Scalable Nonparametric Multiway Data Analysis

Shandian Zhe
Department of Computer Science
Purdue University
szhe@purdue.edu

Zenglin Xu
Department of Computer Science
University of Electronic Science and Technology of China
zlxu@uestc.edu.cn

Xinqi Chu
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
chu36@illinois.edu

Yuan Qi
Department of Computer Science
Purdue University
alanqi@cs.purdue.edu

Youngja Park
IBM Thomas J. Watson Research Center
young_ park@us.ibm.com

Abstract

Multiway data analysis deals with modeling of multiway arrays, i.e., tensors. The goal of multiway data analysis is twofold: predicting missing entries by modeling the interactions between array elements and discovering hidden patterns, such as clusters or communities in each mode. Despite the success of existing tensor factorization approaches, they are either unable to capture nonlinear interactions embedded in data, or computationally expensive to handle massive data. In addition, most of the existing methods lack a principled way to discover latent clusters, which is important for better understanding of the multiway data. To address these issues, we propose a scalable nonparametric tensor decomposition model. It employs the Dirichlet process mixture (DPM) prior to model the latent clusters; it uses local Gaussian processes (GPs) to capture nonlinear relationships and to improve scalability. An efficient online variational Expectation-Maximization algorithm is proposed for learning the model. Experiments on both synthetic and real-world data show that the proposed model is capable to discover latent clusters with higher prediction accuracy than competitive tensor decomposition methods. Furthermore, the proposed model obtains significantly better predictive performance than the state-of-the-art large scale tensor decomposition algorithm, GigaTensor, on two large datasets with billions of entries.

1 Introduction

Many multiple aspect data can be described by multiway arrays, i.e., tensors. For example, an access log database can be represented by an array with three modes (*user, file, action*) and patient-drug responses by an array with four modes (*person, medicine, biomarker, time*). Given such an array, we want to capture complex interactions between the array elements and predict the missing entries (*e.g.,* unknown drug response); furthermore, we want to discover the hidden patterns embedded in the data, such as clusters or communities of the nodes or objects in each mode. For example, we want to find out groups of abnormal users who may threaten the system security; we want to discover sets of people with identical characteristics to develop personalized medicines.

A number of approaches have been proposed for multiway data analysis, such as CANDECOM/PARAFAC (CP) [8], Tucker decomposition [21] and its generalization [4], and infinite Tucker decomposition (InfTucker) [23]. Although very useful, these approaches have their own limitations. For example, the popular multilinear factorization methods, such as PARAFAC and Tucker decomposition, cannot capture the nonlinear relationships between array elements; although nonparametric models, such as InfTucker, can model the nonlinear relationships by latent Gaussian processes (GPs), they suffer a prohibitive high training cost and cannot handle massive data in real applications; besides, most of them lack a principled way to discover the latent clusters, which is important in data analysis and knowledge discovery.

To address these limitations, a novel scalable nonparametric model is proposed in this paper. First, to model the nonlinear interactions between array elements, we exploit a tensor-variate Gaussian process [23] defined on latent factors, where the similarity between array elements can be described by arbitrary kernel or covariance functions. Second, to scale up the model, we relax the global GP used by [23] and employ a local GP assumption instead. Specifically, the whole array is sliced into many small subarrays, each of which is generated from a local latent tensor-variate GP. Moreover, to grasp the hidden clusters, we employ Dirichlet Process Mixture (DPM) prior—a nonparametric prior which can model an undetermined number of clusters—over the latent factors. Finally, an efficient online variational Bayesian Expectation-Maximization (VB-EM) algorithm is developed for model estimation. The algorithm sequentially processes each subarray: in the E-step, it caches global statistics and updates them by calculating local statistics only, resulting in an efficient update of variational posteriors; in the M-step, it optimizes the latent factors using stochastic gradient descent. Compared with the global GP model, i.e.,InfTucker, which requires to store the whole array in the main memory and calculates the huge covariance matrix of all the array elements, the online VB-EM algorithm only stores subarrays and their covariance matrices of much smaller size, and is therefore feasible for large array analysis.

For evaluation, the proposed approach is first examined on three small real-world datasets where InfTucker is feasible. Our model achieves higher prediction accuracy than InfTucker and other mulitlinear alternatives. Moreover, a simulation shows that our approach is able to capture the latent clusters in a tensor which contains nonlinear relationships. Finally, our approach is applied to analyze two real-world large multiway arrays with billions of entries. The comparison with the state-of-the-art large scale tensor decomposition algorithm, GigaTensor [12], shows that our approach obtains significantly better predictive performance.

2 Background on Tensor Decomposition

We first introduce the notations. A K-mode multiway array or tensor is denoted by $\mathcal{M} \in \mathbb{R}^{m_1 \times \ldots \times m_K}$, where the k-th mode has a dimension of m_k, corresponding to m_k nodes or objects. We use $m_{\mathbf{i}}$ ($\mathbf{i} = (i_1, \ldots, i_K)$) to denote $\mathcal{M}$'s entry at location $\mathbf{i}$. Using the vectorization operation, we can stack all of $\mathcal{M}$'s entries in a vector, $\mathrm{vec}(\mathcal{M})$, with size $\prod_{k=1}^{K} m_k$ by 1. In $\mathrm{vec}(\mathcal{M})$, the entry $\mathbf{i} = (i_1, \ldots, i_K)$ of $\mathcal{M}$ is mapped to the entry at position $j = i_K + \sum_{i=1}^{K-1}(i_k - 1)\prod_{k+1}^{K} m_k$. Given a tensor $\mathcal{W} \in \mathbb{R}^{r_1 \times \ldots \times r_K}$ and a matrix $\mathbf{U} \in \mathbb{R}^{s \times r_k}$, a mode-$k$ tensor-matrix multiplication between $\mathcal{W}$ and $\mathbf{U}$ is denoted by $\mathcal{W} \times_k \mathbf{U}$, which is a tensor of size $r_1 \times \ldots \times r_{k-1} \times s \times r_{k+1} \times \ldots \times r_K$. The corresponding entry-wise definition is $(\mathcal{W} \times_k \mathbf{U})_{i_1 \ldots i_{k-1} j i_{k+1} \ldots i_K} = \sum_{i_k=1}^{r_k} w_{i_1 \ldots i_K} u_{j i_k}$.

