ST793 Project: A Blogpost on "Doubly Enhanced EM Algorithm for Model-Based Tensor Clustering" by Mai et al

Ayumi Mutoh, Jisu Oh, Shih-Ni Prim

December 7, 2023

1 Introduction

In recent decades, while tensor data have gained popularity in modern science, their high-dimensional structures often pose challenges for statistical analysis, specifically in model-based clustering. Model-based clustering is a statistical approach to data clustering, where observed data is considered to have been created from a finite combination of component models, such as the Gaussian mixture model (GMM). Since the formalization of the expected-maximization (EM) algorithm by Dempster et al. (1977), the EM algorithm has been widely employed in the majority of model-based clustering applications. While the GMMs can be readily extended to higher-order tensors using the standard EM algorithm, their performance can be further enhanced by integrating the Doubly Enhanced EM algorithm (DEEM), as proposed by Qing Mai and Deng (2022). Mai et al. consider a tensor normal mixture model (TNMM) that incorporates tensor correlation structure and variable selection for clustering and parameter estimation. They developed the DEEM algorithm which enables DEEM to excel in high-dimensional tensor data analysis. Similar to the EM algorithm, DEEM carries out an enhanced E-step and an enhanced M-step.

In this blogpost, we first introduce the DEEM methods with intermediate steps for the theoretical explanation. The objective is to break down the steps, making the derivation more accessible for our readers to follow. Subsequently, we will conduct a simulation study to evaluate the performance of DEEM.

2 Theoretical Derivation

2.1 EM Algorithm

Before delving into DEEM, we would like to review the EM algorithm and its functioning in clustering.

The EM algorithm is an iterative approach that cycles between two steps for maximum likelihood estimation in the presence of latent variables. The observed data Y is incomplete and data Z is missing. The first step is to write down the joint likelihood, $L_c(\theta|Y,Z)$, of the "complete" data (Y,Z). The "E" step of the EM algorithm is to compute the conditional expectation of log-likelihood, $\log L_c(\theta|Y,Z)$, given Y, assuming the true parameter value is $\theta^{(\nu)}$

$$Q(\theta,\theta^{(\nu)},Y) = E_{\theta^{(\nu)}}(\log L_c(\theta|Y,Z)|Y).$$

In the "M" step, we maximize $Q(\theta,\theta^{(\nu)},Y)$ with respect to θ with $\theta^{(\nu)}$ fixed. We repeat the E step and M step until convergence.

The EM algorithm is well-known for its use in unsupervised learning problems such as clustering with a mixture model. The process from Yang et al. (2012) goes as follows:

1. Identify the number of clusters.
2. Define each cluster by generating a Gaussian model.
3. For every observation, calculate the probability that it belongs to each cluster (Ex. observation 12 has 40% probability of belonging to Cluster A and 60% probability of belonging to Cluster B.)
4. Use the above probabilities to recalculate the Gaussian models.
5. Repeat until observations "converge" on their assignments.

Let's consider a simple example. Suppose we have data X_i as shown in Figure 1, which comes from two distinct classes. We use this data to build a Gaussian model for each class. Since we don't know which class each observation belongs to, there is no straightforward way to construct two Gaussian models to partition the data. Therefore, we begin with a random guess of our Gaussian model parameters: $\mu_1, \sigma_1^2, \mu_2, \sigma_2^2$.

We have 'missing' data points X_i that we believe belong to either of the two distributions. After initializing two random Gaussian models, we compute the likelihood of each observation, X_i, being expressed in both of the Gaussian models. The next is the E-step, where we compute the probability that each X_i can belong to any of two distributions. Now we have each point's probability of belonging to either distribution.

In the M-step, we update the parameters, $\mu_1, \sigma_1^2, \mu_2, \sigma_2^2$, of the model to their most likely values. For the new μ_1, we take a weighted average of all the points, weighted by the probability that they belong to the first distribution. Denoting p_i is the probability that X_i belongs to the first distribution.

$$\mu_1 = \frac{p_1X_1 + p_2X_2 + \cdots + p_nX_n}{p_1 + p_2 + \cdots + p_n}$$

The new σ_1^2 can be updated similarly.

$$\sigma_1^2 = \frac{p_1(X_1-\mu_1)^2 + p_2(X_2-\mu_1)^2 + \cdots + p_n(X_n-\mu_1)^2}{p_1+p_2+\cdots+p_n}$$

We repeat this process for μ_2 and σ_2^2 and update our distributions. We iterate through the E-step and M-step until convergence, obtaining two clusters as shown in Figure 2.

