

Residual Bayesian Co-clustering for Matrix Approximation

Hanhuai Shan*

Arindam Banerjee*

Abstract

In recent years, matrix approximation for missing value prediction has emerged as an important problem in a variety of domains such as recommendation systems, e-commerce and online advertisement. While matrix factorization based algorithms typically have good approximation accuracy, such algorithms can be slow especially for large matrices. Further, such algorithms cannot naturally make prediction on new rows or columns. In this paper, we propose residual Bayesian co-clustering (RBC), which learns a generative model corresponding to the matrix from the non-missing entries, and uses the model to predict the missing entries. RBC is an extension of Bayesian co-clustering by taking row and column bias into consideration. The model allows mixed memberships of rows and columns to multiple clusters, and can naturally handle the prediction on new rows and columns which are not used in the training process, given only a few non-missing entries in them. We propose two variational inference based algorithms for learning the model and predicting missing entries. One of the proposed algorithms leads to a parallel RBC which can achieve significant speed-ups. The efficacy of RBC is demonstrated by extensive experimental comparisons with state-of-the-art algorithms on real world datasets.

1 Introduction

The rapid evolution of data mining techniques has led to an increasing realization that real data is usually multi-relational, with multiple entities connecting with each other through various relationships. An important special case of multi-relational data is dyadic data, which contains two entities interacting with each other. For example, users and movies connected by ratings in movie recommendation systems, customers and products connected by purchasing records in e-commerce, webpages and advertisements connected by click-through rate in online advertisement systems [1], etc.. Such dyadic data could be represented as a matrix with rows and columns representing each entity respectively. In most cases, the data matrix is very sparse, in the sense that most of its entries have missing values, because a user can only rate a few movies, a cus-

tommer only buys a few products, and a webpage can only contain a few advertisements, although there are thousands or millions of movies, products and advertisements in total. An important task given such matrices is to do missing value prediction for better recommendation or decision making. The prediction of the rating from a user to a movie helps us to decide whether to recommend the movie to the user. The prediction on the number of products a customer will buy helps us to decide whether to offer any discount to attract the customer. Similarly, the prediction of the click-through rate for an advertisement on a webpage helps us to decide whether to publish the advertisement on that webpage. Such scenarios can be boiled down to a matrix approximation problem: Given part of the entries in a matrix, the task is to approximate the original matrix, especially to predict the missing entries.

Matrix factorization based algorithms, such as singular value decomposition (SVD) [11] and non-negative matrix factorization (NNMF) [12] have been shown to provide accurate predictions. However, these techniques suffer from three limitations: First, although recent years have seen a considerable amount of work in speeding up these algorithms [9, 18], matrix factorization based methods are typically not efficient computationally. Second, such methods cannot effectively do predictions for new objects. However, in a video sharing site such as YouTube, new videos and even new users come into the system at a rapid rate. It is unrealistic to redo training or factorization every time a new video or a new user is added to the system. Third, instead of only predicting the desired entries, such methods need to reconstruct the whole matrix with possibly millions of rows and columns, even though we only need the prediction on a few entries.

In recent years, co-clustering algorithms [7, 1, 2] have been used for matrix approximation. The main idea is to use certain summary statistics of co-clusters as a low parameter representation for the original matrix, and to approximate the original matrix based on these summary statistics as well as the cluster membership for the rows and columns. Most of the co-clustering algorithms are partitional, which allow each row (column) to only belong to one row (column) cluster. However, objects in the real world can usually belong to

*Department of CSE, University of Minnesota, Twin Cities.
{shan,banerjee}@cs.umn.edu

multiple clusters with varying degrees. For example, a movie could fall into the categories of romantic movie and war movie. Therefore, the assumption made by partitional co-clustering algorithms may restrict their ability to generate accurate matrix approximations.

In this paper, we propose residual Bayesian co-clustering (RBC) for matrix approximation. We assume that the data matrix is generated from a generative process using a set of parameters. The learning task is to estimate these parameters after observing non-missing entries in the matrix, then the prediction can be done on the missing entries. As an extension of Bayesian co-clustering (BCC) [17], RBC assumes that each row has a mixed membership to row clusters, with a Dirichlet prior on top, similarly for the columns. For each row-column cluster, i.e., each co-cluster, a univariate parametric component distribution is designated to generate data. Instead of learning the co-cluster structure from the original data matrix as in BCC, RBC infers the co-clustering from the residual matrix obtained by suitably subtracting row and column bias from the original matrix. That is why we refer to the model as “residual Bayesian co-clustering”. RBC can naturally handle sparse matrices. It can also do predictions for new rows and columns not used in the training process, given only a few entries in them. For inference and parameter estimation, we propose two variational EM-style algorithms by introducing two variational distributions, one of which partly decouples the dependency between the mixed memberships of rows and columns, leading to a parallel RBC algorithm. The efficacy of RBC is demonstrated by the experiments on real datasets. In particular, we show that RBC consistently outperforms several co-clustering algorithms as well as NNMF on missing value prediction. Further, RBC is better than or competitive with SVD on all datasets considered, and parallel RBC scales substantially better than SVD as the size of the matrix grows.

The rest of paper is organized as follows: In Section 2, we present an overview of generative mixture models. In Section 3, we propose residual Bayesian co-clustering model. Two variational approaches for learning RBC are presented in Section 4 and 5, and Section 5 also introduces parallel RBC. We present the experimental results in Section 6 and conclude in Section 7.

2 Generative Mixture Models

In this section, we give a brief overview of generative mixture models (GMM) as a background for RBC.

2.1 Finite Mixture Models Finite mixture models (FMM) [6] are one of the most widely studied models for

discovering the latent cluster structure from observed data. It assumes that there are k clusters, each has a distribution to generate the data, and each data point is generated from one of them. The density function of a data point \mathbf{x} in FMM is given by:

$$p(\mathbf{x}|\pi, \Theta) = \sum_{z=1}^k p(z|\pi)p(\mathbf{x}|\theta_z),$$

where π is the prior over k components, and $\Theta = \{\theta_z, [z]_1^k\}$ ($[z]_1^k \equiv z = 1, \dots, k$) are parameters of the distributions $\{p(\mathbf{x}|\theta_z), [z]_1^k\}$ for k clusters respectively. $p(\mathbf{x}|\theta_z)$ is an exponential family distribution [4] with a form of $p_\psi(\mathbf{x}|\theta) = \exp(\langle \mathbf{x}, \theta \rangle - \psi(\theta))p_0(\mathbf{x})$ [3], where θ is the natural parameter, $\psi(\cdot)$ is the cumulant function, and $p_0(\mathbf{x})$ is a non-negative base measure. ψ determines a particular family, such as Gaussian, Poisson, etc., and θ determines a particular distribution in that family. An EM-style algorithm [15, 13] could be used to estimate the parameters and infer the cluster assignment.

2.2 Latent Dirichlet Allocation One key assumption of FMM is that the each data point is only generated from one cluster. Latent Dirichlet allocation (LDA) [5] relaxes this assumption by allowing each data point to have a mixed membership over k clusters. In particular, it assumes there is a separate mixed membership π for each data point, and π is sampled from a Dirichlet distribution $\text{Dir}(\alpha)$. For a sequence of tokens $\mathbf{x} = x_1 \cdots x_n$, each x_l has a separate cluster z sampled from π . LDA with k clusters has a density of the form

$$p(\mathbf{x}|\alpha, \Theta) = \int_{\pi} \text{Dir}(\pi|\alpha) \left(\prod_{l=1}^n \sum_{z_l=1}^k p(z_l|\pi)p(x_l|\theta_{z_l}) \right) d\pi.$$

Getting a closed form expression for the marginal density $p(\mathbf{x}|\alpha, \Theta)$ is intractable. Variational inference [5] and Gibbs sampling [10] are two most popular approaches for inference and parameter estimation.

2.3 Bayesian Co-clustering Bayesian co-clustering [17] extends LDA to do “two-way” clustering on a matrix, by keeping a mixed membership π_{1u} and π_{2v} for each row u and each column v and introducing a separate Dirichlet prior α_1 and α_2 for rows and columns respectively. Each entry x_{uv} in the matrix has a row cluster z_1 and column cluster z_2 sampled from π_{1u} and π_{2v} respectively, and each co-cluster (z_1, z_2) has a certain exponential family distribution $p(x|z_1, z_2, \Theta)$ to generate the entry. The density function for each entry in the matrix is given by:

$$p(x|\alpha_1, \alpha_2, \Theta) = \int_{\pi_1} \int_{\pi_2} p(\pi_1|\alpha_1)p(\pi_2|\alpha_2) \sum_{z_1} \sum_{z_2} p(z_1|\pi_1)p(z_2|\pi_2)p(x|z_1, z_2, \Theta)d\pi_1d\pi_2.$$

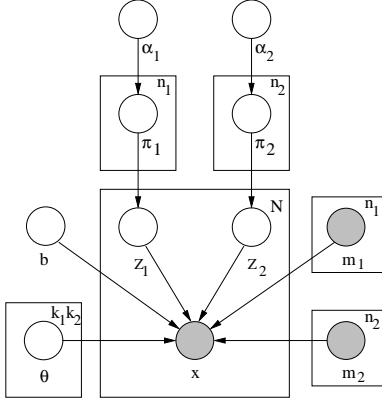


Figure 1: The graphical model for RBC.

The inference and parameter estimation could be done using a variational EM algorithm.

