Scalable Bayesian Matrix Factorization

Avijit Saha**[,1], Rishabh Misra**[,2], and Balaraman Ravindran[1]

[1] Department of CSE, Indian Institute of Technology Madras, India
{avijit, ravi}@cse.iitm.ac.in
[2] Department of CSE, Thapar University, India
rishabhmisra1994@gmail.com

Abstract. Matrix factorization (MF) is the simplest and most well studied factor based model and has been applied successfully in several domains. One of the standard ways to solve MF is by finding maximum a posteriori estimate of the model parameters, which is equivalent to minimizing the regularized objective function. Stochastic gradient descent (SGD) is a common choice to minimize the regularized objective function. However, SGD suffers from the problem of overfitting and entails tedious job of finding the learning rate and regularization parameters. A fully Bayesian treatment of MF avoids these problems. However, the existing Bayesian matrix factorization method based on the Markov chain Monte Carlo (MCMC) technique has cubic time complexity with respect to the target rank, which makes it less scalable. In this paper, we propose the **S**calable **B**ayesian **M**atrix **F**actorization (SBMF), which is a MCMC Gibbs sampling algorithm for MF and has linear time complexity with respect to the target rank and linear space complexity with respect to the number of non-zero observations. Also, we show through extensive experiments on three sufficiently large real word datasets that SBMF incurs only a small loss in the performance and takes much less time as compared to the baseline method for higher latent dimension.

Keywords: Recommender Systems, Matrix Factorization, Bayesian Inference, Markov Chain Monte Carlo, Scalability.

1 Introduction

Factor based models have been used extensively in collaborative filtering. In a factor based model, preferences of each user are represented by a latent factor vector. Matrix factorization (MF) [1–6] is the simplest and most well studied factor based model and has been applied successfully in several domains. Formally, MF recovers a low-rank latent structure of a matrix by approximating it as a product of two low-rank matrices. For delineation, consider a user-movie

** Both the authors contributed equally.

 In: M. Atzmueller, F. Lemmerich (Eds.): Proceedings of 6th International Workshop on Mining Ubiquitous and Social Environments (MUSE), co-located with the ECML PKDD 2015. Published at http://ceur-ws.org

matrix $\boldsymbol{R} \in \mathbb{R}^{I \times J}$ where the r_{ij} cell represents the rating provided to the j^{th} movie by the i^{th} user. MF decomposes the matrix $\boldsymbol{R}$ into two low-rank matrices $\boldsymbol{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2, ..., \boldsymbol{u}_I]^T \in \mathbb{R}^{I \times K}$ and $\boldsymbol{V} = [\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_J]^T \in \mathbb{R}^{J \times K}$ (K is the latent space dimension) such that:

$$\boldsymbol{R} \sim \boldsymbol{U}\boldsymbol{V}^T. \tag{1}$$

Probabilistic Matrix Factorization (PMF) [4] provides a probabilistic interpretation for MF. In PMF, latent factor vectors are assumed to be marginally independent, whereas rating variables, given the latent factor vectors, are assumed to be conditionally independent. PMF considers the conditional distribution of the rating variables (the likelihood term) as:

$$p(\boldsymbol{R}|\boldsymbol{U}, \boldsymbol{V}, \tau^{-1}) = \prod_{(i,j) \in \Omega} \mathcal{N}(r_{ij}|\boldsymbol{u}_i^T \boldsymbol{v}_j, \tau^{-1}), \tag{2}$$

where Ω is the set of all observed entries in $\boldsymbol{R}$ provided during the training and τ is the model precision. Zero-mean spherical Gaussian priors are placed on the latent factor vectors of users and movies. The main drawback of this model is that inferring the posterior distribution over the latent factor vectors, given the ratings, is intractable. PMF handles this intractability by providing a maximum a posteriori estimation of the model parameters by maximizing the log-posterior over the model parameters, which is equivalent to minimizing the regularized square error loss defined as:

$$\sum_{(i,j) \in \Omega} \left(r_{ij} - \boldsymbol{u}_i^T \boldsymbol{v}_j\right)^2 + \lambda \left(||\boldsymbol{U}||_F^2 + ||\boldsymbol{V}||_F^2\right), \tag{3}$$

where λ is the regularization parameter and $||\boldsymbol{X}||_F^2$ is the Frobenius norm of $\boldsymbol{X}$. The optimization problem in Eq. (3) can be solved using stochastic gradient descent (SGD) [2]. SGD is an online algorithm which obviates the need to store the entire dataset in the memory. Although SGD is scalable and enjoys local convergence guarantee [7], it often overfits the data and requires manual tuning of the learning rate and regularization parameters. Hence, maximum a posteriori estimation of MF suffers from the problem of overfitting and entails tedious job of finding the learning rate (if SGD is the choice of optimization) and regularization parameters.