The Tucker decomposition of a K-mode tensor $\mathcal{M}$ is defined by

$$\mathcal{M} = [\![\mathcal{W}; \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(K)}]\!] = \mathcal{W} \times_1 \mathbf{U}^{(1)} \times_2 \ldots \times_K \mathbf{U}^{(K)} \tag{1}$$

where $\mathcal{W} \in \mathbb{R}^{r_1 \times \ldots \times r_K}$ is the core tensor, and $\mathbf{U}^{(k)} \in \mathbb{R}^{m_k \times r_k}$ is the k-th latent factor matrix. The Tucker decomposition can also be represented in a vectorized form, $\mathrm{vec}([\![\mathcal{W}; \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(K)}]\!]) = \mathbf{U}^{(1)} \otimes \ldots \otimes \mathbf{U}^{(K)} \cdot \mathrm{vec}(\mathcal{W})$ where $\otimes$ is the Kronecker product. If we enforce $r_1 = \ldots = r_K$ and restrict the core tensor $\mathcal{W}$ to be diagonal (i.e., $W_{i_1 \ldots i_K} \neq 0$ only if $i_1 = \ldots = i_K$), it reduces to PARAFAC decomposition.

PARAFAC and Tucker decomposition are multilinear factorization methods and cannot model the nonlinear interactions in tensor (see Equation (1)). The infinite Tucker decomposition (InfTucker) [23] is a nonparametric Bayesian model that maps the latent factors into an infinite feature space and then performs the Tucker decomposition with the core tensor $\mathcal{W}$ of infinite size. Based on the feature mapping, InfTucker can capture nonlinear relationships. Specifically, InfTucker is originated by assigning an element-wise standard Gaussian prior over the core tensor $\mathcal{W}$, i.e., $\mathrm{vec}(\mathcal{W}) \sim \mathcal{N}(\mathrm{vec}(\mathcal{W}); \mathbf{0}, \mathbf{I})$; then by marginalizing out $\mathcal{W}$, we can obtain the marginal distribution for the tensor $\mathcal{M}$:

$$p(\mathcal{M}|\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(K)}) = \mathcal{N}(\text{vec}(\mathcal{M}); \mathbf{0}, \Sigma^{(1)} \otimes \ldots \otimes \Sigma^{(K)}) \tag{2}$$

where $\Sigma^{(k)} = \mathbf{U}^{(k)}\mathbf{U}^{(k)\top}$. To capture nonlinear relationships, we replace each row $\mathbf{u}_t^k$ of the latent factors $\mathbf{U}^{(k)}$ by a nonlinear feature mapping $\phi(\mathbf{u}_t^k)$ and then obtain an equivalent nonlinear covariance matrix $\Sigma^{(k)} = k(\mathbf{U}^{(k)}, \mathbf{U}^{(k)})$ where $k(\cdot, \cdot)$ is a nonlinear covariance function. The core tensor $\mathcal{W}$ after feature mapping has the size of the mapped feature vector $\mathbf{u}_t^k$ on mode k, which could be infinite. Because the covariance of $\text{vec}(\mathcal{M})$ is the function of the latent factors $\mathcal{U} = \{\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(K)}\}$, Equation (2) actually defines a Gaussian process on tensor entries, where the input are based on the corresponding latent factors $\mathcal{U}$.

For an easy model interpretation, InfTucker assigns element-wise Laplace priors $p(\mathcal{U})$, which encourage sparse estimation. Given $\mathcal{U}$, a latent real-valued tensor $\mathcal{M}$ is sampled from the tensor-variate GP defined in Equation (2). Given $\mathcal{M}$, the observed tensor $\mathcal{Y}$ is sampled from a noisy model $p(\mathcal{Y}|\mathcal{M})$. For example, we can use probit models for binary observations and Gaussian models for continuous observations. Thus the joint distribution is $p(\mathcal{Y}, \mathcal{M}, \mathcal{U}) = p(\mathcal{U})p(\mathcal{M}|\mathcal{U})p(\mathcal{Y}|\mathcal{M})$.

3 Model

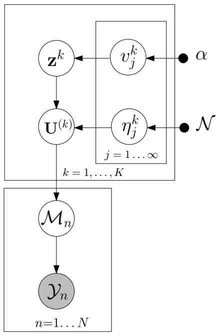


Figure 1: The graphical model representation.

Despite the capability of modeling nonlinear relationships, InfTucker has two bottlenecks: First, it cannot discover the latent cluster structures. Although it uses Laplace prior to enhance model interpretation, the effect is limited because the latent factors do not correspond to cluster memberships; and, their numbers could be different. Second, InfTucker cannot scale up to large data, making it impractical for many real-world applications. This stems from a global GP assumption : All the elements of the tensor $\mathcal{M}$ are sampled from a Gaussian process given the latent factors $\mathcal{U}$. As a result, computing the probability for the global $\mathcal{M}$—$p(\mathcal{M}|\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(K)})$ in Equation (2)– requires computing the Kronecker-product of the covariance matrices and its inverse. The matrix inversion is prohibitively expensive. Although [23] use the property of the Kronecker-product and avoid the naive computation, it still needs to perform eigen-decomposition over the covariance matrix for each mode, which is infeasible for a large dimension m_k.