3 Residual Bayesian Co-clustering

In this section, we propose residual Bayesian co-clustering for matrix approximation. It first learns from the non-missing entries to capture the generative process which generates the matrix, and then uses the learnt process to predict the missing entries. The generative model of Bayesian co-clustering [17] uses a same distribution to generate all the entries in each co-cluster. However, in real cases, the entries of a matrix in a same row or column are usually biased by rows or columns. For example, in a movie rating system, some users are generous in a sense that they tend to give high ratings, then 3 in 1-5 rating scheme indicates a poor movie. Meanwhile, for users who are usually critical, 3 might be a good rating score. In terms of movies, those with famous movie stars tend to get higher ratings while others do not. In such cases, the ratings are biased for each user and each movie. Therefore we propose RBC, which is built on BCC, to take such bias into consideration.

Given an $n_1 \times n_2$ data matrix X , for the purpose of co-clustering, we assume k_1 row clusters $\{z_1 = i, [i]_1^{k_1}\}$ and k_2 column clusters $\{z_2 = j, [j]_1^{k_2}\}$, and two Dirichlet distributions $\text{Dir}(\alpha_1)$ and $\text{Dir}(\alpha_2)$ from which the mixed membership $\{\pi_{1u}, [u]_1^{n_1}\}$ and $\{\pi_{2v}, [v]_1^{n_2}\}$ for rows and columns are generated respectively. Row clusters for entries in row u and column clusters for entries in column v are sampled from discrete distributions $\text{Disc}(\pi_{1u})$ and $\text{Disc}(\pi_{2v})$ respectively. A row cluster i and a column cluster j together decide a co-cluster (i, j) , which has a Gaussian distribution $N(\mu_{ij}, \sigma_{ij}^2)$, where μ_{ij} and σ_{ij}^2 are the mean and variance for co-cluster (i, j) . Note that in principle it is possible to generalize the model by using various types of distributions, but we only discuss the Gaussian case in this paper. Assuming the means of each row u is m_{1u} and the means of each column v is m_{2v} , the generative model for RBC is given as follows:

1. For each row $u, [u]_1^{n_1}$, choose $\pi_{1u} \sim \text{Dir}(\alpha_1)$.
2. For each column $v, [v]_1^{n_2}$, choose $\pi_{2v} \sim \text{Dir}(\alpha_2)$.
3. To generate a non-missing entry in row u and column v ,
 - (a) choose $z_1 = i \sim \text{Disc}(\pi_{1u})$, $z_2 = j \sim \text{Disc}(\pi_{2v})$,
 - (b) choose $x_{uv} \sim N(x | \mu_{z_1 z_2} + b m_{1u} + b m_{2v}, \sigma_{z_1 z_2}^2)$.

The graphical model for RBC is shown in Figure 1, where N is the total number of the non-missing entries. The model only generates the non-missing entries, so it is able to handle matrices with missing values naturally. The generative process defines each entry x_{uv} to be generated from a Gaussian distribution $N(x | \mu_{z_1 z_2} + b m_{1u} + b m_{2v}, \sigma_{z_1 z_2}^2)$, which means that the co-cluster mean $\mu_{z_1 z_2}$ in RBC is actually defined on a matrix after subtracting the effects of row and column means, rather than on the original matrix. b is a parameter to be estimated determining how strong the row and column effects are. In RBC, we assume equal row and column effects by using a same coefficient b for both m_{1u} and m_{2v} , but in principle, we can differentiate these two effects by introducing different coefficients.

For each entry x_{uv} in the data matrix X , given row cluster $z_{1uv} \in \{1 \dots k_1\}$ and column cluster $z_{2uv} \in \{1 \dots k_2\}$, the probability of x_{uv} is defined as:

$$p(x_{uv} | \theta_{z_{1uv} z_{2uv}}, b) = p(x_{uv} | \mu_{z_{1uv} z_{2uv}} + b m_{1u} + b m_{2v}, \sigma_{z_{1uv} z_{2uv}}^2),$$

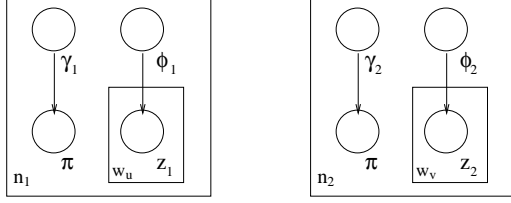
where $\theta_{z_{1uv} z_{2uv}} = \{\mu_{z_{1uv} z_{2uv}}, \sigma_{z_{1uv} z_{2uv}}^2\}$. The generation of x_{uv} not only depends on b and $\theta_{z_{1uv} z_{2uv}}$, but also depends on m_{1u} and m_{2v} , so strictly, the probability should be denoted as $p(x_{uv} | \theta_{z_{1uv} z_{2uv}}, b, m_{1u}, m_{2v})$, but we omit m_{1u} and m_{2v} hereafter for brevity. Accordingly, the marginal probability of x_{uv} is given by:

$$p(x_{uv} | \alpha_1, \alpha_2, b, \Theta) = \int_{\pi_{1u}} \int_{\pi_{2v}} p(\pi_{1u} | \alpha_1) p(\pi_{2v} | \alpha_2) \sum_{z_{1uv}} \sum_{z_{2uv}} p(z_{1uv} | \pi_{1u}) p(z_{2uv} | \pi_{2v}) p(x_{uv} | \theta_{z_{1uv} z_{2uv}}, b) d\pi_{1u} d\pi_{2v},$$

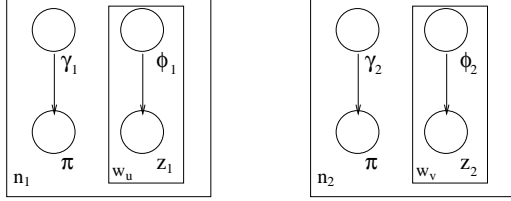
where $\Theta = \{\theta_{ij}, [i]_1^{k_1}, [j]_1^{k_2}\} = \{\mu_{ij}, \sigma_{ij}^2, [i]_1^{k_1}, [j]_1^{k_2}\}$. The probability of the entire matrix X is not the product of all such marginal probabilities, because π_1 for each row and π_2 for each column are sampled only once for all entries in this row or column, so the entries in a same row or a same column are not statistically independent.

The overall joint distribution over all observable and latent variables is given by

$$p(X, \pi_{1u}, \pi_{2v}, z_{1uv}, z_{2uv}, [u]_1^{n_1}, [v]_1^{n_2} | \alpha_1, \alpha_2, b, \Theta) = \prod_u p(\pi_{1u} | \alpha_1) \prod_v p(\pi_{2v} | \alpha_2) \prod_{u,v} \left(p(z_{1uv} | \pi_{1u}) p(z_{2uv} | \pi_{2v}) p(x_{uv} | \theta_{z_{1uv} z_{2uv}}, b) \right)^{\delta_{uv}},$$



(a) row (b) column
Figure 2: Variational distribution q .



(a) row (b) column
Figure 3: Variational distribution q' .

where δ_{uv} is an indicator function which takes value 0 when x_{uv} is missing and 1 otherwise, so only the non-missing entries are considered.

Marginalizing over $\{z_{1uv}, z_{2uv}, [u]_1^{n_1}, [v]_1^{n_2}\}$ and $\{\pi_{1uv}, \pi_{2uv}, [u]_1^{n_1}, [v]_1^{n_2}\}$, the probability of observing the entire matrix X is:

(3.1)

$$p(X|\alpha_1, \alpha_2, b, \Theta) = \int_{u=1, \dots, n_1}^{\pi_{1u}} \int_{v=1, \dots, n_2}^{\pi_{2v}} \left(\prod_u p(\pi_{1u}|\alpha_1) \right) \left(\prod_v p(\pi_{2v}|\alpha_2) \right) \prod_{u,v} \left(\sum_{z_{1uv}} \sum_{z_{2uv}} p(z_{1uv}|\pi_{1u}) p(z_{2uv}|\pi_{2v}) p(x_{uv}|\theta_{z_{1uv}, z_{2uv}}, b) \right)^{\delta_{uv}} d\pi_{11} \cdots d\pi_{1n_1} d\pi_{21} \cdots d\pi_{2n_2}.$$

4 Inference and Learning

Given the data matrix X , the learning task is to estimate model parameters $(\alpha_1^*, \alpha_2^*, b^*, \Theta^*)$ such that the likelihood of observing the matrix X is maximized. While the expectation maximization (EM) family of algorithms [6] are the most general methods to apply, computation of $\log p(X|\alpha_1, \alpha_2, b, \Theta)$ is intractable, implying that a direct application of EM is not feasible. Following [17], we propose a variational inference method, which alternates between obtaining a tractable lower bound of true log-likelihood and choosing the model parameters to maximize the lower bound. The tractable lower bound is selected from an entire family of parameterized lower bounds with a set of free variational parameters, by optimizing the lower bound with respect to the variational parameters.

4.1 Variational Approximation To get a tractable lower bound for $\log p(X|\alpha_1, \alpha_2, b, \Theta)$, we introduce $q(\mathbf{z}_1, \mathbf{z}_2, \boldsymbol{\pi}_1, \boldsymbol{\pi}_2|\gamma_1, \gamma_2, \boldsymbol{\phi}_1, \boldsymbol{\phi}_2)$ as an approximation of the latent variable distribution $p(\mathbf{z}_1, \mathbf{z}_2, \boldsymbol{\pi}_1, \boldsymbol{\pi}_2|\alpha_1, \alpha_2, b, \Theta)$:

$$q(\mathbf{z}_1, \mathbf{z}_2, \boldsymbol{\pi}_1, \boldsymbol{\pi}_2|\gamma_1, \gamma_2, \boldsymbol{\phi}_1, \boldsymbol{\phi}_2) = \left(\prod_{u=1}^{n_1} q(\pi_{1u}|\gamma_{1u}) \right) \left(\prod_{v=1}^{n_2} q(\pi_{2v}|\gamma_{2v}) \right) \left(\prod_{u=1}^{n_1} \prod_{v=1}^{n_2} q(z_{1uv}|\phi_{1u}) q(z_{2uv}|\phi_{2v}) \right),$$

where $\gamma_1 = \{\gamma_{1u}, [u]_1^{n_1}\}$ and $\gamma_2 = \{\gamma_{2v}, [v]_1^{n_2}\}$ are parameters of variational Dirichlet distributions for rows and columns, and $\boldsymbol{\phi}_1 = \{\phi_{1u}, [u]_1^{n_1}\}$ and $\boldsymbol{\phi}_2 = \{\phi_{2v}, [v]_1^{n_2}\}$ are parameters for variational discrete distributions for rows and columns. Figure 2 shows the approximating distribution q as a graphical model, where all entries in a same row u have a same distribution $\text{Disc}(\phi_{1u})$ to generate the row cluster z_1 , similarly for the columns, and w_u and w_v are the number of non-missing entries in row u and column v .