On the other hand, fully Bayesian methods [5, 8–10] for MF do not require manual tuning of the learning rate and regularization parameters and are robust to overfitting. As direct evaluation of posterior is intractable in practice, approximate inference techniques are adopted to learn the posterior distribution. One of the possible choices of approximate inference is to apply variational approximate inference technique [8, 9]. Bayesian MF based on the variational approximation [11–13, 10] considers a simplified factorized distribution and assumes that the latent factors of users are independent of the latent factors of items while approximating the posterior. But this assumption often leads to over simplification and can produce inaccurate results as shown in [5]. On the other hand, Markov chain Monte Carlo (MCMC) based approximation method

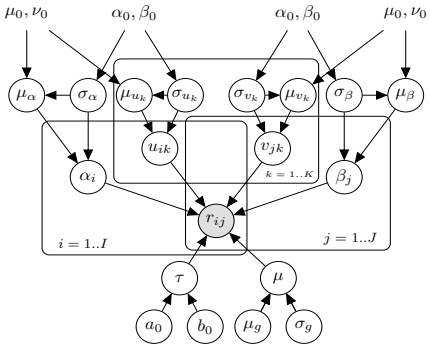

Fig. 1. Graphical model representation of SBMF.

can produce exact results when provided with infinite resources. MCMC based Bayesian Probabilistic Matrix Factorization (BPMF) [5] directly approximates the posterior distribution using the Gibbs sampling technique and outperforms the variational based approximation.

In BPMF, user/item latent factor vectors are assumed to follow a multivariate Gaussian distribution, which results cubic time complexity with respect to the latent factor vector dimension. Though BPMF performs well in many applications, this cubic time complexity makes it difficult to apply BPMF on very large datasets. In this paper, we propose the **S**calable **B**ayesian **M**atrix **F**actorization (SBMF) based on the MCMC Gibbs sampling, where we assume univariate Gaussian priors on each dimension of the latent factor. Due to this assumption, the complexity of SBMF reduces to linear with respect to the latent factor vector dimension. We also consider user and item bias terms in SBMF which are missing in BPMF. These bias terms capture the variation in rating values that are independent of any user-item interaction. Also, the proposed SBMF algorithm is parallelized for multicore environments. We show through extensive experiments on three large scale real world datasets that the adopted univariate approximation in SBMF results in only a small performance loss and provides significant speed up when compared with the baseline method BPMF for higher values of latent dimension.

2 Method

2.1 Model

Fig. 1 shows a graphical model representation of SBMF. Consider Ω as the set of observed entries in $\boldsymbol{R}$ provided during the training phase. The observed data r_{ij} is assumed to be generated as follows:

$$r_{ij} = \mu + \alpha_i + \beta_j + \boldsymbol{u}_i^T \boldsymbol{v}_j + \epsilon_{ij}, \tag{4}$$

where $(i, j) \in \Omega$, μ is the global bias, α_i is the bias associated with the i^{th} user, β_j is the bias associated with the j^{th} item, $\boldsymbol{u}_i$ is the latent factor vector of dimension K associated with the i^{th} user, and $\boldsymbol{v}_j$ is the latent factor vector of dimension K associated with the j^{th} item. Uncertainty in the model is absorbed by the noise ϵ_{ij} which is generated as $\epsilon_{ij} \sim \mathcal{N}(0, \tau^{-1})$, where τ is the precision parameter. Bias terms are particularly helpful in capturing the individual bias for user/item: a user may have the tendency to rate all the items higher than the other users or an item may get higher ratings if it is perceived better than the others [2].

The conditional on the observed entries of $\boldsymbol{R}$ (the likelihood term) can be written as follows:

$$p(\boldsymbol{R}|\boldsymbol{\Theta}) = \prod_{(i,j)\in\Omega} \mathcal{N}(r_{ij}|\mu + \alpha_i + \beta_j + \boldsymbol{u}_i^T \boldsymbol{v}_j, \tau^{-1}), \tag{5}$$

where $\boldsymbol{\Theta} = \{\tau, \mu, \{\alpha_i\}, \{\beta_j\}, \boldsymbol{U}, \boldsymbol{V}\}$. We place independent univariate priors on all the model parameters in $\boldsymbol{\Theta}$ as follows:

