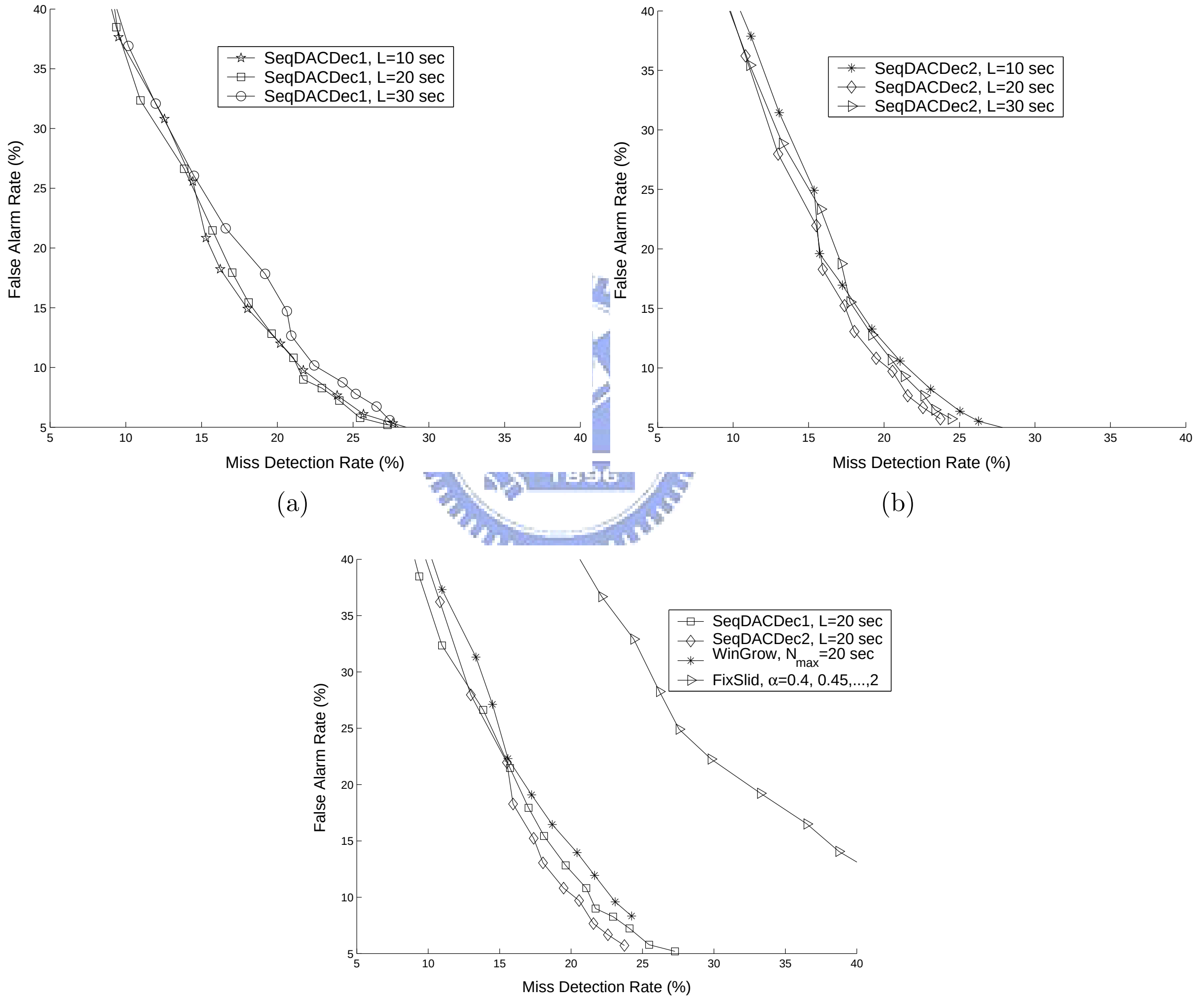
Approach	WinGrow	SeqDACDec1	SeqDACDec2	FixSlid
CPU time	5162.08 sec	1911.17 sec	3386.84 sec	221.28 sec
Speedup over WinGrow	1	2.70	1.52	23.33
EER (in %)	M:18.69 F:16.46	M:17.03 F: 17.94	M:17.39 F:15.23	M:27.13 F:25.76

namely SeqDACDec1 and SeqDACDec2, outperform the other approaches. Table 5.1 shows the CPU times of all the approaches in the EER case. The programs were run on a PC with a 3.2GHz Intel Pentium IV CPU. From the table, we observe that SeqDACDec1 and SeqDACDec2 are more efficient than WinGrow.

Next, we conducted experiments on HUB4-98 with the parameters tuned with the MATBN3hr data set. Figure 5.13 shows the ROC curves for all approaches; we see that SeqDACDec1 and SeqDACDec2 achieve the best performance. Table 5.2 summarizes the CPU time required by different approaches in the EER case. Comparing Table 5.2 to Table 5.1, it is clear that every approach achieves a higher speedup over WinGrow on HUB4-98 than on MATBN3hr. This is because the homogeneous segments in HUB4-98 are longer than those in MATBN3hr on average, as shown in Figure 5.10, and these approaches achieve higher speedup over WinGrow for an audio stream comprised of longer homogeneous segments, as mentioned in Section 5.3 (*cf.* Eqs. (5.6), (5.10), (5.11), and (5.14)).

Table 5.2: The CPU time of different audio segmentation approaches evaluated on HUB4-98 in the EER case and the associated EERs.