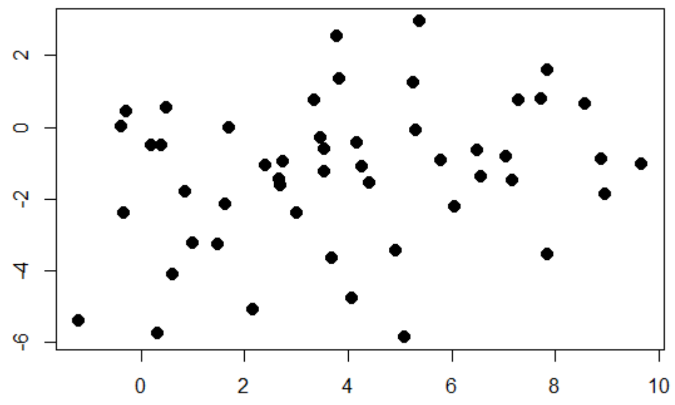


Figure 1: Mixture of two Gaussian Distributions

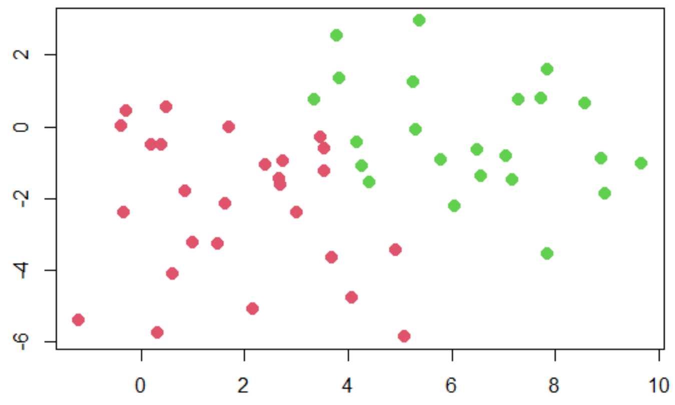
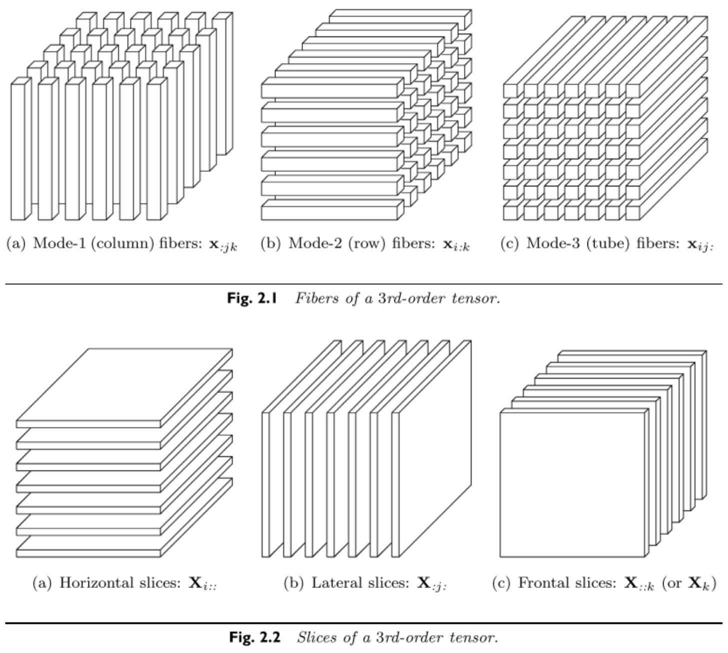


Figure 2: Clusters Found by EM algorithm

2.2 Tensor

While the term "tensor" might sound unfamiliar to some, tensors are simply multi-way arrays. Data is often structured as matrices, and they are in fact second-order tensors. When we use the term "tensor," we usually mean tensors of third- and higher-order. The "order" means the dimension of a tensor, and it is sometimes called the "mode." You can think of a third-order tensor as a cube. As shown in Figure 3, a tensor can be manipulated similarly as a matrix. In a matrix, we can talk about rows and columns. In a third-order tensor, we can talk about **fibers** when you fix two modes and keep all values of one mode. The index is then the mode that has all the values. **Slices** are when you fix one mode and keep all values for the rest of the modes. The index is then the mode that is fixed.



(a) Mode-1 (column) fibers: $\mathbf{x}_{:jk}$
(b) Mode-2 (row) fibers: $\mathbf{x}_{i:k}$
(c) Mode-3 (tube) fibers: $\mathbf{x}_{ij:}$
Fig. 2.1 Fibers of a 3rd-order tensor.
(a) Horizontal slices: $\mathbf{X}_{i::}$
(b) Lateral slices: $\mathbf{X}_{:j:}$
(c) Frontal slices: $\mathbf{X}_{::k}$ (or $\mathbf{X}_k$)
Fig. 2.2 Slices of a 3rd-order tensor.

Figure 3: Dimensions and Terminology of a Tensor, taken from Kolda and Bader (2009)

Before we continue onto the DEEM algorithm, some concepts and notations are necessary to understand derivations in the following section. Note that the following notations are taken from Kolda and Bader (2009). First of all, we should go over the concept of matricization. If we want to matricize a third-order tensor, we can think of cutting a cube into slices and putting the slices side-by-side to make them into a matrix. We now borrow an example (Example 2.1) given in Kolda and Bader (2009) to demonstrate how to matricize a third-order tensor $X \in \mathbb{R}^{3\times4\times2}$:

$$X_1 = \begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix}, X_2 = \begin{bmatrix} 13 & 16 & 19 & 22 \\ 14 & 17 & 20 & 23 \\ 15 & 18 & 21 & 24 \end{bmatrix}.$$

The three mode-n unfoldings/matricizations are then

$$X_1 = \begin{bmatrix} 1 & 4 & 7 & 10 & 13 & 16 & 19 & 22 \\ 2 & 5 & 8 & 11 & 14 & 17 & 20 & 23 \\ 3 & 6 & 9 & 12 & 15 & 18 & 21 & 24 \end{bmatrix},$$

$$X_2 = \begin{bmatrix} 1 & 2 & 3 & 13 & 14 & 15 \\ 4 & 5 & 6 & 16 & 17 & 18 \\ 7 & 8 & 9 & 19 & 20 & 21 \\ 10 & 11 & 12 & 22 & 23 & 24 \end{bmatrix},$$

$$X_3 = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & \cdots & 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 & 17 & \cdots & 21 & 22 & 23 & 24 \end{bmatrix}.$$

The example above shows that mode-1 unfolding is to put X_1 and X_2 side-by-side and mode-2 unfolding is to stack on top of each other.