$$p(\mu) = \mathcal{N}(\mu|\mu_g, \sigma_g^{-1}), \tag{6}$$

$$p(\alpha_i) = \mathcal{N}(\alpha_i|\mu_\alpha, \sigma_\alpha^{-1}), \tag{7}$$

$$p(\beta_j) = \mathcal{N}(\beta_j|\mu_\beta, \sigma_\beta^{-1}), \tag{8}$$

$$p(\boldsymbol{U}) = \prod_{i=1}^{I} \prod_{k=1}^{K} \mathcal{N}(u_{ik}|\mu_{u_k}, \sigma_{u_k}^{-1}), \tag{9}$$

$$p(\boldsymbol{V}) = \prod_{j=1}^{J} \prod_{k=1}^{K} \mathcal{N}(v_{jk}|\mu_{v_k}, \sigma_{v_k}^{-1}), \tag{10}$$

$$p(\tau) = \mathcal{N}(\tau|a_0, b_0). \tag{11}$$

We further place Normal-Gamma priors on all the hyperparameters $\boldsymbol{\Theta}_H = \{\mu_\alpha, \sigma_\alpha, \mu_\beta, \sigma_\beta, \{\mu_{u_k}, \sigma_{u_k}\}, \{\mu_{v_k}, \sigma_{v_k}\}\}$ as follows:

$$p(\mu_\alpha, \sigma_\alpha) = \mathcal{NG}\left(\mu_\alpha, \sigma_\alpha|\mu_0, \nu_0, \alpha_0, \beta_0\right), \tag{12}$$

$$p(\mu_\beta, \sigma_\beta) = \mathcal{NG}\left(\mu_\beta, \sigma_\beta|\mu_0, \nu_0, \alpha_0, \beta_0\right), \tag{13}$$

$$p(\mu_{u_k}, \sigma_{u_k}) = \mathcal{NG}\left(\mu_{u_k}, \sigma_{u_k}|\mu_0, \nu_0, \alpha_0, \beta_0\right), \tag{14}$$

$$p(\mu_{v_k}, \sigma_{v_k}) = \mathcal{NG}\left(\mu_{v_k}, \sigma_{v_k}|\mu_0, \nu_0, \alpha_0, \beta_0\right). \tag{15}$$

We denote $\{a_0, b_0, \mu_g, \sigma_g, \mu_0, \nu_0, \alpha_0, \beta_0\}$ as $\boldsymbol{\Theta}_0$ for notational convenience. The joint distribution of the observations and the hidden variables can be written as:

$$p(\boldsymbol{R}, \boldsymbol{\Theta}, \boldsymbol{\Theta}_H|\boldsymbol{\Theta}_0) = p(\boldsymbol{R}|\boldsymbol{\Theta})p(\mu)\prod_{i=1}^{I} p(\alpha_i)\prod_{j=1}^{J} p(\beta_j)p(\boldsymbol{U})p(\boldsymbol{V})p(\mu_\alpha, \sigma_\alpha)$$
$$p(\mu_\beta, \sigma_\beta)\prod_{k=1}^{K} p(\mu_{u_k}, \sigma_{u_k})p(\mu_{v_k}, \sigma_{v_k}). \tag{16}$$

2.2 Inference

Since evaluation of the joint distribution in Eq. (16) is intractable, we adopt a Gibbs sampling based approximate inference technique. As all our model parameters are conditionally conjugate [10], equations for Gibbs sampling can be written in closed form using the joint distribution as given in Eq. (16). Replacing Eq. (5)-(15) in Eq. (16), the sampling distribution of u_{ik} can be written as follows:

$$p(u_{ik}|-) \sim \mathcal{N}\left(u_{ik}|\mu^*, \sigma^*\right), \tag{17}$$

where,

$$\sigma^* = \left(\sigma_{u_k} + \tau \sum_{j \in \Omega_i} v_{jk}^2\right)^{-1}, \tag{18}$$

$$\mu^* = \sigma^* \left(\sigma_{u_k}\mu_{u_k} + \tau \sum_{j \in \Omega_i} v_{jk} \left(r_{ij} - \left(\mu + \alpha_i + \beta_j + \sum_{l=1 \& l \neq k}^{K} u_{il}v_{jl}\right)\right)\right). \tag{19}$$

Here, Ω_i is the set of items rated by the i^{th} user in the training set. Now, directly sampling u_{ik} from Eq. (17) requires $O(K|\Omega_i|)$ complexity. However if we precompute a quantity $e_{ij} = r_{ij} - (\mu + \alpha_i + \beta_j + u_i^T v_j)$ for all $(i,j) \in \Omega$ and write Eq. (19) as:

$$\mu^* = \sigma^* \left(\sigma_{u_k}\mu_{u_k} + \tau \sum_{j \in \Omega_i} v_{jk} \left(e_{ij} + u_{ik}v_{jk}\right)\right), \tag{20}$$

then the sampling complexity of u_{ik} reduces to $O(|\Omega_i|)$. Table 1 shows the space and time complexities of SBMF and BPMF. We sample model parameters in parallel whenever they are independent to each other. Algorithm 1 describes the detailed Gibbs sampling procedure.

