Maximum Weighted Likelihood via Rival Penalized EM for Density Mixture Clustering with Automatic Model Selection

Yiu-ming Cheung, *Member*, *IEEE*

Abstract—Expectation-Maximization (EM) algorithm [10] has been extensively used in density mixture clustering problems, but it is unable to perform model selection automatically. This paper, therefore, proposes to learn the model parameters via maximizing a weighted likelihood. Under a specific weight design, we give out a Rival Penalized Expectation-Maximization (RPEM) algorithm, which makes the components in a density mixture compete each other at each time step. Not only are the associated parameters of the winner updated to adapt to an input, but also all rivals' parameters are penalized with the strength proportional to the corresponding posterior density probabilities. Compared to the EM algorithm [10], the RPEM is able to fade out the redundant densities from a density mixture during the learning process. Hence, it can automatically select an appropriate number of densities in density mixture clustering. We experimentally demonstrate its outstanding performance on Gaussian mixtures and color image segmentation problem. Moreover, a simplified version of RPEM generalizes our recently proposed RPCCL algorithm [8] so that it is applicable to elliptical clusters as well with any input proportion. Compared to the existing heuristic RPCL [25] and its variants, this generalized RPCCL (G-RPCCL) circumvents the difficult preselection of the so-called delearning rate. Additionally, a special setting of the G-RPCCL not only degenerates to RPCL and its Type A variant, but also gives a guidance to choose an appropriate delearning rate for them. Subsequently, we propose a stochastic version of RPCL and its Type A variant, respectively, in which the difficult selection problem of delearning rate has been novelly circumvented. The experiments show the promising results of this stochastic implementation.

Index Terms—Maximum weighted likelihood, rival penalized Expectation-Maximization algorithm, generalized rival penalization controlled competitive learning, cluster number, stochastic implementation.

1 Introduction

As a statistical tool, clustering analysis has been widely applied to a variety of scientific areas such as vector quantization [7], [15], data mining [18], image processing [14], [22], statistical data analysis [6], [13], and so forth. In the past, the clustering analysis has been extensively studied along two directions: 1) hierarchical clustering and 2) nonhierarchical clustering. In this paper, we concentrate on the latter only, which aims to partition N inputs (also called *observations* or *data points* interchangeably hereinafter), written as $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$, into k^* true clusters so that the inputs within the same cluster are similar under a measure, but those in different clusters are not.

Conventionally, the k-means [16] is a popular clustering algorithm that updates the seed points, i.e., those learnable points in the input space representing the cluster centers, via minimizing a mean-square-error function. The k-means has been widely used in a variety of applications, but it has at least two major drawbacks:

1. It implies that the shapes of data clusters are spherical because it performs clustering based on the Euclidean distance only.
2. It needs to preassign the cluster number k, which is an estimate of k^*. When k is exactly equal to k^*, the k-means algorithm can correctly find out the clustering centers as shown in Fig. 1b. Otherwise, it will lead to an incorrect clustering result as depicted in Figs. 1a and 1c, where some of the seed points are not located at the centers of the corresponding clusters. Instead, they are at some boundary points between different clusters or at points far away from the cluster centers.

In the literature, a broad view of the clustering problem has been formulated within the framework of density estimates [5], [17], [19], [21], in which the probability density of inputs is represented by a finite mixture model. Each mixture component represents the density distribution of a data cluster. Consequently, clustering can be viewed as identifying the dense regions of the input densities and therefore named *density mixture clustering*. In particular, it is called *Gaussian mixture clustering* when each mixture component is a Gaussian density. Often, the Expectation-Maximization (EM) algorithm [10], [12] provides a general solution for the parameter estimate in a density mixture model. Unfortunately, it also needs to preassign an appropriate number of densities analogous to the k-means algorithm. Otherwise, the EM will mostly give out a poor estimate result.

In the past decades, some works have been done toward determining the correct number of clusters or densities along two major lines. The first one is to formulate the cluster number selection as the choice of component number in a finite mixture model. Consequently, there have been some criteria proposed for model selection, such as AIC [2], [3], CAIC [4], and SIC [21]. However, these conventional criteria may overestimate or underestimate the cluster number due to the difficulty of choosing an

- *The author is with the Department of Computer Science, Rm. 709, 7/F, Sir Run Run Shaw Building, Hong Kong Baptist University, Kowloon Tong, Kowloon, Hong Kong, P.R. China. E-mail: ymc@comp.hkbu.edu.hk.*

Manuscript received 19 Mar. 2004; revised 28 Sept. 2004; accepted 13 Dec. 2004; published online 20 Apr. 2005.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0075-0304.

1041-4347/05/$20.00 © 2005 IEEE Published by the IEEE Computer Society

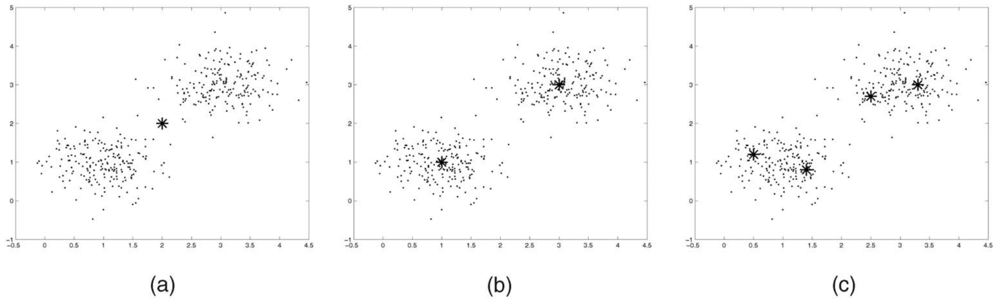

Fig. 1. The results from the *k*-means algorithm with (a) $k = 1$, (b) $k = 2$, and (c) $k = 4$, respectively, where "$*$" denotes the locations of converged seed points. It can be seen that the algorithm can find out the cluster centers correctly as shown in Fig. 1b when k is equal to the true cluster number $k^* = 2$. In contrast, if k is misspecified, the performance of the *k*-means algorithm will deteriorate seriously as shown in Figs. 1a and 1c, where some seed points do not locate at the centers of the corresponding clusters. Instead, they are either at some boundary points between different clusters or at points biased from some cluster centers.

appropriate penalty function. Recently, a new criteria from Ying-Yang Machine theory [23] has been presented as well for cluster number selection, which need not select an appropriate penalty function in general. Some empirical studies have found that it can determine a correct cluster number, whose computation is, however, laborious. The other approach is to introduce a mechanism into an algorithm so that an appropriate cluster number k can be automatically selected online. Hence, this direction generally gives a promising way to develop a robust clustering algorithm in terms of the cluster number.

In the literature, one example is called incremental clustering [11], [12] that gradually increases the number clusters under the control of an appropriate threshold value, which is, however, hard to be decided in advance as well. Another typical example is the heuristic Rival Penalized Competitive Learning (RPCL) algorithm and its variants [24], [25]. The basic idea in each of them is that, for each input, not only the winner of the seed points is updated to adapt to the input, but also its nearest rival (i.e., the second winner) is delearned by a much smaller fixed learning rate (also called *delearning rate* hereinafter). The empirical studies have shown that the RPCL can indeed automatically select the correct cluster number by gradually driving extra seed points far away from the input data set. However, some empirical studies have also found that its performance is sensitive to the selection of the delearning rate. If the rate is not well preselected, the RPCL may completely break down. To circumvent this awkward situation, we have recently proposed an improved version, namely, *Rival Penalization Controlled Competitive Learning* (RPCCL) [8], in which the rival is more penalized if its distance to the winner is smaller than the distance between the winner and the input, and vice versa. The RPCCL always fixes the delearning rate at the same value as the learning rate, which is, however, absolutely prohibited in the RPCL [26]. The experiments in [8] have shown the promising results. Nevertheless, as well as the RPCL and its variants, such a penalization scheme is still heuristically proposed without any theoretical guidance.

In this paper, we therefore propose to learn the model parameters via maximizing a weighted likelihood, which is developed from the likelihood function of inputs with a designable weight. Under a specific weight design, we then give out a maximum weighted likelihood (MWL) approach named *Rival Penalized Expectation-Maximization* (RPEM) algorithm, which makes the components in a density mixture compete with each other, and the rivals intrinsically penalized with a dynamic control during the learning. Not only are the associated parameters of the winner updated to adapt to an input, but also all rivals' parameters are penalized with the strength proportional to the corresponding posterior density probabilities. Compared to the EM, such a rival penalization mechanism enables the RPEM to fade out the redundant densities in the density mixture. In other words, the RPEM has the capability of automatically selecting an appropriate number of densities in density mixture clustering. The numerical simulations have demonstrated its outstanding performance on Gaussian mixtures and the color image segmentation problem. Moreover, we show that a simplified version of RPEM actually generalizes the RPCCL algorithm [8] so that it is applicable to ellipse-shaped clusters as well with any input proportion. Compared to the existing RPCL and its variants, this generalized RPCCL (G-RPCCL), as well as the RPCCL, circumvents the difficult preselection of the delearning rate. Additionally, a special setting of G-RPCCL further degenerates to the RPCL and its Type A version, but meanwhile giving out a guidance to choose an appropriate delearning rate. Subsequently, we propose a stochastic version of RPCL and its Type A variant, respectively, in which the difficult selection problem of the delearning rate is novelly circumvented. The experiments have shown the promising results of this stochastic implementation.

The remainder of this paper is organized as follows: Section 2 will overview the EM algorithm in the density mixture clustering. Section 3 goes into the details of describing the MWL learning framework, through which the RPEM algorithm is then proposed and experimentally demonstrated by using both of synthetic and real data. In Section 4, we will give out the details of G-RPCCL and show the relations between it and the existing RPCCL, RPCL, and its Type A variant, respectively. Subsequently, not only is a simple efficient method of choosing an appropriate delearning rate in the RPCL and its Type A variant suggested, but the stochastic implementations of RPCL and its Type A variant are also presented and demonstrated accordingly. Lastly, we draw a conclusion in Section 5.

2 Overview of EM Algorithm in Density Mixture Clustering

Suppose N observations: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ are independently and identically distributed (iid) from the following mixture-of-density model:

$$p(\mathbf{x}|\boldsymbol{\Theta}^*) = \sum_{j=1}^{k^*} \alpha_j^* p(\mathbf{x}|\theta_j^*) \tag{1}$$

with

$$\sum_{j=1}^{k} \alpha_j^* = 1 \qquad \text{and} \qquad \forall\, 1 \leq j \leq k^*, \quad \alpha_j^* > 0, \tag{2}$$

where $\boldsymbol{\Theta}^* = \{\alpha_j^*, \theta_j^*\}_{j=1}^{k^*}$ denotes the set of true model parameters and k^* is the mixture number of densities in the model. In (1), each mixture component $p(\mathbf{x}|\theta_j^*)$ is a multivariate probability density function (pdf) of $\mathbf{x}$ with the parameter θ_j^*, which can be interpreted as the pdf of those data points that form a corresponding cluster, say Cluster j, and the associated α_j^* is the proportion of data points that belong to Cluster j. Hence, if the parameter set $\boldsymbol{\Theta}^*$ is known, the clustering task becomes straightforward, which classifies an input $\mathbf{x}_t$ into a certain cluster based on the posterior probability of a density that $\mathbf{x}_t$ comes from, written as:

$$h(j|\mathbf{x}_t, \boldsymbol{\Theta}^*) = \frac{\alpha_j^* p(\mathbf{x}_t|\theta_j^*)}{\sum_{i=1}^{k^*} \alpha_i^* p(\mathbf{x}_t|\theta_i^*)}, \qquad 1 \leq j \leq k^*. \tag{3}$$

For instance, we can perform clustering by taking the winner-take-all rule, i.e., assign an input $\mathbf{x}_t$ to Cluster c if $c = \arg\max_j h(j|\mathbf{x}_t, \boldsymbol{\Theta}^*)$, or taking its soft version which assigns $\mathbf{x}_t$ to Cluster j with the probability $h(j|\mathbf{x}_t, \boldsymbol{\Theta}^*)$. Hence, how to estimate $\boldsymbol{\Theta}^*$ from a set of observations becomes a key issue in density mixture clustering. Hereinafter, we will concentrate on the parameter estimate of a density mixture, particularly when the true mixture number k^* is unknown.