This concept is intuitive but much more awkward when we want to define it formally. In Kolda and Bader (2009), the mode-n matricization of a tensor $X \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_M}$ is denoted by $X_{(n)}$, which is a matrix of the dimension $(I_n, \prod_{p \neq I_n} I_p)$. (The first dimension comes from the dimension of mode n, and the second dimension comes from the product of all the other dimensions.) The tensor element $(i_1, \cdots, i_M)$ is mapped to the matrix element (i_n, j) in the following manner:

$$j = 1 + \sum_{k=1, k\neq n}^{M} (i_k - 1) J_k \quad \text{with} \quad J_k = \prod_{m=1, m\neq n}^{k-1} I_m.$$

Next, the notation $[\![\cdot]\!]$ is defined as:

$$[\![G; A^{(1)}, A^{(2)}, \cdots, A^{(M)}]\!] := G \times_1 A^{(1)} \times_2 A^{(2)} \cdots \times_M A^{(M)},$$

where A are matrices and X and G are tensors. The symbol $\times_n$ means n-mode (matrix) product of a tensor $G \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_M}$ and the matrices $A^{(n)} \in \mathbb{R}^{J \times I_n}$. $G \times_n A^{(n)}$ is then a tensor of the dimension $I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_M$. We can write the elements of $G \times_n A^{(n)}$ as:

$$(G \times_n A^{(n)})_{i_1 \cdots i_{n-1}\, j\, i_{n+1} \cdots i_N} = \sum_{i_n=1}^{I_n} g_{i_1\, i_2 \cdots i_N} \cdot a^{(n)}_{j\, i_n}.$$

Also, for any two tensors A and B in $\mathbb{R}^{I_1 \times I_2 \times \cdots \times I_M}$, we define their inner product by

$$\langle A, B \rangle = \sum_{J \in I_1 \times I_2 \times \cdots \times I_N} a_J b_J.$$

With that, we are ready to learn about the DEEM algorithm. If you are interested in knowing more about tensors, Kolda and Bader (2009) has lots of great details. So be sure to check it out!

2.3 Doubly Enhanced EM Algorithm

In this subsection, we introduce the doubly enhanced EM (DEEM) algorithm and discuss its theoretical properties. Algorithm 1 from Qing Mai and Deng (2022) is provided in the Appendix as our Figure 6.

Let Z denote the random tensor in $\mathbb{R}^{p_1, \times \cdots \times p_M}$ such that every element in Z is distributed as iid $N(0,1)$. Then we say that a random tensor X has a tensor normal distribution, denoted by $X \sim \mathrm{TN}(\mu; \Sigma_1, \ldots, \Sigma_M)$, if $X = \mu + [\![Z; \Sigma_1^{1/2}, \ldots, \Sigma_M^{1/2}]\!]$, where $\mu \in \mathbb{R}^{p_1, \times \cdots \times p_M}$ is the total mean and each $\Sigma_i \in \mathbb{R}^{p_i \times p_i}$ means the covariance matrix within ith class. In other words, a tensor normal regression is multivariate normal distribution generalized to a high dimension. An M^{th} order random tensor X has a total of $\sum_{i=1}^{M} p_i$ means and M covariate matrices each of the dimension $p_i \times p_i$ for $i = 1, \cdots, M$. We can find that the density of X has the form

$$p(X|\mu; \Sigma_1, \ldots, \Sigma_M) = \frac{1}{(2\pi)^{p/2} |\Sigma_1|^{q_i/2} \cdots |\Sigma_M|^{q_M/2}} \exp\left(-\frac{1}{2} \left\langle [\![X - \mu; \Sigma_1^{-1}, \ldots, \Sigma_M^{-1}]\!], X - \mu \right\rangle \right), \tag{1}$$

where $p = p_1 p_2 \cdots p_M$ and $q_i = p/p_i$.

We will consider independent tensor-variate observations in $\mathbb{R}^{p_1, \times \cdots \times p_M}$ drawn from K clusters with the same within-class covariance matrices; suppose that μ_i's are the mean tensor of the kth cluster. Let π_k be the probability of an observation to be taken from the kth cluster.

Then the sample $\{X_i\}_{i=1}^{n}$ from a mixture of the tensor normal distributions can be written as the following:

$$X_i \sim \sum_{k=1}^{K} \pi_k \mathrm{TN}(\mu_k; \Sigma_1, \ldots, \Sigma_M), \quad i = 1, 2, \ldots, n,$$

or equivalently,

$$P(Y_i = k) = \pi_k X_i | Y_i = k \sim \mathrm{TN}(\mu_k; \Sigma_1, \ldots, \Sigma_M), \quad i = 1, 2, \ldots, n. \tag{2}$$

Hence, Y_i indicates the number of the cluster from which X_i was taken, and if $Y_i = k$ is given, X_i has the tensor normal distribution with the mean μ_k of the cluster k and the within-class covariance matrices $\Sigma_1, \ldots, \Sigma_M$. (Recall we assume that the clusters have the same within-class matrices.)

Suppose that $\{X_i\}_{i=1}^{n}$ is a sample from the model (2). Let $\theta = \{\pi_i, \mu_i, \Sigma_j : 1 \leq i \leq K, 1 \leq j \leq M\}$ denote the set of all parameters in the model. If we can observe Y_i, then the complete log-likelihood can be obtained as follows:

$$\ell_c(\theta; X, Y) = \log \prod_{i=1}^{n} \pi_{Y_i} p(X_i|\mu_{Y_i}; \Sigma_1, \dots \Sigma_M) = \sum_{i=1}^{n} [\log \pi_{Y_i} + \log p(X_i|\mu_{Y_i}, \Sigma_1, \dots, \Sigma_M)].$$