Approach	WinGrow	SeqDACDec1	SeqDACDec2	FixSlid
CPU time	8418.23 sec	2003.62 sec	3853.48 sec	201.57 sec
Speedup over WinGrow	1	4.20	2.18	41.76
EER (in %)	M:28.8 F:31.08	M:27.96 F:25.67	M:27.19 F:26.37	M:35.56 F:38.04



(a)

(b)

(c)

Figure 5.12: The ROC curves for MATBN3hr obtained by (a) SeqDACDec1 with $N_{min} = 2$ seconds and analysis windows of different size; (b) SeqDACDec2 with $N_{min} = 2$ seconds and analysis windows of different size; and (c) SeqDACDec1 with $N_{min} = 2$ seconds and $L = 20$ seconds, SeqDACDec2 with $N_{min} = 2$ seconds and $L = 20$ seconds, WinGrow with $N_{min} = 3$ seconds and $N_{max} = 20$ seconds, and FixSlid with a 2-second window.

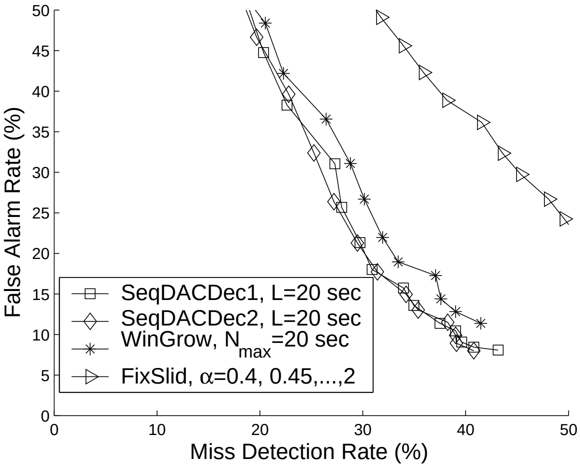


Figure 5.13: The ROC curves for HUB4-98.

Chapter 6

Conclusion and future work

6.1 Conclusion

In this thesis, we propose new learning algorithms for probabilistic model-based clustering. The proposed SGML algorithm tries to tackle two long standing critical problems in the EM-based Gaussian mixture modeling; namely, 1) the difficulty in determining the number of Gaussian components and 2) the sensitivity to model initialization. A fast version of the SGML, called fastSGML, is also presented. It splits multiple components in each splitting step and, thus, needs a much lower computation cost than SGML. We conducted experiments on clustering of a synthetic data set and the speaker identification task. Experiment results on the speaker identification task show that the proposed algorithms can automatically determine an appropriate model complexity for speaker GMMs though no significant improvements in identification accuracy are obtained compared to the best performance of the baseline systems.

Considering the learning of a probabilistic self-organizing map (PbSOM) as a model-based clustering process, we develop a coupling-likelihood mixture model for PbSOM, and derive three EM-type learning algorithms, namely the SOCEM, SOEM, and SODAEM algorithms, for learning the model (PbSOM). The proposed algorithms improve Kohonen's learning algorithms by including a cost function, an EM-based convergence property, and a probabilistic framework. In addition, the proposed algorithms provide some insights into the choice of neighborhood size that would ensure topographic ordering. From the experiment results, we observe that the learning performance of SOCEM is very sensitive to the initial setting of the reference models when the neighborhood is small. Conversely, it is not sensitive to the initial condition when the neighborhood is sufficiently large. To deal with the initialization problem, we first run SOCEM with a large neighborhood, and then gradually reduce the neighborhood size until the learning converges to the desired map. When using a small neighborhood, SOEM is less sensitive to the initialization than SOCEM. However, to learn an ordered map, SOEM still needs to start with a large neighborhood. In both SOCEM and SOEM, the neighborhood shrinking can be interpreted as an annealing process that overcomes the initialization issue. Alternatively, we can apply SODAEM, which is a deterministic annealing variant of SOCEM and SOEM, to learn a map. In our experiments, SODAEM overcomes the initialization issue of SOCEM and SOEM via the annealing process controlled by the temperature parameter. Moreover, through the comparison of SOCEM and Kohonen's batch algorithm, we can also apply the DA interpretation of neighborhood shrinking to Kohonen's algorithms to explain why they need to start with a large neighborhood size. We have also shown that the SOCEM and SOEM algorithms can be interpreted, respectively, as topology-constrained deterministic annealing variants of the CEM and EM algorithms for Gaussian model-based clustering. The experiment results show that our proposed PbSOM learning algorithms achieve an effective data clustering performance, while maintaining the topology-preserving property.

Moreover, we propose two BIC-based audio segmentation approaches that employ divide-and-conquer strategies for acoustic change detection. In contrast to the leading and highly accurate window-growing-based approach, which searches for acoustic changes in a bottom-up manner by using a sequentially size-growing analysis window, the proposed DACDec1 and DACDec2 approaches search for acoustic changes in a top-down manner. We compared our approaches to leading approaches analytically by performing computational cost analysis. The results of experiments conducted on broadcast news data demonstrate that the proposed approaches are more efficient and achieve higher segmentation accuracy than the existing approaches discussed in this thesis.

6.2 Future work

Some research ideas derived on the basis of this dissertation are as follows.

For SGML and fastSGML: In addition to the application to speaker identification, one may also apply the SGML and fastSGML algorithms to learn the universal background model (UBM) for speaker verification [3, 91]. UBM is a GMM with a large component number, say 512 or 1024, which was usually empirically determined. SGML and fastSGML could be used to automatically determine an appropriate component number for UBM.

For PbSOM: Kohonens's SOM algorithm has been successfully applied to many applications, such as document processing [78, 79], image processing [80], and speech precessing [81, 82, 83]. One may apply the SOCEM, SOEM, and SODAEM algorithms to the above tasks, evaluate the performance, and compare the results to those obtained by Kohonen's algorithm. In addition, there is one idea on the perspective of model learning. The PbSOM algorithms presented in this thesis are derived based on the maximum likelihood criterion. It is worth to think the possibility of deriving PbSOM learning algorithms based on other criteria, for example, Maximum A Posteriori (MAP).