After introducing the variational distribution q , a direct application of Jensen's inequality [14] gives a lower bound to $\log p(X|\alpha_1, \alpha_2, b, \Theta)$:

$$(4.2) \quad \log p(X|\alpha_1, \alpha_2, b, \Theta) \geq E_q[\log p(X, \mathbf{z}_1, \mathbf{z}_2, \boldsymbol{\pi}_1, \boldsymbol{\pi}_2|\alpha_1, \alpha_2, b, \Theta)] - E_q[\log q(\mathbf{z}_1, \mathbf{z}_2, \boldsymbol{\pi}_1, \boldsymbol{\pi}_2|\gamma_1, \gamma_2, \boldsymbol{\phi}_1, \boldsymbol{\phi}_2)],$$

which is denoted as $L(\gamma_1, \gamma_2, \boldsymbol{\phi}_1, \boldsymbol{\phi}_2; \alpha_1, \alpha_2, b, \Theta)$ and could be expanded as

$$L(\gamma_1, \gamma_2, \boldsymbol{\phi}_1, \boldsymbol{\phi}_2; \alpha_1, \alpha_2, b, \Theta) = E_q[\log p(\boldsymbol{\pi}_1|\alpha_1)] + E_q[\log p(\boldsymbol{\pi}_2|\alpha_2)] + E_q[\log p(\mathbf{z}_1|\boldsymbol{\pi}_1)] + E_q[\log p(\mathbf{z}_2|\boldsymbol{\pi}_2)] + E_q[p(X|\mathbf{z}_1, \mathbf{z}_2, b, \Theta)] - E_q[\log q(\boldsymbol{\pi}_1|\gamma_1)] - E_q[\log q(\boldsymbol{\pi}_2|\gamma_2)] - E_q[\log q(\mathbf{z}_1|\boldsymbol{\phi}_1)] - E_q[\log q(\mathbf{z}_2|\boldsymbol{\phi}_2)].$$

The expression of each type of term in L is listed in Table 1, where ϕ_{1ui} is the i^{th} component of $\boldsymbol{\phi}_1$ for row u , ϕ_{2vj} is the j^{th} component of $\boldsymbol{\phi}_2$ for column v , and similarly for γ_{1ui} and γ_{2vj} , and $\Psi(\cdot)$ is the digamma function [5]. The expressions for the rest of the terms are in a similar form.

4.1.1 Inference In the inference step, given $(\alpha_1, \alpha_2, b, \Theta)$, we obtain the tightest lower bound to $\log p(X|\alpha_1, \alpha_2, b, \Theta)$ by maximizing the lower bound function $L(\gamma_1, \gamma_2, \boldsymbol{\phi}_1, \boldsymbol{\phi}_2; \alpha_1, \alpha_2, b, \Theta)$ with respect to variational parameters $(\gamma_1, \gamma_2, \boldsymbol{\phi}_1, \boldsymbol{\phi}_2)$, which gives:

Term	Expression
$E_q[\log p(\boldsymbol{\pi}_1 \alpha_1)]$	$\sum_{u=1}^{n_1} \sum_{i=1}^{k_1} (\alpha_{1i} - 1) (\Psi(\gamma_{1ui}) - \Psi(\sum_{l=1}^{k_1} \gamma_{1ul})) + n_1 \log \Gamma(\sum_{i=1}^{k_1} \alpha_{1i}) - n_1 \sum_{i=1}^{k_1} \log \Gamma(\alpha_{1i})$
$E_q[\log p(\mathbf{z}_1 \boldsymbol{\pi}_1)]$	$\sum_{u=1}^{n_1} \sum_{i=1}^{k_1} w_u \phi_{1ui} (\Psi(\gamma_{1ui}) - \Psi(\sum_{l=1}^{k_1} \gamma_{1ul}))$
$E_q[\log q(\boldsymbol{\pi}_1 \gamma_1)]$	$\sum_{u=1}^{n_1} \sum_{i=1}^{k_1} (\gamma_{1ui} - 1) (\Psi(\gamma_{1ui}) - \Psi(\sum_{l=1}^{k_1} \gamma_{1ul})) + \sum_{u=1}^{n_1} \log \Gamma(\sum_{i=1}^{k_1} \gamma_{1ui}) - \sum_{u=1}^{n_1} \sum_{i=1}^{k_1} \log \Gamma(\gamma_{1ui})$
$E_q[\log q(\mathbf{z}_1 \phi_1)]$	$\sum_{u=1}^{n_1} \sum_{v=1}^{n_2} \sum_{i=1}^{k_1} \sum_{j=1}^{k_2} \delta_{uv} \phi_{1ui} \phi_{2vj} \log \phi_{1ui}$
$E_q[\log p(X \mathbf{z}_1, \mathbf{z}_2, b, \Theta)]$	$\sum_{u=1}^{n_1} \sum_{v=1}^{n_2} \sum_{i=1}^{k_1} \sum_{j=1}^{k_2} \delta_{uv} \phi_{1ui} \phi_{2vj} \left(-\frac{(x_{uv} - \mu_{ij} - bm_{1u} - bm_{2v})^2}{2\sigma_{ij}^2} - \log \sqrt{2\pi\sigma_{ij}^2} \right)$

Table 1: Expressions for terms in $L(\gamma_1, \gamma_2, \phi_1, \phi_2; \alpha_1, \alpha_2, b, \Theta)$ using q .

(4.3)

$$\phi_{1ui} \propto \exp\left(\Psi(\gamma_{1ui})\right) \times \exp\left(\frac{\sum_{v,j} \delta_{uv} \phi_{2vj} \left(-\frac{(x_{uv} - \mu_{ij} - bm_{1u} - bm_{2v})^2}{2\sigma_{ij}^2} - \log \sigma_{ij}\right)}{w_u}\right)$$

(4.4)

$$\phi_{2vj} \propto \exp\left(\Psi(\gamma_{2vj})\right) \times \exp\left(\frac{\sum_{u,i} \delta_{uv} \phi_{1ui} \left(-\frac{(x_{uv} - \mu_{ij} - bm_{1u} - bm_{2v})^2}{2\sigma_{ij}^2} - \log \sigma_{ij}\right)}{w_v}\right)$$

(4.5)

$$\gamma_{1ui} = \alpha_{1i} + w_u \phi_{1ui}$$

(4.6)

$$\gamma_{2vj} = \alpha_{2j} + w_v \phi_{2vj}$$

where $[i]_1^{k_1}, [j]_1^{k_2}, [u]_1^{n_1}, [v]_1^{n_2}$. The solution is not in a closed form. ϕ_1 and ϕ_2 depend on each other. γ_1 and γ_2 depend on ϕ_1 and ϕ_2 respectively.

4.1.2 Parameter Estimation The optimum values of $(\gamma_1^*, \gamma_2^*, \phi_1^*, \phi_2^*)$ from the inference step gives a tightest lower bound $L(\gamma_1^*, \gamma_2^*, \phi_1^*, \phi_2^*; \alpha_1, \alpha_2, b, \Theta)$ as the surrogate objective function. Maximizing L yields the estimation of model parameters. In particular, the update functions for α_1 and α_2 are given as [17]:

$$(4.7) \quad \alpha'_1 = \alpha_1 + \eta H(\alpha_1)^{-1} g(\alpha_1)$$

$$(4.8) \quad \alpha'_2 = \alpha_2 + \eta H(\alpha_2)^{-1} g(\alpha_2),$$

where $H(\cdot)$ and $g(\cdot)$ are the Hessian matrix and gradient at α_1 or α_2 , and $0 < \eta \leq 1$ is used for line search to determine the step size in the update direction.

The update functions for $\Theta = \{\mu_{ij}, \sigma_{ij}^2, [i]_1^{k_1}, [j]_1^{k_2}\}$ and b are given by

$$(4.9) \quad \mu_{ij} = \frac{\sum_{u,v} \delta_{uv} \phi_{1ui} \phi_{2vj} (x_{uv} - bm_{1u} - bm_{2v})}{\sum_{u,v} \delta_{uv} \phi_{1ui} \phi_{2vj}}$$

$$(4.10) \quad \sigma_{ij}^2 = \frac{\sum_{u,v} \delta_{uv} \phi_{1ui} \phi_{2vj} (x_{uv} - bm_{1u} - bm_{2v} - \mu_{ij})^2}{\sum_{u,v} \delta_{uv} \phi_{1ui} \phi_{2vj}}$$

$$(4.11) \quad b = \frac{\sum_{u,v,i,j} \delta_{uv} \phi_{1ui} \phi_{2vj} (x_{uv} - \mu_{ij})(m_{1u} + m_{2v})}{\sum_{u,v,i,j} \delta_{uv} \phi_{1ui} \phi_{2vj} (m_{1u} + m_{2v})^2},$$

where $[i]_1^{k_1}$ and $[j]_1^{k_2}$. The solution is not in a closed form since b and Θ depend on each other.