But, in general, we cannot observe Y_i; hence, from an initial value $\widetilde{\theta}^{(0)}$, we create a sequence $\widetilde{\theta}^{(t)}$ through the E-step to obtain the Q function

$$Q(\theta; \widetilde{\theta}^{(t)}) = E_{Y|X,\widetilde{\theta}^{(t)}}[\ell_c(\theta; X, Y)] = \sum_{i=1}^{n} \sum_{k=1}^{K} \widetilde{\xi}_{ik}^{(t)} [\log \pi_k + \log p(X_i|\mu_k, \Sigma_1, \dots \Sigma_M)]$$

where

$$\widetilde{\xi}_{ik}^{(t)} = P(Y_i = k|X, \widetilde{\theta}^{(t)}) = \frac{\widetilde{\pi}_k^{(t)} p(X_i|\widetilde{\mu}_k^{(t)}, \widetilde{\Sigma}_1^{(t)}, \dots, \widetilde{\Sigma}_M^{(t)})}{\sum_{j=1}^{K} \widetilde{\pi}_j^{(t)} p(X_i|\widetilde{\mu}_j^{(t)}, \widetilde{\Sigma}_1^{(t)}, \dots, \widetilde{\Sigma}_M^{(t)})} \tag{3}$$

and the M-step to update the parameter

$$\widetilde{\theta}^{(t+1)} = \operatorname*{argmax}_{\theta} Q(\theta; \widetilde{\theta}^{(t)}).$$

Then the EM sequence $\widetilde{\theta}^{(t)}$ converges to the MLE, but there are some issues in our situation: Getting the updates for π_k and μ_k is quite easy and straightforward, but it is challenging to obtain the updates for the covariance matrices Σ_i. When we compute $\widehat{\xi}_{ik}^{(t)}$ in (3), all the elements in X_i are used, and the standard EM algorithm does not involve a process for variable selection. Thus, due to an excessive number of parameters in the model, it may lead to the accumulation of errors, which potentially results in inaccurate estimates.

To overcome these problems, we introduce the enhanced E-step, where we replace $\widetilde{\xi}_{ik}^{(t)}$ with an estimator $\widehat{\xi}^{(t)}$ that can be calculated relatively faster under the sparsity assumption. We want to find the objective function Q^{DEEM} that has a better property than the standard Q function above. First, it can be seen that

$$\xi_{i1} = P(Y_i = 1|X_i, \theta) = \frac{\pi_1}{\pi_1 + \sum_{k=2}^{K} \pi_k \exp\left[\langle X_i - (\mu_k + \mu_1)/2, B_k \rangle\right]} \tag{4}$$

and

$$\xi_{ik} = P(Y_i = k|X_i, \theta) = \frac{\pi_k \exp\left[\langle X_i - (\mu_k + \mu_1)/2, B_k \rangle\right]}{\pi_1 + \sum_{j=2}^{K} \pi_j \exp\left[\langle X_i - (\mu_j + \mu_1)/2, B_j \rangle\right]} \tag{5}$$

for $k \geq 2$, where

$$B_k = [\![\mu_k - \mu_1; \Sigma_1^{-1}, \dots, \Sigma_M^{-1}]\!] \in \mathbb{R}^{p_1, \times \cdots \times p_M}.$$

Note that we are considering the clustering problem, that is, we are interested in recovering Y_i's and finding a method that minimizes the clustering error. It can be shown that the covariance matrices Σ_i are nuisance parameters in the optimal clustering rule, i.e., the values of Σ_i are not used in the optimal clustering rule if we already know B_k's. To cover more general cases, we do not impose conditions on Σ_i. Instead, we assume the sparsity condition on B; for $B_k = [b_k^J]_J$, where $J = (j_1, \dots, j_M)$ denotes an index of the tensor, we impose the condition $b_2^J = \cdots = b_K^J = 0$ for almost every J. In other words, if $D = \{J : b_k^J \neq 0 \text{ for some } k = 2, \dots, K\}$, then we assume that the number of elements in D is significantly smaller than $p = p_1 p_2 \cdots p_M$. This assumption comes from the belief that, in the high-dimensional setting, most of the variables are not significant in estimation. The above expressions for ξ_{ik} show that this assumption reduces the computational cost and improves the estimation efficiency.

If we accept the fact that $(B_2, \dots, B_K)$ minimizes the quantity

$$\sum_{k=2}^{K} \left(\langle B_k, [\![B_k, \Sigma_1, \dots, \Sigma_M]\!] \rangle - 2\langle B_k, \mu_k - \mu_1 \rangle \right),$$

then it is reasonable to obtain the sequence of estimates $\widehat{B}_k^{(t+1)}$ by solving the optimization problem

$$\operatorname*{argmin}_{B_2,\dots,B_K} \sum_{k=2}^{K} \left(\langle B_k, [\![B_k, \widehat{\Sigma}_1^{(t)}, \dots, \widehat{\Sigma}_M^{(t)}]\!] \rangle - 2\langle B_k, \widehat{\mu}_k^{(t)} - \widehat{\mu}_1^{(t)} \rangle \right) + \lambda^{(t+1)} \sum_J \left(\sum_{k=2}^{K} (b_k^J)^2 \right),$$

where we added the lasso penalty term to satisfy the sparsity assumption to some extent. Using these $\widehat{B}_k^{(t+1)}$, we can obtain the sequence $\widehat{\xi}_{ik}^{(t+1)}$ by replacing the parameters in (4) and (5) with their estimates. Then, the objective Q^{DEEM} is defined using $\widehat{\xi}^{(t)}$ as follows:

$$Q^{\mathrm{DEEM}}(\theta; \widehat{\theta}^{(t)}) = \sum_{i=1}^{n} \sum_{k=1}^{K} \widehat{\xi}_{ik}^{(t)} [\log \pi_k + \log p(X_i | \mu_k, \Sigma_1, \dots, \Sigma_M)].$$

In light of the sparsity assumption, $\widehat{\xi}_{ik}^{(t)}$ can be computed based on the values of relatively smaller variables.