For audio segmentation: One may apply the audio segmentation approaches presented in this thesis to some audio processing tasks, for example, audio indexing [97], automatic transcription of audio recordings [98], speaker tracking [99], and speaker diarization [100].

Appendix A

A.1 The SOCEM, SOEM, and SODAEM algorithms where mixture weights are learned

Theoretically, the mixture weights of the coupling-likelihood mixture model in Eq. (4.6) can be learned automatically. Following the derivations of the SOCEM, SOEM, and SODAEM algorithms in Sections 4.2, 4.3, and 4.4, the learning rules for the mixture weights are derived as follows.

- **Posterior distribution:**
  For SOCEM and SOEM,

$$\gamma_{k|i}^{(t)} = \frac{w_s(k)^{(t)} \exp(\sum_{l=1}^{G} h_{kl} \log r_l(\mathbf{x}_i; \boldsymbol{\theta}_l^{(t)}))}{\sum_{j=1}^{G} w_s(j)^{(t)} \exp(\sum_{l=1}^{G} h_{jl} \log r_l(\mathbf{x}_i; \boldsymbol{\theta}_l^{(t)}))}. \tag{A.1}$$

  For SODAEM,

$$\tau_{k|i}^{(t)} = \frac{(w_s(k)^{(t)} \exp(\sum_{l=1}^{G} h_{kl} \log r_l(\mathbf{x}_i; \boldsymbol{\theta}_l^{(t)})))^{\beta}}{\sum_{j=1}^{G} (w_s(j)^{(t)} \exp(\sum_{l=1}^{G} h_{jl} \log r_l(\mathbf{x}_i; \boldsymbol{\theta}_l^{(t)})))^{\beta}}. \tag{A.2}$$

- **Re-estimation formulae**:
  For SOCEM,

$$w_s(k)^{(t+1)} = \frac{1}{N} |\hat{\mathcal{P}}_k^{(t)}|. \tag{A.3}$$

  For SOEM,

$$w_s(k)^{(t+1)} = \frac{1}{N} \sum_{i=1}^{N} \gamma_{k|i}^{(t)}. \tag{A.4}$$

  For SODAEM,

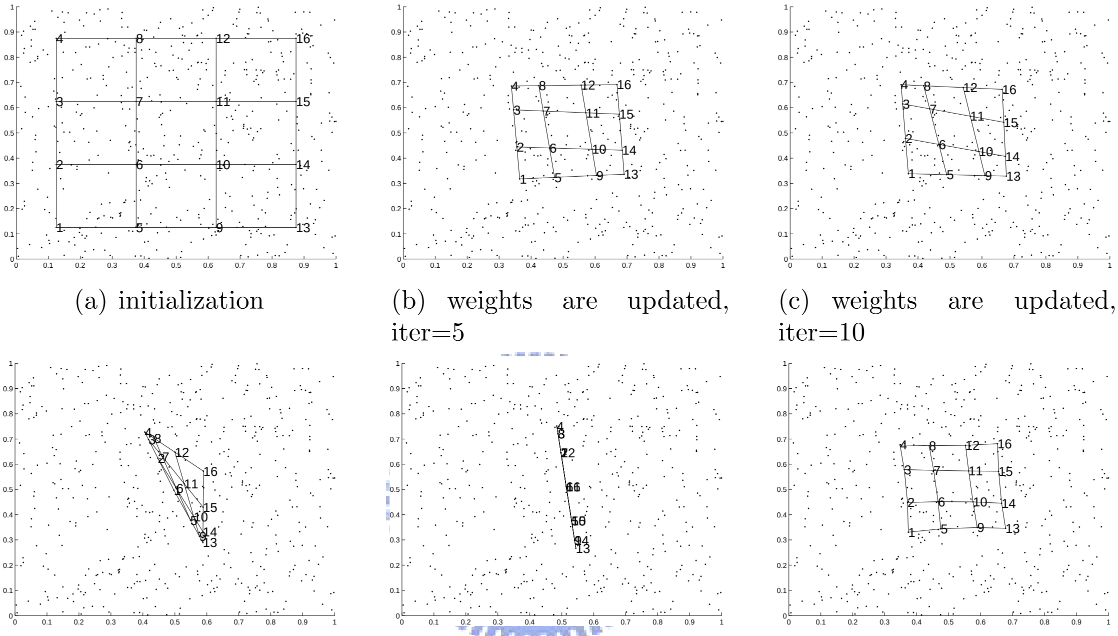
$$w_s(k)^{(t+1)} = \frac{1}{N} \sum_{i=1}^{N} \tau_{k|i}^{(t)}. \tag{A.5}$$

Table A.1: The mixture weights learned by SOCEM with the initialization in Figure A.1 (a). The mixture weights are initialized at $\frac{1}{16}$.

weight index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Initial	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$
iter=5	0.202	0	0	0.304	0	0	0	0	0	0	0	0	0.286	0	0	0.208
iter=10	0.144	0	0	0.354	0	0	0	0	0	0	0	0	0.344	0	0	0.158
iter=16	0	0	0	0.454	0	0	0	0	0	0	0	0	0.494	0	0	0.052
iter=18	0	0	0	0.504	0	0	0	0	0	0	0	0	0.496	0	0	0

The mean vectors and covariance matrices in SOCEM, SOEM, and SODAEM algorithms are updated using Eqs. (4.11)-(4.12), Eqs. (4.18)-(4.19), and Eqs. (4.22)-(4.23), respectively, where $\gamma_{k|i}^{(t)}$ and $\tau_{k|i}^{(t)}$ are computed by Eqs. (A.1) and (A.2), respectively.