To obtain an estimate of $\boldsymbol{\Theta}^*$, written as $\boldsymbol{\Theta} = \{\alpha_j, \theta_j\}_{j=1}^{k}$, Dempster et al. [10] assumed that each observation $\mathbf{x}_t$ is accompanied by its hidden label

$$\mathbf{x}_{t,h} = [x_{t,h}^{(1)}, x_{t,h}^{(2)}, \ldots, x_{t,h}^{(k)}]^T, \qquad 1 \leq t \leq N \tag{4}$$

with

$$x_{t,h}^{(j)} = \begin{cases} 1, & \text{if } \mathbf{x}_t \text{ is drawn from } p(\mathbf{x}|\theta_j) \\ 0, & \text{otherwise,} \end{cases} \qquad \text{and} \qquad \sum_{j=1}^{k} x_{t,h}^{(j)} = 1 \tag{5}$$

that shows which density $\mathbf{x}_t$ comes from, where T denotes the transpose operation of a matrix. Since each $\mathbf{x}_t$ is randomly drawn from one of k densities, it is therefore that $\mathbf{x}_{1,h}, \mathbf{x}_{2,h}, \ldots, \mathbf{x}_{N,h}$ are also iid. The paper [10] further assumes that each $\mathbf{x}_{t,h}$ is from a multinomial distribution consisting of one draw on k densities with probabilities $\alpha_1, \alpha_2, \ldots, \alpha_k$, respectively. That is, the marginal density distribution of $\mathbf{x}_{t,h}$ is

$$p(\mathbf{x}_{t,h}|\boldsymbol{\Theta}) = \prod_{j=1}^{k} (\alpha_j)^{x_{t,h}^{(j)}}. \tag{6}$$

Hence, the joint pdf of $\mathbf{x}_{1,h}, \mathbf{x}_{2,h}, \ldots, \mathbf{x}_{N,h}$ is:

$$p(\mathbf{x}_{1,h}, \mathbf{x}_{2,h}, \ldots, \mathbf{x}_{N,h}|\boldsymbol{\Theta}) = \prod_{t=1}^{N} p(\mathbf{x}_{t,h}|\boldsymbol{\Theta}) = \prod_{t=1}^{N}\prod_{j=1}^{k} (\alpha_j)^{x_{t,h}^{(j)}}. \tag{7}$$

Furthermore, supposing $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ are conditionally independent as given $\mathbf{x}_{1,h}, \mathbf{x}_{2,h}, \ldots, \mathbf{x}_{N,h}$, respectively, we then have

$$p(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N|\mathbf{x}_{1,h}, \mathbf{x}_{2,h}, \ldots, \mathbf{x}_{N,h}, \boldsymbol{\Theta}) = \prod_{t=1}^{N} p(\mathbf{x}_t|\mathbf{x}_{t,h}, \boldsymbol{\Theta}) \tag{8}$$

with

$$p(\mathbf{x}_t|\mathbf{x}_{t,h}, \boldsymbol{\Theta}) = \prod_{j=1}^{k} p(\mathbf{x}_t|\theta_j)^{x_{t,h}^{(j)}} \tag{9}$$

upon the fact that $\mathbf{x}_t$ exclusively depends on $\mathbf{x}_{t,h}$. Consequently, the joint pdf of the complete data:

$$\begin{aligned} \mathbf{X}_N &= (\mathbf{x}_1^T, \mathbf{x}_2^T, \ldots, \mathbf{x}_N^T)^T \quad \text{and} \\ \mathbf{X}_{N,h} &= (\mathbf{x}_{1,h}^T, \mathbf{x}_{2,h}^T, \ldots, \mathbf{x}_{N,h}^T)^T, \end{aligned} \tag{10}$$

is given by

$$p(\mathbf{X}_N, \mathbf{X}_{N,h}|\boldsymbol{\Theta}) = \prod_{t=1}^{N}\prod_{j=1}^{k} [\alpha_j p(\mathbf{x}_t|\theta_j)]^{x_{t,h}^{(j)}}. \tag{11}$$

Hence, the empirical log likelihood function of $\boldsymbol{\Theta}$ is:

$$\tilde{L}(\boldsymbol{\Theta}; \mathbf{X}_N) = \frac{1}{N}\sum_{t=1}^{N}\sum_{j=1}^{k} x_{t,h}^{(j)} [\ln \alpha_j + \ln p(\mathbf{x}_t|\theta_j)]. \tag{12}$$

Since those hidden labels $\mathbf{x}_{t,h}$s in (12) are unknown, the incomplete data theory [10] therefore treats them as missing data and replaces each $\mathbf{x}_{t,h}$ by its expected value conditioned on $\mathbf{x}_t$, i.e., $E(\mathbf{x}_{t,h}|\mathbf{x}_t, \boldsymbol{\Theta})$, which is actually the posterior density probability $h(j|\mathbf{x}_t, \boldsymbol{\Theta})$. Accordingly, given a specific k, an optimal solution of $\boldsymbol{\Theta}$ can be obtained through maximizing the following log likelihood function:

$$\begin{aligned} L(\boldsymbol{\Theta}; \mathbf{X}_N) &= \frac{1}{N}\sum_{t=1}^{N} \iota_t(\boldsymbol{\Theta}; \mathbf{x}_t) \\ &= \frac{1}{N}\sum_{t=1}^{N}\sum_{j=1}^{k} h(j|\mathbf{x}_t, \boldsymbol{\Theta}) \ln[\alpha_j p(\mathbf{x}_t|\theta_j)] \end{aligned} \tag{13}$$

with

$$\iota_t(\boldsymbol{\Theta}; \mathbf{x}_t) = \sum_{j=1}^{k} h(j|\mathbf{x}_t, \boldsymbol{\Theta}) \ln[\alpha_j p(\mathbf{x}_t|\theta_j)], \tag{14}$$

where $\alpha_j > 0$ for $\forall\, 1 \leq j \leq k$ and $\sum_{j=1}^{k} \alpha_j = 1$.

In implementation, maximizing (13) can be achieved by the Expectation-Maximization (EM) algorithm [10]. Alternatively, we can also learn $\boldsymbol{\Theta}$ via an adaptive version of EM. That is, after assigning some initial value to $\boldsymbol{\Theta}$, we perform the following two steps at each time step t:

1. **E-Step**. We fix $\boldsymbol{\Theta}^{(\text{old})}$ and calculate

$$\begin{aligned} h(j|\mathbf{x}_t, \boldsymbol{\Theta}^{(\text{old})}) &= \frac{\alpha_j^{(\text{old})} p(\mathbf{x}_t|\theta_j^{(\text{old})})}{p(\mathbf{x}_t|\boldsymbol{\Theta}^{(\text{old})})} \\ &= \frac{\alpha_j^{(\text{old})} p(\mathbf{x}_t|\theta_j^{(\text{old})})}{\sum_{i=1}^{k} \alpha_i^{(\text{old})} p(\mathbf{x}_t|\theta_i^{(\text{old})})}, \end{aligned} \tag{15}$$

where $j = 1, 2, \ldots, k$.

2. **M-Step**. Fixing $h(j|\mathbf{x}_t, \boldsymbol{\Theta}^{(\text{old})})$s calculated in (15), we use a stochastic gradient ascent method to update $\boldsymbol{\Theta}$, i.e., update $\boldsymbol{\Theta}$ with a small step toward the direction of maximizing (14). Subsequently, we have

$$\boldsymbol{\Theta}^{(\text{new})} = \boldsymbol{\Theta}^{(\text{old})} + \eta \frac{\partial l_t(\boldsymbol{\Theta}; \mathbf{x}_t)}{\partial \boldsymbol{\Theta}} |_{\boldsymbol{\Theta}^{(\text{old})}}, \qquad (16)$$

where η is a small positive learning step.

The above two steps are performed for each input until $\boldsymbol{\Theta}$ converges. In general, this adaptive EM algorithm, as well as the original one in [10], does not contain a mechanism for automatic model section, i.e., a mechanism to select an appropriate value of k. As a result, $\boldsymbol{\Theta}^*$ can be well-estimated only when k happens to be equal to the unknown true value k^*. Otherwise, the EM will mostly give out a poor estimation.

3 Maximum Weighted Likelihood and Rival Penalized EM Algorithm

3.1 A General MWL Learning Framework

Recall that an input $\mathbf{x}$ comes from the mixture-of-density model of (1), the maximum likelihood (ML) estimate $\boldsymbol{\Theta}$ of $\boldsymbol{\Theta}^*$ can be obtained via maximizing the following cost function

$$\ell(\boldsymbol{\Theta}) = \int \ln\ p(\mathbf{x}|\boldsymbol{\Theta}) dF(\mathbf{x}), \qquad (17)$$

with

$$\begin{aligned} &p(\mathbf{x}|\boldsymbol{\Theta}) = \sum_{j=1}^{k} \alpha_j p(\mathbf{x}|\theta_j), \\ &\sum_{j=1}^{k} \alpha_j = 1, \text{ and} \\ &\forall\, 1 \leq j \leq k, \quad \alpha_j > 0, \end{aligned} \qquad (18)$$

where $F(\mathbf{u}) = \int_{-\infty}^{\mathbf{u}} p(\mathbf{x}) d\mathbf{x}$ is the cumulative probability function of $\mathbf{x}$. Hereinafter, we suppose that k is not smaller than k^* and $p(\mathbf{x}|\boldsymbol{\Theta})$ is an identifiable model, i.e., for any two possible values of $\boldsymbol{\Theta}$, denoted as $\boldsymbol{\Theta}_1$ and $\boldsymbol{\Theta}_2$, $p(\mathbf{x}|\boldsymbol{\Theta}_1) = p(\mathbf{x}|\boldsymbol{\Theta}_2)$ if and only if $\boldsymbol{\Theta}_1 = \boldsymbol{\Theta}_2$. It can be seen that (17) can be further represented as

$$\begin{aligned} \ell(\boldsymbol{\Theta}) &= \int \ln p(\mathbf{x}|\boldsymbol{\Theta}) dF(\mathbf{x}) \\ &= \int \sum_{j=1}^{k} g(j|\mathbf{x}, \boldsymbol{\Theta}) \ln p(\mathbf{x}|\boldsymbol{\Theta}) dF(\mathbf{x}) \\ &= \int [g(1|\mathbf{x}, \boldsymbol{\Theta}) \ln p(\mathbf{x}|\boldsymbol{\Theta}) + g(2|\mathbf{x}, \boldsymbol{\Theta}) \ln p(\mathbf{x}|\boldsymbol{\Theta}) \\ &\quad + \ldots, + g(k|\mathbf{x}, \boldsymbol{\Theta}) \ln p(\mathbf{x}|\boldsymbol{\Theta})] dF(\mathbf{x}), \end{aligned} \qquad (19)$$

where $g(j|\mathbf{x}, \boldsymbol{\Theta})$s are the designable weights satisfying

$$\sum_{j=1}^{k} g(j|\mathbf{x}, \boldsymbol{\Theta}) = 1, \qquad (20)$$

in which each $g(j|\mathbf{x}, \boldsymbol{\Theta})$ is generally a function of $\mathbf{x}$ and $\boldsymbol{\Theta}$. By Baye's formula, we know that the probability of $\mathbf{x}$ coming from the jth density as given $\mathbf{x}$ is

$$h(j|\mathbf{x}, \boldsymbol{\Theta}) = \frac{\alpha_j p(\mathbf{x}|\theta_j)}{p(\mathbf{x}|\boldsymbol{\Theta})}. \qquad (21)$$