4.1.3 EM algorithm Starting from an initial guess of model parameters $(\alpha_1^{(0)}, \alpha_2^{(0)}, b^{(0)}, \Theta^{(0)})$, to find out the optimum $(\alpha_1^{(*)}, \alpha_2^{(*)}, b^{(*)}, \Theta^{(*)})$ that maximizes $\log p(X|\alpha_1, \alpha_2, b, \Theta)$, an EM-style algorithm alternates between the following two steps until convergence:

1. **E-step:** Given $(\alpha_1^{(t-1)}, \alpha_2^{(t-1)}, b^{(t-1)}, \Theta^{(t-1)})$, in the t^{th} iteration, get the variational parameters

$$(\gamma_1^{(t)}, \gamma_2^{(t)}, \phi_1^{(t)}, \phi_2^{(t)})$$

$$= \operatorname{argmax}_{(\gamma_1, \gamma_2, \phi_1, \phi_2)} L(\gamma_1, \gamma_2, \phi_1, \phi_2; \alpha_1^{(t-1)}, \alpha_2^{(t-1)}, b^{(t-1)}, \Theta^{(t-1)}).$$

Then, $L(\gamma_1^{(t)}, \gamma_2^{(t)}, \phi_1^{(t)}, \phi_2^{(t)}; \alpha_1, \alpha_2, b, \Theta)$ serves as the lower bound function to $\log p(X|\alpha_1, \alpha_2, b, \Theta)$. Note that since there are dependencies between ϕ_1, ϕ_2, γ_1 and γ_2 , there is an inner loop between ϕ_1 and ϕ_2 , and an outer loop between (ϕ_1, ϕ_2) and (γ_1, γ_2) until convergence.

2. **M-step:** Update the model parameters as follows:

$$(\alpha_1^{(t)}, \alpha_2^{(t)}, b^{(t)}, \Theta^{(t)}) = \operatorname{argmax}_{(\alpha_1, \alpha_2, b, \Theta)} L(\gamma_1^{(t)}, \gamma_2^{(t)}, \phi_1^{(t)}, \phi_2^{(t)}; \alpha_1, \alpha_2, b, \Theta).$$

Because of the dependency between b and Θ as in (4.9)-(4.11), the M-step goes from (4.9) to (4.11) for several iterations until convergence.

After t iterations, the objective function becomes $L(\gamma_1^{(t)}, \gamma_2^{(t)}, \phi_1^{(t)}, \phi_2^{(t)}; \alpha_1^{(t)}, \alpha_2^{(t)}, b^{(t)}, \Theta^{(t)})$. In $(t+1)^{\text{th}}$ iteration,

$$L(\gamma_{1,2}^{(t)}, \phi_{1,2}^{(t)}; \alpha_{1,2}^{(t)}, b^{(t)}, \Theta^{(t)})$$

$$(4.12) \quad \leq L(\gamma_{1,2}^{(t+1)}, \phi_{1,2}^{(t+1)}; \alpha_{1,2}^{(t)}, b^{(t)}, \Theta^{(t)})$$

$$(4.13) \quad \leq L(\gamma_{1,2}^{(t+1)}, \phi_{1,2}^{(t+1)}; \alpha_{1,2}^{(t+1)}, b^{(t+1)}, \Theta^{(t+1)}).$$

The first inequality holds because (4.12) is the maximum of $L(\gamma_1, \gamma_2, \phi_1, \phi_2; \alpha_1^{(t)}, \alpha_2^{(t)}, b^{(t)}, \Theta^{(t)})$ in E-step, and the second inequality holds because (4.13) is the maximum of $L(\gamma_1^{(t+1)}, \gamma_2^{(t+1)}, \phi_1^{(t+1)}, \phi_2^{(t+1)}; \alpha_1, \alpha_2, b, \Theta)$ in M-step. Therefore, the objective function is non-decreasing until convergence [14].

4.1.4 Prediction The prediction of the missing entries in existent rows and columns is straightforward. After running variational EM, we get not only the model parameters $(\alpha_1^*, \alpha_2^*, b^*, \Theta^*)$, but also the variational parameters $(\phi_1, \phi_2, \gamma_1, \gamma_2)$, where $\phi_{1u} = [\phi_{1u1}, \phi_{1u2}, \dots, \phi_{1uk_1}]$ gives the mixed-membership of

Term	Expression
$E_q[\log p(\boldsymbol{\pi}_1 \alpha_1)]$	$\sum_{u=1}^{n_1} \sum_{i=1}^{k_1} (\alpha_{1i} - 1)(\Psi(\gamma_{1ui}) - \Psi(\sum_{l=1}^{k_1} \gamma_{1ul})) + n_1 \log \Gamma(\sum_{i=1}^{k_1} \alpha_{1i}) - n_1 \sum_{i=1}^{k_1} \log \Gamma(\alpha_{1i})$
$E_q[\log p(\mathbf{z}_1 \boldsymbol{\pi}_1)]$	$\sum_{u=1}^{n_1} \sum_{v=1}^{n_2} \sum_{i=1}^{k_1} \sum_{j=1}^{k_2} \phi_{1uvi} \phi_{2uvj} (\Psi(\gamma_{1ui}) - \Psi(\sum_{l=1}^{k_1} \gamma_{1ul}))$
$E_q[\log q(\boldsymbol{\pi}_1 \gamma_1)]$	$\sum_{u=1}^{n_1} \sum_{i=1}^{k_1} (\gamma_{1ui} - 1)(\Psi(\gamma_{1ui}) - \Psi(\sum_{l=1}^{k_1} \gamma_{1ul})) + \sum_{u=1}^{n_1} \log \Gamma(\sum_{i=1}^{k_1} \gamma_{1ui}) - \sum_{u=1}^{n_1} \sum_{i=1}^{k_1} \log \Gamma(\gamma_{1ui})$
$E_q[\log q(\mathbf{z}_1 \phi_1)]$	$\sum_{u=1}^{n_1} \sum_{v=1}^{n_2} \sum_{i=1}^{k_1} \sum_{j=1}^{k_2} \delta_{uv} \phi_{1uvi} \phi_{2uvj} \log \phi_{1uvi}$
$E_q[\log p(X \mathbf{z}_1, \mathbf{z}_2, b, \Theta)]$	$\sum_{u=1}^{n_1} \sum_{v=1}^{n_2} \sum_{i=1}^{k_1} \sum_{j=1}^{k_2} \delta_{uv} \phi_{1uvi} \phi_{2uvj} \left(-\frac{(x_{uv} - \mu_{ij} - b m_{1u} - b m_{2v})^2}{2\sigma_{ij}^2} - \log \sqrt{2\pi\sigma_{ij}^2} \right)$

Table 2: Expressions for terms in $L(\gamma_1, \gamma_2, \phi_1, \phi_2; \alpha_1, \alpha_2, b, \Theta)$ using q' .

each row u belonging to each row cluster i , $[i]_1^{k_1}$. Similarly, ϕ_{2v} gives the mixed-membership of each column v belonging to each column cluster j , $[j]_1^{k_2}$. Therefore, $\phi_{1ui}\phi_{2vj}$ gives the mixed membership of x_{uv} to each co-cluster (i, j) . The prediction of each entry r_{uv} of the residual matrix R is given by $\hat{r}_{uv} = \phi_{1ui}\phi_{2vj}\mu_{ij}^*$, which is the solution of [3]

$$\underset{\hat{r}_{uv}}{\operatorname{argmin}} E_{(i,j) \sim \phi_{1ui}\phi_{2vj}} [\|\mu_{ij}^* - \hat{r}_{uv}\|^2].$$

Therefore, after adjusting the effect of row and column means, the prediction of each entry \hat{x}_{uv} is

$$(4.14) \quad \hat{x}_{uv} = \sum_{i=1}^{k_1} \sum_{j=1}^{k_2} \phi_{1ui}\phi_{2vj}\mu_{ij}^* + b^* m_{1u} + b^* m_{2v},$$

The whole matrix could be approximated using

$$(4.15) \quad \hat{X} = G_1^T M^* G_2 + b^* \mathbf{m}_1 \mathbf{e}_1^T + b^* \mathbf{e}_2 \mathbf{m}_2^T,$$

where G_1 and G_2 are matrices with ϕ_{1u} and ϕ_{2v} in columns respectively, M^* is a $k_1 \times k_2$ matrix with μ_{ij}^* on the $(i, j)^{th}$ entry, $\mathbf{m}_1 = [m_{11}, \dots, m_{1n_1}]^T$, $\mathbf{m}_2 = [m_{21}, \dots, m_{2n_2}]^T$, and \mathbf{e}_1 and \mathbf{e}_2 are $n_2 \times 1$ and $n_1 \times 1$ vectors consisting all ones respectively.

RBC is also able to handle the prediction of missing entries on new rows or columns which are not used in training process. When there are new users coming into the recommendation system and giving two or three ratings, RBC is able to predict the rest of the ratings for the new users without retraining the model. In particular, given an $n_1 \times n_2$ data matrix X , we first run RBC on it to get the model parameters $(\alpha_1^*, \alpha_2^*, b^*, \Theta^*)$. For h new coming rows \mathbf{s} with a few non-missing entries, appending it to X yields an $(n_1 + h) \times n_2$ matrix Z . We run E-step on Z given $(\alpha_1^*, \alpha_2^*, b^*, \Theta^*)$ to get $(\phi_1, \phi_2, \gamma_1, \gamma_2)$ for Z . The prediction then can be performed following (4.14) for the missing entries in \mathbf{s} . The algorithm is similarly applicable to new columns.

5 Parallel RBC

In very large matrices with millions of rows and columns, it is desirable to run missing value prediction in a distributed and parallel way for high efficiency. In this section, we propose parallel RBC which is applicable to large scale matrices.