Table 1. Complexity Comparison

Method	Time Complexity	Space Complexity
SBMF	$O(\|\Omega\|K)$	$O((I+J)K)$
BPMF	$O(\|\Omega\|K^2 + (I+J)K^3)$	$O((I+J)K)$

Algorithm 1 **S**calable **B**ayesian **M**arix **F**actorization (SBMF)

Require: $\boldsymbol{\Theta}_0$, initialize $\boldsymbol{\Theta}$ and $\boldsymbol{\Theta}_H$.
Ensure: Compute e_{ij} for all $(i,j) \in \Omega$
for $t = 1$ **to** T **do**
  // Sample hyperparameters
  $\alpha^* = \alpha_0 + \frac{1}{2}(I+1)$, $\beta^* = \beta_0 + \frac{1}{2}(\nu_0 (\mu_\alpha - \mu_0)^2 + \sum_{i=1}^{I} (\alpha_i - \mu_\alpha))$. Sample $\sigma_\alpha \sim \Gamma(\alpha^*, \beta^*)$.
  $\sigma^* = (\nu_0\sigma_\alpha + \sigma_\alpha I)^{-1}$, $\mu^* = \sigma^*(\nu_0\sigma_\alpha\mu_0 + \sigma_\alpha \sum_{i=1}^{I} \alpha_i)$. Sample $\mu_\alpha \sim \mathcal{N}(\mu^*, \sigma^*)$.
  $\alpha^* = \beta_0 + \frac{1}{2}(J+1)$, $\beta^* = \beta_0 + \frac{1}{2}(\nu_0 (\mu_\beta - \mu_0)^2 + \sum_{j=1}^{J} (\beta_j - \mu_\beta))$. Sample $\sigma_\beta \sim \Gamma(\alpha^*, \beta^*)$.
  $\sigma^* = (\nu_0\sigma_\beta + \sigma_\beta J)^{-1}$, $\mu^* = \sigma^*(\nu_0\sigma_\beta\mu_0 + \sigma_\beta \sum_{j=1}^{J} \beta_j)$. Sample $\mu_\beta \sim \mathcal{N}(\mu^*, \sigma^*)$.
  for $k = 1$ to K **do in parallel**
    $\alpha^* = \alpha_0 + \frac{1}{2}(I+1)$, $\beta^* = \beta_0 + \frac{1}{2}(\nu_0 (\mu_{u_k} - \mu_0)^2 + \sum_{i=1}^{I} (u_{ik} - \mu_{u_k}))$.
    Sample $\sigma_{u_k} \sim \Gamma(\alpha^*, \beta^*)$.
    $\sigma^* = (\nu_0\sigma_{u_k} + \sigma_{u_k} I)^{-1}$, $\mu^* = \sigma^*(\nu_0\sigma_{u_k}\mu_0 + \sigma_{u_k} \sum_{i=1}^{I} u_{ik})$. Sample $\mu_{u_k} \sim \mathcal{N}(\mu^*, \sigma^*)$.
    $\alpha^* = \beta_0 + \frac{1}{2}(J+1)$, $\beta^* = \beta_0 + \frac{1}{2}(\nu_0 (\mu_{v_k} - \mu_0)^2 + \sum_{j=1}^{J} (v_{jk} - \mu_{v_k}))$.
    Sample $\sigma_{v_k} \sim \Gamma(\alpha^*, \beta^*)$.
    $\sigma^* = (\nu_0\sigma_{v_k} + \sigma_{v_k} J)^{-1}$, $\mu^* = \sigma^*(\nu_0\sigma_{v_k}\mu_0 + \sigma_{v_k} \sum_{j=1}^{J} v_{jk})$. Sample $\mu_{v_k} \sim \mathcal{N}(\mu^*, \sigma^*)$.
  end for
  $a_0^* = a_0 + \frac{1}{2}|\Omega|$, $b_0^* = b_0 + \frac{1}{2} \sum_{(i,j)\in\Omega} e_{ij}^2$. Sample $\tau \sim \Gamma(a_0^*, b_0^*)$.
  // Sample model parameters
  $\sigma^* = (\sigma_g + \tau|\Omega|)^{-1}$, $\mu^* = \sigma^*(\sigma_g\mu_g + \tau \sum_{(i,j)\in\Omega} (e_{ij} + \mu))$. Sample $\mu \sim \mathcal{N}(\mu^*, \sigma^*)$.
  for $(i,j) \in \Omega$ **do in parallel**
    $e_{ij} = e_{ij} + (\mu_{old} - \mu)$
  end for
  for $i = 1$ **to** I **do in parallel**
    $\sigma^* = (\sigma_\alpha + \tau|\Omega_i|)^{-1}$, $\mu^* = \sigma^*(\sigma_\alpha\mu_\alpha + \tau \sum_{j\in\Omega_i} (e_{ij} + \alpha_i))$. Sample $\alpha_i \sim \mathcal{N}(\mu^*, \sigma^*)$.
    for $j \in \Omega_i$ **do**
      $e_{ij} = e_{ij} + (\alpha_{old} - \alpha_i)$
    end for
    for $k = 1$ **to** K **do**
      $\sigma^* = (\sigma_{u_k} + \tau \sum_{j\in\Omega_i} v_{jk}^2)^{-1}$, $\mu^* = \sigma^*(\sigma_{u_k}\mu_{u_k} + \tau \sum_{j\in\Omega_i} v_{jk} (e_{ij} + u_{ik}v_{jk}))$.
      Sample $u_{ik} \sim \mathcal{N}(\mu^*, \sigma^*)$.
      for $j \in \Omega_i$ **do**
        $e_{ij} = e_{ij} + v_{jk}(u_{ik}^{old} - u_{ik})$
      end for
    end for
  end for
  for $j = 1$ **to** J **do in parallel**
    $\sigma^* = (\sigma_\beta + \tau|\Omega_j|)^{-1}$, $\mu^* = \sigma^*(\sigma_\beta\mu_\beta + \tau \sum_{i\in\Omega_j} (e_{ij} + \beta_j))$. Sample $\beta_j \sim \mathcal{N}(\mu^*, \sigma^*)$.
    for $i \in \Omega_j$ **do**
      $e_{ij} = e_{ij} + (\beta_{old} - \beta_j)$
    end for
    for $k = 1$ **to** K **do**
      $\sigma^* = (\sigma_{v_k} + \tau \sum_{i\in\Omega_j} u_{ik}^2)^{-1}$, $\mu^* = \sigma^*(\sigma_{v_k}\mu_{v_k} + \tau \sum_{i\in\Omega_j} u_{ik} (e_{ij} + u_{ik}v_{jk}))$.
      Sample $v_{jk} \sim \mathcal{N}(\mu^*, \sigma^*)$.
      for $i \in \Omega_j$ **do**
        $e_{ij} = e_{ij} + u_{ik}(v_{jk}^{old} - v_{jk})$
      end for
    end for
  end for
end for