To overcome these bottlenecks, we propose a novel, scalable nonparametric model: First, we assign a Dirichlet process mixture (DPM) [2] prior over the latent factors. DPM is a nonparametric mixture model that has unbounded number of mixture components (i.e., cluster centres). Using DPM can neatly capture an undetermined number of latent clusters. Then, we break the whole array into many, smaller subarrays, where each subarray is sampled from a separate, local tensor-variate GP based on the latent factors. This local GP assumption enables fast computation over subarrays and sequentially processing each subarry enables efficient online learning algorithm. The graphical representation of our model is shown in Figure 1.

Specifically, we assign the DPM prior over the latent factors $\mathbf{U}^{(k)}$ in each mode k. For the convenience of inference, we use the stick-breaking construction [18]: An infinite collection of random variables $\mathbf{v}^k = \{v_1^k, v_2^k, \ldots\}$ and an infinite set of atoms (i.e., cluster centres) $\boldsymbol{\eta}^k = \{\boldsymbol{\eta}_1^k, \boldsymbol{\eta}_2^k, \ldots\}$ are first generated by

$$p(\mathbf{v}^k|\alpha) = \prod_{j=1}^{\infty} \text{Beta}(v_j^k|1, \alpha), \qquad\qquad p(\boldsymbol{\eta}^k) = \prod_{j=1}^{\infty} \mathcal{N}(\boldsymbol{\eta}_j^k|\mathbf{0}, \mathbf{I})$$

where $\alpha > 0$ and the base measure is standard Gaussian. Then, to generate the latent factors $\mathbf{u}_t^k$ (which corresponds to t-th row in $\mathbf{U}^{(k)}$), a cluster assignment variable $\mathbf{z}_t^k$ is first sampled and $\mathbf{u}_t^k$ is generated according to the assigned cluster center,

$$p(\mathbf{u}_t^k, z_t^k|\mathbf{v}^k, \boldsymbol{\eta}^k) = p(z_t^k|\mathbf{v}^k)p(\mathbf{u}_t^k|z_t^k, \boldsymbol{\eta}^k) = \prod_{j=1}^{\infty} \left(\pi_j(\mathbf{v}_k)\right)^{\mathbb{1}(z_t^k = j)} \cdot \mathcal{N}(\mathbf{u}_t^k|\boldsymbol{\eta}_{z_t^k}^k, \lambda_k \mathbf{I})$$

where $\pi_j(\mathbf{v}^k) = v_j^k \prod_{i=1}^{j-1}(1 - v_i^k)$ and λ_k is the variance parameter which controls how far away $\mathbf{u}_t^k$ is from the cluster center. We use $\mathbf{z}^k = \{z_1^k, \ldots, z_{m_k}^k\}$ to denote the set of cluster assignment variables in mode k.

Given the latent factors $\mathcal{U} = \{\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(K)}\}$, we use Gaussian process to generate the observed array. As we mentioned earlier, the global GP used by InfTucker will cause a prohibitive high computational cost and therefore we use the local GP assumption instead: we break the whole array $\mathcal{Y}$ into many smaller subarrays $\{\mathcal{Y}_1, \ldots, \mathcal{Y}_N\}$; for each subarray $\mathcal{Y}_n$, a latent real-valued subarray $\mathcal{M}_n$ is generated by a local GP based on the corresponding subset of the latent factors $\mathcal{U}_n = \{\mathbf{U}_n^{(1)}, \ldots, \mathbf{U}_n^{(K)}\}$ and the noisy observation $\mathcal{Y}_n$ is sampled according to $\mathcal{M}_n$,

$$p(\mathcal{Y}_n, \mathcal{M}_n|\mathcal{U}) = p(\mathcal{M}_n|\mathcal{U}_n)p(\mathcal{Y}_n|\mathcal{M}_n) = \mathcal{N}(\mathrm{vec}(\mathcal{M}_n); \mathbf{0}, \Sigma_n^{(1)} \otimes \ldots \otimes \Sigma_n^{(K)}))p(\mathcal{Y}_n|\mathcal{M}_n)$$

where $\Sigma_n^{(k)} = k(\mathbf{U}_n^{(k)}, \mathbf{U}_n^{(k)})$ is the k-th mode covariance matrix over the sub-factors $\mathcal{U}_n$.

Now, the joint probability of our model is given by

$$p(\mathcal{U}, \{\mathbf{z}^k, \mathbf{v}^k, \boldsymbol{\eta}^k\}_{k=1}^K, \{\mathcal{M}_n, \mathcal{Y}_n\}_{n=1}^N)$$

$$= \prod_{k=1}^K p(\mathbf{v}^k|\alpha)p(\boldsymbol{\eta}^k) \prod_{t=1}^{m_k} p(z_t^k|\mathbf{v}^k)p(\mathbf{u}_t^k|z_t^k, \boldsymbol{\eta}^k) \prod_{n=1}^N p(\mathcal{M}_n|\mathbf{U}_n^{(1)}, \ldots, \mathbf{U}_n^{(K)})p(\mathcal{Y}_n|\mathcal{M}_n). \quad (3)$$

Compared with the joint probability of InfTucker, the joint probability of our model gets rid of the global factor $p(\mathcal{M}|\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(K)})$ and uses the product of smaller local factors $\prod_{n=1}^N p(\mathcal{M}_n|\mathbf{U}_n^{(1)}, \ldots, \mathbf{U}_n^{(K)})$ instead. These local factors requires much less memory and processing time than the global factor. More important, the additive nature of these local factors in the log domain enables us to design an efficient online learning algorithm.

4 Model Estimation

Now, we present our online variational Bayes Expectation Maximization (VB-EM) algorithm for model estimation:We randomly shuffle the subarrays and sequentially process each subarray with VB-EM: In the E-step, we use variational approximation and, in the M-step, we apply stochastic gradient descent (SGD) to maximize the variational lower bound over the latent factors. The details are given in the following paragraphs.