Subsequently, we have

$$p(\mathbf{x}|\boldsymbol{\Theta}) = \frac{\alpha_j p(\mathbf{x}|\theta_j)}{h(j|\mathbf{x}, \boldsymbol{\Theta})}, \quad \text{for} \quad \forall j, 1 \leq j \leq k \qquad (22)$$

as long as $h(j|\mathbf{x}, \boldsymbol{\Theta})$ is not equal to zero. Putting (22) into (19), we obtain

$$\begin{aligned} \ell(\boldsymbol{\Theta}) &= \int \Big[g(1|\mathbf{x}, \boldsymbol{\Theta}) \ln p(\mathbf{x}|\boldsymbol{\Theta}) + g(2|\mathbf{x}, \boldsymbol{\Theta}) \ln p(\mathbf{x}|\boldsymbol{\Theta}) \\ &\quad + \ldots, + g(k|\mathbf{x}, \boldsymbol{\Theta}) \ln p(\mathbf{x}|\boldsymbol{\Theta})] dF(\mathbf{x}) \\ &= \int [g(1|\mathbf{x}, \boldsymbol{\Theta}) \ln \frac{\alpha_1 p(\mathbf{x}|\theta_1)}{h(1|\mathbf{x}, \boldsymbol{\Theta})} + g(2|\mathbf{x}, \boldsymbol{\Theta}) \ln \frac{\alpha_2 p(\mathbf{x}|\theta_2)}{h(2|\mathbf{x}, \boldsymbol{\Theta})} \\ &\quad + \ldots, + g(k|\mathbf{x}, \boldsymbol{\Theta}) \ln \frac{\alpha_k p(\mathbf{x}|\theta_k)}{h(k|\mathbf{x}, \boldsymbol{\Theta})} \Big] dF(\mathbf{x}) \\ &= \int \sum_{j=1}^{k} g(j|\mathbf{x}, \boldsymbol{\Theta}) \ln \frac{\alpha_j p(\mathbf{x}|\theta_j)}{h(j|\mathbf{x}, \boldsymbol{\Theta})} dF(\mathbf{x}) \\ &= \int \sum_{j=1}^{k} g(j|\mathbf{x}, \boldsymbol{\Theta}) \ln[\alpha_j p(\mathbf{x}|\theta_j)] dF(\mathbf{x}) \\ &\quad - \int \sum_{j=1}^{k} g(j|\mathbf{x}, \boldsymbol{\Theta}) \ln h(j|\mathbf{x}, \boldsymbol{\Theta}) dF(\mathbf{x}). \end{aligned} \qquad (23)$$

We name (23) Weighted Likelihood (WL) function. We then have the following result:

Theorem 1. *If $p(\mathbf{x}|\boldsymbol{\Theta})$ is an identifiable model, (23) reaches the global maximum if and only if $\boldsymbol{\Theta} = \boldsymbol{\Theta}^*$.*

Proof. By (17), we know that

$$\begin{aligned} \max_{\boldsymbol{\Theta}} \ell(\boldsymbol{\Theta}) &= \max_{\boldsymbol{\Theta}} \int p(\mathbf{x}) \ln p(\mathbf{x}|\boldsymbol{\Theta}) d\mathbf{x} \\ &\Leftrightarrow \min_{\boldsymbol{\Theta}} [- \int p(\mathbf{x}) \ln p(\mathbf{x}|\boldsymbol{\Theta}) d\mathbf{x}] \\ &\Leftrightarrow \min_{\boldsymbol{\Theta}} \Big[\int p(\mathbf{x}) \ln p(\mathbf{x}|\boldsymbol{\Theta}^*) d\mathbf{x} \\ &\quad - \int p(\mathbf{x}) \ln p(\mathbf{x}|\boldsymbol{\Theta}) d\mathbf{x} \Big] \\ &= \min_{\boldsymbol{\Theta}} \int p(\mathbf{x}) \ln \frac{p(\mathbf{x}|\boldsymbol{\Theta}^*)}{p(\mathbf{x}|\boldsymbol{\Theta})} d\mathbf{x}, \end{aligned} \qquad (24)$$

where $A \Leftrightarrow B$ means that A is equivalent to B.

Since $p(\mathbf{x}) = p(\mathbf{x}|\boldsymbol{\Theta}^*)$, we then have

$$\max_{\boldsymbol{\Theta}} \ell(\boldsymbol{\Theta}) \Leftrightarrow \min_{\boldsymbol{\Theta}} \Big[\int p(\mathbf{x}|\boldsymbol{\Theta}^*) \ln \frac{p(\mathbf{x}|\boldsymbol{\Theta}^*)}{p(\mathbf{x}|\boldsymbol{\Theta})} d\mathbf{x} \Big] \geq 0, \qquad (25)$$

in which "=" is held if and only if $p(\mathbf{x}|\boldsymbol{\Theta}^*) = p(\mathbf{x}|\boldsymbol{\Theta})$ is based on the property of Kullback-Leibler divergence as shown in the later Lemma 2. That is, "=" is held if and only if $\boldsymbol{\Theta} = \boldsymbol{\Theta}^*$ because $p(\mathbf{x}|\boldsymbol{\Theta})$ is an identifiable model.□

As N is large enough, the empirical WL function of (23), written as $Q(\boldsymbol{\Theta}; \mathbf{X}_N)$, can be further given as

$$\begin{aligned} Q(\boldsymbol{\Theta};\mathbf{X}_N) = &\frac{1}{N}\sum_{t=1}^{N}\sum_{j=1}^{k} g(j|\mathbf{x}_t,\boldsymbol{\Theta})\ln[\alpha_j p(\mathbf{x}_t|\theta_j)] \\ &-\frac{1}{N}\sum_{t=1}^{N}\sum_{j=1}^{k} g(j|\mathbf{x}_t,\boldsymbol{\Theta})\ln h(j|\mathbf{x}_t,\boldsymbol{\Theta}), \end{aligned} \tag{26}$$

which is the sum of two terms. The first term is actually a generalized version of the EM cost function in (13). It uses the general $g(j|\mathbf{x}_t,\boldsymbol{\Theta})$ to approximate the hidden label $x_{t,h}^{(j)}$, but not $h(j|\mathbf{x}_t,\boldsymbol{\Theta})$. Evidently, this term degenerates to the EM cost function when $g(j|\mathbf{x}_t,\boldsymbol{\Theta})$ is equal to $h(j|\mathbf{x}_t,\boldsymbol{\Theta})$ for any j. The second term is a kind of measure to evaluate the uncertainty of which density that the input $\mathbf{x}_t$ comes from. For instance, as $g(j|\mathbf{x}_t,\boldsymbol{\Theta}) \equiv h(j|\mathbf{x}_t,\boldsymbol{\Theta})$ for $\forall j, t$, the second term is exactly the conditional entropy of the densities. In general, the learning of $\boldsymbol{\Theta}$ toward maximizing the first term of (26) is to reduce such an uncertainty, but the learning of maximizing (26) is to increase the value of second term. In other words, the second term is serving as a regularization term in the learning of $\boldsymbol{\Theta}$. In (23) and (26), we have not considered the case that $h(j|\mathbf{x}_t,\boldsymbol{\Theta}) = 0$ for some j as given an input $\mathbf{x}_t$. Clearly, if $h(j|\mathbf{x}_t,\boldsymbol{\Theta}) = 0$ holds for some j, the maximum value of $Q(\boldsymbol{\Theta};\mathbf{X}_N)$ in (26) may not exist. To avoid this awkward situation, we therefore further request

$$\forall\, j, \quad g(j|\mathbf{x}_t,\boldsymbol{\Theta}) = 0 \quad \text{if} \quad h(j|\mathbf{x}_t,\boldsymbol{\Theta}) = 0 \tag{27}$$

in designing $g(j|\mathbf{x}_t,\boldsymbol{\Theta})$, which has a variety of choices as long as the loose conditions stated in (20) and (27) are satisfied. For instance, given an input $\mathbf{x}_t$, we can let $g(j|\mathbf{x}_t,\boldsymbol{\Theta})$ be some probability function, i.e., $\sum_{j=1}^{k} g(j|\mathbf{x}_t,\boldsymbol{\Theta}) = 1$ and $g(j|\mathbf{x}_t,\boldsymbol{\Theta}) \geq 0$ for any $1 \leq j \leq k$. A typical example is to let $g(j|\mathbf{x}_t,\boldsymbol{\Theta}) = h(j|\mathbf{x}_t,\boldsymbol{\Theta})$ or

$$I(j|\mathbf{x}_t,\boldsymbol{\Theta}) = \begin{cases} 1, & \text{if } j = c = \arg\max_{1\leq i\leq k} h(i|\mathbf{x}_t,\boldsymbol{\Theta}) \\ 0, & \text{otherwise.} \end{cases} \tag{28}$$

In the first design, (26) degenerates to the Kullback-Leibler divergence function derived from Ying-Yang Machine with the backward architecture, e.g., see [23]. In contrast, the second design leads (26) to be the cost function of hard-cut EM [23]. In the subsequent sections, we will prefer to investigate one specific $g(j|\mathbf{x}_t,\boldsymbol{\Theta})$ only with

$$g(j|\mathbf{x}_t,\boldsymbol{\Theta}) = (1+\xi_t)\varphi(j|\mathbf{x}_t,\boldsymbol{\Theta}) - \xi_t h(j|\mathbf{x}_t,\boldsymbol{\Theta}), \tag{29}$$

where $\varphi(j|\mathbf{x}_t,\boldsymbol{\Theta})$ is a special probability function, named *indicator function*, i.e., given any input $\mathbf{x}_t$, we have $\phi(j|\mathbf{x}_t,\boldsymbol{\Theta}) = 0$ or 1 for $1 \leq j \leq k$ and $\sum_{j=1}^{k}\varphi(j|\mathbf{x}_t,\boldsymbol{\Theta}) = 1$. Furthermore, ξ_t is generally a coefficient varying with the time step t. Hereinafter, we set ξ_t at 1 with $t = 1, 2, \ldots, N$ for simplicity. That is, (29) becomes

$$g(j|\mathbf{x}_t,\boldsymbol{\Theta}) = 2\varphi(j|\mathbf{x}_t,\boldsymbol{\Theta}) - h(j|\mathbf{x}_t,\boldsymbol{\Theta}). \tag{30}$$

After designing the weights, the learning of $\boldsymbol{\Theta}$ can then be accomplished toward maximizing (26). We therefore name such a learning as *Maximum Weighted Likelihood* (MWL) learning approach.

