Temporal Collaborative Filtering with Bayesian Probabilistic Tensor Factorization

Liang Xiong* Xi Chen* Tzu-Kuo Huang* Jeff Schneider† Jaime G. Carbonell‡

Abstract

Real-world relational data are seldom stationary, yet traditional collaborative filtering algorithms generally rely on this assumption. Motivated by our sales prediction problem, we propose a factor-based algorithm that is able to take time into account. By introducing additional factors for time, we formalize this problem as a tensor factorization with a special constraint on the time dimension. Further, we provide a fully Bayesian treatment to avoid tuning parameters and achieve automatic model complexity control. To learn the model we develop an efficient sampling procedure that is capable of analyzing large-scale data sets. This new algorithm, called *Bayesian Probabilistic Tensor Factorization* (BPTF), is evaluated on several real-world problems including sales prediction and movie recommendation. Empirical results demonstrate the superiority of our temporal model.

1 Introduction

Learning from relational data has always been a central topic in the fields of data mining and machine learning. In many real world problems, instead of the attributes of individual entities, we are given only the data about relationships between them. For example, in recommendation systems the data we have are the preference scores of users toward different items. This learning problem has been extensively investigated under the *Collaborative Filtering* framework. Nowadays collaborative filtering plays a vital role in various automatic recommendation systems and has been used in many online applications such as *Amazon.com, eBay,* and *Netflix*.

Successful as they are, one limitation of most existing collaborative filtering algorithms is that they are static models in which relations are assumed to be fixed at different time. However, real relational data is often evolving over time and exhibits strong temporal patterns. To motivate the proposed model, let us consider the following problem. A shoe production company sells many types of shoes to its retailer customers. Now this company wants to predict the orders that will arrive for the ongoing season based on this season's initial orders and historical sales data. Having this prediction, the company can make more informed decisions on the marketing strategy and inventory planning.

The traditional way to treat this kind of problems is using time-series forecasting or statistical regression models. Typical time-series models such as *autoregressive moving average* (ARMA) and *exponential smoothing* [5] use past data to make predictions. But they are not suitable for our problem because each season this fashion company introduces a lot of new products, for which there is no historical data. Regression models are also not appropriate since few product attributes are available due to the complexity of domain knowledge and policy issues. What we have is only the transaction data recording the involved customer, product, and quantity of each order. Moreover, both of these two paradigms cannot exploit the "collaboration" between entities and hence are expected to perform poorly when the data is sparse. For these reasons, we rely on collaborative filtering to make the prediction.

Nevertheless, even if collaborative filtering is able to handle our relational data, traditional static methods are incapable of learning the shift of product designs and customers' preferences, especially considering that we are facing the volatile and fast-moving fashion business. The preference of the market can change from season to season and even within each season. In this case, trying to explain all the data with one fixed global model would be ineffective. On the other hand, if we only use recent data or down-weigh past instances, a lot of useful information would be lost, making the already very sparse data set even worse.

To solve this problem, we propose a factorization based method that is able to model time-evolving relational data. This method is based on probabilistic latent factor models [18, 19]. In addition to the factors that are used to characterize entities, we introduce another set of latent features for each different time period. Intuitively, these additional factors represent the population-level preference of latent features at each

*Machine Learning Department, Carnegie Mellon University
†Robotics Institute, Carnegie Mellon University
‡Language Technology Institute, Carnegie Mellon University

particular time, so that they are able to capture concepts like "high-heeled shoes lost their popularity this fall" or "orders of golf shoes tend to arrive late". A special treatment is made to the time factors to ensure that the evolution of factors is smooth. This model learns the inherent factors for entities using all the available data, as well as adapts these factors to different time periods. It can be formulated as a probabilistic tensor factorization problem, thus is widely applicable to various relational data sets.

One outstanding problem for many relational data is that they are often very sparse. Taking the Netflix Prize[1] data for example, there are $17,770$ movies and $480,189$ users, but only $99,072,112$ training ratings. In matrix terms this means that we are trying to complete a huge matrix with only 1.16% of its entries given. This phenomenon presents two challenges for us. The first one is how to avoid over-fitting, and the second is how to take advantage of this sparsity to accelerate computation. To address the first problem, we extend our approach using Bayesian techniques. By introducing priors on the parameters, we can effectively average over various models and ease the pain of tuning parameters. We call the resulting algorithm *Bayesian Probabilistic Tensor Factorization* (BPTF). And for scalability, we develop an efficient *Markov Chain Monte Carlo* (MCMC) procedure for the learning process so that this algorithm can be scaled to problems like Netflix.