4.1 Variational approximation

We use variational inference to approximate the posteriors of the latent variables $(\{\mathbf{v}^k\}_k, \{\boldsymbol{\eta}^k\}_k, \{\mathbf{z}^k\}_k, \{\mathcal{M}_n\}_n)$—the random variables for stick-breaking construction, the cluster centres, the cluster assignments and the latent subarrays. Specifically, a fully factorized distribution is used $\prod_k q(\mathbf{v}^k)q(\boldsymbol{\eta}^k)q(\mathbf{z}^k)\prod_n q(\mathcal{M}_n)$ to approximate $p(\{\mathbf{v}^k, \boldsymbol{\eta}^k, \mathbf{z}^k\}_{k=1}^K, \{\mathcal{M}_n\}_{n=1}^N|\{\mathcal{Y}_n\}_{n=1}^N, \mathcal{U})$—the exact posterior. The variational inference minimizes the Kullback-Leibler (KL) divergence between the approximate and the exact posteriors by coordinate descent. The variational update for each $q(\mathcal{M}_n)$ is the same as that for $q(\mathcal{M})$ in [23]. The other latent variables come from the DPM prior, and they are infinite (*e.g.,* $\mathbf{v}^k$ and $\boldsymbol{\eta}^k$) or have infinite number of supports (*e.g.,* $\mathbf{z}^k$). Hence we introduce a truncated variational posterior proposed by [3]: We set a truncation level T_k for each mode k and set $q(v_{T_k}^k = 1) = 1$ so that $q(z_t^k > T_k) = 0$. Therefore, each $q(z_t^k)$ has only T_k supports and we only need to consider T_k posteriors for $\mathbf{v}^k$ and $\boldsymbol{\eta}^k$. The variational distributions $q(z_t^k), q(v_j^k)$ and $q(\boldsymbol{\eta}_j^k)(1 \le t \le m_k, 1 \le j \le T_k)$ are then given by

$$q(z_t^k) = \mathrm{Multi}(z_t^k|\phi_{t1}^k, \ldots, \phi_{tT_k}^k), \quad q(v_j^k) = \mathrm{Beta}(v_j^k|\gamma_{j1}^k, \gamma_{j2}^k), \quad q(\boldsymbol{\eta}_j^k) = \mathcal{N}(\boldsymbol{\eta}_j^k|\boldsymbol{\mu}_j^k, s_j^k\mathbf{I}). \quad (4)$$

The parameters of the distributions are calculated by

$$\phi_{tj}^k \propto \exp\left(\mathbb{E}_q\left[\log v_j^k\right] + \sum_{i=1}^{j-1}\mathbb{E}_q\left[\log(1 - v_i^k)\right] - \frac{1}{2\lambda_k}\mathbb{E}_q\left[\|\boldsymbol{\eta}_j^k\|^2\right] + \frac{1}{\lambda_k}{\mathbf{u}_t^k}^\top\mathbb{E}_q\left[\boldsymbol{\eta}_j^k\right]\right), \quad (5)$$

$$\gamma_{j1}^k = 1 + \sum_{t=1}^{m_k}\phi_{tj}^k\ , \ \ \gamma_{j2}^k = \alpha + \sum_{t=1}^{m_k}\sum_{i=j+1}^{T_k}\phi_{ti}^k\ , \ s_j^k = \frac{1}{1 + {\lambda_k}^{-1}\sum_{t=1}^{m_k}\phi_{tj}^k},\ \boldsymbol{\mu}_j^k = \frac{\sum_{t=1}^{m_k}\phi_{tj}^k\mathbf{u}_t^k}{\lambda_k + \sum_{t=1}^{m_k}\phi_{tj}^k}. \quad (6)$$

The moments required to calculate the parameters are given by $\mathbb{E}_q\left[\log v_j^k\right] = \psi(\gamma_{j1}^k) - \psi(\gamma_{j1}^k + \gamma_{j2}^k)$, $\mathbb{E}_q\left[\log(1 - v_j^k)\right] = \psi(\gamma_{j2}^k) - \psi(\gamma_{j1}^k + \gamma_{j2}^k)$, $\mathbb{E}_q\left[\boldsymbol{\eta}_j^k\right] = \boldsymbol{\mu}_j^k$ and $\mathbb{E}_q\left[\|\boldsymbol{\eta}_j^k\|^2\right] = \|\boldsymbol{\mu}_j^k\|^2 + r_k s_j^k$, where $\psi(x) = \frac{d}{dx}\ln\Gamma(x)$.

4.2 Efficient online VB-EM algorithm

Given the variational distributions, we can estimate the latent factors $\mathcal{U}$ by maximizing the expected log joint probability, $\mathbb{E}_q\left[\log(p(\mathcal{U}, \{\mathbf{z}^k, \mathbf{v}^k, \boldsymbol{\eta}^k\}_{k=1}^K, \{\mathcal{M}_n, \mathcal{Y}_n\}_{n=1}^N))\right]$. The traditional variational EM algorithm can be applied here. However, in each iteration the algorithm requires to pass all the subarrays. It can therefore be slow to apply for large arrays because we need generate a large number of subarrays for analysis, and it is not naturally suited to dynamic arrays with increasing size over time. Therefore, we propose an online VB-EM algorithm for efficient model inference and it turns out our algorithm leads to a better performance for our problem.

Specifically, we randomly shuffle the subarrays and sequentially process each subarray with VB-EM. For each subarray $\mathcal{Y}_n$, we use variational inference to update the approximate posteriors of the local variables (i.e., the latent subtensor $\mathcal{M}_n$ and the cluster assignment variables $\{\mathbf{z}^k_{\mathbb{I}_{nk}}\}_k$ for the sub-factors $\mathcal{U}_n$, where $\mathbb{I}_{nk}$ is the index set of $\mathcal{U}_n$ in k-th mode), and the global variables (i.e., $\{\mathbf{v}^k\}_k$ and $\{\boldsymbol{\eta}^k\}_k$) (E-step); then we update the sub-factors $\mathcal{U}_n$ using stochastic gradient descent (SGD) (M-step).