However, in our experience, if the mixture weights are learned in the three algorithms, the learning of topological order is frequently dominated by some particular mixture components, which makes it difficult to obtain an ordered map. As an example, we applied SOCEM to the synthetic data set, which consisted of 500 points uniformly distributed in a unit square. The network structure was a 4 × 4 equally spaced square lattice in a unit square. All the mixture weights were set at 1/16 initially. The value of σ in the neighborhood function (i.e., Eq. (2.4)) was set at 0.4. The results are shown in Figs. A.1 (a)-(e). From the figures, we observe that the map shrinks to near a line after the algorithm converges (with 18 iterations). This phenomenon can be verified by inspecting the values of mixture weights during the learning process. As shown in Table A.1, after the algorithm converges, most of the mixture weights become zero and the learning only maximizes the local coupling-likelihoods of neurons 4 and 13, whose mixture weights are 0.504 and 0.496, respectively. In contrast, as shown in Figure A.1 (f), if the mixture weights are equally fixed at 1/16 throughout the learning process, SOCEM converges to an ordered map. For SOEM and SODAEM, we obtained the similar results.



(a) initialization

(b) weights are updated, iter=5

(c) weights are updated, iter=10

(d) weights are updated, iter=16

(e) weights are updated, iter=18

(f) fixed equal weights

Figure A.1: The map-learning process obtained by running the SOCEM algorithm on the synthetic data with an ordered initialization in (a). Simulation 1 ((a)-(e)): The mixture weights are initialized at $\frac{1}{16}$, and updated in the learning process; the algorithm starts with the initialization in (a) and converges to the unordered map in (e). Simulation 2 ((a) and (f)): SOCEM is performed with equal mixture weights throughout the learning process; the algorithm starts with the initialization in (a) and converges to the map in (f). The network structure is a 4 × 4 square lattice; the value of σ is set at 0.4.

Appendix B

B.1 Compute $T_1(k)$

$T_1(k)$ is expressed as

$$T_1(k) = \frac{1}{k}\sum_{i=1}^{k}(T_1(i-1)+T_1(k-i)) + (k+1)^2m^2\tau$$
$$= \frac{2}{k}\sum_{i=1}^{k}T_1(i-1) + (k+1)^2m^2\tau, \tag{B.1}$$

where $T_1(0) = m^2\tau$. To solve this recursive equation, we can apply the technique used for analyzing the time cost of the Quicksort algorithm [111]. First, we multiply both sides of Eq. (B.1) by k as follows:

$$kT_1(k) = 2\sum_{i=1}^{k}T_1(i-1) + k(k+1)^2m^2\tau. \tag{B.2}$$

Replacing k in Eq. (B.2) with $k-1$, we obtain

$$(k-1)T_1(k-1) = 2\sum_{i=1}^{k-1}T_1(i-1) + (k-1)k^2m^2\tau. \tag{B.3}$$

Subtracting Eq. (B.3) from Eq. (B.2), we obtain

$$kT_1(k) - (k-1)T_1(k-1) = 2T_1(k-1) + (3k^2+k)m^2\tau. \tag{B.4}$$

Rearranging the terms in Eq. (B.4) yields

$$\frac{T_1(k)}{k+1} = \frac{T_1(k-1)}{k} + \frac{(3k+1)m^2\tau}{(k+1)}. \tag{B.5}$$

Let $a_k = T_1(k)/(k+1)$, then Eq. (B.5) can be rewritten as

$$a_k = a_{k-1} + (3 - \frac{2}{k+1})m^2\tau, \tag{B.6}$$

where $a_0 = m^2\tau$. Recursively substituting the $a_k s'$ in Eq. (B.6), we obtain

$$\begin{aligned} a_k &= a_0 + (3k - (\frac{2}{2} + \frac{2}{3} + \cdots + \frac{2}{k+1}))m^2\tau \\ &= (3k + 3 - 2\sum_{i=1}^{k+1}\frac{1}{i})m^2\tau. \end{aligned} \tag{B.7}$$

Because $\sum_{i=1}^{k+1}\frac{1}{i} \approx \ln(k+1)$ [111], we have

$$a_k \approx (3k + 3 - 2\ln(k+1))m^2\tau. \tag{B.8}$$

Since $a_k = T_1(k)/(k+1)$, $T_1(k)$ can be expressed as

$$\begin{aligned} T_1(k) &= a_k(k+1) \\ &\approx (3(k+1)^2 - 2(k+1)\ln(k+1))m^2\tau. \end{aligned} \tag{B.9}$$

B.2 Compute $T'(m)$

$T'(m)$ is expressed as

$$T'(m) = \frac{1}{m}\sum_{i=1}^{m}(T'(i-1) + T'(m-i)) + m^2\tau + m\tau, \tag{B.10}$$

where $T'(0) = 0$. Similar to the manipulation of Eq. (B.1) in Appendix B.1, by setting $a_m = T'(m)/(m+1)$, we have $a_0 = 0$ and

$$\begin{aligned} a_m &= a_{m-1} + \frac{(3m-1)\tau}{m+1} \\ &= a_{m-1} + (3 - \frac{4}{m+1})\tau \\ &= a_0 + (3m - (\frac{4}{2} + \frac{4}{3} + \cdots + \frac{4}{m+1}))\tau \\ &\approx (3m + 4 - 4\ln(m+1))\tau. \end{aligned} \tag{B.11}$$

Since $a_m = T'(m)/(m+1)$, we have

$$\begin{aligned} T'(m) &= a_m(m+1) \\ &\approx (3m + 4 - 4\ln(m+1))(m+1)\tau. \end{aligned} \tag{B.12}$$