In M-step, the parameters can be updated inductively from the proposed Q^{DEEM} function: The estimates for π_k and μ_k can be obtained by the formula

$$\widehat{\pi}_k^{(t+1)} = \frac{1}{n} \sum_{i=1}^{n} \widehat{\xi}_{ik}^{(t+1)} \quad \text{and} \quad \widehat{\mu}_k^{(t+1)} = \frac{\sum_{i=1}^{n} \widehat{\xi}_{ik}^{(t+1)} X_i}{\sum_{i=1}^{n} \widehat{\xi}_{ik}^{(t+1)}}, \quad k = 1, 2, \dots, K.$$

Then given $\xi_{ik}^{(t+1)}$, we calculate the intermediate covariance matrices

$$\breve{\Sigma}_j^{(t+1)} = \frac{1}{nq_j} \sum_{i=1}^{n} \sum_{k=1}^{K} \widehat{\xi}_{ik}^{(t+1)} (X_i - \widehat{\mu}_k^T(t+1))_{(j)} (X_i - \widehat{\mu}_k^T(t+1))_{(j)}^T,$$

and the conditional variance of $X_i^{1\ldots1}$

$$(\widehat{\sigma}_1^{11})^{(t+1)} = \frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}\widehat{\xi}_{ik}^{(t+1)}(X_i^{1\ldots1} - (\widehat{\mu}_k^{1\ldots1})^{(t+1)})^2.$$

The target covariance estimator is given by scaling the intermediate covariances with $(\widehat{\sigma}_1^{11})^{(t+1)}$ and $(\breve{\sigma}_1^{11})^{(t+1)}$:

$$\widehat{\Sigma}_j^{(t+1)} = \begin{cases} \dfrac{1}{(\breve{\sigma}_j^{11})^{(t+1)}}\breve{\Sigma}_j^{(t+1)} & \text{if } j \geq 2, \\ \dfrac{(\widehat{\sigma}_1^{11})^{(t+1)}}{(\breve{\sigma}_1^{11})^{(t+1)}}\breve{\Sigma}_1^{(t+1)} & \text{if } j = 1. \end{cases}$$

The general process is summarized in Algorithm 1 in the Appendix as our Figure 6.

Now we are interested in how this sequence of parameters $\widehat{\theta}^{(t)}$ behave. In fact, under some initialization condition, it can be seen that there are some constant C and $0 < \kappa < 1/2$ such that for large t, with a probability $\geq 1 - O(\prod_{i=1}^{M} p_i^{-1})$,

$$\|\widehat{B}^{(t)} - B\| \leq C\sqrt{\frac{s\Sigma_{i=1}^{M}\log p_i}{n}},$$

where $s = o(\sqrt{n/\sum_i \log p_i})$ and d_0 is a measure of the difference between the initial value $\widehat{\theta}^{(0)}$ and the true parameter θ. This result implies that if the number of iterations t is large, the DEEM estimator $\widehat{B}^{(t)}$ converges to the true parameter B. Here, the condition $s = o(\sqrt{n/\sum_i \log p_i})$ means the sparsity assumption.

We consider the error rate of DEEM and the optimal clustering rule:

$$R(DEEM) = \min_{\Pi} P(\Pi(\widehat{Y}_i^{DEEM} \neq Y_i)) \quad \text{and} \quad R(Opt) = P(\widehat{Y}_i^{opt} \neq Y_i),$$

where Π denotes the permutation operator and $\widehat{Y}_i^{DEEM} = \text{argmax}_k\, \widehat{\xi}_{ik}$. Here, the optimal clustering rule is optimal in the sense that it minimizes the clustering error. From the result above, we can show that if t is large, then with probability $\geq 1 - O(\prod_{i=1}^{M} p_i^{-1})$

$$R(DEEM) - R(Opt) \leq C\frac{s\sum_{i=1}^{M}\log p_i}{n}.$$

Consequently, this result shows that the error rate of DEEM converges to the error rate of the optimal clustering rule if t is large.

3 Simulation Study

3.1 Data Generation

For our simulation studies, we follow the framework used in Qing Mai and Deng (2022). For each setting, K denotes the number of mixture groups, and noise is generated as a M^{th}-order tensor:

$$\mathbf{X}_i \sim \sum_{k=1}^{K} \pi_k^* \mathrm{TN}(\boldsymbol{\pi}_k^*; \boldsymbol{\Sigma}_1^*, \cdots, \boldsymbol{\Sigma}_{\boldsymbol{M}}^*), i = 1, \cdots, n \tag{6}$$

For $K-1$ mixture groups, the $\mathbf{X}_i$ is given as a given $\mathbf{B}_k$ plus the noise above. For 1 mixture group, the values are simply the noise. Qing Mai and Deng designate two types of $\boldsymbol{\Sigma}_k^*$:

$$\boldsymbol{\Omega} = \begin{cases} AR(\rho): & \omega_{ij} = \rho^{|i-j|} \\ CS(\rho): & \omega_{ij} = \rho + (1-\rho)1(i=j). \end{cases}$$

The covariance matrices are not sparse if they are set using the two formats above. For each setting, we generate 100 independent datasets, the same number of replicates as used by the authors, and present the mean error rate and standard deviation.

3.2 Settings

The settings are provided in Table 1. Note that, for B_k^*, the indices not included in the subscript is 0. In other words, B_k^* is a sparse tensor. Both the DEEM and EM algorithms require K to be given. λ is a tuning parameter for regularization; however, due to the long computation time, we experiment with several different values of λ and set it at 0.05 instead of tuning it for each setting. We encourage interested readers to try out two functions–**tune_K** and **tune_lambda**–in the R package **TensorClustering**.

We choose four settings from the seven settings, because these settings are increasingly more computationally expensive, and we believe that they demonstrate the advantage of the DEEM algorithm compared to the classical EM algorithm in terms of accuracy, as shown in Table 2. We also run three extra settings, which we call M8, M9, M10 to avoid confusion with the seven settings in the paper. The results for these extra settings are shown in Tables 4 and 5.