The naive computation for $q(\mathbf{v}^k)$ and $q(\boldsymbol{\eta}^k)$ by Equation (6) will involve all the factors $\mathbf{U}^{(k)}$ in k-th mode and all the statistics from $q(\mathbf{z}^k)$ (i.e., $\{\phi^k_{tj}\}_{t,j}$), therefore is low efficient. To improve the efficiency, we observe that the calculation for each $q(v^k_j)$ and $q(\boldsymbol{\eta}^k_j)$ relies on three statistics: $\Psi^k_1 = \sum_{t=1}^{m_k} \phi^k_{tj}, \Psi^k_2 = \sum_{t=1}^{m_k} \sum_{i=j+1}^{T_k} \phi^k_{ti}$ and $\Psi^k_3 = \sum_{t=1}^{m_k} \phi^k_{tj} \mathbf{u}^k_n$. These statistics are additive. The processing of $\mathcal{Y}_n$ only changes a subset of statistics $\{\phi^k_{tj} : t \in \mathbb{I}_{nk}\}$; the summation over the remaining statistics will not change and there is no need to compute it again. Therefore, we can cache the three global statistics, and calculate the corresponding local statistics with respect to $\mathcal{Y}_n$: $\Psi^k_{1n} = \sum_{t \in \mathbb{I}_{nk}} \phi^k_{tj}, \Psi^k_{2n} = \sum_{t \in \mathbb{I}_{nk}} \sum_{i=j+1}^{T_k} \phi^k_{ti}$ and $\Psi^k_{3n} = \sum_{t \in \mathbb{I}_{nk}} \phi^k_{tj} \mathbf{u}^k_t$. After computing $\{\phi^k_{tj} : t \in \mathbb{I}_{nk}\}$ for $q(\mathbf{z}^k_{\mathbb{I}_{nk}})$ by (5), we update $\{\Psi^k_1, \Psi^k_2, \Psi^k_3\}$ by simply subtracting the old local statistics and then adding the new ones, i.e., ${\Psi^k_s}^{(\text{new})} = \Psi^k_s - {\Psi^k_{sn}}^{(\text{old})} + {\Psi^k_{sn}}^{(\text{new})} (s = 1, 2, 3)$. Then we can update $q(\mathbf{v}^k)$ and $q(\boldsymbol{\eta}^k)$ accordingly based on the global statistics. This procedure can repeat during the iterations in the E-step to cyclically update local variational posterior $q(\mathbf{z}^k_{\mathbb{I}_{nk}})$ and the global variational posteriors $q(\mathbf{v}^k)$ and $q(\boldsymbol{\eta}^k)$. The calculation only involves statistics which associate with the subarray and hence is much more efficient than the naive computation.

Given the required variational posteriors, we perform SGD to optimize $\mathcal{U}$. First, we derive the expected log likelihood function with respect to $\mathcal{U}$, then rearrange it into to a summation form $f(\mathcal{U}) = \sum_{n=1}^N g_n(\mathcal{U})$, where $g_n(\mathcal{U}) = \frac{1}{N}\mathbb{E}_q\left[\log(p(\mathcal{U}|\{\mathbf{z}^k, \boldsymbol{\eta}^k\}_{k=1}^K))\right] + \mathbb{E}_q\left[\log(p(\mathcal{M}_n|\mathcal{U}))\right]$. Then for each subarray $\mathcal{Y}_n$, we have the following update: $\mathcal{U}_n = \mathcal{U}_n + \rho\frac{\partial g_n}{\partial \mathcal{U}_n}$. The gradient $\frac{\partial g_n}{\partial \mathcal{U}_n}$ has a form similar to that of the expected log joint probability with respect to global latent factors $\mathcal{U}$ in InfTucker. The main difference is from the terms regarding the DPM prior, of which the gradient is trivial. Hence we omit the detailed equation and refer the detail to the paper by [23].

4.3 Strategies to generate subarrays

Here we discuss three ways to generate subarrays used in our training. i) **Uniform sampling.** This is the simplest method: We just uniformly sample a set of indexes of size $\overline{m}_k$, for each mode k, to define a subarray. To make multiple subarrays, we just repeat this process so that each subarray has the same size. ii) **Weighted sampling.** This strategy exploits the information in the data. It samples a set indexes in each mode, based on weights, rather than uniformly. The weights are calculated from the degree of the indexes. The degree of an index is defined as the number of nonzero entries containing that index. The weighted sampling strategy gives more weights to those higher degree indexes so that the sampled subarrays may contain more nonzero elements, as compared with the uniform sampling strategy. iii) **Grid sampling.** It ensures the coverage of every element of the whole array. Specifically, we randomly permute the indexes in each mode, then partition the permuted indexes into multiple segments with the same size, and repeat this process for each mode to generate a grid. In this grid, each (hyper-)cube contains a subarray. We can repeat this whole process to generate more subarrays.

4.4 Predicting array entries by bagging

To predict the values of unknown entries, the global GP model needs to infer the posterior distribution of the whole latent array $q(\mathcal{M})$. For large arrays, this inference is computationally prohibitive.

To overcome this hurdle, we apply a bagging strategy which learns the prediction by simply aggregating predictions on a collection of small subarrays. Because our approach can quickly provide predictions on the small subarrays, it achieves fast final predictions. Note that Bagging [9] has been widely used to improve prediction accuracy for many machine learning methods such as neural networks and decision trees. For our model, we first generate subarrays and find their corresponding latent factors, then use them to learn predictive means of the unknown elements following the global GP prediction algorithm (but on the subsets here), and finally aggregate the predictive means by averaging. As we sample subarrays from the whole array, our prediction can be viewed as nonparametric bootstrap prediction [6].