B.3 Compute $T_2(k)$

$T_2(k)$ is expressed as

$$T_2(k) = \frac{1}{k}\sum_{i=1}^{k}(T_2(i-1) + T_2(k-i)) + (k+1)^2m^2\tau + 2m\tau, \tag{B.13}$$

where $T_2(0) \leq (3m+4-4\ln(m+1))(m+1)\tau$. Similar to the manipulation of Eq. (B.1) in Appendix B.1, by setting $a_k = T_2(k)/(k+1)$, we have

$$\begin{aligned} a_k &= a_{k-1} + \frac{(3k^2+k)m^2\tau + 2m\tau}{k(k+1)}, \\ &= a_{k-1} + (3 - \frac{2}{k+1})m^2\tau + 2(\frac{1}{k} - \frac{1}{k+1})m\tau, \\ &\approx a_0 + (3k+2-2\ln(k+1))m^2\tau + 2(\ln k - \ln(k+1)+1)m\tau. \end{aligned} \tag{B.14}$$

Substituting $a_0 = T_2(0)$ into Eq. (B.14), we obtain

$$\begin{aligned} a_k \leq\ & (3k+5-2\ln(k+1))m^2\tau \\ & +(9+2\ln k - 2\ln(k+1) - 4\ln(m+1))m\tau + (4-4\ln(m+1))\tau. \end{aligned} \tag{B.15}$$

Since $a_k = T_2(k)/(k+1)$, we have

$$\begin{aligned} T_2(k) \leq\ & (3k+5-2\ln(k+1))(k+1)m^2\tau \\ & +(9+2\ln k - 2\ln(k+1) - 4\ln(m+1))(k+1)m\tau \\ & +(4-4\ln(m+1))(k+1)\tau. \end{aligned} \tag{B.16}$$

Bibliography

[1] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Computing Surveys*, vol. 31, no. 3, pp. 264–323, 1999.

[2] R. Xu and D. Wunsch, "Survey of clustering algorithms," *IEEE Trans. Neural Networks*, vol. 16, no. 3, pp. 645–678, 2005.

[3] D. A. Reynolds, T. F. Quatieri, and R. B. Dunn, "Speaker verification using adapted Gaussian mixture models," *Digital Signal Processing*, vol. 10, no. 1-3, pp. 19–41, 2000.

[4] D. A. Reynolds and R. C. Rose, "Robust text-independent speaker identification using Gaussian mixture speaker models," *IEEE Trans. Speech and Audio Processing*, vol. 3, no. 1, pp. 72–83, 1995.

[5] S.-S. Cheng, Y.-Y. Xu, H.-M. Wang, and H.-C. Fu, "Automatic construction of regression class tree for MLLR via model-based hierarchical clustering," *Lecture Notes in Computer Science*, pp. 390–398, 2006.

[6] H. C. Fu, H. Y. Chang, Y. Y. Xu, and H. T. Pao, "User adaptive handwriting recognition by self-growing probabilistic decision-based neural networks," *IEEE Trans. Neural Networks*, vol. 11, no. 6, pp. 1373–1384, 2000.

[7] R. Wehrens, L. M. C. Buydens, C. Fraley, and A. E. Raftery, "Model-based clustering for image segmentation and large datasets via sampling," *Journal of Classification*, vol. 21, no. 2, pp. 231–253, 2004.

[8] G. J. McLachlan, R. W. Bean, and D. Peel, "A mixture model-based approach to the clustering of microarray expression data," *Bioinformatics*, vol. 18, no. 3, pp. 413–422, 2002.

[9] J. C. Mar and G. J. McLachlan, "Model-based clustering in gene expression microarrays: An application to breast cancer data," *International Journal of Software Engineering and Knowledge Engineering*, vol. 13, no. 6, pp. 579–592, 2003.

[10] C. Fraley and A. E. Raftery, “How many clusters? Which clustering method? Answers via model-based cluster analysis,” *The Computer Journal*, vol. 41, no. 8, pp. 578–588, 1998.

[11] C. Fraley and A. E. Raftery, “Model-based clustering, discriminant analysis, and density estimation,” *Journal of the American Statistical Association*, vol. 97, no. 458, pp. 611–631, 2002.

[12] S. Zhong and J. Ghosh, “A unified framework for model-based clustering,” *Journal of Machine Learning Research*, vol. 4, no. 6, pp. 1001–1037, 2003.

[13] C. Fraley and A. E. Raftery, “Bayesian regularization for normal mixture estimation and model-based clustering,” *Journal of Classification*, vol. 24, no. 2, pp. 155–181, 2007.

[14] M. S. Oh and A. E. Raftery, “Model-based clustering with dissimilarities: A Bayesian approach,” *Journal of Computational and Graphical Statistics*, vol. 16, no. 3, pp. 559–585, 2007.

[15] M. J. Symons, “Clustering criteria and multivariate normal mixture,” *Biometrics*, vol. 37, pp. 35–43, 1981.

[16] S. Ganesalingam, “Classification and mixture approach to clustering via maximum likelihood,” *Applied Statistics*, vol. 38, no. 3, pp. 455–466, 1989.

[17] G. Celeux and G. Govaert, “A classification EM algorithm for clustering and two stochastic versions,” *Computational Statistics & Data Analysis*, vol. 14, no. 3, pp. 315–332, 1992.

[18] J. D. Banfield and A. E. Raftery, “Model-based Gaussian and non-Gaussian clustering,” *Biometrics*, vol. 49, no. 3, pp. 803–821, 1993.

[19] A. P. Dempster, N. M. Laird, and D. B. Rubin, “Maximum likelihood from incomplete data via the EM algorithm,” *J. Roy. Statistical Soc. B*, vol. 39, 1977.

[20] J. A. Bilmes, “A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models,” Technical Reports TR-97-021, International Computer Science Insitute, April 1998.

[21] L. Xu and M. Jordan, “On convergence properties of the EM algorithm for Gaussian mixtures,” *Neural Computation*, vol. 8, no. 1, pp. 129–151, 1996.

[22] G. J. McLachlan and T. Krishnan, *The EM algorithm and extensions.* New York: John Wiley, 1977.

[23] N. Ueda and R. Nakano, "Deterministic annealing EM algorithm," *Neural Networks*, vol. 11, no. 2, pp. 271–282, 1998.