5.1 Fully Factorized Variational Distribution

The update expressions for ϕ_1 and ϕ_2 in (4.3) and (4.4) show that the update of ϕ_{1u} for each row u depends on ϕ_{2v} of all columns v , $[v]_1^{n_2}$, and the update of ϕ_{2v} for each column v depends on ϕ_{1u} of all rows u , $[u]_1^{n_1}$. Such dependency makes it difficult to parallelize RBC. Therefore, before we propose parallel RBC, we first propose a new variational inference algorithm using a fully factorized variational distribution. RBC with this new inference algorithm is referred to as RBC-FF.

Given objective function in (3.1), to get a tractable lower bound, we have approximated the latent variable distribution $p(\mathbf{z}_1, \mathbf{z}_2, \boldsymbol{\pi}_1, \boldsymbol{\pi}_2|\alpha_1, \alpha_2, b, \Theta, X)$ using a variational distribution $q(\mathbf{z}_1, \mathbf{z}_2, \boldsymbol{\pi}_1, \boldsymbol{\pi}_2|\gamma_1, \gamma_2, \phi_1, \phi_2)$, where all the entries in a same row u (column v) share a same discrete distribution with variational parameter ϕ_{1u} (ϕ_{2v}), as shown by the graphical model in Figure 2. In this section, we propose a new variational distribution q' as a family of fully factorized distributions:

$$q'(\mathbf{z}_1, \mathbf{z}_2, \boldsymbol{\pi}_1, \boldsymbol{\pi}_2|\gamma_1, \gamma_2, \phi) = \left(\prod_{u=1}^{n_1} q'(\pi_{1u}|\gamma_{1u}) \right) \left(\prod_{v=1}^{n_2} q'(\pi_{2v}|\gamma_{2v}) \right) \left(\prod_{u=1}^{n_1} \prod_{v=1}^{n_2} q'(z_{1uv}|\phi_{1uv}) q'(z_{2uv}|\phi_{2uv}) \right).$$

The graphical model of q' is in Figure 3. Instead of sharing a discrete distribution with other entries in a same row (column), each entry x_{uv} has its own $\text{Disc}(\phi_{1uv})$ and $\text{Disc}(\phi_{2uv})$ for z_1 and z_2 respectively. The disadvantage of using q' instead of q is that the number of parameters for variational discrete distributions increases from $(n_1 + n_2)$ to $2N$, where $N = n_1 n_2$ in a full matrix. However, as we will show, using q' reduces the dependency between rows and columns in the inference step, hence facilitates parallel RBC.

Using q' , each type of term in the lower bound function $L(\gamma_1, \gamma_2, \phi_1, \phi_2; \alpha_1, \alpha_2, b, \Theta)$ is given in Table 2. The summation of all terms in L has a coupling of the form $\phi_{1uvi} \times \phi_{2uvj}$. Letting $\Phi_{uvij} = \phi_{1uvi} \phi_{2uvj}$, Φ_{uv} is a $k_1 \times k_2$ matrix which denotes the probability of x_{uv} belonging to each co-cluster (i, j) , and Φ is used to denote the collection of Φ_{uv} for all N non-missing entries.

In the inference step, given $(\alpha_1, \alpha_2, b, \Theta)$, the update functions for variational parameters are given as follows:

$$(5.16) \quad \Phi_{uvij} \propto \exp\left(\Psi(\gamma_{1ui}) + \Psi(\gamma_{2vj})\right) \times \exp\left(-\frac{(x_{uv} - \mu_{ij} - bm_{1u} - bm_{2v})^2}{2\sigma_{ij}^2} - \log \sigma_{ij}\right)$$

$$(5.17) \quad \gamma_{1ui} = \alpha_{1i} + \sum_{v=1}^{n_2} \sum_{j=1}^{k_2} \delta_{uv} \Phi_{uvij}$$

$$(5.18) \quad \gamma_{2vj} = \alpha_{2j} + \sum_{u=1}^{n_1} \sum_{i=1}^{k_1} \delta_{uv} \Phi_{uvij},$$

where $[i]_1^{k_1}, [j]_1^{k_2}, [u]_1^{n_1}, [v]_1^{n_2}$.

In parameter estimation, the updates for α_1 and α_2 are the same as in (4.7) and (4.8), and the updates for $\Theta = \{\mu_{ij}, \sigma_{ij}^2, [i]_1^{k_1}, [j]_1^{k_2}\}$ and b are given by

$$(5.19) \quad \mu_{ij} = \frac{\sum_{u,v} \delta_{uv} \Phi_{uvij} (x_{uv} - bm_{1u} - bm_{2v})}{\sum_{u,v} \delta_{uv} \Phi_{uvij}}$$

$$(5.20) \quad \sigma_{ij}^2 = \frac{\sum_{u,v} \delta_{uv} \Phi_{uvij} (x_{uv} - bm_{1u} - bm_{2v} - \mu_{ij})^2}{\sum_{u,v} \delta_{uv} \Phi_{uvij}}$$

$$(5.21) \quad b = \frac{\sum_{u,v,i,j} \delta_{uv} \Phi_{uvij} (x_{uv} - \mu_{ij})(m_{1u} + m_{2v})}{\sum_{u,v,i,j} \delta_{uv} \Phi_{uvij} (m_{1u} + m_{2v})^2},$$

where $[i]_1^{k_1}, [j]_1^{k_2}$.

We have a similar variational EM algorithm as in Section 4.1.3. In particular, the algorithm alternates between the E-step, updating variational parameters $(\gamma_1, \gamma_2, \Phi)$, and the M-step, updating model parameters $(\alpha_1, \alpha_2, b, \Theta)$ until convergence. The objective function is guaranteed to be non-decreasing.

5.2 Prediction For each entry x_{uv} , since Φ_{uvij} gives its mixed-membership to each co-cluster (i, j) , so $\sum_{j=1}^{k_2} \Phi_{uvij}$ and $\sum_{i=1}^{k_1} \Phi_{uvij}$ give its mixed-membership to the row cluster i and column cluster j respectively. We take the average of row mixed-membership of all entries in a same row u , i.e., $\frac{1}{w_u} \sum_{v,j} \delta_{uv} \Phi_{uvij}$, as the mixed-membership to row clusters for row u . Similarly, we use $\frac{1}{w_v} \sum_{u,i} \delta_{uv} \Phi_{uvij}$ as the mixed-membership to column clusters for column v . Therefore, the prediction of x_{uv} in an existent row and column is given by

$$(5.22) \quad \hat{x}_{uv} = \sum_{i=1}^{k_1} \sum_{j=1}^{k_2} \left(\frac{1}{w_u} \sum_{v=1}^{n_2} \sum_{j=1}^{k_2} \delta_{uv} \Phi_{uvij} \right) \left(\frac{1}{w_v} \sum_{u=1}^{n_1} \sum_{i=1}^{k_1} \delta_{uv} \Phi_{uvij} \right) \mu_{ij}^* + b^* m_{1u} + b^* m_{2v}.$$

We still use (4.15) to predict the matrix, except that G_1 and G_2 have $\frac{1}{w_u} \sum_{v,j} \delta_{uv} \Phi_{uvij}$ and $\frac{1}{w_v} \sum_{u,i} \delta_{uv} \Phi_{uvij}$ in columns respectively.

To predict the entries in a new row, we follow a similar strategy as in Section 4.1.4. Given an $n_1 \times n_2$ data matrix X , we first run RBC-FF on it to get the

model parameters $(\alpha_1^*, \alpha_2^*, b^*, \Theta^*)$, then append h new coming rows \mathbf{s} to X to get an $(n_1 + h) \times n_2$ matrix Z . Given the model parameters, an E-step on Z yields $(\Phi, \gamma_1, \gamma_2)$, which are used in (5.22) to predict the missing entries in \mathbf{s} . The algorithm is also applicable to the new columns.

5.3 Parallel RBC We parallelize RBC based on RBC-FF. It employs a client-server mode, where multiple clients run parts of the algorithm in parallel, and pass intermediate results to the server, which then gathers all the intermediate results to do other computations. Since we are running an EM algorithm, the process goes on for several iterations. For brevity, we call the parallelized algorithm parallel RBC instead of parallel RBC-FF.

In the E-step, the data matrix is divided to several parts and each part is assigned to one client, which calculates Φ_{uvij} for only the entries assigned to it. The division and assignment of the data matrix could be arbitrary, as long as each client knows the indexes (u, v) in the original matrix for its entries. In the extreme case, each client can only have one entry. Without loss of generality, we assume that the original data matrix X is divided to d parts and each client c has access to one submatrix Y^c , $[c]_1^d$. In addition, each client also needs model parameters α_1, α_2, b , and Θ , as well as $\{m_{1u}, m_{2v}, [u]_1^{n_1}, [v]_1^{n_2}\}$. Given γ_1 and γ_2 , each client c can calculate $\Phi^c = \{\Phi_{uvij}, x_{uv} \in Y^c\}$ for all the entries assigned to it using (5.16). The client then calculates the summation as

$$(5.23) \quad \Phi_{1ui}^c = \sum_{x_{uv} \in Y^c} \sum_{j=1}^{k_2} \delta_{uv} \Phi_{uvij}^c$$

$$(5.24) \quad \Phi_{2vj}^c = \sum_{x_{uv} \in Y^c} \sum_{i=1}^{k_1} \delta_{uv} \Phi_{uvij}^c,$$

for $[i]_1^{k_1}, [j]_1^{k_2}, [u]_1^{n_1}, [v]_1^{n_2}$. The clients pass the summations to the server. The server updates γ_1 and γ_2 using these intermediate summations without accessing Φ_{uvij} :

$$(5.25) \quad \gamma_{1ui} = \alpha_{1i} + \sum_{c=1}^d \Phi_{1ui}^c$$

$$(5.26) \quad \gamma_{2vj} = \alpha_{2j} + \sum_{c=1}^d \Phi_{2vj}^c,$$

which are equivalent with (5.17) and (5.18) respectively. The updated γ_1 and γ_2 are passed to the clients for further updating Φ_{uvij} . Another more straightforward option to parallelize the E-step is to calculate Φ^c in each client and pass all these $\{\Phi^c, [c]_1^d\}$ to the server for updating γ_1 and γ_2 . However, it has two limitations