5 Related work

Multiway data is common in real applications. Many excellent works have been proposed based on the multilinear factorization approaches, such as [19, 1, 10, 12]. However, the interactions and patterns in multiway data can be complex, and it is natural to exploit powerful nonparametric models. InfTucker [23] is a nonparametric model based on GP and can capture the nonlinear relationships. Our work enhances InfTucker by introducing DPM to discover the latent cluster patterns. In theory, our work, as well as InfTucker, can be considered as instances of random function prior models [15] .Recently, nonparametric modeling has also been used to infer the appropriate number of latent factors for PARAFAC decomposition [16], which is very interesting, but has a different goal with our work.

The global GP model is impractical for large multiway arrays due to a prohibitive high computation cost. Hence we resort to a relaxed local GP assumption. Many works have been proposed for training of local GPs. For example, [17] proposed an infinite GP mixture model; [13] used partitions of GP to analyze spatial data; [7] proposed treed GP; and [5] proposed multiresolution GP coupling nestedly partitioned GPs for time-series analysis. However, these great works focus on GP models with known input locations. Our work uses local GPs to boost the latent GP model on large multiway arrays, where the input is unkown (and need to be estimated) and the model estimation could be challenging.

Our online VB-EM algorithm is also related to online learning of DP or Hierarchical DP [22, 11]. While these excellent works focus on DP, our algorithm is designed for the inference of a combination of latent DP and GP model.

6 Experiment

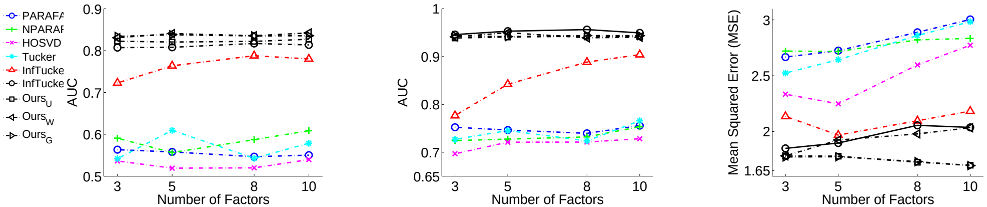
6.1 Missing value prediction

Datasets. First, two binary datasets, *Digg* and *Enron*, and one continuous dataset, *Alog* were used for examination. *Digg* is extracted from a social news website `digg.com` and describes a three-way interaction (news, keyword, topic). It contains $581 \times 124 \times 48$ elements, of which 0.024% are non-zero. *Enron*, extracted from the Enron email dataset (`www.cs.cmu.edu/~./enron/`), depicts a three-way relationship (sender, receiver, time). *Enron* is of size $203 \times 203 \times 200$ where 0.01% elements are non-zero. *Alog* is extracted from an access log from a file management system. It records three-way interactions (user, action, resource) and contains $200 \times 100 \times 200$ elements, of which 0.33% are nonzeros.

Competing methods. We compared our approach with the following tensor decomposition methods: PARAFAC, nonnegative PARAFAC (N-PARAFAC) [20], high order SVD (HOSVD) [14], Tucker and InfTucker. We also implemented the InfTucker model with a DPM prior, denoted by InfTucker-DPM, which extends InfTucker by assigning DPM priors over the latent factors.

Parameter settings. The number of latent factors was chosen from the set $\{3, 5, 8, 10\}$. All the methods were evaluated by a 5-fold cross validation: the nonzero entries were randomly split into 5 folds and 4 folds were used for training; the remaining non-zero entries and 0.1% zero entries were used for testing so that the evaluation will not be dominated by the large portion of zero entries. RBF kernels were consistently employed in InfTucker, InfTucker-DPM and our approach, with parameters chosen by another cross-validation and so did the hyperparameter of the Laplace prior of InfTucker. In the proposed approach, the size of subarray is set to $40 \times 40 \times 40$ for all the three datasets; three sampling strategies described in Section 4.3 were used and 500 subarrays were generated by each strategy; the learning rate ρ was tuned from the range $\{10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}\}$. For both InfTucker-DPM and our approach, the variational truncation level in each mode is set to one-tenth of the dimension of each mode. For bagging prediction, we randomly sampled 10 subarrays, each with a size of $40 \times 40 \times 40$. The area-under-curve (AUC) is used to evaluate performance on *Digg* and *Enron*, and the mean squared error (MSE) on *Alog*. We then report the average results from the 5-fold cross validation.

Results. As shown in Figure 2, both InfTucker-DPM and our model achieve higher prediction accuracy than InfTucker and the other alternatives. A t-test shows that InfTucker-DPM and our approach significantly outperform InfTucker ($p < 0.05$) in almost all the cases. The results demonstrate that DPM priors can benefit the prediction task. Moreover, our local GP based model and the online VB-EM algorithm can achieve comparable or sometimes even better results than the model based on global GP, i.e., InfTucker-DPM.



(a) *digg* (b) *enron* (c) *alog*

Figure 2: The prediction results on small datasets. The results are averaged over 5 runs. OURS$_{\text{U}}$, OURS$_{\text{W}}$ and OURS$_{\text{G}}$ refer to our method based on the uniform, weighted, and grid sampling strategies, respectively.