[24] Y. Ishikawa and R. Nakano, "Landscape of a likelihood surface for a Gaussian mixture and its use for the EM algorithm," in *Proc. Int. Joint Conf. Neural Networks*, pp. 1434–1440, 2006.

[25] Y. Ishikawa and R. Nakano, "EM algorithm with PIP initialization and temperature-based selection," in *Proc. Int. Conf. Knowledge-Based Intelligent Information and Engineering Systems, Part III*, pp. 58–66, 2008.

[26] M. Takada and R. Nakano, "Multi-thread search with deterministic annealing EM algorithm," in *Proc. Int. Joint Conf. Neural Networks*, pp. 1034–1038, 2002.

[27] M. Takada and R. Nakano, "Threshold-based dynamic annealing for multi-thread DAEM and its extreme," pp. 501–506, 2003.

[28] F. Pernkopf and D. Bouchaffra, "Genetic-based EM algorithm for learning Gaussian mixture models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 8, pp. 1344–1348, 2005.

[29] U. Fayyad, C. Reina, and P. S. Bradley, "Initialization of iterative refinement clustering algorithms," in *Proc. Int. Conf. Knowledge Discovery and Data Mining*, pp. 194–198, AAAI Press, 1998.

[30] N. Ueda, R. Nakano, Z. Ghahramani, and G. Hinton, "SMEM algorithm for mixture models," *Neural Computation*, vol. 12, no. 9, pp. 2109–2128, 2000.

[31] S. J. Young and P. Woodland, "State clustering in hidden Markov model-based continuous speech recognition," *Computer Speech and Language*, vol. 8, no. 4, pp. 369–383, 1994.

[32] N. Vlassis and A. Likas, "A greedy EM algorithm for Gaussian mixture learning," in *Neural Processing Letters*, pp. 77–87, 2002.

[33] C. Biernacki, G. Celeux, and G. Govaert, "Choosing starting values for the EM algorithm for getting the highest likelihood in multivariate Gaussian mixture models," *Comput. Stat. Data Anal.*, vol. 41, no. 3-4, pp. 561–575, 2003.

[34] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification.* Wiley-Interscience Publication, 2000.

[35] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning.* Springer, 2001.

[36] W. D. Furman and B. G. Lindsay, “Testing for the number of components in a mixture of normal distributions using moment estimators,” *Comput. Stat. Data Anal.*, vol. 17, no. 5, pp. 473–492, 1994.

[37] G. J. McLachlan and N. Khan, “On a resampling approach for tests on the number of clusters with mixture model-based clustering of tissue samples,” *Journal of Multivariate Analysis*, vol. 90, pp. 90–105, 2004.

[38] L. F. James, C. E. Priebe, and D. J. Marchette, “Consistent estimation of mixture complexity,” *The Annsls of Statisitcs*, vol. 29, no. 5, pp. 1281–1296, 2001.

[39] A. P. Benavent, F. E. Ruiz, and J. M. S. Martinez, “EBEM: An entropy-based EM algorithm for Gaussian mixture models,” in *Proc. Int. Conf. Pattern Recognition*, vol. 2, 2006.

[40] K. P. Burnham and D. R. Anderson, *Model Selection and Inference: A Practical Information-Theoretic Approach*. New York: Springer-Verlag, 1998.

[41] G. Schwarz, “Estimation the dimension of a model,” *Annals of Statistics*, vol. 6, no. 2, pp. 461–464, 1978.

[42] A. E. Raftery, “Bayesian model selection in social research,” *Sociological Methodology*, vol. 25, pp. 111–163, 1995.

[43] C. Biernacki, G. Celeux, and G. Govaert, “Assessing a mixture model for clustering with the integrated completed likelihood,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 7, pp. 719–725, 2000.

[44] J. Rissanen, “Modelling by shortest data description,” *Automatica*, vol. 14, pp. 465–471, 1978.

[45] J. J. Oliver, R. A. Baxter, and C. S. Wallace, “Unsupervised learning using MML,” in *Proc. 13th Int. Conf. Machine Learning*, pp. 364–372, 1996.

[46] A. Corduneanu and C. M. Bishop, “Variational Bayesian model selection for mixture distributions,” in *Proc. Int. Workshop on Artificial Intelligence and Statistics*, 2001.

[47] C. M. Bishop, *Pattern Recognition and Machine Learning*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[48] T. Kohonen, *Self-organizing Maps*. Springer, 2001.

[49] T. Kohonen, “The self-organizing maps,” *Neurocomputing*, vol. 21, no. 1-3, pp. 1–6, 1998.

[50] C. Bishop, M. Svensén, and C. Williams, “The generative topographic mapping,” *Neural Computation*, vol. 10, no. 1, pp. 215–234, 1998.

[51] V. V. Tolat, “An analysis of Kohonen’s self-organizing maps using a system of energy functions,” *Biological Cybernetics*, vol. 64, no. 2, pp. 155–164, 1990.

[52] E. Erwin, K. Obermayer, and K. Schulten, “Self-organizing maps: ordering, convergence properties and energy functions,” *Biological Cybernetics*, vol. 67, no. 1, pp. 47–55, 1992.

[53] Y. Cheng, “Convergence and ordering of Kohonen’s batch map,” *Neural Computation*, vol. 9, no. 8, pp. 1667–1676, 1997.

[54] S. P. Luttrell, “Self-organization: A derivation from first principles of a class of learning algorithm,” in *Proc. IEEE Int. Joint Conf. Neural Networks*, pp. II–495–II–498, 1989.

[55] S. P. Luttrell, “Code vector density in topographic mappings: Scalar case,” *IEEE Trans. Neural Networks*, vol. 2, no. 4, pp. 427–436, 1991.

[56] T. Graepel, M. Burger, and K. Obermayer, “Phase transitions in stochastic self-organization maps,” *Physical Review E*, vol. 56, no. 4, pp. 3876–3890, 1997.