Setting	Parameters
M1	$K = 2, p = 10 \times 10 \times 4, \mathbf{\Sigma}_1^* = CS(0.3), \mathbf{\Sigma}_2^* = AR(0.8), \mathbf{\Sigma}_3^* = CS(0.3), \mathbf{B}^*_{2,[1:6,1,1]} = 0.5$
M3	$K = 3, p = 10 \times 10 \times 4, \mathbf{\Sigma}_1^* = CS(0.3), \mathbf{\Sigma}_2^* = AR(0.8), \mathbf{\Sigma}_3^* = CS(0.5), \mathbf{B}^*_{2,[1:6,1,1]} = 0.5, \mathbf{B}^*_{3,[1:6,1,1]} = -0.5$
M4	$K = 4, p = 10 \times 10 \times 4, \mathbf{\Sigma}_1^* = \mathbf{I}_{10}, \mathbf{\Sigma}_2^* = AR(0.8), \mathbf{\Sigma}_3^* = \mathbf{I}_4, \mathbf{B}^*_{2,[1:6,1,1]} = 0.8, \mathbf{B}^*_{3,[1:6,1,1]} = -0.8$
M5	$K = 6, p = 10 \times 10 \times 4, \mathbf{\Sigma}_1^* = AR(0.9), \mathbf{\Sigma}_2^* = CS(0.6), \mathbf{\Sigma}_3^* = AR(0.9), \mathbf{B}^*_{2,[1:6,1,1]} = 0.6, \mathbf{B}^*_{3,[1:6,1,1]} = 1.2, \mathbf{B}^*_{4,[1:6,1,1]} = 1.8, \mathbf{B}^*_{5,[1:6,1,1]} = 2.4, \mathbf{B}^*_{6,[1:6,1,1]} = 3$
M8	$K = 2, p = 10 \times 10 \times 10, \mathbf{\Sigma}_1^* = AR(0.5), \mathbf{\Sigma}_2^* = CS(0.5), \mathbf{\Sigma}_3^* = AR(0.5), \mathbf{B}^*_{2,[1:6,1,1]} = 1.5$
M9	$K = 2, p = 10 \times 10 \times 10, \mathbf{\Sigma}_1^* = \mathbf{I}_{10}, \mathbf{\Sigma}_2^* = \mathbf{I}_{10}, \mathbf{\Sigma}_3^* = \mathbf{I}_{10}, \mathbf{B}^*_{2,[1:6,1,1]} = 1.5$
M10	$K = 2, p = 10 \times 10 \times 4 \times 4 \times 4, \mathbf{\Sigma}_1^* = AR(0.5), \mathbf{\Sigma}_2^* = CS(0.5), \mathbf{\Sigma}_3^* = AR(0.5), \mathbf{\Sigma}_4^* = \mathbf{I}_4, \mathbf{\Sigma}_5^* = \mathbf{I}_4, \mathbf{B}^*_{2,[1:6,1,1,1,1]} = 5$

Table 1: Simulation settings

3.3 Metrics

Note that it is as straightforward to calculate the mean error rate for a clustering problem as it is for a classification problem. Both methods return labels for the groups; however, the group labels do not matter. For example, if there are five observations and their true group labels are $(1, 1, 2, 2, 2)$ and the methods return $(2, 2, 1, 1, 1)$, the error rate should be 0. In the paper, the authors explain that the mean clustering error rate is calculated by:

$$\min_{\Pi} \frac{1}{n} \sum_{i=1}^{n} 1(\hat{Y}_i \neq \Pi(Y_i)) \text{ over all possible permutations } \Pi : \{1, \cdots, \} \mapsto \{1, \cdots, K\}$$

We create a function to permute the true labels, compare the estimated labels and the true labels, and return the lowest error rate. To compare the speed of the two methods, we also record the computation time. Tables 3 and 5 provides the mean computation time and standard error (in parentheses) for each setting.

3.4 External R Packages and Functions

For the DEEM algorithm, we use the function DEEM; for the standard EM algorithm, we use the function TGMM. Both functions are from the R package TensorClustering. We use the Trnorm function from the R package Tlasso to generate tensor noise with designated covariance matrices. We use the permutations function in the gtools package to permute true labels. In short, be sure to install the three R packages: TensorClustering, Tlasso, and gtools if you would like to reproduce our simulation.

3.5 Simulation result

The error rates and computation time are shown in Tables 2 and 3. It is clear that DEEM has lower mean error rates in all four settings. The error rates are in general higher than those given in the article, possibly because the hyperparameters are not tuned for each setting in our case.

The computation time tells a different story, however. As seen in Table 3, DEEM is not always the winner in terms of time. As the setting becomes more complicated and estimating the clusters becomes more challenging, it takes longer for DEEM to converge. For setting M5, it is possible that DEEM reached the maximum iterations for some runs.

Setting	DEEM	EM
M1	0.41 (0.05)	0.45 (0.03)
M3	0.46 (0.09)	0.56 (0.05)
M4	0.35 (0.03)	0.57 (0.06)
M5	0.31 (0.11)	0.43 (0.06)

Table 2: Error Rates from 100 Replicates

Setting	DEEM	EM
M1	0.72 (0.45)	0.93 (0.39)
M3	13.95 (7.78)	7.74 (3.99)
M4	15.8 (0.81)	21.9 (9.68)
M5	332.96 (124.38)	14.66 (5.88)

Table 3: Computation Time (seconds) from 100 Replicates

Next, we transform the values in the tables into figures. As shown in Figure 4, DEEM always has lower mean error rates. However, as the model becomes complicated, DEEM's error rates become more varied, even though the mean rate is still lower. In Figure 5, the story seems more complicated. (Note that we cannot use the same y-axis for all four plots, because the computation time for DEEM for M5 is so long, which would make some of the boxes very small and not informative.) For the two settings M1 and M4, DEEM has a lower computation time. For M3, the computation time for DEEM is much more varied, and EM has an overall shorter computation time. For M5, DEEM has a very long computation time; in fact, the 100 replicates took almost 10 hours. It is unclear if the reduced error rate is worth the computational cost.