6.2 Latent cluster discovery

To examine the ability of discovering latent clusters, we simulated a synthetic tensor of size $100 \times 100 \times 100$. First, a set of latent factors $\{\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \mathbf{U}^{(3)}\}$ were sampled from a Gaussian mixture model (GMM) and then the tensor elements were sampled based on the latent factors. We set the number of the mixture components in GMM to 3, with centers located at $\{(2,2),(2,-2),(-2,-2)\}$ and the covariance matrix for each component to $0.5\mathbf{I}$. The selecting probability of each component is $\frac{1}{3}$. Given $\mathbf{u}_i^1, \mathbf{u}_j^2, \mathbf{u}_k^3$, a tensor element y_{ijk} was generated by a nonlinear function:

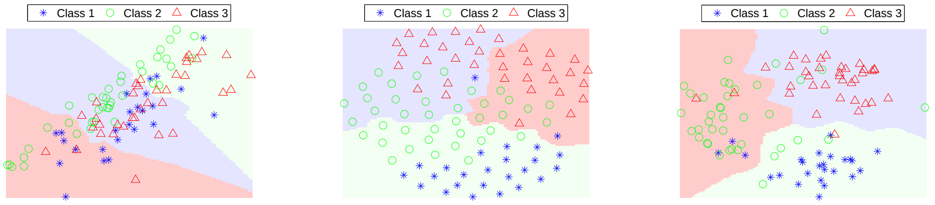
$$x_{ijk} = \|\mathbf{u}_i^1 - \mathbf{u}_j^2\|^2 + \|\mathbf{u}_i^1 - \mathbf{u}_k^3\|^2 + \|\mathbf{u}_j^2 - \mathbf{u}_k^3\|^2, y_{ijk} = \log(x_{ijk}^{\frac{3}{2}} + x_{ijk} + 1) - \cos(\sqrt{x_{ijk}}) + \epsilon_{ijk}$$

where ϵ_{ijk} is a random noise sampled from a Gaussian distribution $\mathcal{N}(0, 10)$. Given the data, we ran our approach to recover the cluster structure of the latent factors. For comparison, we also ran PARAFAC and InfTucker, and then used k-means to find clusters. We set $k = 3$, the exact number of clusters. In our approach, the size of subarray was set to $10 \times 10 \times 10$; 1000 subarrays were sampled with the uniform sampling strategy; and the truncation level was set to 10 for each mode. The parameters for InfTucker and our approach were obtained by cross-validation. Figure 3 displays the estimated clusters of all the methods in the first mode. The cluster regions are filled with different background colors, where the marker of each point (i.e., latent factor) exhibits its ground-truth class (i.e., the GMM component from which it is originally sampled). In Figure 3a, the points of different classes are largely mixed, implying that PARAFAC failed to capture the nonlinear relationships in data. In Figure 3b, points of the same class stay continuously, indicating that InfTucker successfully captures the nonlinear relationships. However, the points are distributed almost uniformly and the cluster structures is difficult to reveal. In Figure 3c, the points are well separated into three clusters. Although with a few missing assignments, the results demonstrates that our approach not only captured the nonlinear relationships but also identified the latent cluster structures.

For a quantitative evaluation, we calculated the purity of the estimated clusters [24]. The purity is calculated by $\mathrm{Purity}(\Omega, \mathrm{C}) = \frac{1}{\mathrm{N}} \sum_{\mathrm{k}} \max_{\mathrm{j}} |\omega_{\mathrm{k}} \cap \mathrm{c}_{\mathrm{j}}|$ where $\Omega = \{\omega_1, \ldots, \omega_K\}$ is the cluster assignment determined by the algorithm and $C = \{c_1, \ldots, c_J\}$ is the ground-truth classes. Higher purity means better cluster quality; a perfect cluster assignment has a pu-

Table 1: The purity of the estimated clusters.

Method	Mode 1	Mode 2	Mode 3
PARAFAC	0.42	0.44	0.42
InfTucker	0.62	0.69	0.75
Our model	**0.84**	**0.84**	**0.88**



(a) PARAFAC

(b) InfTucker

(c) Our model

Figure 3: The estimated latent clusters.

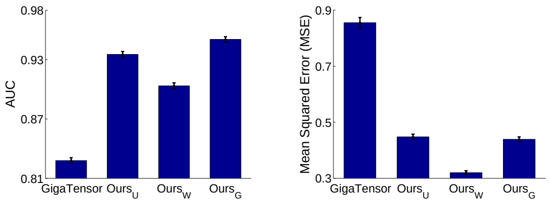
rity of one. The purity of the estimated clusters based on the three methods are listed in Table 1. As we can see, our model obtains the highest purity, implying the best recovered cluster structure.

6.3 Large multiway array analysis

Two large real datasets were employed for analysis: (1) DBLP, of size 10K × 200 × 10K, depicts a three-way bibliography relationship (author, conference, keyword). We parsed the original DBLP xml file (`http://dblp.uni-trier.de/xml/`) and selected the 10K most prolific authors, the 200 most popular conferences and 10K most common keywords to construct a binary-valued tensor. (2) ACC , of size 3K × 150 × 30K, describes the (user, action, resource) interaction and was extracted from access logs of a source code version control system in a large company. We selected the 3K most active users, the 30K most popular resources for analysis. The count of each interaction is highly varied (i.e., from just once to millions). Hence we took logarithm and obtained a real-valued tensor. In total, DBLP contains 20 billions of elements and ACC has 13.5 billions of entries. To the best of our knowledge, there is no nonparametric models which can deal with tensor data at this scale.

Our approach was compared with the state-of-the-art large scale tensor decomposition method, GigaTensor. GigaTensor is developed with the Map-Reduce framework. We used the original GigaTensor software package and its default settings. We ran GigaTensor on a Hadoop cluster with 16 computers and our online algorithm on a single computer.