3.2 Rival Penalized EM Algorithm

By considering the specific weights in (30) and putting them into (26), we then have

$$Q(\boldsymbol{\Theta};\mathbf{X}_N) = \frac{1}{N}\sum_{t=1}^{N} q_t(\boldsymbol{\Theta};\mathbf{x}_t) \tag{31}$$

with

$$q_t(\boldsymbol{\Theta};\mathbf{x}_t) = R_t(\boldsymbol{\Theta};\mathbf{x}_t) + H_t(\boldsymbol{\Theta};\mathbf{x}_t) \tag{32}$$

and

$$R_t(\boldsymbol{\Theta};\mathbf{x}_t) = \sum_{j=1}^{k}[2\varphi(j|\mathbf{x}_t,\boldsymbol{\Theta}) - h(j|\mathbf{x}_t,\boldsymbol{\Theta})]\ln[\alpha_j p(\mathbf{x}_t|\theta_j)] \tag{33}$$

$$H_t(\boldsymbol{\Theta};\mathbf{x}_t) = -\sum_{j=1}^{k}[2\varphi(j|\mathbf{x}_t,\boldsymbol{\Theta}) - h(j|\mathbf{x}_t,\boldsymbol{\Theta})]\ln h(j|\mathbf{x}_t,\boldsymbol{\Theta}), \tag{34}$$

where $q_t(\boldsymbol{\Theta};\mathbf{x}_t)$ is called an instantaneous cost function at time step t because its value depends on $\boldsymbol{\Theta}$ and the current input $\mathbf{x}_t$ only. Before estimating $\boldsymbol{\Theta}$ via maximizing $Q(\boldsymbol{\Theta};\mathbf{X}_N)$ in (31) , we need to specify $\varphi(j|\mathbf{x}_t,\boldsymbol{\Theta})$. One choice is to simply let

$$\varphi(j|\mathbf{x}_t,\boldsymbol{\Theta}) = I(j|\mathbf{x}_t,\boldsymbol{\Theta}) \tag{35}$$

as that of (28). It should be noted that, if the number of maximum values of $h(j|\mathbf{x},\boldsymbol{\Theta})$s is more than one, we can randomly select one index among them as c and let $\varphi(c|\mathbf{x}_t,\boldsymbol{\Theta}) = 1$; meanwhile, the others are equal to zero. Subsequently, we can always guarantee $\varphi(j|\mathbf{x}_t,\boldsymbol{\Theta})$ to be an indicator function. As a result, as well as the adaptive EM algorithm in Section 2, we can learn $\boldsymbol{\Theta}$ via maximizing (31) adaptively. That is, after assigning some initial value to $\boldsymbol{\Theta}$, we perform the following two steps as given an input $\mathbf{x}_t$:

1. **Step A.1**. Fixing $\boldsymbol{\Theta}^{(\text{old})}$, we compute $h(j|\mathbf{x}_t,\boldsymbol{\Theta}^{(\text{old})})$ and $\varphi(j|\mathbf{x}_t,\boldsymbol{\Theta}^{(\text{old})})$ via (21) and (35), respectively.
2. **Step A.2**. Fixing $h(j|\mathbf{x}_t,\boldsymbol{\Theta})$s calculated in **Step A.1**, we update $\boldsymbol{\Theta}$ with a small step towards the direction of maximizing (32). To avoid the constraint on α_js during the optimization, we therefore let α_js be the soft-max function of k new free variables β_js with

$$\alpha_j = \frac{\exp(\beta_j)}{\sum_{i=1}^{k}\exp(\beta_i)}, \quad \text{for} \quad 1 \leq j \leq k, \tag{36}$$

and update β_js directly instead of α_js. As a result, we update $\boldsymbol{\Theta}$ by

$$\begin{aligned} \beta_c^{(\text{new})} &= \beta_c^{(\text{old})} + \eta_\beta \frac{\partial q_t(\boldsymbol{\Theta};\mathbf{x}_t)}{\partial \beta_c}\big|_{\boldsymbol{\Theta}^{(\text{old})}} \\ &= \beta_c^{(\text{old})} + \eta_\beta[2 - h(c|\mathbf{x}_t,\boldsymbol{\Theta}^{(\text{old})}) - \alpha_c^{(\text{old})}], \end{aligned} \tag{37}$$

$$\begin{aligned} \theta_c^{(\text{new})} &= \theta_c^{(\text{old})} + \eta \frac{\partial q_t(\boldsymbol{\Theta};\mathbf{x}_t)}{\partial \theta_c}\big|_{\boldsymbol{\Theta}^{(\text{old})}} \\ &= \theta_c^{(\text{old})} + \eta[2 - h(c|\mathbf{x}_t,\boldsymbol{\Theta}^{(\text{old})})]\frac{\partial \ln p(\mathbf{x}_t|\theta_c)}{\partial \theta_c}\big|_{\theta_c^{(\text{old})}}, \end{aligned} \tag{38}$$

meanwhile

$$\begin{aligned}\beta_r^{(\text{new})} &= \beta_r^{(\text{old})} + \eta_\beta \frac{\partial q_t(\mathbf{\Theta}; \mathbf{x}_t)}{\partial \beta_r}|_{\mathbf{\Theta}^{(\text{old})}} \\ &= \beta_r^{(\text{old})} - \eta_\beta [h(r|\mathbf{x}_t, \mathbf{\Theta}^{(\text{old})}) + \alpha_r^{(\text{old})}],\end{aligned} \tag{39}$$

$$\begin{aligned}\theta_r^{(\text{new})} &= \theta_r^{(\text{old})} + \eta \frac{\partial q_t(\mathbf{\Theta}; \mathbf{x}_t)}{\partial \theta_r}|_{\mathbf{\Theta}^{(\text{old})}} \\ &= \theta_r^{(\text{old})} - \eta h(r|\mathbf{x}_t, \mathbf{\Theta}^{(\text{old})}) \frac{\partial \ln p(\mathbf{x}_t|\theta_r)}{\partial \theta_r}|_{\theta_r^{(\text{old})}},\end{aligned} \tag{40}$$

where $\alpha_j^{(\text{old})}$ is computed via (36) in terms of $\beta_j^{(\text{old})}$, c is given by (28), and $r = 1, 2, \ldots, k$ but $r \neq c$. Furthermore, the positive η_β is the learning rate of β_js. We should let $\eta_\beta \ll \eta$ upon the sensitivity of α_js to the small changes of β_js in (36).

The above two steps are implemented for each input until $\mathbf{\Theta}$ converges. It can be seen that, at each time step t, **Step A.2** not only updates the associated parameters of the winning mixture component, i.e., the cth one, to adapt to the input, but all of those rival components are also penalized toward minimizing $q_t(\mathbf{\Theta}; \mathbf{x}_t)$ with the force strength proportional to $h(r|\mathbf{x}_t, \mathbf{\Theta})$s, respectively. The larger the $h(r|\mathbf{x}_t, \mathbf{\Theta})$ is, the stronger the penalized force is. We therefore name this algorithm *Rival Penalized EM* (RPEM), whose intrinsic rival-penalized mechanism enables the RPEM to fade out the redundant components from a density mixture as shown in the later Section 3.3.

To show that the learning of $\mathbf{\Theta}$ via the RPEM does converge, we regard $h(j|\mathbf{x}_t, \mathbf{\Theta})$ in (32) as a free probability function (i.e., it can be specified as any probability function), denoted as $\tilde{h}(j|\mathbf{x}_t)$. Under the circumstances, we let

$$\tilde{\varphi}(j|\mathbf{x}_t, \mathbf{\Theta}) = \begin{cases} 1, & \text{if } j = c = \arg\max_{1 \leq i \leq k} \left\{ h(i|\mathbf{x}_t, \mathbf{\Theta}) | \frac{\tilde{h}(i|\mathbf{x}_t)}{h(i|\mathbf{x}_t, \mathbf{\Theta})} \geq 1 \right\} \\ 0, & \text{otherwise,} \end{cases} \tag{41}$$

where $h(j|\mathbf{x}_t, \mathbf{\Theta})$ is still given by (21). Subsequently, (32) becomes

$$\begin{aligned} q_t(\mathbf{\Theta}; \mathbf{x}_t) &= \sum_{j=1}^{k} \tilde{g}(j|\mathbf{x}_t, \mathbf{\Theta}) \ln[\alpha_j p(\mathbf{x}_t|\theta_j)] \\ &\quad - \sum_{j=1}^{k} \tilde{g}(j|\mathbf{x}_t, \mathbf{\Theta}) \ln \tilde{h}(j|\mathbf{x}_t) \end{aligned} \tag{42}$$

with

$$\tilde{g}(j|\mathbf{x}_t, \mathbf{\Theta}) = 2\tilde{\varphi}(j|\mathbf{x}_t, \mathbf{\Theta}) - h(j|\mathbf{x}_t, \mathbf{\Theta}), \tag{43}$$

in which there are two independent parameters: $\mathbf{\Theta}$ and $\tilde{h}(j|\mathbf{x}_t)$s. Hence, their updates toward the direction of maximizing $q_t(\mathbf{\Theta}; \mathbf{x}_t)$ can be performed in a zig-zag way at each time step. That is, we first fix $\mathbf{\Theta}$ and update $\tilde{h}(j|\mathbf{x}_t)$ as given an input $\mathbf{x}_t$, followed by fixing $\tilde{h}(j|\mathbf{x}_t)$ and updating $\mathbf{\Theta}$. The details are as follows:

1. **Step B.1**. Given an input $\mathbf{x}_t$, we fix $\mathbf{\Theta}^{(\text{old})}$ and compute $\tilde{h}(j|\mathbf{x}_t)$ via maximizing (42), whereby $\tilde{\varphi}(j|\mathbf{x}_t, \mathbf{\Theta}^{(\text{old})})$ is calculated via (41).
2. **Step B.2**. Fixing those $\tilde{h}(j|\mathbf{x}_t)$s calculated in **Step B.1**, we update $\mathbf{\Theta}$ with a small step toward the direction of maximizing (42) similar to the previous **Step A.2**, i.e., update

$$\beta_c^{(\text{new})} = \beta_c^{(\text{old})} + \eta_\beta \Big[2 - \tilde{h}(c|\mathbf{x}_t) - \alpha_c^{(\text{old})} \Big], \tag{44}$$

$$\theta_c^{(\text{new})} = \theta_c^{(\text{old})} + \eta \big[2 - \tilde{h}(c|\mathbf{x}_t) \big] \frac{\partial \ln p(\mathbf{x}_t|\theta_c)}{\partial \theta_c}|_{\theta_c^{(\text{old})}}, \tag{45}$$

and

$$\beta_r^{(\text{new})} = \beta_r^{(\text{old})} - \eta_\beta \Big[\tilde{h}(r|\mathbf{x}_t) + \alpha_r^{(\text{old})} \Big] \tag{46}$$

$$\theta_r^{(\text{new})} = \theta_r^{(\text{old})} - \eta \tilde{h}(r|\mathbf{x}_t) \frac{\partial \ln p(\mathbf{x}_t|\theta_r)}{\partial \theta_r}|_{\theta_r^{(\text{old})}}, \tag{47}$$

where $r = 1, 2, \ldots, k$, and but $r \neq c$.

In the above, **Step B.1** and **Step B.2** both always increase the value of $q_t(\mathbf{X}_N; \mathbf{\Theta})$, the convergence of $\mathbf{\Theta}$ learning via this algorithm is therefore guaranteed. It can be seen that $\tilde{\varphi}(j|\mathbf{x}_t, \mathbf{\Theta})$ defined in (41) will be equal to $\varphi(j|\mathbf{x}_t, \mathbf{\Theta})$ in (35) as long as $\tilde{h}(j|\mathbf{x}_t) = h(j|\mathbf{x}_t, \mathbf{\Theta})$. Under the circumstances, this alternative algorithm will be the same as the previous RPEM. In the following, we will show that $\tilde{h}(j|\mathbf{x}_t) = h(j|\mathbf{x}_t, \mathbf{\Theta})$ is exactly the maximum point of (42) when $\mathbf{\Theta}$ is fixed in **Step B.1**. Before giving out the theorem, we first present two useful lemmas as follows.

Lemma 2. *Given any two probability density functions of a continuous-valued variable* $\mathbf{u}$, *denoted as* $p_1(\mathbf{u})$ *and* $p_2(\mathbf{u})$, *respectively, based on the Kullback-Leibler divergence property, we have*

$$\int p_1(\mathbf{u}) \ln \frac{p_1(\mathbf{u})}{p_2(\mathbf{u})} d\mathbf{u} \geq 0, \tag{48}$$

and "=" is held if and only if $p_1(\mathbf{u}) = p_2(\mathbf{u})$.

If $\mathbf{x}$ takes one of k discrete values only, the conclusion of Lemma 2 is also true, in which (48) becomes

$$\sum_{j=1}^{k} p_1(\mathbf{u}_j) \ln \frac{p_1(\mathbf{u}_j)}{p_2(\mathbf{u}_j)} \geq 0, \tag{49}$$

where $\mathbf{u}_j$ denotes the jth possible value of $\mathbf{u}$, and $p_1(\mathbf{u})$ and $p_2(\mathbf{u})$ are two probability functions, but not the pdfs anymore.