The modeling of temporal effects in collaborative filtering has also been called for in many other problems since the preferences of users are often subject to change in recommendation systems. Remarkably, the latest progress in the *Netflix Prize* contest is attributed to a temporal model [12]. The winner identifies strong temporal patterns in the data, and exploits them to achieve a significant improvement leading to the best performance attained by a single algorithm. In our experiments we applied our BPTF model to the sales prediction problem as well as two movie recommendation data sets. The empirical results show that using the temporal modeling, consistent improvement of prediction accuracy can be achieved over static methods at the cost of few additional parameters.

The rest of this paper is organized as follows. First we introduce some preliminaries about factorization based collaborative filtering. Then in section 3 we describe the proposed model and its learning procedure. Some related works are discussed in section 4. Section 5 presents the empirical performance and efficiency of our method. Finally we make our conclusions.

[1] http://www.netflixprize.com/

2 Preliminaries

First we introduce some symbols and settings. Suppose we are dealing with pair-wise relationships between two types of entities $\{u_i\}$ and $\{v_j\}$, which we shall call "user" and "item" respectively, and for some u_i and v_j we observe a relational score R_{ij} which we shall call as "rating". Thus each instance of the data is a tuple (u_i, v_j, R_{ij}), which in the movie recommendation case means that user u_i gives rating R_{ij} to movie v_j. Assuming that there are N users and M items, these tuples are usually organized into a sparse matrix $\mathbf{R} \in \mathbb{R}^{N\times M}$ using $(u_i,\, v_j)$ as the index and R_{ij} as the entry value.

Typical collaborative filtering algorithms can be categorized into two classes: neighborhood methods and factorization methods. Generally factor-based algorithms are considered more effective than those based on neighborhood. But these two class are often complementary and the best performance is often obtained by blending them [2]. A practical survey of this field can be found in [11].

One representative factor-based method for collaborative filtering is *Probabilistic Matrix Factorization* (PMF) [18]. PMF assigns a D-dimensional latent feature vector for each user and movie, denoted as $U_i, V_j \in \mathbb{R}^D$, and model each rating as the inner-product of corresponding latent features, *i.e.* $R_{ij} \approx U_i' V_j$ where U_i' is the transpose of U_i. Formally, the following conditional distribution is assumed:

$$p(\mathbf{R}|\mathbf{U},\mathbf{V},\alpha) = \prod_{i=1}^{N}\prod_{j=1}^{M}\left[\mathcal{N}\left(R_{ij}|U_i'V_j,\alpha^{-1}\right)\right]^{I_{ij}}, \tag{2.1}$$

where $\{U_i\}, \{V_j\}$ consist the columns of $\mathbf{U} \in \mathbb{R}^{D\times N}$ and $\mathbf{V} \in \mathbb{R}^{D\times M}$, $\mathcal{N}(\cdot|\cdot,\cdot)$ denotes the Gaussian distribution, α is the observation precision, and I_{ij} is the indicator that R_{ij} has been observed. Zero-mean Gaussian prior are imposed on U_i and V_j to control model complexity.

This model can be learned by estimating the value of $\mathbf{U}$ and $\mathbf{V}$ using maximum likelihood. It turns out that this learning procedure actually corresponds to the following weighted regularized matrix factorization:

$$\begin{aligned}\mathbf{U},\mathbf{V} = \arg\min_{\mathbf{U},\mathbf{V}} &\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}\left(R_{ij} - U_i'V_j\right)^2 \\ &+\frac{\lambda_U}{2}\sum_{i}^{N}\|U_i\|^2 + \frac{\lambda_V}{2}\sum_{j}^{N}\|V_j\|^2.\end{aligned} \tag{2.2}$$

These formulations reflect the basic ideas of factorization based collaborative filtering.

The above optimization can be done efficiently using gradient descent. This model is very successful

in the Netflix Prize contest in terms of speed-accuracy trade-off. The drawback though is that it requires fine tuning of both the model and the training procedure to predict accurately and avoid over-fitting. This process is computationally expensive on large data sets.

In the next section we first extend PMF to the tensor version so that it can take time into account, with the time factors being specially treated. We then provide a Bayesian treatment to avoid fine parameter tuning and achieve model averaging automatically.

3 Proposed Methods

We present the proposed method in two parts. First we extend PMF to *tensor factorization* to model temporal relational data, and formulate a *maximum a posteriori* (MAP) scheme for estimating the factors. Then we apply a fully Bayesian treatment to deal with the tuning of prior parameters and derive an almost *parameter-free* probabilistic tensor factorization algorithm. Finally an efficient learning procedure is developed.