μ_{ij}	$\left(\sum_{u,v} \delta_{uv} \Phi_{uvij} x_{uv} - b \sum_{u,v} \delta_{uv} \Phi_{uvij} (m_{1u} + m_{2v}) \right) / \sum_{u,v} \delta_{uv} \Phi_{uvij}$
σ_{ij}^2	$\left(\sum_{u,v} \delta_{uv} \Phi_{uvij} x_{uv}^2 + b^2 \sum_{u,v} \delta_{uv} \Phi_{uvij} (m_{1u} + m_{2v})^2 + \mu_{ij}^2 \sum_{u,v} \delta_{uv} \Phi_{uvij} \right. \\ \left. - 2b \sum_{u,v} \delta_{uv} \Phi_{uvij} x_{uv} (m_{1u} + m_{2v}) - 2\mu_{ij} \sum_{u,v} \delta_{uv} \Phi_{uvij} x_{uv} + 2b\mu_{ij} \sum_{u,v} \delta_{uv} \Phi_{uvij} (m_{1u} + m_{2v}) \right) / \sum_{u,v} \delta_{uv} \Phi_{uvij}$
b	$\left(\sum_{i,j} \sum_{u,v} \delta_{uv} \Phi_{uvij} x_{uv} (m_{1u} + m_{2v}) - \sum_{i,j} \mu_{ij} \sum_{u,v} \delta_{uv} \Phi_{uvij} (m_{1u} + m_{2v}) \right) / \sum_{i,j} \sum_{u,v} \delta_{uv} \Phi_{uvij} (m_{1u} + m_{2v})^2$

Table 3: Expanded update expressions for μ_{ij} , σ_{ij}^2 , and b in (5.19)-(5.21).

compared to our strategy: First, for very large matrices with millions of entries, the number of Φ_{uvij} is huge, which will be a huge communication cost to pass them from clients to the server for several iterations. Second, the computation for γ_1 and γ_2 cannot be parallelized. In our algorithm, we make most of the computation (both E-step and M-step) in parallel, and avoid passing the huge number of parameters Φ_{uvij} back and forth.

In the M-step, the updating of α_1 and α_2 is performed on the server side following (4.7) and (4.8). For the updating of b and Θ as in (5.19)-(5.21), there are dependencies with each other. However, a closer examination gives us such observation: $\{\mu_{ij}, [i]_1^{k_1}, [j]_1^{k_2}\}$ and b depend on each other, but they do not depend on $\{\sigma_{ij}^2, [i]_1^{k_1}, [j]_1^{k_2}\}$. Therefore, we can run (5.19) and (5.21) iteratively until convergence to get $\{\mu_{ij}, [i]_1^{k_1}, [j]_1^{k_2}\}$ and b , which are then used to calculate $\{\sigma_{ij}^2, [i]_1^{k_1}, [j]_1^{k_2}\}$ using (5.20) in one step. We expand (5.19)-(5.21) to get the expressions in Table 3. Letting

$$(5.27) \quad A_{ij}^c = \sum_{x_{uv} \in Y^c} \delta_{uv} \Phi_{uvij}^c x_{uv}$$

$$(5.28) \quad B_{ij}^c = \sum_{x_{uv} \in Y^c} \delta_{uv} \Phi_{uvij}^c (m_{1u} + m_{2v})$$

$$(5.29) \quad C_{ij}^c = \sum_{x_{uv} \in Y^c} \delta_{uv} \Phi_{uvij}^c$$

$$(5.30) \quad D_{ij}^c = \sum_{x_{uv} \in Y^c} \delta_{uv} \Phi_{uvij}^c x_{uv}^2$$

$$(5.31) \quad E_{ij}^c = \sum_{x_{uv} \in Y^c} \delta_{uv} \Phi_{uvij}^c (m_{1u} + m_{2v})^2$$

$$(5.32) \quad F_{ij}^c = \sum_{x_{uv} \in Y^c} \delta_{uv} \Phi_{uvij}^c x_{uv} (m_{1u} + m_{2v}),$$

we have

$$(5.33) \quad \mu_{ij} = \left(\sum_c A_{ij}^c - b \sum_c B_{ij}^c \right) / \sum_c C_{ij}^c$$

$$(5.34) \quad \sigma_{ij} = \left(\sum_c D_{ij}^c + b^2 \sum_c E_{ij}^c + \mu_{ij}^2 \sum_c C_{ij}^c \right. \\ \left. - 2b \sum_c F_{ij}^c - 2\mu_{ij} \sum_c A_{ij}^c + 2b\mu_{ij} \sum_c B_{ij}^c \right) / \sum_c C_{ij}^c$$

$$(5.35) \quad b = \left(\sum_{i,j} \sum_c F_{ij}^c - \sum_{i,j} \mu_{ij} \sum_c B_{ij}^c \right) / \sum_{i,j} \sum_c E_{ij}^c.$$

The M-step is performed as follows: First each client

c calculates $\{A_{ij}^c, B_{ij}^c, C_{ij}^c, D_{ij}^c, E_{ij}^c, F_{ij}^c, [i]_1^{k_1}, [j]_1^{k_2}\}$ and passes them to the server. The server alternates between (5.33) and (5.35) till convergence to get updated $\{\mu_{ij}, [i]_1^{k_1}, [j]_1^{k_2}\}$ and b , it then updates $\{\sigma_{ij}^2, [i]_1^{k_1}, [j]_1^{k_2}\}$ using (5.34). By organizing update equations as in (5.33)-(5.35), there are two advantages: First, most part of updates ($A^c \dots F^c$) are calculated in parallel in clients for only once each M-step, since they do not change across the iterations, so the computation amount is very small. Second, The updates only depend on ($A^c \dots F^c$), which are $6 k_1 \times k_2$ matrices, instead of Φ , which is a set of $N k_1 \times k_2$ matrices. In real data matrices where $N \gg 6$, the communication cost for passing ($A^c \dots F^c$) is orders of magnitude smaller than passing Φ .

Putting everything together, to run parallel RBC, the server needs access to the whole data matrix X , and each client c needs access to a submatrix Y^c with their indexes (u, v) in X , initial values for the model parameters ($\alpha_1, \alpha_2, b, \Theta$), as well as common initial values of the variational parameters γ_1 and γ_2 to start the E-step. The algorithm runs as follows:

1. E-step:

- (a) Each client c calculates Φ^c for each $x_{uv} \in Y^c$ using (5.16).
- (b) Each client c calculates $\{\Phi_{1ui}^c, [i]_1^{k_1}\}$ and $\{\Phi_{2vj}^c, [j]_1^{k_2}\}$ following (5.23) and (5.24), and passes them to the server.
- (c) Server updates γ_1 and γ_2 following (5.25) and (5.26), and passes them to the clients.
- (d) Go back to (a) until convergence.

2. M-step:

- (a) Each client c calculates ($A^c \dots F^c$) for $[i]_1^{k_1}, [j]_1^{k_2}$ and passes them to the server.
- (b) The server alternates between (5.33) and (5.35) until convergence to get updated $\{\mu_{ij}, [i]_1^{k_1}, [j]_1^{k_2}\}$ and b , then updates $\{\sigma_{ij}^2, [i]_1^{k_1}, [j]_1^{k_2}\}$ following (5.34). The server passes b and Θ to the clients.
- (c) The server updates α_1 and α_2 following (4.7) and (4.8) till convergence and passes them to the clients.

3. Go back to E-step until convergence.

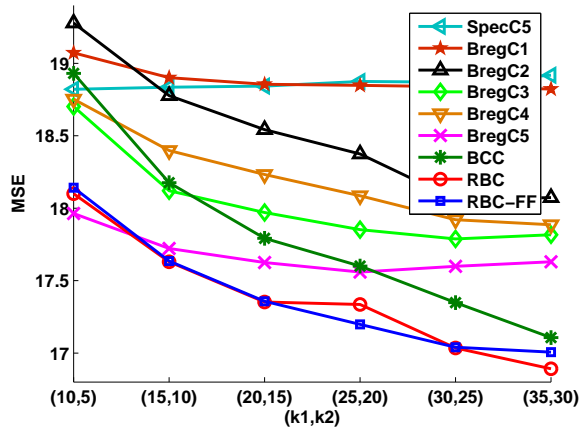


Figure 4: MSE on Jester compared to other co-clustering algorithms with different choices of (k_1, k_2) .

For prediction of entries in existent rows and columns, since the server has the final $\{\Phi_{1ui}^c, [i]_1^{k_1}\}$ and $\{\Phi_{2vj}^c, [j]_1^{k_2}\}$ passed from the clients. It can use

$$(5.36) \quad \hat{x}_{uv} = \sum_{i=1}^{k_1} \sum_{j=1}^{k_2} \left(\frac{1}{w_u} \sum_{c=1}^d \Phi_{1ui}^c \right) \left(\frac{1}{w_v} \sum_{c=1}^d \Phi_{2vj}^c \right) \mu_{ij}^* + b^* m_{1u} + b^* m_{2v}$$

to do prediction, which is equivalent to (5.22). For prediction of entries in new rows \mathbf{s} , the non-missing entries in \mathbf{s} will be assigned to the clients, as well as the updated means of rows and columns. The clients run a parallel E-step to get new $\{\Phi_{1ui}^c, [i]_1^{k_1}\}$ and $\{\Phi_{2vj}^c, [j]_1^{k_2}\}$ and pass them to the server. The server predicts the missing entries using (5.36).