Lemma 3. *Suppose the discrete variable* ν *takes one of* k *possible values:* $1, 2, \ldots, k$. *Given any two probability functions, denoted as* $p_1(\nu)$ *and* $p_2(\nu)$, *we define a function*

$$\phi(\nu = j) = \begin{cases} 1, & \text{if } j = c = \arg\max_{1 \leq i \leq k} \Big\{ p_2(\nu = i) | \frac{p_1(\nu=i)}{p_2(\nu=i)} \geq 1 \Big\} \\ 0, & \text{otherwise,} \end{cases} \tag{50}$$

where we can randomly choose one in case such a c *is not unique. Then, we have*

$$\begin{aligned}&\sum_{j=1}^{k}\phi(\nu=j)=1, \quad \text{and}\\ &\phi(\nu=j)=0 \text{ for } \forall\, 1\le j\le k,\ j\neq c.\end{aligned} \tag{51}$$

Proof. Given any two probability functions $p_1(\nu)$ and $p_2(\nu)$ with ν taking one of k possible values: $1, 2, \ldots, k$, there must exist at least one j so that $\frac{p_1(\nu=j)}{p_2(\nu=j)}\ge 1$. Otherwise, we have

$$\sum_{i=1}^{k}p_1(\nu=i)<\sum_{i=1}^{k}p_2(\nu=i)=1 \tag{52}$$

which contradicts the $p_1(\nu)$'s property that

$$\sum_{i=1}^{k}p_1(\nu=i)=1.$$

Suppose there are m possible values of ν, denoted as $i_1, i_2, \ldots, i_m$ with $m\ge 1$, such that $\frac{p_1(\nu=i_\iota)}{p_2(\nu=i_\iota)}\ge 1$, where $1\le\iota\le m$. Without loss of generality, we suppose there is one unique maximum value, denoted as $p_2(\nu=i_c)$, among these m $p_2(\nu=i_\iota)$s. Otherwise, we can randomly take one from the several maximum values. By definition of function ϕ in (50), we therefore know that $\phi(\nu=c)=1$ and $\phi(\nu=j)=0$, where $j=1,2,\ldots,k$, but $j\neq c$. □

Subsequently, we present the theorem as follows.

Theorem 4. *When* $\boldsymbol{\Theta}$ *is fixed,* $q_t(\boldsymbol{\Theta};\mathbf{x}_t)$ *in (42) reaches the maximum value if and only if* $\tilde{h}(j|\mathbf{x}_t)$ *is equal to* $h(j|\mathbf{x}_t,\boldsymbol{\Theta})$ *of (21).*

Proof. We consider a specific function, denoted as $\bar{q}_t(\boldsymbol{\Theta};\mathbf{x}_t)$, with

$$\begin{aligned}\bar{q}_t(\boldsymbol{\Theta};\mathbf{x}_t)&=\sum_{j=1}^{k}[2\tilde{\varphi}(j|\mathbf{x}_t,\boldsymbol{\Theta})-h(j|\mathbf{x}_t,\boldsymbol{\Theta})]\ln\frac{\tilde{h}(j|\mathbf{x}_t)}{\alpha_j p(\mathbf{x}_t|\theta_j)}\\ &=\sum_{j=1}^{k}[2\tilde{\varphi}(j|\mathbf{x}_t,\boldsymbol{\Theta})-h(j|\mathbf{x}_t,\boldsymbol{\Theta})]\ln\frac{\tilde{h}(j|\mathbf{x}_t)}{h(j|\mathbf{x}_t,\boldsymbol{\Theta})}+C\\ &=2\sum_{j=1}^{k}\tilde{\varphi}(j|\mathbf{x}_t,\boldsymbol{\Theta})\ln\frac{\tilde{h}(j|\mathbf{x}_t)}{h(j|\mathbf{x}_t,\boldsymbol{\Theta})}+\sum_{j=1}^{k}h(j|\mathbf{x}_t,\boldsymbol{\Theta})\ln\\ &\quad\frac{h(j|\mathbf{x}_t,\boldsymbol{\Theta})}{\tilde{h}(j|\mathbf{x}_t)}+C,\end{aligned} \tag{53}$$

where C is a constant term independent from the choice of $\tilde{h}(j|\mathbf{x}_t)$s and $\boldsymbol{\Theta}$. From Lemma 3, we know that $\tilde{\varphi}(j|\mathbf{x}_t,\boldsymbol{\Theta})$s are an indicator function. By putting $\tilde{\varphi}(j|\mathbf{x}_t,\boldsymbol{\Theta})$ of (41) into (53), we therefore have

$$\begin{aligned}\bar{q}_t(\boldsymbol{\Theta};\mathbf{x}_t)&=2\tilde{\varphi}(c|\mathbf{x}_t,\boldsymbol{\Theta})\ln\frac{\tilde{h}(c|\mathbf{x}_t)}{h(c|\mathbf{x}_t,\boldsymbol{\Theta})}\\ &\quad+\sum_{j=1}^{k}h(j|\mathbf{x}_t,\boldsymbol{\Theta})\ln\frac{h(j|\mathbf{x}_t,\boldsymbol{\Theta})}{\tilde{h}(j|\mathbf{x}_t)}+C.\end{aligned} \tag{54}$$

Based on the definition of $\tilde{\varphi}(j|\mathbf{x}_t,\boldsymbol{\Theta})$, we know that the first term

$$2\tilde{\varphi}(c|\mathbf{x}_t,\boldsymbol{\Theta})\ln\frac{\tilde{h}(c|\mathbf{x}_t)}{h(c|\mathbf{x}_t,\boldsymbol{\Theta})}\ge 0, \tag{55}$$

and "=" is held if and only if $\tilde{h}(c|\mathbf{x}_t)=h(c|\mathbf{x}_t,\boldsymbol{\Theta})$. Furthermore, from Lemma 2, we also know that the second term of (54) reaches the minimum as $\tilde{h}(j|\mathbf{x}_t)=h(j|\mathbf{x}_t,\boldsymbol{\Theta})$ for all $1\le j\le k$. That is, (54) reaches the minimum value if and only if $\tilde{h}(j|\mathbf{x}_t)=h(j|\mathbf{x}_t,\boldsymbol{\Theta})$ for all j. Because of $\bar{q}_t(\boldsymbol{\Theta};\mathbf{x}_t)=-q_t(\boldsymbol{\Theta};\mathbf{x}_t)$, we know that (32) then reaches the maximum value, accordingly. □

Please note that update of $\tilde{h}(j|\mathbf{x}_t)$ via maximizing (42) as given $\boldsymbol{\Theta}$ does not often lead to $\tilde{h}(j|\mathbf{x}_t)=h(j|\mathbf{x}_t,\boldsymbol{\Theta})$ unless $\tilde{\varphi}(j|\mathbf{x}_t,\boldsymbol{\Theta})$ is well-designed. Under the circumstances, the parameter estimate via **Step B.1** and **Step B.2** may not be the same as the one via **Step A.1** and **Step A.2**. To save space, we leave their discussion elsewhere. In the following, we will further study the RPEM in detail under the Gaussian density mixtures.

Suppose N inputs $\{\mathbf{x}_t\}_{t=1}^{N}$ are all iid distributed and come from a Gaussian density mixture, i.e., the $p(\mathbf{x}|\boldsymbol{\Theta})$ in (18) is

$$p(\mathbf{x}|\boldsymbol{\Theta})=\sum_{j=1}^{k}\alpha_j p(\mathbf{x}_t|\theta_j)=\sum_{j=1}^{k}\alpha_j G(\mathbf{x}_t|\mathbf{m}_j,\boldsymbol{\Sigma}_j), \tag{56}$$

with

$$p(\mathbf{x}_t|\theta_j)=G(\mathbf{x}_t|\mathbf{m}_j,\boldsymbol{\Sigma}_j), \tag{57}$$

where $\boldsymbol{\Theta}=\{\alpha_j,\mathbf{m},\boldsymbol{\Sigma}_j\}_{j=1}^{k}$ and $G(\mathbf{x}|\mathbf{m},\boldsymbol{\Sigma})$ denotes a multivariate Gaussian density function of $\mathbf{x}$ with the mean $\mathbf{m}$, and covariance matrix $\boldsymbol{\Sigma}$. By putting (57) into (32), we then have

$$\begin{aligned}q_t(\boldsymbol{\Theta};\mathbf{x}_t)&=\sum_{j=1}^{k}g(j|\mathbf{x}_t,\boldsymbol{\Theta})\ln[\alpha_j G(\mathbf{x}_t|\mathbf{m}_j,\boldsymbol{\Sigma}_j)]\\ &\quad-\sum_{j=1}^{k}g(j|\mathbf{x}_t,\boldsymbol{\Theta})\ln h(j|\mathbf{x}_t,\boldsymbol{\Theta})\\ &=\sum_{j=1}^{k}[2I(j|\mathbf{x}_t,\boldsymbol{\Theta})-h(j|\mathbf{x}_t,\boldsymbol{\Theta})]\ln[\alpha_j G(\mathbf{x}_t|\mathbf{m}_j,\boldsymbol{\Sigma}_j)]\\ &\quad-\sum_{j=1}^{k}[2I(j|\mathbf{x}_t,\boldsymbol{\Theta})-h(j|\mathbf{x}_t,\boldsymbol{\Theta})]\ln h(j|\mathbf{x}_t,\boldsymbol{\Theta}).\end{aligned} \tag{58}$$

As a result, the detailed algorithm of RPEM can be given as follows:

Initialization. Given a specific k ($k\ge k^*$), we initialize the parameter $\boldsymbol{\Theta}$. Then, at each time step t, we implement the following two steps:

1. **Step C.1**. Given an input $\mathbf{x}_t$, we fix $\boldsymbol{\Theta}^{(\text{old})}$ and calculate

$$h(j|\mathbf{x}_t,\boldsymbol{\Theta}^{(\text{old})})=\frac{\alpha_j^{(\text{old})}G(\mathbf{x}_t|\mathbf{m}_j^{(\text{old})},\boldsymbol{\Sigma}_j^{(\text{old})})}{p(\mathbf{x}_t|\boldsymbol{\Theta}^{(\text{old})})}, \tag{59}$$

$$\begin{aligned}g(j|\mathbf{x}_t,\boldsymbol{\Theta}^{(\text{old})})&=2I(j|\mathbf{x}_t,\boldsymbol{\Theta}^{(\text{old})})-h(j|\mathbf{x}_t,\boldsymbol{\Theta}^{(\text{old})}),\\ &\qquad 1\le j\le k,\end{aligned} \tag{60}$$

where α_js are calculated by (36), $p(\mathbf{x}_t|\mathbf{\Theta}^{(\text{old})})$ is given by (56), and $I(j|\mathbf{x}_t, \mathbf{\Theta}^{(\text{old})})$ is given by (28).