3 Experiments

3.1 Datasets

In this section, we show empirical results on three large real world movie-rating datasets[3,4] to validate the effectiveness of SBMF. The details of these datasets are provided in Table 2. Both the Movielens datasets are publicly available and 90:10 split is used to create their train and test sets. For Netflix, the probe data is used as the test set.

3.2 Experimental Setup and Parameter Selection

All the experiments are run on an Intel i5 machine with 16GB RAM. We have considered the serial as well as the parallel implementation of SBMF for all the experiments. In the parallel implementation, SBMF is parallelized in multicore environment using OpenMP library. Although BPMF can also be parallelized, the base paper [5] and it's publicly available code provide only the serial implementation. So in our experiments, we have compared only the serial implementation of BPMF against the serial and the parallel implementations of SBMF. Serial and parallel versions of the SBMF are denoted as SBMF-S and SBMF-P, respectively. Since the performance of both SBMF and BPMF depend on the dimension of latent factor vector (K), it is necessary to investigate how the models work with different values of K. So three sets of experiments are run for each dataset corresponding to $K = \{50, 100, 200\}$ for SBMF-S, SBMF-P, and BPMF. As our main aim is to validate that SBMF is more scalable as compared to BPMF under same conditions, we choose 50 burn-in iterations for all the experiments of SBMF-S, SBMF-P, and BPMF. In Gibbs sampling process burn-in refers to the practice of discarding an initial portion of a Markov chain sample, so that the effect of initial values on the posterior inference is minimized. Note that, if SBMF takes less time than BPMF for a particular burn-in period, then increasing the number of burn-in iterations will make SBMF more scalable as compared to BPMF. Additionally, we allow the methods to have 100 collection iterations.