6 Experimental Results

In this section, we present experimental results of missing value prediction for RBC and RBC-FF, and running time for parallel RBC, compared to other algorithms. Missing value prediction includes the prediction for entries in existent rows and columns, as well as in new rows and columns.

Three datasets are used in our experiments—Jester, Movielens, and Foodmart:

1. Jester¹: Jester is a joke rating dataset. The original dataset contains 4.1 million continuous ratings ($[-10, 10]$) of 100 jokes from 73,421 users. We pick 1000 users who rate all 100 jokes and use this dense data matrix in our experiment.
2. Movielens²: Movielens is a movie recommendation dataset created by the Grouplens Research Project.

¹<http://goldberg.berkeley.edu/jester-data/>

²<http://www.grouplens.org/node/73>

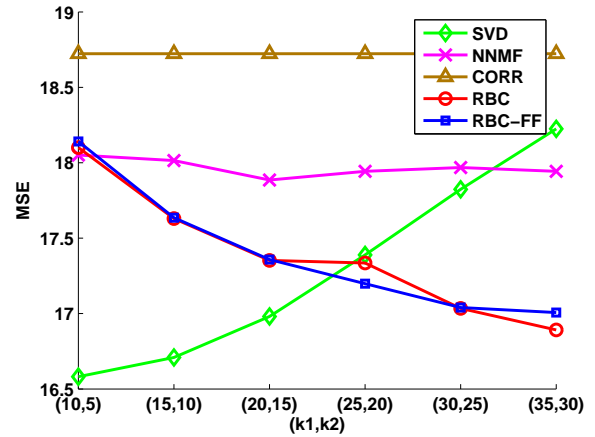


Figure 5: MSE on Jester compared to SVD, NNMF, and CORR with different choices of (k_1, k_2) .

It contains 100,000 ratings (1-5) in a sparse matrix with 1682 movies and 943 users. Following [1], we replace each rating x_{uv} with $\sqrt{6 - x_{uv}}$ to make the distribution closer to Gaussian.

3. Foodmart: Foodmart comes with Microsoft SQL server. It contains transaction data for a fictitious retailer. There are 164,558 sales records in a sparse data matrix for 7803 customers and 1559 products. Each entry is the number of products bought by the customer. We treat the entries with value 0 as missing entries and remove rows and columns with less than 10 non-missing entries.

6.1 Missing Value Prediction for Existent Rows and Columns

We compare RBC and RBC-FF to other algorithms in terms of missing value prediction for existent rows and columns. We run the algorithms using 10-fold cross validation, where in each fold, 10% of the non-missing entries in the data matrix are held out as test data, and the remaining 90% entries are used for training. The training data has at least one entry of the original data matrix in each row and column, so the prediction is performed on “existent” rows and columns.

6.1.1 RBC vs. Other Co-clustering Algorithms

We first compare RBC and RBC-FF to other co-clustering algorithms: Bregman co-clustering [2] with 6 schemes (C1-C6), Bayesian co-clustering [17], and co-clustering based on spectral graph partitioning [8]. For Bregman co-clustering with different schemes, missing value prediction is based on the summary statistics it preserves in each scheme (BregC1-BregC6). For Bayesian co-clustering, it uses $\hat{x}_{uv} = \sum_{i,j} \phi_{1ui} \phi_{2vj} \mu_{ij}^*$ to do prediction. For spectral graph partitioning, we use two prediction strategies following C2 and C5 (SpecC2

k_1, k_2	SpecC2	SpecC5	BregC1	BregC2	BregC3	BregC4	BregC5	BregC6	BCC	RBC	RBC-FF
5,10	0.1175 ± 0.0019	0.0979 ± 0.0013	0.0956 ± 0.0015	0.1073 ± 0.0026	0.0949 ± 0.0015	0.1201 ± 0.0033	0.1073 ± 0.0026	0.1715 ± 0.0080	0.0957 ± 0.0012	0.0943 ± 0.0012	0.0943 ± 0.0010
10,15	0.1141 ± 0.0016	0.0963 ± 0.0013	0.0948 ± 0.0013	0.0959 ± 0.0013	0.0942 ± 0.0012	0.1173 ± 0.0040	0.1090 ± 0.0037	0.2603 ± 0.0084	0.0953 ± 0.0011	0.0935 ± 0.0010	0.0935 ± 0.0011
15,20	0.1136 ± 0.0014	0.0960 ± 0.0009	0.0944 ± 0.0010	0.1100 ± 0.0040	0.0954 ± 0.0012	0.1178 ± 0.0048	0.1100 ± 0.0040	0.3399 ± 0.1112	0.0952 ± 0.0013	0.0931 ± 0.0013	0.0931 ± 0.0013

Table 4: MSE on MovieLens compared to other co-clustering algorithms with different choices of (k_1, k_2) .

k_1, k_2	SpecC2	SpecC5	BregC1	BregC2	BregC3	BregC4	BregC5	BregC6	BCC	RBC	RBC-FF
10,5	0.9758 ± 0.0221	0.9159 ± 0.0199	0.9123 ± 0.0194	0.9765 ± 0.0212	0.9819 ± 0.0217	0.9855 ± 0.0221	0.9415 ± 0.0154	1.4148 ± 0.0168	0.9591 ± 0.0212	0.9119 ± 0.0196	0.9136 ± 0.0197
15,10	0.9767 ± 0.0214	0.9170 ± 0.0191	0.9126 ± 0.0200	1.0178 ± 0.0236	1.0206 ± 0.0237	1.0269 ± 0.0239	0.9875 ± 0.0229	2.0442 ± 0.0262	0.9582 ± 0.0217	0.9111 ± 0.0202	0.9113 ± 0.0204
20,15	0.9785 ± 0.0209	0.9187 ± 0.0192	0.9129 ± 0.0196	1.0561 ± 0.0227	1.0648 ± 0.0222	1.0670 ± 0.0164	1.0304 ± 0.0217	2.9876 ± 0.0505	0.9580 ± 0.0215	0.9106 ± 0.0198	0.9112 ± 0.0217

Table 5: MSE on Foodmart compared to other co-clustering algorithms with different choices of (k_1, k_2) .

and SpecC5). In particular, given row cluster i for row u and column cluster j for column v , C2 predicts x_{uv} following $\hat{x}_{uv} = \mu_{ij}^*$, and C5 predicts x_{uv} following $\hat{x}_{uv} = \mu_{ij}^* + m_{1u} + m_{2v} - \mu_{1i} - \mu_{2j}$, where μ_{1i} and μ_{2j} are the mean for row cluster i and column cluster j respectively. We compare these algorithms using different number of row clusters k_1 and column clusters k_2 . Since spectral graph partitioning keeps the same number of row and column clusters, we set the cluster number to be $\max(k_1, k_2)$.

The average mean square error (MSE) over 10-fold cross validation on Jester is shown in Figure 4. To avoid clutter on the plots, we do not show the results corresponding to SpecC2 and BregC6, which have very poor performance. Table 4 and 5 show the results on MovieLens and Foodmart for all eleven algorithms. As a baseline, The average MSE on Jester, MovieLens and Foodmart using the mean of the training data are 26.9410, 0.1308, and 0.9743 respectively. The observations are as follows: (1) Overall, RBC and RBC-FF achieve lower MSE than other co-clustering algorithms in almost all cases, including different datasets and different number of clusters. The only exception is on Jester with $(k_1, k_2) = (10, 5)$, where BregC5 has the lowest MSE. Such observations indicate that RBC can generate more accurate predictions than other co-clustering algorithms. (2) The fact that RBC and RBC-FF have better performance than BCC shows the advantage to do co-clustering on residue after removing the bias of the rows and columns. (3) The fact that RBC and RBC-FF have better performance than Bregman co-clustering is probably because they are mixed-membership models. Allowing each row (column) to belong to multiple row (column) clusters with varying degrees may yield a more reasonable co-clustering, which further helps missing value prediction. Another evidence for this viewpoint

k_1, k_2	SVD	NNMF	CORR	RBC	RBC-FF
5,10	0.0986 ± 0.0012	0.1086 ± 0.0012	0.4118 ± 0.0061	0.0943 ± 0.0012	0.0943 ± 0.0010
10,15	0.0988 ± 0.0011	0.1078 ± 0.0013	0.4118 ± 0.0061	0.0935 ± 0.0010	0.0935 ± 0.0011
15,20	0.0991 ± 0.0011	0.1080 ± 0.0012	0.4118 ± 0.0061	0.0931 ± 0.0013	0.0931 ± 0.0013

Table 6: MSE on MovieLens compared to SVD, NNMF and CORR with different choices of (k_1, k_2) .

k_1, k_2	SVD	NNMF	CORR	RBC	RBC-FF
10,5	0.8998 ± 0.0210	0.9197 ± 0.0212	1.4528 ± 0.0281	0.9119 ± 0.0196	0.9136 ± 0.0197
15,10	0.8995 ± 0.0208	0.9216 ± 0.0207	1.4528 ± 0.0281	0.9111 ± 0.0202	0.9113 ± 0.0204
20,15	0.9021 ± 0.0211	0.9202 ± 0.0208	1.4528 ± 0.0281	0.9106 ± 0.0198	0.9112 ± 0.0217

Table 7: MSE on Foodmart compared to SVD, NNMF, and CORR with different choices of (k_1, k_2) .

is that BCC, which is the mixed-membership version of BregC2, achieves lower MSE than BregC2. (4) The performance of RBC and RBC-FF improves as k_1 and k_2 increase, because more parameters are used for approximation with a larger k_1 or k_2 . (5) The results from RBC and RBC-FF are very close to each other, indicating that parallel RBC, which is built on RBC-FF, does not sacrifice accuracy.