2. **Step C.2**. Fixing $h(j|\mathbf{x}_t, \mathbf{\Theta}^{(\text{old})})$s, we update $\mathbf{\Theta}$ by using a stochastic gradient ascent method in the same way as **Step A.2** and **Step B.2**. We notice that all subsequent computations involve $\mathbf{\Sigma}_j^{-1}$s only rather than $\mathbf{\Sigma}_j$s. To save computing costs and ensure the learning of $\mathbf{\Sigma}_j$ more stable, we therefore directly update $\mathbf{\Sigma}_j^{-1}$s rather than $\mathbf{\Sigma}_j$s. It turns out that the update of $\mathbf{\Theta}$ is given as follows:

$$\begin{aligned}\beta_j^{(\text{new})} &= \beta_j^{(\text{old})} + \eta_\beta \frac{\partial q_t(\mathbf{\Theta}; \mathbf{x}_t)}{\partial \beta_j}|_{\mathbf{\Theta}^{(\text{old})}} \\ &= \beta_j^{(\text{old})} + \eta_\beta[g(j|\mathbf{x}_t, \mathbf{\Theta}^{(\text{old})}) - \alpha_j^{(\text{old})}],\end{aligned} \tag{61}$$

$$\begin{aligned}\mathbf{m}_j^{(\text{new})} &= \mathbf{m}_j^{(\text{old})} + \eta \frac{\partial q_t(\mathbf{\Theta}; \mathbf{x}_t)}{\partial \mathbf{m}_j}|_{\mathbf{\Theta}^{(\text{old})}} \\ &= \mathbf{m}_j^{(\text{old})} + \eta g(j|\mathbf{x}_t, \mathbf{\Theta}^{(\text{old})})\mathbf{\Sigma}_j^{-1^{(\text{old})}}(\mathbf{x}_t - \mathbf{m}_j^{(\text{old})}),\end{aligned} \tag{62}$$

$$\begin{aligned}\mathbf{\Sigma}_j^{-1^{(\text{new})}} &= \mathbf{\Sigma}_j^{-1(\text{old})} + \eta \mathbf{\Sigma}_j^{-1(\text{old})} \frac{\partial q_t(\mathbf{\Theta}; \mathbf{x}_t)}{\partial \mathbf{\Sigma}_j^{-1}} \mathbf{\Sigma}_j^{-1^{(\text{old})}}|_{\mathbf{\Theta}^{(\text{old})}} \\ &= [1 + \eta g(j|\mathbf{x}_t, \mathbf{\Theta}^{(\text{old})})]\mathbf{\Sigma}_j^{-1(\text{old})} \\ &\quad - \eta g(j|\mathbf{x}_t, \mathbf{\Theta}^{(\text{old})})\mathbf{U}_{t,j}\end{aligned} \tag{63}$$

with

$$\begin{aligned}\mathbf{U}_{t,j} = [&\mathbf{\Sigma}_j^{-1^{(\text{old})}}(\mathbf{x}_t - \mathbf{m}_j^{(\text{old})})(\mathbf{x}_t - \mathbf{m}_j^{(\text{old})})^T \mathbf{\Sigma}_j^{-1^{(\text{old})}}], \\ &1 \le j \le k.\end{aligned} \tag{64}$$

Please note that, to simplify the computation of $\mathbf{\Sigma}_j^{-1}$s' update, (63) has updated $\mathbf{\Sigma}_j^{-1}$ along the direction of $\mathbf{\Sigma}_j^{-1} \frac{\partial q_t(\mathbf{\Theta};\mathbf{x}_t)}{\partial \mathbf{\Sigma}_j^{-1}} \mathbf{\Sigma}_j^{-1}$, i.e., along the direction with an acute angle of $\frac{\partial q_t(\mathbf{\Theta};\mathbf{x}_t)}{\partial \mathbf{\Sigma}_j^{-1}}$. In the next section, we will experimentally demonstrate the performance of RPEM.

3.3 Experimental Simulation

We have conducted six experiments to demonstrate the performance of RPEM. Because of the space limitation, we leave the details of each experiment in Section 1 of the online Appendix, which can be found on the IEEE Computer Society Digital Library at: http://computer.org/tkde/archives.htm. In the following, we just summarize the results.

3.3.1 Experiment 1

With the data from a mixture of three bivariate Gaussian densities, we first investigated the performance of RPEM provided that the number k of seed points is equal to the true mixture number $k^* = 3$. The experiment verifies the convergence of Q value in (26), and shows that the RPEM has given the well estimate of the true parameters as well as the EM. Then, we further investigated the performance robustness of RPEM when $k = 7 > k^*$. It was found that the RPEM led three redundant densities to fade out in the mixture model through the learning. That is, the RPEM can automatically make the model selection during the parameter learning process. In comparison, the EM is unable to estimate the parameters correctly and select a model automatically in this case.

3.3.2 Experiment 2

Similar to Experiment 1, we investigated the RPEM with the data from a mixture of three bivariate Gaussian densities, in which, however, the data clusters were considerably overlapped. Once again, we first set $k = 3$, and performed the RPEM and EM. We found that the parameter estimate of RPEM is slightly better than the EM, and the RPEM learning is much faster than the EM. This scenario is also consistent with the qualitative analysis in [25]. That is, the rival penalization mechanism can indeed speed up the convergence of the seed points. We are going to theoretically analyze the convergence properties of RPEM elsewhere because of the space limitation in this paper.

Furthermore, we also investigated the RPEM performance when k was much larger than k^*. We arbitrarily set $k = 25$. The experimental results show that the RPEM can successfully recognize the input data set from the mixture of the three densities, but the EM cannot.

3.3.3 Experiment 3

This experiment further investigated the RPEM when k and k^* were both large. We generated the data from a mixture of 10 bivariate Gaussian densities and set k at 30. The numerical results show that the RPEM can work very well.

3.3.4 Experiment 4

In this experiment, we investigated the robustness of RPEM in 10 data clusters that are seriously overlapped. The results show that the RPEM has led the model parameters into a local maximum solution and identified seven clusters only, but not the true 10 ones. Nevertheless, we found that some of the data clusters have been seriously overlapped, which may be more reasonable to regard as a single cluster, rather than count on an individual basis. In this viewpoint, the results given by the RPEM are acceptable and correct even if the clusters are seriously overlapped.

3.3.5 Experiment 5

In the previous experiments, we considered the bivariate data points only for easy visual demonstration. This experiment showed the RPEM performance on high-dimensional data. We generated the data points from a mixture of four 30-dimension Gaussians and set k at 7. The numerical results show that the RPEM has pushed the extra α_js very close to zero and the associated seed points far away from the input data set. Consequently, it has led three redundant densities to fade out from the mixture, while the other four densities are well-recognized. That is, the RPEM has successfully identified the true data distribution.

3.3.6 Experiment 6

This experiment demonstrated the performance of RPEM in color image segmentation in comparison with the common k-means algorithm. We initially assigned 10 seed points and learned them by RPEM and *k*-means algorithm, respectively. In this trial, we used the benchmark Beach image with 64×64 pixels, in which the sky color is close to the sea color. This implies that the region of sky seriously overlaps the region of sea in HSV color space. Subsequently, it leads the RPEM to be trapped into a local optimal solution similar to the case in Experiment 4. Nevertheless, the results given by the RPEM in this experiment are still acceptable. In contrast, the *k*-means cannot make correct image segmentation at all.

The above six experiments have shown the outstanding performance of RPEM. In the following, we will further develop a simplified variant of RPEM as a general rival penalized competitive learning approach. Also, its relations with the existing RPCCL, RPCL, and its Type A variants are discussed, respectively. In particular, a new way to choose an appropriate delearning rate in the RPCL and its Type A variant is suggested and implemented in a stochastic way.

4 A Simplified Variant of RPEM Algorithm

4.1 Generalized RPCCL Algorithm and Its Stochastic Implementations

Suppose we ignore the difference between $h(c|\mathbf{x}_t, \boldsymbol{\Theta})$ and $I(c|\mathbf{x}_t, \boldsymbol{\Theta})$ in comparison with $h(c|\mathbf{x}_t, \boldsymbol{\Theta})$. $g(j|\mathbf{x}_t, \boldsymbol{\Theta})$ in (30) then becomes

$$\begin{aligned} g(j|\mathbf{x}_t, \boldsymbol{\Theta}) &= 2\varphi(j|\mathbf{x}_t, \boldsymbol{\Theta}) - h(j|\mathbf{x}_t, \boldsymbol{\Theta}) \\ &= 2I(j|\mathbf{x}_t, \boldsymbol{\Theta}) - h(j|\mathbf{x}_t, \boldsymbol{\Theta}) \\ &\approx \begin{cases} 1, & \text{if } j = c = \arg\max_i [I(i|\mathbf{x}_t, \boldsymbol{\Theta})], \\ -h(j|\mathbf{x}_t, \boldsymbol{\Theta}), & \text{otherwise}, \end{cases} \end{aligned} \tag{65}$$

where $\varphi(j|\mathbf{x}_t, \boldsymbol{\Theta})$ is given by (35). Hence, the $R_t(\boldsymbol{\Theta}; \mathbf{x}_t)$ function in (33) can then be simplified as

$$\begin{aligned} R_t(\mathbf{x}_t; \boldsymbol{\Theta}) &= \sum_{j=1}^{k} [2I(j|\mathbf{x}_t, \boldsymbol{\Theta}) - h(j|\mathbf{x}_t, \boldsymbol{\Theta})] \ln[\alpha_j p(\mathbf{x}_t|\theta_j)] \\ &\approx \ln[\alpha_c p(\mathbf{x}_t|\theta_c)] - \sum_{j=1, j\neq c}^{k} h(j|\mathbf{x}_t, \boldsymbol{\Theta}) \ln[\alpha_j p(\mathbf{x}_t|\theta_j)]. \end{aligned} \tag{66}$$

Subsequently, we can obtain a variant of the RPEM algorithm as follows:

1. **Step D.1**. Fixing $\boldsymbol{\Theta}^{(\text{old})}$, we compute $h(j|\mathbf{x}_t, \boldsymbol{\Theta}^{(\text{old})})$ and $g(j|\mathbf{x}_t, \boldsymbol{\Theta}^{(\text{old})})$ via (59) and (65), respectively.
2. **Step D.2**. Fixing $h(j|\mathbf{x}_t, \boldsymbol{\Theta}^{(\text{old})})$s and $g(j|\mathbf{x}_t, \boldsymbol{\Theta}^{(\text{old})})$s calculated in **Step D.1**, we update $\boldsymbol{\Theta}$ with a small step toward the direction of maximizing (66), which can be realized by two separate updates:

   - **Awarded Update**. We update the parameters of the winner among the mixture components, i.e., β_c and θ_c with $c = \arg\max_j [I(j|\mathbf{x}_t, \boldsymbol{\Theta}^{(\text{old})})]$, by

$$\begin{aligned} \beta_c^{(\text{new})} &= \beta_c^{(\text{old})} + \eta_\beta \frac{\partial R_t(\boldsymbol{\Theta}; \mathbf{x}_t)}{\partial \beta_c} |_{\boldsymbol{\Theta}^{(\text{old})}} \\ &= \beta_c^{(\text{old})} + \eta_\beta \sum_{j=1}^{k} [g(j|\mathbf{x}_t, \boldsymbol{\Theta}^{(\text{old})})(\delta_{jc} - \alpha_c^{(\text{old})})], \\ &= \beta_c^{(\text{old})} + \eta[1 - h(c|\mathbf{x}_t, \boldsymbol{\Theta}^{(\text{old})})\alpha_c^{(\text{old})}], \end{aligned} \tag{67}$$

$$\begin{aligned} \theta_c^{(\text{new})} &= \theta_c^{(\text{old})} + \eta \frac{\partial R_t(\boldsymbol{\Theta}; \mathbf{x}_t)}{\partial \theta_c} |_{\boldsymbol{\Theta}^{(\text{old})}} \\ &= \theta_c^{(\text{old})} + \eta \frac{\partial \ln p(\mathbf{x}_t|\theta_c)}{\partial \theta_c} |_{\theta_c^{(\text{old})}}, \end{aligned} \tag{68}$$

   where δ_{jc} is the Kronecker delta function.
   - **Penalized Update**. We update those β_js and θ_js with $1 \leq j \leq k$, but $j \neq c$:

$$\begin{aligned} \beta_j^{(\text{new})} &= \beta_j^{(\text{old})} + \eta_\beta \frac{\partial R_t(\boldsymbol{\Theta}; \mathbf{x}_t)}{\partial \beta_j} |_{\boldsymbol{\Theta}^{(\text{old})}} \\ &= \beta_j^{(\text{old})} + \eta_\beta \sum_{u=1}^{k} [g(u|\mathbf{x}_t, \boldsymbol{\Theta}^{(\text{old})})(\delta_{uj} - \alpha_j^{(\text{old})})] \\ &= \beta_j^{(\text{old})} - \eta_\beta [h(j|\mathbf{x}_t, \boldsymbol{\Theta}^{(\text{old})}) \\ &\quad + h(c|\mathbf{x}_t, \boldsymbol{\Theta}^{(\text{old})})\alpha_j^{(\text{old})})], \end{aligned} \tag{69}$$

$$\begin{aligned} \theta_j^{(\text{new})} &= \theta_j^{(\text{old})} + \eta \frac{\partial R_t(\boldsymbol{\Theta}; \mathbf{x}_t)}{\partial \theta_j} |_{\boldsymbol{\Theta}^{(\text{old})}} \\ &= \theta_j^{(\text{old})} - \eta h(j|\mathbf{x}_t, \boldsymbol{\Theta}^{(\text{old})}) \frac{\partial \ln p(\mathbf{x}_t|\theta_j)}{\partial \theta_j} |_{\theta_j^{(\text{old})}}. \end{aligned} \tag{70}$$

In this algorithm, all mixture components are competing with each other at each time step to update to adapt the input. Not only are the winner's parameters β_c and θ_c updated toward maximizing the value of $R_t(\boldsymbol{\Theta}; \mathbf{x}_t)$ in (66), but those rival's parameters $\{\beta_j, \theta_j\}_{j=1, j\neq c}^{k}$ are penalized to update toward the reverse direction of maximizing $R_t(\boldsymbol{\Theta}; \mathbf{x}_t)$. This is the exact procedure of a rival penalized competitive learning. Furthermore, (69) and (70) show that the *rival penalization strength*, denoted as $\eta h(j|\mathbf{x}_t, \boldsymbol{\Theta})$, is dynamically controlled. The larger $h(j|\mathbf{x}_t, \boldsymbol{\Theta})$ is, the larger the penalization strength is. That is, the rival is more penalized if its winning chance is closer to the winner. Such a penalization mechanism is the same as the one in *Rival Penalization Controlled Competitive Learning* (RPCCL) algorithm [8]. We therefore call the algorithm described by **Step D.1** and **Step D.2** *Generalized RPCCL* (G-RPCCL), which extends the RPCCL at least three-fold:

1. The rival penalization mechanism in the G-RPCCL is systematically developed from the MWL framework, while the one in the RPCCL of [8] is heuristically proposed.
2. The G-RPCCL is applicable to the elliptical clusters with any input proportion, but the RPCCL in [8] may not.