In SBMF, we initialize parameters in $\boldsymbol{\Theta}$ using a Gaussian distribution with 0 mean and 0.01 variance. All the parameters in $\boldsymbol{\Theta}_H$ are set to 0. Also, $a_0, b_0, \nu_0, \alpha_0$, and β_0 are set to 1, μ_0 and μ_g are set to 0, and σ_g is initialized to 0.01. In BPMF, we use standard parameter setting as provided in the paper [5]. We collect samples of user and item latent factor vectors and bias terms from the collection iterations and approximate a rating r_{ij} as:

$$\hat{r}_{ij}^t = \frac{1}{C} \sum_{c=1}^{C} \left(\mu^c + \alpha_i^c + \beta_j^c + \boldsymbol{u}_i^c . \boldsymbol{v}_j^c \right), \tag{21}$$

[3] http://grouplens.org/datasets/movielens/
[4] http://www.netflixprize.com/

Table 2. Dataset Description

Dataset	No. of users	No. of movies	No. of ratings
Movielens 10m	71567	10681	**10m**
Movielens 20m	138493	27278	**20m**
Netflix	480189	17770	**100m**

where $\boldsymbol{u}_i^c$ and $\boldsymbol{v}_j^c$ are the c^{th} drawn samples of the i^{th} user and the j^{th} item latent factor vectors respectively, μ^c, α_i^c, and β_j^c are the c^{th} drawn samples of the global bias, the i^{th} user bias, and the j^{th} item bias, respectively. C is the number of drawn samples. Then the Root Mean Square Error (RMSE) [2] is used as the evaluation metric for all the experiments. The code for SBMF will be publicly available [5].

3.3 Results

In all the graphs of Fig. 2 , X-axis represents the time elapsed since the starting of experiment and Y-axis presents the RMSE value. Since we allow 50 burn-in iterations for all the experiments and each iteration of BPMF takes more time than SBMF-P, collection iterations of SBMF-P begin earlier than BPMF. Thus we get the initial RMSE value of SBMF-P earlier. Similarly, each iteration of SBMF-S takes less time as compared to BPMF (except for $K = \{50, 100\}$ in the Netflix dataset). We believe that in Netflix dataset (for $K = \{50, 100\}$), BPMF takes less time than SBMF-S because BPMF is implemented in Matlab where matrix computations are efficient. On the other hand, SBMF is implemented in C++ where the matrix storage is unoptimized. As the Netflix data is large with respect to the number of entries and the number of users and items, number of matrix operations are more in it as compared to the other datasets. So for lower values of K, the cost of matrix operations for SBMF-S dominates the cost incurred due to $O(K^3)$ complexity of BPMF. Thus BPMF takes less time than SBMF-S. However, with large values of K, BPMF starts taking more time as the $O(K^3)$ complexity of BPMF becomes dominating. We leave the task of optimizing the code of SBMF as future work.

We can observe from the Fig. 2 that SBMF-P takes much less time in all the experiments than BPMF and incurs only a small loss in the performance. Similarly, SBMF-S also takes less time than the BPMF (except for $K = \{50, 100\}$ in Netflix dataset) and incurs only a small performance loss. Important point to note is that total time difference between both of the variants of SBMF and BPMF increases with the dimension of latent factor vector and the speedup is significantly high for $K = 200$. Table 3 shows the final RMSE values and the total time taken correspond to each dataset and K. We find that the RMSE values for SBMF-S and SBMF-P are very close for all the experiments. We also observe that increasing the latent space dimension reduces the RMSE value in the Netflix