[57] T. Graepel, M. Burger, and K. Obermayer, “Self-organizing maps: Generalizations and new optimization techniques,” *Neurocomputing*, vol. 21, 1998.

[58] T. Heskes, “Self-organizing maps, vector quantization, and mixture modeling,” *IEEE Trans. Neural Networks*, vol. 12, no. 6, pp. 1299–1350, 2001.

[59] T. W. S. Chow and S. Wu, “An online cellular probabilistic self-organizing map for static and dynamical data sets,” *IEEE Trans. Circuit and Systems, Part I*, vol. 51, no. 4, pp. 732–747, 2004.

[60] S. Wu and T. W. S. Chow, “Prsom: A new visualization method by hybridizing multidimensional scaling and self-organizing map,” *IEEE Trans. Neural Networks*, vol. 16, no. 6, pp. 1362–1380, 2005.

[61] S. P. Luttrell, “A Bayesian analysis of self-organizing maps,” *Neural Computation*, vol. 6, no. 5, pp. 767–794, 1994.

[62] F. Anouar, F. Badran, and S. Thiria, “Probabilistic self-organizing map and radial basis function networks,” *Neurocomputing*, vol. 20, no. 1-3, pp. 83–96, 1998.

[63] J. Lampinen and T. Kostiainen, “Generative probability density model in the self-organizing map,” in *Self-organizing neural networks: Recent advances and applications* (U. Seiffert and L. Jain, eds.), pp. 75–94, Physica Verlag, 2002.

[64] M. M. V. Hulle, "Joint entropy maximization in kernel-based topographic maps," *Neural Computation*, vol. 14, no. 8, pp. 1887–1906, 2002.

[65] M. M. V. Hulle, "Maximum likelihood topographic map formation," *Neural Computation*, vol. 17, no. 3, pp. 503–513, 2005.

[66] J. J. Verbeek, N. Vlassis, and B. J. A. Kröse, "Self-organizing mixture models," *Neurocomputing*, vol. 63, pp. 99–123, 2005.

[67] J. Sum, C. S. Leung, L. W. Chan, and L. Xu, "Yet another algorithm which can generate topography map," *IEEE Trans. Neural Networks*, vol. 8, no. 5, pp. 1204–1207, 1997.

[68] L. D. Consortium, *2001 NIST Speaker Recognition Evaluation Corpus.* http://www.ldc.upenn.edu/.

[69] D. A. Reynolds, "Speaker identification and verification using Gaussian mixture speaker models," *Speech Communication*, vol. 17, pp. 91–108, 1995.

[70] S. Chen and P. S. Gopalakrishnan, "Speaker, environment and channel change detection and clustering via the Bayesian information criterion," in *Proc. DARPA Broadcast News Transcription and Understanding Workshop*, (Lansdowne, VA), pp. 127–132, Feb. 1998.

[71] M. Cettolo, M. Vescovi, and R. Rizzi, "Evaluation of BIC-based algorithms for audio segmentation," *Computer Speech and Language*, vol. 19, pp. 147–170, 2005.

[72] S. S. Cheng, H. M. Wang, , and H. C. Fu, "A model-selection-based self-splitting Gaussian mixture learning with application to speaker identification," *EURASIP Journal on Applied Signal Processing*, vol. 2004, no. 17, pp. 2626–2639, 2004.

[73] S.-S. Cheng, H.-C. Fu, and H.-M. Wang, "Model-based clustering by probabilistic self-organizing maps," *To appear in IEEE Trans. Neural Network.*

[74] S.-S. Cheng, H.-M. Wang, and H.-C. Fu, "BIC-based audio segmentation by divide-and-conquer," in *Proc. Int. Conf. Acoustics, Speech and Signal Processing*, pp. 4841–4844, 2008.

[75] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.*, vol. 28, no. 1, pp. 84–95, 1980.

[76] S. C. Chu and J. Roddick, "Pattern clustering using incremental splitting for non-uniformly distributed data," in *Proc. Int. conf. Knowledge-Based Intelligent Information Engineering Systems and Allied Technologies*, pp. 1037–1041, 2001.

[77] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2002.

[78] T. Kohonen, S. Kaski, K. Lagus, J. Salojarvi, V. P. J. Honkela, and A. Saarela, "Self organization of a massive document collection," *IEEE Trans. Neural Networks: Special Issue on Neural Networks for Data Mining and Knowledge Discovery*, vol. 11, no. 3, pp. 574–585, 2000.

[79] K. Lagus, S. Kaski, and T. Kohonen, "Mining massive document collections by the WEBSOM method," *Information Sciences*, vol. 163, pp. 135–156, 2004.

[80] L. Gupta and S. Das, "Texture edge detection using multi-resolution features and SOM," in *Proc. IEEE Int. Conf. Pattern Recognition*, 2006.

[81] M. Kurimo, "Training mixture density HMMs with SOM and LVQ," *Computer Speech and Language*, vol. 11, no. 4, pp. 321–341, 1997.

[82] M. Kurimo, "Fast latent semantic indexing of spoken documents by using self-organizing maps," in *Proc. Int. Conf. Acoustics, Speech and Signal Processing*, pp. 2425–2428, 2000.

[83] M. Kurimo, "Thematic indexing of spoken documents by using self-organizing maps," *Speech Communication*, vol. 38, no. 1-2, pp. 29–44, 2002.

[84] B. D. Ripley, *Pattern Recognition and neural networks.* Cambridge University Press, 1996.

[85] R. E. Kass and A. E. Raftery, "Bayes factors," *Journal of the American Statistical Association*, vol. 90, no. 430, pp. 773–795, 1995.

[86] A. M. T. Tantrum and W. Stuetzle, "Hierarchical model-based clustering of large datasets through fractionation and refractionation," in *Proc. SIGKDD*, 2002.