6.1.2 RBC vs. Other Prediction Algorithms

We then compare RBC and RBC-FF to other missing value prediction algorithms, such as singular value decomposition (SVD) [11], non-negative matrix factorization (NNMF) [12], and correlation based algorithms (CORR) [16]. SVD and NNMF predict the missing entries based on low parameter matrix approximation, and CORR tries to find similar rows as neighbors of the row with the missing entry, and predicts the missing entry using a combination of the neighbors. The missing en-

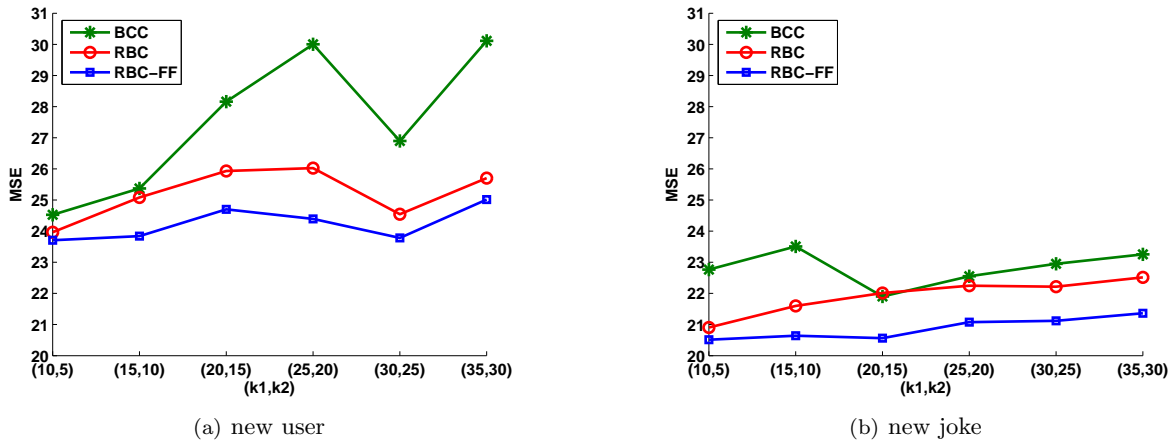


Figure 6: MSE of RBC and BCC for missing value prediction for new users and jokes on Jester.

tries are filled with means of non-missing entries in a same row for SVD and NNMF. The rank of SVD and NNMF is set to be $\max(k_1, k_2)$. For CORR, the number of neighbors is fixed to be 100. Since the ratings in Jester is from -10 to 10, we first add 11 to the training data to run NNMF, and then subtract 11 from the predicted value to get the real prediction.

Figure 5 shows the average MSE over 10-fold cross validation on Jester, and Table 6 and 7 show the results on Movielens and Foodmart respectively. On Jester, when (k_1, k_2) is small, SVD has the lowest MSE. When (k_1, k_2) increases, MSE of SVD increases, but MSE of RBC and RBC-FF decreases and becomes the lowest. On Movielens, RBC and RBC-FF perform the best among all other algorithms. On Foodmart, SVD is the best, but RBC and RBC-FF still achieve lower MSE than NNMF and CORR. Therefore, overall, RBC and RBC-FF have better performance than NNMF and CORR, and have competitive performance with SVD, depending on the datasets and parameters, but as we will show in Section 6.3, parallel RBC scales substantially more efficiently than SVD.

6.2 Missing Value Prediction for New Rows and Columns We show the result for RBC and RBC-FF in the missing value prediction for new rows and columns on Jester. SVD and NNMF can not handle such cases, neither can Bregman co-clustering and spectral graph partitioning. Therefore, we only compare our algorithm to BCC. Each time we randomly hold out 5 rows (columns) as the test set, and train the model using the rest of data. After that, given three random entries in each row (column) of the test set, we try to predict the rest of the entries. We repeat the process for 10 times. The average MSE for RBC, RBC-FF and BCC are presented in Figure 6. As a baseline, the average MSE for rows and columns are 26.5059 and 27.5702

respectively.

The observations are as follows: (1) Overall, RBC and RBC-FF have a lower MSE than BCC, indicating that other than the better performance on existent rows and columns, RBC and RBC-FF also generate more accurate prediction than BCC on new rows and columns. (2) Different from the results in Figure 4, where RBC and RBC-FF have similar performance, in predicting the entries for new rows and columns, RBC-FF performs better than RBC. (3) Another difference is that in Figure 4, the MSE of three algorithms decreases as (k_1, k_2) increases, while in Figure 6, the overall trend of the curves is going up, although not very clear, especially for BCC. A possible reason is that when (k_1, k_2) increases, it is getting more and more difficult to figure out the correct mixed membership to all row (column) clusters for the new rows (columns) with very few entries given. (4) The prediction on new jokes is better than that on new users, which indicates that it is easier to predict the rest of the ratings each joke will receive from different users, than to predict the rest of the ratings each user will give to different jokes. This observation is surprising since for each new joke, we need to predict 997 ratings given 3, while for each new user, we only need to predict 97 ratings given 3. Intuitively, it might be because the ratings each joke gets from different users is more consistent than the ratings each user gives to different jokes.

6.3 Running Time of Parallel RBC We show the running time comparison between parallel RBC and SVD. We randomly generate nine matrices with increasing scales from 500^2 to 4500^2 in steps of 500. To simulate the parallel RBC using one processor, we run RBC-FF and take the running time which would be spent on the client side and server side, denoting as T_c and T_s respectively, so the total time for parallel

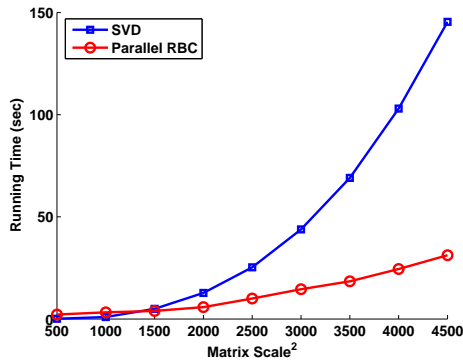


Figure 7: Running time of parallel RBC and SVD on different scale of matrices.

RBC would be approximately $T_c/d + T_s$, where d is the number of clients. Assuming there are ten processors, the running time of RBC and SVD is shown in Figure 7.

When the data matrix is small, both RBC and SVD run very fast, and SVD is slightly faster. When the scale of matrix increases, the running time of SVD increases rapidly, as shown by the steep curve. Comparatively, the curve for parallel RBC only goes up slowly, and its advantage in terms of computational efficiency becomes more and more distinct. In the experiment, we ignored the communication overhead, so the numbers shown in the figure might not be very accurate. However, the trend of the curves is clear. Moreover, in our experiment, we assume that there are only ten processors. In large scale systems with hundreds of processors, parallel RBC could be orders of magnitude faster.

7 Conclusion

In this paper, we have proposed residual Bayesian co-clustering for matrix approximation. It extends Bayesian co-clustering by taking row and column bias into consideration, hence captures a more reasonable generative process of the matrix. Two variational inference algorithms are proposed. One shares a variational distribution among all entries in the same row/column to keep dependency between rows and columns. The other uses a fully factorized variational distribution, which reduces such dependency and leads to parallel RBC. In the experiments of missing value prediction, RBC generates more accurate prediction than several other co-clustering algorithms, and competitive results with matrix factorization based algorithms, such as SVD and NNMF. Moreover, SVD and most of other algorithms cannot do prediction for new rows and columns while RBC can naturally handle such problems. In addition, parallel RBC is much more efficient than SVD, making it applicable to very large matrices in real data.

Acknowledgements: The research was supported by NASA grant NNX08AC36A and NSF grant IIS-0812183.

References

- [1] D. Aggarwal and S. Merugu. Predictive discrete latent factor models for large scale dyadic data. In *KDD*, 2007.
- [2] A. Banerjee, I. Dhillon, J. Ghosh, S. Merugu, and D. Modha. A generalized maximum entropy approach to Bregman co-clustering and matrix approximation. *JMLR*, 2007.
- [3] A. Banerjee, S. Merugu, I. Dhillon, and J. Ghosh. Clustering with Bregman divergences. *JMLR*, 2005.
- [4] O. Barndorff-Nielsen. *Information and Exponential Families in Statistical Theory*. John Wiley and Sons, 1978.
- [5] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet allocation. *JMLR*, 2003.
- [6] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 1977.
- [7] M. Deodhar and J. Ghosh. A framework for simultaneous co-clustering and learning from complex data. In *KDD*, 2007.
- [8] I. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *KDD*, 2001.
- [9] P. Drineas, R. Kannan, and M. Mahoney. Fast monte carlo algorithms for matrices ii: Computing a low rank approximation to a matrix. *SIAM Journal on Computing*, 2006.
- [10] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *PAMI*, 1984.
- [11] G. Golub and C. Van Loan. *Matrix Computations*. John Hopkins University Press, 1996.
- [12] D. Lee and H. Seung. Algorithms for non-negative matrix factorization. In *NIPS*, 2001.
- [13] G. McLachlan and T. Krishnan. *The EM algorithm and Extensions*. Wiley-Interscience, 1996.
- [14] R. Neal and G. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In M. Jordan, editor, *Learning in Graphical Models*. MIT Press, 1998.
- [15] R. Redner and H. Walker. Mixture densities, maximum likelihood and the EM algorithm. *SIAM Review*.
- [16] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *WWW*, 2001.
- [17] H. Shan and A. Banerjee. Bayesian co-clustering. In *ICDM*, 2008.
- [18] J. Sun, Y. Xie, H. Zhang, and C. Faloutsos. Less is more: Sparse graph mining with compact matrix decomposition. In *SDM*, 2007.