[5] https://github.com/avijit1990, https://github.com/rishabhmisra

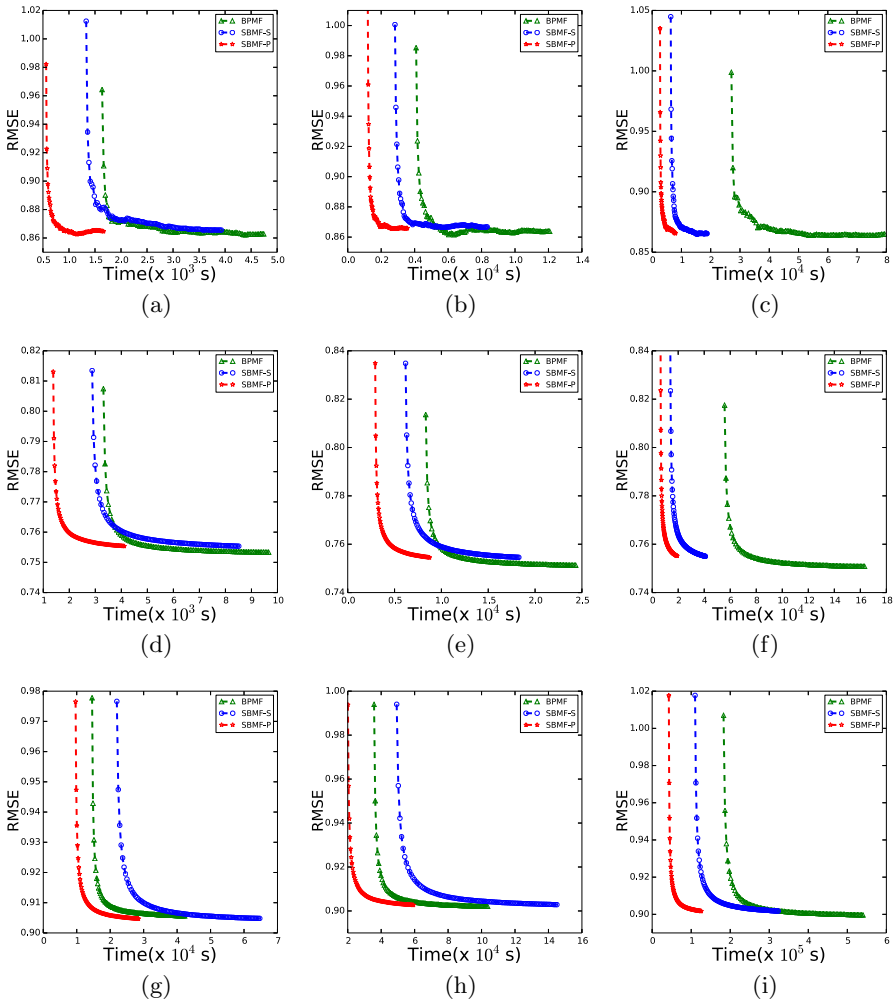

Fig. 2. Left, middle, and right columns show results for $K = 50, 100$, and 200, respectively. {a,b,c}, {d,e,f}, and {g,h,i} are results on Movielens 10m, Movielens 20m, and Netflix datasets, respectively.

dataset. With high latent dimension, the running time of BPMF is significantly high due to its cubic time complexity with respect to the latent space dimension and it takes approximately 150 hours on Netflix dataset with $K = 200$. However, SBMF has linear time complexity with respect to the latent space dimension and SBMF-P and SBMF-S take only 35 and 90 hours (approximately) respectively on the Netflix dataset with $K = 200$. Thus SBMF is more suited for large datasets with large latent space dimension. Similar speed up patterns are found on the other datasets also.

Table 3. Results Comparison

		$K = 50$		$K = 100$		$K = 200$	
Dataset	Method	RMSE	Time(Hr)	RMSE	Time(Hr)	RMSE	Time(Hr)
Movielens 10m	BPMF	**0.8629**	1.317	**0.8638**	3.517	**0.8651**	22.058
	SBMF-S	0.8655	1.091	0.8667	2.316	0.8654	5.205
	SBMF-P	0.8646	**0.462**	0.8659	**0.990**	0.8657	**2.214**
Movielens 20m	BPMF	**0.7534**	2.683	**0.7513**	6.761	**0.7508**	45.355
	SBMF-S	0.7553	2.364	0.7545	5.073	0.7549	11.378
	SBMF-P	0.7553	**1.142**	0.7545	**2.427**	0.7551	**5.321**
Netflix	BPMF	0.9057	11.739	**0.9021**	28.797	**0.8997**	150.026
	SBMF-S	0.9048	17.973	0.9028	40.287	0.9017	89.809
	SBMF-P	**0.9047**	**7.902**	0.9026	**16.477**	0.9017	**34.934**

4 Related Work

MF [1–6] is widely used in several domains because of performance and scalability. Stochastic gradient descent [2] is the simplest method to solve MF but it often suffers from the problem of overfitting and requires manual tuning of the learning rate and regularization parameters. Thus many Bayesian methods [5, 11, 13] have been developed for MF that automatically select all the model parameters and avoid the problem of overfitting. Variational Bayesian approximation based MF [11] considers a simplified distribution to approximate the posterior. But this method does not scale well on large datasets. Consequently, scalable variational Bayesian methods [12, 13] have been proposed to scale to large datasets. However variational approximation based Bayesian method might give inaccurate results [5] because of its over simplistic assumptions. Thus, Gibbs sampling based MF [5] has been proposed which gives better performance than the variational Bayesian MF counter part.