[87] HTK, *Hidden Markov Model Toolkit.* http://htk.eng.cam.ac.uk/.

[88] B. L. Pellom and J. H. L. Hansen, "An efficient scoring algorithm for Gaussian mixture model based speaker identification," *Signal Processing Letters, IEEE*, vol. 5, no. 11, pp. 281–284, 1998.

[89] C. Miyajima, Y. Hattori, K. Tokuda, T. Masuko, T. Kobayashi, and T. Kitamura, "Speaker identification using Gaussian mixture models based on multi-space probability distribution," *Proc. of ICASSP*, vol. 84, pp. 847–855, 2001.

[90] L. Wang, K. Chen, and H. S. Chi, "Capture interspeaker information with a neural network for speaker identification," *IEEE Trans. Neural Networks*, vol. 13, pp. 436–445, 2002.

[91] F. Bimbot, J.-F. Bonastre, C. Fredouille, G. Gravier, I. Magrin-Chagnolleau, S. Meignier, T. Merlin, J. Ortega-García, D. Petrovska-Delacrétaz, and D. A. Reynolds, "A tutorial on text-independent speaker verification," *EURASIP J. Appl. Signal Process.*, vol. 2004, no. 1, pp. 430–451, 2004.

[92] *The VIMAS speech codec.* http://www.vimas.com.

[93] C. Ambroise and G. Govaert, "Constrained clustering and Kohonen self-organizing maps," *Journal of Classification*, vol. 13, no. 2, pp. 299–313, 1996.

[94] K. Rose, E. Gurewitz, and G. C. Fox, "Vector quantization by deterministic annealing," *IEEE Trans. Inform. Theory*, vol. 38, no. 4, pp. 1249–1257, 1992.

[95] K. Rose, "Deterministic annealing for clustering, compression, classification, regression, and related optimization problems," *Proceedings of The IEEE*, vol. 86, no. 11, pp. 2210–2239, 1998.

[96] A. Asuncion and D. Newman, *UCI Machine Learning Repository.* 2007.

[97] H. Meinedo and J. Neto, "Audio segmentation, classification and clustering in a broadcast news task," in *Proc. Int. Conf. Acoust., Speech, Signal Processing*, (Hong Kong, China), pp. II–5–II–8, Apr. 2003.

[98] P. C. Woodland, M. J. F. Gales, D. Pye, and S. J. Young, "The development of the 1996 HTK broadcast news transcription system," in *Proc. DARPA Speech Recognition Workshop*, (Lansdowne, VA), pp. 73–78, Feb. 1997.

[99] J. F. Bonastre, P. Delacourt, C. Fredouille, T. Merlin, and C. J. Wellekens, "A speaker tracking system based on speaker turn detection for NIST evaluation," in *Proc. Int. Conf. Acoust., Speech, Signal Processing*, (Istanbul, Turkey), pp. 1177–1180, Jun. 2000.

[100] S. E. Tranter and D. A. Reynolds, "An overview of automatic speaker diarization systems," *IEEE Trans. Audio, Speech and Language Processing*, vol. 14, no. 5, pp. 1557–1565, 2006.

[101] M. Siegler, U. Jain, B. Raj, and R. Stern, "Automatic segmentation, classification and clustering of broadcast news audio," in *Proc. DARPA Speech Recognition Workshop*, (Chantilly, VA), pp. 97–99, Feb. 1997.

[102] A. Tritschler and R. A. Gopinath, "Improved speaker segmentation and segments clustering using the Bayesian information criterion," in *Proc. Eur. Conf. Speech Communication and Technology*, (Budapest, Hungary), pp. 679–682, Sep. 1999.

[103] P. Sivakumaran, J. Fortuna, and A. M. Ariyaeeinia, "On the use of the Bayesian information criterion in multiple speaker detection," in *Proc. Eur. Conf. Speech Communication and Technology*, (Aalborg, Denmark), pp. 795–798, Sep. 2001.

[104] P. Delacourt and C. J. Wellekens, "DISTBIC: A speaker-based segmentation for audio data indexing," *Speech Communication*, vol. 32, no. 1, pp. 111–126, 2000.

[105] J. W. Hung, H. M. Wang, and L. S. Lee, "Automatic metric-based speech segmentation for broadcast news via principal component analysis," in *Proc. Int. Conf. Spoken Language Processing*, (Beijing, China), pp. 121–124, Oct. 2000.

[106] R. Bakis, S. Chen, P. S. Gopalakrishnan, R. Gopinath, S. Maes, L. Polymenakos, and M. Franz, "Transcription of broadcast news shows with the IBM large vocabulary speech recognition system," in *Proc. DARPA Speech Recognition Workshop*, (Chantilly, VA), pp. 67–72, Feb. 1997.

[107] B. W. Zhou and J. H. L. Hansen, "Efficient audio stream segmentation via the combined T^2 statistic and Bayesian information criterion," *IEEE Trans. Speech and Audio Processing*, vol. 13, no. 4, pp. 467–474, 2005.

[108] S. S. Cheng and H. M. Wang, "A sequential metric-based audio segmentation method via the Bayesian information criterion," in *Proc. Eur. Conf. Speech Communication and Technology*, (Geneva, Switzerland), pp. 945–948, Sep. 2003.

[109] H. M. Wang, B. Chen, J. W. Kuo, and S. S. Cheng, "MATBN: A Mandarin Chinese broadcast news corpus," *Int. Journal of Computational Linguistics and Chinese Language Processing*, vol. 10, no. 2, pp. 219–236, 2005.

[110] NIST, *1998 DARPA/NIST HUB-4 broadcast news evaluation test data.* http://www.nist.gov/speech/tests/bnr/hub4_98/hub4_98.htm.

[111] U. Manber, *Introduction to Algorithms: A Creative Approach.* Addison-Wesley, 1989.