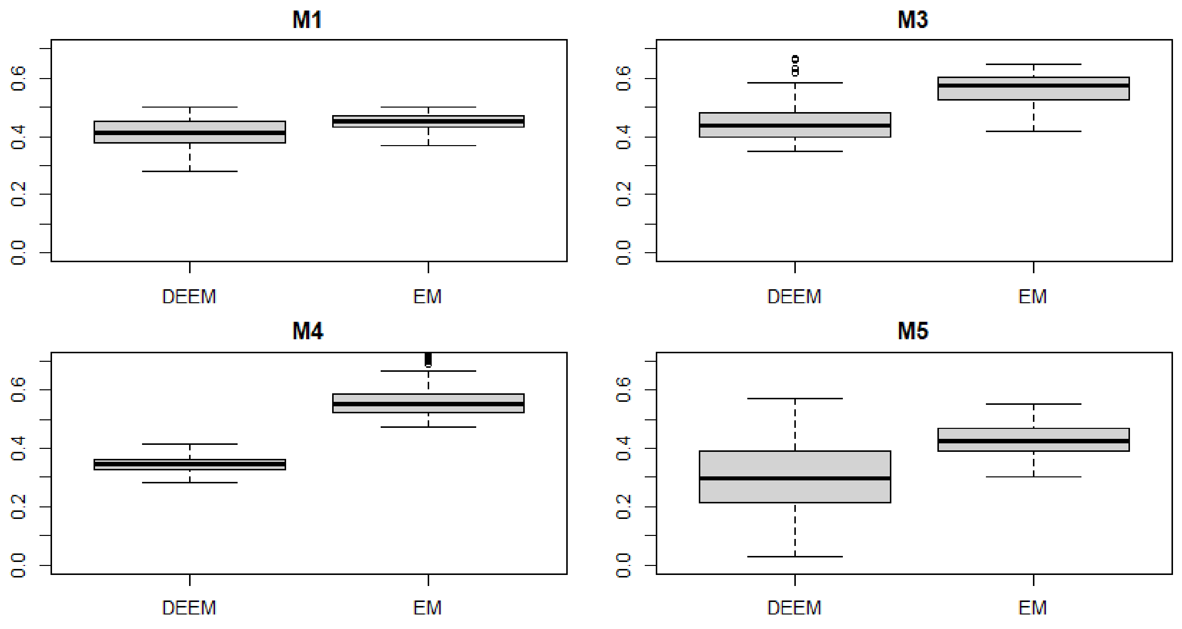


Figure 4: Boxplots of Mean Error Rates from 100 Replicates

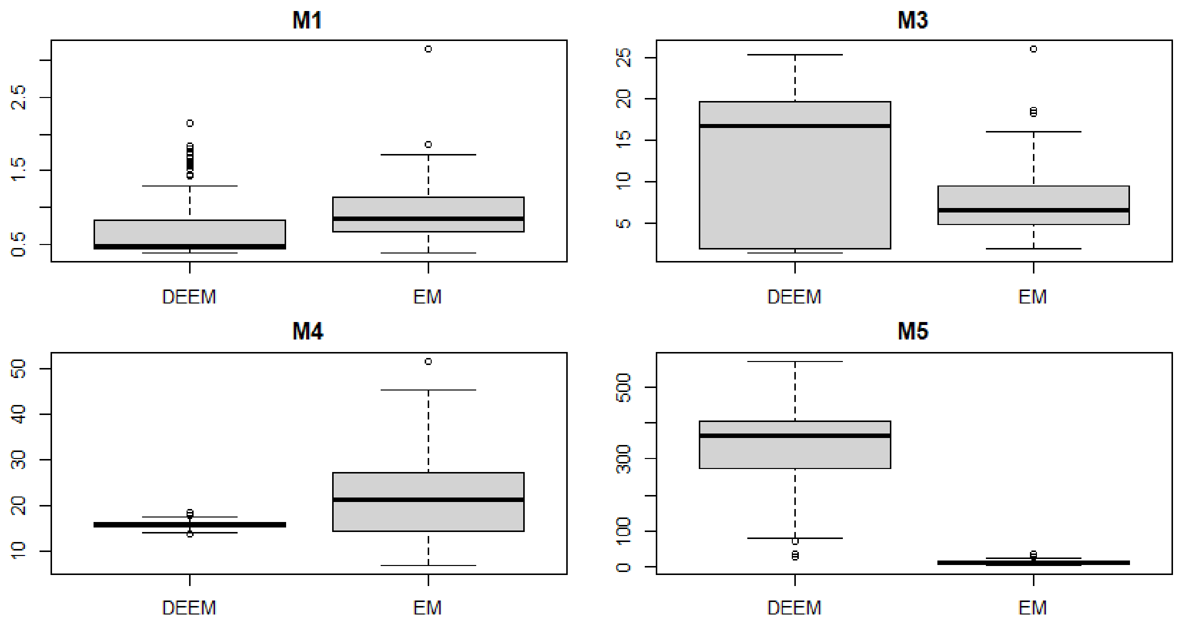


Figure 5: Boxplots of Mean Computation Time (in seconds) from 100 Replicates

In terms of the extra settings, we noticed that the error for DEEM is much lower than EM for M8, for which we set two clusters with the covariance matrices to have a moderate correlation. For M9, we let the setting be an easy case, since the covariance matrices are all identity matrices, and DEEM still has a much lower mean error rate than EM. For M10, we use a fifth-order tensor and make the estimating task an easy one. As seen in Table 4, DEEM has a much lower mean error rate than EM, although its computation time is again quite long.

One recurrent problem from the simulation is that DEEM has a much longer running time than EM. We leave this question about DEEM's computation time to interested readers.