Since performance of MF depends on the latent dimensionality, several non-parametric MF methods [14–16] have been proposed that set the number of latent factors automatically. Non-negative matrix factorization (NMF) [3] is a variant of MF, which recovers two low rank matrices, each of which is non-negative. Bayesian NMF [6, 17] considers Poisson likelihood and different type of priors and generates a family of MF model based on the prior imposed on the latent factor. Also, in real world the preferences of user changes over time. To incorporate this dynamics into the model, several dynamic MF models [18, 19] have been developed.

5 Conclusion and Future Work

We have proposed the **S**calable **B**ayesian **M**atrix **F**actorization (SBMF), which is a Markov chain Monte Carlo based Gibbs sampling algorithm for matrix factorization and has linear time complexity with respect to the target rank and linear space complexity with respect to the number of non-zero observations.

SBMF gives competitive performance in less time as compared to the baseline method. Experiments on several real world datasets show the effectiveness of SBMF. In future, it would be interesting to extend this method in applications like matrix factorization with side-information, where the time complexity is cubic with respect to the number of features (which can be very large in practice).

6 Acknowledgement

This project was supported by Ericsson India.

References

1. N. Srebro and T. Jaakkola, "Weighted low-rank approximations," in *Proc. of ICML*, pp. 720–727, AAAI Press, 2003.
2. Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, pp. 30–37, Aug. 2009.
3. D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Proc. of NIPS*, pp. 556–562, 2000.
4. R. Salakhutdinov and A. Mnih, "Probabilistic matrix factorization," in *Proc. of NIPS*, 2007.
5. R. Salakhutdinov and A. Mnih, "Bayesian probabilistic matrix factorization using markov chain monte carlo," in *Proc. of ICML*, pp. 880–887, 2008.
6. P. Gopalan, J. Hofman, and D. Blei, "Scalable recommendation with Poisson factorization," *CoRR*, vol. abs/1311.1704, 2013.
7. M.-A. Sato, "Online model selection based on the variational Bayes," *Neural Computation*, vol. 13, no. 7, pp. 1649–1681, 2001.
8. M. J. Beal, "Variational algorithms for approximate Bayesian inference," in *PhD. Thesis, Gatsby Computational Neuroscience Unit, University College London.*, 2003.
9. D. Tzikas, A. Likas, and N. Galatsanos, "The variational approximation for Bayesian inference," *IEEE Signal Processing Magazine*, vol. 25, pp. 131–146, Nov. 2008.
10. M. Hoffman, D. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," *JMLR*, vol. 14, pp. 1303–1347, may 2013.
11. Y. Lim and Y. Teh, "Variational bayesian approach to movie rating prediction," in *Proc. of KDDCup*, 2007.
12. J. Silva and L. Carin, "Active learning for online bayesian matrix factorization," in *Proc. of KDD*, pp. 325–333, 2012.
13. Y. Kim and S. Choi, "Scalable variational Bayesian matrix factorization with side information," in *Proc. of AISTATS*, pp. 493–502, 2014.
14. M. Zhou and L. Carin, "Negative binomial process count and mixture modeling," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 99, no. PrePrints, p. 1, 2013.
15. M. Zhou, L. Hannah, D. B. Dunson, and L. Carin, "Beta-negative binomial process and poisson factor analysis," in *Proc. of AISTATS*, pp. 1462–1471, 2012.
16. M. Xu, J. Zhu, and B. Zhang, "Nonparametric max-margin matrix factorization for collaborative prediction," in *Proc. of NIPS*, pp. 64–72, 2012.

17. P. Gopalan, F. Ruiz, R. Ranganath, and D. Blei, "Bayesian nonparametric Poisson factorization for Recommendation Systems," in *Proc. of AISTATS*, pp. 275–283, 2014.
18. Y. Koren, "Factorization meets the neighborhood: A multifaceted collaborative filtering model," in *Proc. of KDD*, pp. 426–434, 2008.
19. L. Xiong, X. Chen, T. K. Huang, J. Schneider, and J. G. Carbonell, "Temporal collaborative filtering with Bayesian probabilistic tensor factorization.," in *Proc. of SDM*, pp. 211–222, SIAM, 2010.