The number of latent factors were set to 3 for both datasets. The DBLP and ACC datasets contain 0.001% and 0.009% nonzero elements, respectively. We randomly chose 80% of nonzero entries for training, and then sampled 50 test datasets from the remaining entries. Each test dataset comprises 200 nonzero elements and $1,800$ zero elements. For prediction of our method, we randomly sampled 10 subarrays of size $100\times100\times100$ for bagging. We used RBF kernel; to tune the kernel parameters, we drew a subarray of size $2000 \times 150 \times 2000$ for each training array and then performed cross-validation to obtain the best parameters. The size of subtensor used by our online algorithm were chosen from $\{100 \times 100 \times 100, 125 \times 125 \times 125, 150 \times 150 \times 150\}$. To identify the number of subtensors, during the cross validation, we chose the number from $\{1, 2, 3\} \times P$ where P is the number of subtensors which can cover the same quantity of entries in the whole array. The learning rate were chosen from $\{10^{-7}, 10^{-8}, 10^{-9}\}$.



(a) DBLP

(b) ACC

Figure 4: Prediction results on large multiway data. The results are averaged over 50 test datasets.

Figure 4 shows the AUC and MSE for DBLP and ACC datasets. It turns out that regardless of the subarray sampling strategy, our model outperforms GigaTensor significantly. It improves the AUC of GigaTensor on DBLP by 12%, and the MSE on ACC by 53% on average. Because GigaTensor is a distributed algorithm for PARAFAC decomposition, the results actually show that our model consistently outperforms PARAFAC in large arrays.

As to the speed, GigaTensor is several times faster than our model. The reason is that GigaTensor can exploit multiple computational units in a cluster and perform parallel decomposition while our model was carried out on a single computer. However, our method makes nonparametric model—a more powerful tool— feasible for large multiway data analysis and thus practical for real applications.

7 Conclusion

In this paper, we present a scalable nonparametric Bayesian model, and an efficient online VB-EM learning algorithm for large multiway data analysis. In the future work, we plan to develop a distributed learning algorithm, like GigaTensor on MapReduce, to further scale up our model to even larger data, say, trillions of elements.

References

[1] Evrim Acar, Daniel M Dunlavy, Tamara G Kolda, and Morten M**o**rup. Scalable tensor factorizations for incomplete data. *Chemometrics and Intelligent Laboratory Systems*, 106(1):41–56, 2011.

[2] Charles E Antoniak. Mixtures of dirichlet processes with applications to bayesian nonparametric problems. *The annals of statistics*, pages 1152–1174, 1974.

[3] David M Blei, Michael I Jordan, et al. Variational inference for dirichlet process mixtures. *Bayesian analysis*, 1(1):121–143, 2006.

[4] Wei Chu and Zoubin Ghahramani. Probabilistic models for incomplete multi-dimensional arrays. *AISTATS*, 2009.

[5] David B Dunson and Emily B Fox. Multiresolution gaussian processes. In *Advances in Neural Information Processing Systems*, pages 737–745, 2012.

[6] Tadayoshi Fushiki, Fumiyasu Komaki, and Kazuyuki Aihara. Nonparametric bootstrap prediction. *Bernoulli*, 11(2):293–307, 2005.

[7] Robert B Gramacy and Herbert KH Lee. Bayesian treed gaussian process models with an application to computer modeling. *Journal of the American Statistical Association*, 103(483), 2008.

[8] R. A. Harshman. Foundations of the PARAFAC procedure: Model and conditions for an"explanatory"multi-mode factor analysis. *UCLA Working Papers in Phonetics*, 16:1–84, 1970.

[9] Trevor. Hastie, Robert. Tibshirani, and J Jerome H Friedman. *The elements of statistical learning*, volume 1. Springer New York, 2001.

[10] P.D. Hoff. Hierarchical multilinear models for multiway data. *Computational Statistics & Data Analysis*, 55:530–543, 2011.

[11] Michael C Hughes and Erik Sudderth. Memoized online variational inference for dirichlet process mixture models. In *Advances in Neural Information Processing Systems*, pages 1133–1141, 2013.

[12] U Kang, Evangelos Papalexakis, Abhay Harpale, and Christos Faloutsos. Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 316–324. ACM, 2012.

[13] Hyoung-Moon Kim, Bani K Mallick, and CC Holmes. Analyzing nonstationary spatial data using piecewise gaussian processes. *Journal of the American Statistical Association*, 100(470):653–668, 2005.

[14] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl*, 21:1253–1278, 2000.

[15] James Robert Lloyd, Peter Orbanz, Zoubin Ghahramani, and Daniel M. Roy. Random function priors for exchangeable arrays with applications to graphs and relational data. In *NIPS*, pages 1007–1015, 2012.

[16] Piyush Rai, Yingjian Wang, Shengbo Guo, Gary Chen, David Dunson, and Lawrence Carin. Scalable Bayesian low-rank decomposition of incomplete multiway tensors. In *Proceedings of the 31th International Conference on Machine Learning (ICML)*, 2014.

[17] Carl Edward Rasmussen and Zoubin Ghahramani. Infinite mixtures of gaussian process experts. *Advances in neural information processing systems*, 2:881–888, 2002.

[18] Jayaram Sethuraman. A constructive definition of dirichlet priors. Technical report, DTIC Document, 1991.

[19] Amnon Shashua and Tamir Hazan. Non-negative tensor factorization with applications to statistics and computer vision. In *Proceedings of the 22nd ICML*, 2005.

[20] Amnon Shashua and Tamir Hazan. Non-negative tensor factorization with applications to statistics and computer vision. In *Proceedings of the 22th International Conference on Machine Learning (ICML)*, pages 792–799, 2005.

[21] Ledyard Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31:279–311, 1966.

[22] Chong Wang, John W Paisley, and David M Blei. Online variational inference for the hierarchical dirichlet process. In *International Conference on Artificial Intelligence and Statistics*, pages 752–760, 2011.

[23] Zenglin Xu, Feng Yan, and Yuan Qi. Infinite Tucker decomposition: Nonparametric Bayesian models for multiway data analysis. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, 2012.

[24] Ying Zhao and George Karypis. Criterion functions for document clustering: Experiments and analysis. Technical report, 2002.