Setting	DEEM	EM
M8	0.28 (0.11)	0.45 (0.03)
M9	0.18 (0.05)	0.34 (0.10)
M10	0.0003 (0.002)	0.31 (0.13)

Table 4: Error Rates from 100 Replicates

Setting	DEEM	EM
M8	23.7 (4,3)	0.17 (0.07)
M9	7.99 (1.88)	0.32 (0.09)
M10	72.8 (211.41)	1.51 (0.08)

Table 5: Computation Time (seconds) from 100 Replicates

4 Summary

In this blogpost, we review a new method proposed by Qing Mai and Deng (2022), which is essentially an upgraded version of the classical EM algorithm. This new method, DEEM, tends to have lower error rates on tensor data. However, despite the paper's claim that the enhanced M step in the DEEM algorithm facilitates fast covariance estimation, we have encountered situations where the running time could be prohibitive. While DEEM proves to be efficient and effective in handling tensor data, there remains potential for further enhancement.

References

Dempster, A., Laird, N. and Rubin, D. (1977) Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, **39**, 1–22.

Kolda, T. G. and Bader, B. W. (2009) Tensor decompositions and applications. *SIAM Review*, **51**, 455–500. URL`https://doi.org/10.1137/07070111X`.

Qing Mai, Xin Zhang, Y. P. and Deng, K. (2022) A doubly enhanced em algorithm for model-based tensor clustering. *Journal of the American Statistical Association*, **117**, 2120–2134.

Yang, M.-S., Lai, C.-Y. and Lin, C.-Y. (2012) A robust em clustering algorithm for gaussian mixture models. *Pattern Recognition*, **45**, 3950–3961.

Appendix

Algorithm 1 DEEM algorithm

```
1: **Initialize** $\hat{\pi}_k^{(0)}, \hat{\mu}_k^{(0)}, \hat{\Sigma}_m^{(0)}$
2: **for** $t = 0, 1, \ldots,$ **do**                ▷ *repeat the following steps until convergence*
     1. The enhanced E-step:
        (a) Minimize the following convex objective function over $\mathbf{B}_2, \ldots, \mathbf{B}_k \in \mathbb{R}^{p_1 \times \cdots \times p_M}$:
            $$\sum_{k=2}^{k} (\langle \mathbf{B}_k, [\![\mathbf{B}_k, \widehat{\Sigma}_1^{(t)}, \ldots, \widehat{\Sigma}_M^{(t)}]\!] \rangle - 2\langle \mathbf{B}_k, \widehat{\mu}_k^{(t)} - \widehat{\mu}_1^{(t)} \rangle) + \lambda^{(t+1)} \sum_{J} \sqrt{\sum_{k=2}^{K} b_{k,J}^2}$$
            In here, $\hat{\mathbf{B}}_2^{(t+1)}, \ldots, \hat{\mathbf{B}}_K^{(t+1)}$ denote the solution.
        (b) For $i = 1, \ldots, n$ and $k = 1, \ldots, K$, calculate the probabilities
            $$\widehat{\xi}_{ik}^{(t+1)} = \begin{cases} \frac{\widehat{\pi}_1^{(t+1)}}{\widehat{\pi}_1^{(t+1)} + \sum_{j=2}^{K} \widehat{\pi}_j^{(t+1)} \exp\left[\langle X_i - \frac{1}{2}(\widehat{\mu}_j^{(t+1)} + \widehat{\mu}_1^{(t+1)}), \widehat{B}_j^{(t+1)} \rangle\right]}, & k = 1 \\ \frac{\widehat{\pi}_k^{(t+1)} \exp\left[\langle X_i - \frac{1}{2}(\widehat{\mu}_k^{(t+1)} + \widehat{\mu}_1^{(t+1)}), \widehat{B}_k^{(t+1)} \rangle\right]}{\widehat{\pi}_1^{(t+1)} + \sum_{j=2}^{K} \widehat{\pi}_j^{(t+1)} \exp\left[\langle X_i - \frac{1}{2}(\widehat{\mu}_j^{(t+1)} + \widehat{\mu}_1^{(t+1)}), \widehat{B}_j^{(t+1)} \rangle\right]}, & k > 1 \end{cases}$$
     2. The enhanced M-step:
        (a) Update $\widehat{\pi}_k^{(t+1)} = \sum_{i=1}^{n} \widehat{\xi}_{ik}^{(t+1)}/n$ and $\widehat{\mu}_k^{(t+1)} = \sum_{i=1}^{n} \widehat{\xi}_{ik}^{(t+1)} X_i / \sum_{i=1}^{n} \widehat{\xi}_{ik}^{(t+1)}$
        (b) Compute intermediate covariance estimators
            $$\breve{\Sigma}_m^{(t+1)} = \frac{1}{nq_m} \sum_{i=1}^{n} \sum_{k=1}^{K} \widehat{\xi}_{ik}^{(t+1)} (X_i - \widehat{\mu}_k^{(t+1)})_{(m)} (X_i - \widehat{\mu}^{(t+1)})_{(m)}^T$$
        (c) Scale $\breve{\Sigma}_m^{(t+1)}$ to be
            $$\widehat{\Sigma}_m^{(t+1)} = \begin{cases} \{\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} \widehat{\xi}_{ik}^{(t+1)} (X_{i,1\ldots1} - \widehat{\mu}_{k,1\ldots1}^{(t+1)})^2\} \breve{\Sigma}_m^{(t+1)} / \breve{\sigma}_{1,11}^{(t+1)}, & m = 1 \\ \breve{\Sigma}_m^{(t+1)} / \breve{\sigma}_{1,11}^{(t+1)}, & m > 1 \end{cases}$$
3: **end for**
4: Output $\widehat{\xi}_{ik}, \widehat{\pi}_k, \widehat{\Sigma}_m$ at convergence.
```

Figure 6: DEEM algorithm