TABLE 1
The General Procedures of Simplified G-RPCCL Algorithm

Initialization	Given a specific k ($k \geq k^*$), we initialize the parameter $\mathbf{\Theta}$.
Step 1	Given an input $\mathbf{x}_t$, we fix $\mathbf{\Theta}^{(\text{old})}$, and calculate $h(j \mid \mathbf{x}_t, \mathbf{\Theta}^{(\text{old})})$ and $I(j \mid \mathbf{x}_t, \mathbf{\Theta}^{(\text{old})})$ by Eq.(21) and Eq.(28), respectively.
Step 2	Fixing $h(j \mid \mathbf{x}_t, \mathbf{\Theta}^{(\text{old})})$s and $I(j \mid \mathbf{x}_t, \mathbf{\Theta}^{(\text{old})})$s, we update: $\alpha_c^{(\text{new})} = \frac{n_c^{(\text{new})}}{\sum_{j=1,j\neq c}^{k} n_j^{(\text{old})} + n_c^{(\text{new})}}$, $\boldsymbol{\theta}_c^{(\text{new})} = \boldsymbol{\theta}_c^{(\text{old})} + \eta \frac{\partial \ln p(\mathbf{x}_t \mid \boldsymbol{\theta}_c)}{\partial \boldsymbol{\theta}_c} \big\vert_{\boldsymbol{\theta}_c^{(\text{old})}}$, $\boldsymbol{\theta}_r^{(\text{new})} = \boldsymbol{\theta}_r^{(\text{old})} - \eta h(r \mid \mathbf{x}_t, \mathbf{\Theta}^{(\text{old})}) \frac{\partial \ln p(\mathbf{x}_t \mid \boldsymbol{\theta}_r)}{\partial \boldsymbol{\theta}_r} \big\vert_{\boldsymbol{\theta}_r^{(\text{old})}}$, where $c = \arg\max_j [I(j \mid \mathbf{x}_t, \mathbf{\Theta}^{(\text{old})})]$, and $r = \arg\max_{j,j\neq c} [I(j \mid \mathbf{x}_t, \mathbf{\Theta}^{(\text{old})})]$, and $n_j^{(\text{new})}$ is given by Eq.(72).
	The above two steps are repeatedly implemented for each input until $\mathbf{\Theta}$ converges.

TABLE 2
Simplified G-RPCCL Algorithm in a Gaussian Mixture

Initialization	Given a specific k ($k \geq k^*$), we initialize the parameter $\mathbf{\Theta}$.
Step 1	Given an input $\mathbf{x}_t$, we fix $\mathbf{\Theta}^{(\text{old})}$ and calculate $h(j \mid \mathbf{x}_t, \mathbf{\Theta}^{(\text{old})}) = \frac{\alpha_j^{(\text{old})} G(\mathbf{x}_t \mid \mathbf{m}_j^{(\text{old})}, \mathbf{\Sigma}_j^{(\text{old})})}{\sum_{i=1}^{k} \alpha_i^{(\text{old})} G(\mathbf{x}_t \mid \mathbf{m}_i^{(\text{old})}, \mathbf{\Sigma}_i^{(\text{old})})}$, whereby $I(j \mid \mathbf{x}_t, \mathbf{\Theta}^{(\text{old})})$ is obtained via Eq.(28).
Step 2	Fixing $h(j \mid \mathbf{x}_t, \mathbf{\Theta}^{(\text{old})})$s and $I(j \mid \mathbf{x}_t, \mathbf{\Theta}^{(\text{old})})$s, we update: $\alpha_c^{(\text{new})} = \frac{n_c^{(\text{new})}}{\sum_{j=1,j\neq c}^{k} n_j^{(\text{old})} + n_c^{(\text{new})}}$ $\mathbf{m}_c^{(\text{new})} = \mathbf{m}_c^{(\text{old})} + \eta \mathbf{\Sigma}_c^{-1\,(\text{old})} (\mathbf{x}_t - \mathbf{m}_c^{(\text{old})})$ $\mathbf{\Sigma}_c^{-1\,(\text{new})} = (1+\eta) \mathbf{\Sigma}_c^{-1\,(\text{old})} - \eta \mathbf{U}_{t,c}$ $\mathbf{m}_r^{(\text{new})} = \mathbf{m}_r^{(\text{old})} - \eta h(r \mid \mathbf{x}_t, \mathbf{\Theta}^{(\text{old})}) \mathbf{\Sigma}_r^{-1\,(\text{old})} (\mathbf{x}_t - \mathbf{m}_r^{(\text{old})})$ $\mathbf{\Sigma}_r^{-1\,(\text{new})} = [1 - \eta h(r \mid \mathbf{x}_t, \mathbf{\Theta}^{(\text{old})})] \mathbf{\Sigma}_r^{-1\,(\text{old})} + \eta h(r \mid \mathbf{x}_t, \mathbf{\Theta}^{(\text{old})}) \mathbf{U}_{t,r}$ where $c = \arg\max_j [I(j \mid \mathbf{x}_t, \mathbf{\Theta}^{(\text{old})})]$, and $r = \arg\max_{j,j\neq c} [I(j \mid \mathbf{x}_t, \mathbf{\Theta}^{(\text{old})})]$, $n_j^{(\text{new})}$ is given by Eq.(72), and $\mathbf{U}_{t,\tau} = [\mathbf{\Sigma}_\tau^{-1\,(\text{old})} (\mathbf{x}_t - \mathbf{m}_\tau^{(\text{old})})(\mathbf{x}_t - \mathbf{m}_\tau^{(\text{old})})^T \mathbf{\Sigma}_\tau^{-1\,(\text{old})}]$, where $\tau = c, r$. Please note that, to circumvent the calculation of $\mathbf{\Sigma}_\tau$, we have updated $\mathbf{\Sigma}_\tau^{-1}$ along the direction of $\mathbf{\Sigma}_\tau^{-1} \frac{\partial R_t(\mathbf{x}_t; \mathbf{\Theta})}{\partial \mathbf{\Sigma}_\tau^{-1}} \mathbf{\Sigma}_\tau^{-1}$ similar to the previous-stated RPEM.
	The above two steps are repeatedly implemented for each input until $\mathbf{\Theta}$ converges.

3. At each time step, the G-RPCCL penalizes all rivals rather than the nearest rival (i.e., the rival with the subscript $r = \arg\max_{j,j\neq c} I(j|\mathbf{x}_t, \mathbf{\Theta})$) only, like RPCCL.

It is certain that the learning of G-RPCCL can be further simplified if we penalize the nearest rival only like the RPCL [25] and RPCCL [8] rather than all rivals. Furthermore, we notice that α_j can be estimated by $E[h(j|\mathbf{x}_t, \mathbf{\Theta})]$. Thus, instead of using the soft-max function, we can simply estimate α_c by

$$\alpha_c^{(\text{new})} = \frac{n_c^{(\text{new})}}{\sum_{j=1,j\neq c}^{k} n_j^{(\text{old})} + n_c^{(\text{new})}} \tag{71}$$

with

$$n_j^{(\text{new})} = \begin{cases} n_j^{(\text{old})} + 1, & \text{if } j = c = \arg\max_i [I(i|\mathbf{x}_t, \mathbf{\Theta}^{(\text{old})})] \\ n_j^{(\text{old})}, & \text{otherwise}, \end{cases} \tag{72}$$

where each n_js should be initialized at a positive integer value, e.g., let $n_j = 1$. As pointed out in [9], the updating of α_c only in (71) is, in effect, to update those α_js ($j \neq c$) automatically with a small step toward the direction of minimizing $R_t(\mathbf{\Theta}; \mathbf{x}_t)$. Hence, we can save computing costs without updating other α_js. Table 1 summarizes the general steps of this simplified G-RPCCL.

Particularly, if each mixture component $p(\mathbf{x}_t|\theta_j)$ is a Gaussian density as given by (57), $R_t(\mathbf{\Theta}; \mathbf{x}_t)$ in (66) then becomes

$$\begin{aligned} R_t(\mathbf{\Theta}; \mathbf{x}_t) &= \ln[\alpha_c G(\mathbf{x}_t|\mathbf{m}_c, \mathbf{\Sigma}_c)] \\ &\quad - \sum_{j=1,j\neq c}^{k} h(j|\mathbf{x}_t, \mathbf{\Theta}) \ln[\alpha_j G(\mathbf{x}_t|\mathbf{m}_j, \mathbf{\Sigma}_j)]. \end{aligned} \tag{73}$$

Subsequently, the detailed implementation of Simplified G-RPCCL algorithm in a Gaussian mixture is given out in Table 2.

If we always fix the rival penalization strength at its mean further, i.e., $\bar{\eta}_r = \eta E_{\mathbf{x}}[h(r|\mathbf{x}, \mathbf{\Theta})]$, this Simplified G-RPCCL then degenerates to the RPCL (Type A) [24], but gives out a guidance to choose the fixed delearning rate $\bar{\eta}_r$. In implementation, an estimate of $\bar{\eta}_r$ can be given by calculating the sample mean $\bar{h}_r$ of those $h(r|\mathbf{x}_t, \mathbf{\Theta})$s based on

TABLE 3
Stochastic RPCL Algorithm

Initialization	Given a specific k ($k \geq k^*$), we initialize the parameter $\mathbf{\Theta}$.
Step 1	Given an input $\mathbf{x}_t$, we fix $\mathbf{\Theta}^{(\text{old})} = \{\mathbf{m}_j^{(\text{old})}\}_{j=1}^k$, calculate $h(j\vert\mathbf{x}_t, \mathbf{\Theta}^{(\text{old})}) = \frac{\exp(-0.5\Vert\mathbf{x}_t - \mathbf{m}_j^{(\text{old})}\Vert^2)}{\sum_{r=1}^k \exp(-0.5\Vert\mathbf{x}_t - \mathbf{m}_r^{(\text{old})}\Vert^2)}$, and $I(j\vert\mathbf{x}_t, \mathbf{\Theta}^{(\text{old})})$ is given by Eq.(77).
Step 2	Fixing $h(j\vert\mathbf{x}_t, \mathbf{\Theta}^{(\text{old})})$s, we update: $\mathbf{m}_c^{(\text{new})} = \mathbf{m}_c^{(\text{old})} + \eta(\mathbf{x}_t - \mathbf{m}_c^{(\text{old})})$. We randomly generate a value $\gamma \in [0, 1]$ that is uniformly distributed. If $\gamma \leq h(r\vert\mathbf{x}_t, \mathbf{\Theta}^{(\text{old})})$, we perform $\mathbf{m}_r^{(\text{new})} = \mathbf{m}_r^{(\text{old})} - \eta(\mathbf{x}_t - \mathbf{m}_r^{(\text{old})})$, where $c = \arg\max_j[I(j\vert\mathbf{x}_t, \mathbf{\Theta}^{(\text{old})})]$, and $r = \arg\max_{j,j\neq c}[I(j\vert\mathbf{x}_t, \mathbf{\Theta}^{(\text{old})})]$; otherwise, we do nothing.
	The above two steps are repeatedly implemented for each input until $\mathbf{\Theta}$ converges.

the available data points, or adaptively learned via the following updating equation:

$$\bar{h}_r^{(\text{new})} = (1-\eta)\bar{h}_r^{(\text{old})} + \eta h(r|\mathbf{x}_t, \mathbf{\Theta}), \quad \text{for} \quad \forall t, t = 1, 2, \ldots, N. \tag{74}$$

In contrast, a more simple and efficient way is to penalize the nearest rival stochastically, i.e., at each time step, we generate a random uniformly-distributed number $\gamma \in [0, 1]$, if $\gamma \leq h(r|\mathbf{x}_t, \mathbf{\Theta})$, we then penalize the rival with the strength η, otherwise, the rival is not penalized. We call this *Stochastic RPCL (Type A)*. If we further always fix

$$\forall \quad 1 \leq j \leq k, \quad \alpha_j = \frac{1}{k}, \quad \text{and} \quad \mathbf{\Sigma}_j = \mathbf{I} \tag{75}$$

during the learning, where $\mathbf{I}$ is the identity matrix, the indicator function $I(j|\mathbf{x}_t, \mathbf{\Theta})$ of (28) is then equivalent to:

$$I(j|\mathbf{x}_t, \mathbf{\Theta}) = \begin{cases} 1, & \text{if } j = c = \arg\min_{1\leq r\leq k} \|\mathbf{x}_t - \mathbf{m}_r\| \\ 0, & \text{otherwise,} \end{cases} \tag{76}$$

from which it can be seen that the winning of seed points is exclusively determined by the Euclidean distance between the input and the seed points. As a result, if some seed points are initialized far away from the input data set in comparison with other seed points, they will immediately become dead without learning chance any more in the subsequent learning. This scenario is called the *dead unit* problem as pointed out in [1]. To circumvent this awkward situation, Ahalt et al. [1] suggests gradually reducing the winning chance of a frequent winning seed point. That is, we add the relative winning frequency of a seed point into the indicator function. Subsequently, we have:

$$I(j|\mathbf{x}_t, \mathbf{\Theta}) = \begin{cases} 1, & \text{if } j = c = \arg\min_{1\leq r\leq k} \gamma_r\|\mathbf{x}_t - \mathbf{m}_r\| \\ 0, & \text{otherwise,} \end{cases} \tag{77}$$

with

$$\gamma_r = \frac{n_r}{\sum_{i=1}^k n_i},$$

where n_i is the past winning frequency of the seed point $\mathbf{m}_i$. It can be seen that the learning rule of the seed points $\mathbf{m}_j$s in this algorithm is exactly the one in the RPCL [25]. We therefore name this algorithm *Stochastic RPCL* (S-RPCL). Table 3 gives out its details. Compared to the existing RPCL, this new one has novelly circumvented the selection problem of the delearning rate by fixing it at the learning rate, which is, however, strictly prohibited in the RPCL as pointed out in [25]. In the next section, we will demonstrate the performance of S-RPCL in comparison with the RPCL to show the effectiveness of this stochastic method.

4.2 Experimental Demonstrations

We conducted two experiments to compare the S-RPCL and the RPCL. In each experiment, we used six seed points, whose initial positions were randomly assigned in the input space. Moreover, we randomly set the learning rate at 0.001, while letting the delearning rate $r = 0.0001$ by default when using the RPCL. Because of the space limitation, we summarize the experimental results only as follows. For more details, interested readers can refer to Section 2 of the online Appendix on the IEEE Computer Society Digital Library at: http://computer. org/tkde/archives.htm.

4.2.1 Experiment 1

We used the 1,000 data points from a mixture of three Gaussian distribution, in which the data clusters were well-separated. The experimental results show that the S-RPCL has put three seed points into the three cluster centers, meanwhile driving the other three extra seed points far away from the input data set. Based on the rival penalization equation in **Step 2** of Table 3, we know that the rival penalization strength will nonlinearly decrease as the extra seed points leave the input data set, and they will finally become stable outside the input data set. In contrast, although the RPCL could work as well in this case, the RPCL always penalizes the extra seed points even if they are much farther away from the input data set. Consequently, the seed points as a whole will not tend to convergence, but those learned by the S-RPCL will.

4.2.2 Experiment 2

We further investigated the performances of S-RPCL and RPCL in the three moderate overlapping clusters. We found that the S-RPCL had given the correct results, but the RPCL had not. We then further investigated the performance of RPCL by adjusting the delearning rate $r = 0.0001$ along two directions: from 0.0001 to 0.00001 and from 0.0001 to 0.0009, respectively, with a constant step: 0.00001. Unfortunately, we could not find an appropriate r in all cases we had tried so far to make RPCL successfully work.

5 Conclusion

This paper has developed an MWL learning framework from the ML, through which a new RPEM algorithm has been proposed for density mixture clustering. The RPEM learns the density parameters by making mixture components compete each other at each time step. Not only are the associated parameters of the winning density component updated to adapt to an input, but also all rivals' parameters are penalized with the strength proportional to the corresponding posterior density probabilities. Compared to the EM algorithm, this intrinsic rival penalization mechanism enables the RPEM to automatically select an appropriate number of densities by fading out the redundant densities from a density mixture. The numerical experiments have shown its outstanding performance in both of synthetic and real-life data. Moreover, the G-RPCCL developed from the RPEM has further generalized our recently proposed RPCCL algorithm so that it is applicable to elliptical clusters as well with any input proportion. Compared to the existing RPCL and its variants, the G-RPCCL need not select the delearning rate. Additionally, we have shown that a special setting of the G-RPCCL not only includes them as its special cases, but also gives a guidance to choose an appropriate delearning rate for them. Subsequently, we have proposed a stochastic version of RPCL and its Type A variant, respectively, in which the selection problem of delearning rate has been novelly circumvented. The experiments have shown the promising results of this stochastic implementation.

Acknowledgments

The author would like to sincerely thank the editor and three anonymous reviewers for their valuable comments and insightful suggestions. Also, the author thanks Mr. Lap-tak Law for conducting the experiment of RPEM on color image segmentation problem. This work was supported by the Research Grant Council of Hong Kong SAR under Project HKBU 2156/04E.

References

[1] S.C. Ahalt, A.K. Krishnamurty, P. Chen, and D.E. Melton, "Competitive Learning Algorithms for Vector Quantization," *Neural Networks,* vol. 3, pp. 277-291, 1990.

[2] H. Akaike, "Information Theory and an Extension of the Maximum Likelihood Principle," *Proc. Second Int'l Symp. Information Theory,* pp. 267-281, 1973.

[3] H. Akaike, "A New Look at the Statistical Model Identfication," *IEEE Trans. Automatic Control AC-19,* pp. 716-723, 1974.

[4] H. Bozdogan, "Model Selection and Akaike's Information Criterion: The General Theory and Its Analytical Extensions," *Psychometrika,* vol. 52, no. 3, pp. 345-370, 1987.

[5] H. Bozdogan, "Mixture-Model Cluster Analysis Using Model Selection Criteria and a New Information Measure of Complexity," *Proc. First US/Japan Conf. the Frontiers of Statistical Modeling,* vol. 2, pp. 69-113, 1994.

[6] J. Banfield and A. Raftery, "Model-Based Gaussian and Non-Gaussian Clustering," *Biometrics,* vol. 49, pp. 803-821, 1993.

[7] B. Fritzke, "The LBG-U Method for Vector Quantization—An Improvement over LBG Inspired From Neural Networks," *Neural Processing Letters,* vol. 5, no. 1, pp. 35-45, 1997.

[8] Y.M. Cheung, "Rival Penalization Controlled Competitive Learning for Data Clustering with Unknown Cluster Number," *Proc. Ninth Int'l Conf. Neural Information Processing (Paper ID: 1983 in CD-ROM Proceeding),* Nov. 2002.

[9] Y.M. Cheung, "k^*-Means— A New Generalized k-Means Clustering Algorithm," *Pattern Recognition Letters,* vol. 24, no. 15, pp. 2883-2893, 2003.

[10] A.P. Dempster, N.M. Laird, and D.B. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *J. Royal Statistical Soc., Series B,* vol. 39, no. 1, pp. 1-38, 1977.

[11] P.A. Devijver and J. Kittler, *Pattern Recognition: A Statistical Approach.* Prentice-Hall, 1982.

[12] R.O. Dua and P.E. Hart, *Pattern Classification and Scene Analysis.* Wiley, 1973.

[13] L. Kaufman and P. Rousseeuw, *Finding Groups in Data.* New York: John Wiley and Sons, 1989.

[14] Y.W. Lim and S.U. Lee, "On the Color Image Segmentation Algorithm Based on the Thresholding and the Fuzzy C-Means Techniques," *Pattern Recognition,* vol. 23, no. 9, pp. 935-952, 1990.

[15] Y. Linde, A. Buzo, and R.M. Gray, "An Algorithm for Vector Quantizer Design," *IEEE Trans. Comm.,* COM-28, pp. 84-95, 1980.

[16] J.B. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations," *Proc. Fifth Berkeley Symp. Math. Statistics and Probability,* vol. 1, pp. 281-297, Berkeley, Calif.: Univ. of California Press, 1967.

[17] G.J. McLachlan and K.E. Basford, *Mixture Models: Inference and Application to Clustering.* Dekker, 1988.

[18] U. Fayyad, G. Piatetsky-Shpiro, P. Smyth, and R. Uthurusamy, *Advances in Knowledge Discovery and Data Mining.* MIT Press, 1996.

[19] R.A. Redner and H.F. Waler, "Mixture Densities, Maximum Likelihood, and the EM Algorithm," *SIAM Rev.,* vol. 26, pp. 195-239, 1984.

[20] G. Schwarz, "Estimating the Dimension of a Model," *The Annals of Statistics,* vol. 6, no. 2, pp. 461-464, 1978.

[21] B.W. Silverman, *Density Estimation for Statistics and Data Analysis.* London: Chapman & Hall, 1986.

[22] T. Uchiyama and M.A. Arib, "Color Image Segmentation Using Competitive Learning," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 16, no. 12, Dec. 1994.

[23] L. Xu, "Bayesian Ying-Yang Machine, Clustering and Number of Clusters," *Pattern Recognition Letters,* vol. 18, nos. 11-13, pp. 1167-1178, 1997.

[24] L. Xu, "Rival Penalized Competitive Learning, Finite Mixture, and Multisets Clustering," *Proc. Int'l Joint Conf. Neural Networks,* vol. 2, pp. 2525-2530, 1998.

[25] L. Xu, A. Krzyżak, and E. Oja, "Rival Penalized Competitive Learning for Clustering Analysis, RBF Net, and Curve Detection," *IEEE Trans. Neural Networks,* vol. 4, pp. 636-648, 1993.

Yiu-ming Cheung received the PhD degree from the Department of Computer Science and Engineering at the Chinese University of Hong Kong in 2000. He then became a visiting assistant professor at the same institution. Since 2001, he has been an assistant professor in the Department of Computer Science at Hong Kong Baptist University. His research interests include machine learning, information security, signal processing, pattern recognition, and data mining. Currently, he is the chairman of the Computational Intelligence Chapter of the IEEE Hong Kong Section. Also, he is a member of the IASTED Technical Committee on Neural Networks and a member of the editorial boards of two international journals: *International Journal of Computational Intelligence* and *International Journal of Signal Processing*, respectively. He has been listed in the 21st & 22nd edition of Marquis Who'sWho in the World, and 8th Edition of Who'sWho in Science and Engineering. He is a member of the IEEE.