3.1 Probabilistic Tensor Factorization for Temporal Relational Data In PMF each rating is mainly determined by the inner product of a user feature vector and an item feature vector. To model their time-evolving behavior, we make use of the tensor notation. We can denote a rating as R_{ij}^k where i, j index users and items as before, and k indexes the *time slice* in which the rating was given. Then similar to the static case, we can organize these ratings into a *three-dimensional tensor* $\mathbf{R} \in \mathbb{R}^{N \times M \times K}$, whose three dimensions correspond to user, item, and time slices with sizes N, M, and K, respectively.

Extending the idea of PMF, we assume that each entry R_{ij}^k can be expressed as the inner-product of three D-dimensional vectors:

$$R_{ij}^k \approx < U_i, V_j, T_k > \equiv \sum_{d=1}^{D} U_{di} V_{dj} T_{dk}, \tag{3.3}$$

where U_i, V_j are for uses and items while T_k is the additional latent feature vector (or factors) for the k-th time slice. Using matrix representations $\mathbf{U} \equiv [U_1\ U_2\ \cdots\ U_N], \mathbf{V} \equiv [V_1\ V_2\ \cdots\ V_M]$, and $\mathbf{T} \equiv [T_1\ T_2\ \cdots\ T_K]$, we can also express Eq. (3.3) as a *three-way tensor factorization* of $\mathbf{R}$:

$$\mathbf{R} \approx \sum_{d=1}^{D} U_{d,:} \circ V_{d,:} \circ T_{d,:}, \tag{3.4}$$

where $U_{d,:}, V_{d,:}$ and $T_{d,:}$ represent the d-th rows of $\mathbf{U}, \mathbf{V}$ and $\mathbf{T}$, and $\circ$ denotes the vector outer product. This is an instance of the CANDECOMP/PARAFAC (CP) decomposition [10], for which a graphical illustration is in Figure 1.

An interpretation of the factorization (3.3) is that a rating depends not only on how similar a user's preferences and an item's features are (as in PMF), but also on how much these preferences/features match with the "current trend" as reflected in the time feature vectors. For instance, if a user likes green shoes but the overall trend of this year is that few people wears them on the street, then this user is probably not going to buy them neither.

To account for the randomness in ratings, we consider the following probabilistic model:

$$R_{ij}^k | \mathbf{U}, \mathbf{V}, \mathbf{T} \sim \mathcal{N}(< U_i, V_j, T_k >, \alpha^{-1}), \tag{3.5}$$

i.e, the conditional distribution of R_{ij}^k given $\mathbf{U}, \mathbf{V}$, and $\mathbf{T}$ is a Gaussian distribution with mean $< U_i, V_j, T_k >$ and precision α. Note that if T_k is an all-one vector then this model is equivalent to PMF. Since many entries in $\mathbf{R}$ are missing, estimation based on the model (3.5) may overfit the observed entries and fail to predict the missing entries well. To deal with this issue, we follow the usual Bayesian scheme by placing prior distributions on $\mathbf{U}, \mathbf{V}$, and $\mathbf{T}$. Specifically we impose zero-mean, independent Gaussian priors on user and feature vectors:

$$\begin{aligned} U_i &\sim \mathcal{N}(0, \sigma_U^2 \mathbf{I}), \qquad i = 1 \ldots N, \\ V_j &\sim \mathcal{N}(0, \sigma_V^2 \mathbf{I}), \qquad j = 1 \ldots M, \end{aligned} \tag{3.6}$$

where $\mathbf{I}$ is the D-by-D identity matrix.

As for the time factors, since they account for the evolution of global trends, a reasonable prior belief is that they change smoothly over time. Therefore we further assume that each time feature vector depends only on its immediate predecessor, and use the following conditional prior for $\mathbf{T}$:

$$T_k \sim \mathcal{N}(T_{k-1}, \sigma_{dT}^2 \mathbf{I}), \qquad k = 1, \ldots, K. \tag{3.7}$$

For the initial time feature vector T_0, we assume

$$T_0 \sim \mathcal{N}(\mu_T, \sigma_0^2 \mathbf{I}), \tag{3.8}$$

where μ_T is D-by-1 column vector. We call this model the *Probabilistic Tensor Factorization* (PTF).

Having the observational model (3.5) and the priors, we may estimate the latent features $\mathbf{U}, \mathbf{V}$, and $\mathbf{T}$ by maximizing the logarithm of the posterior distribution, which takes the following form assuming ratings are

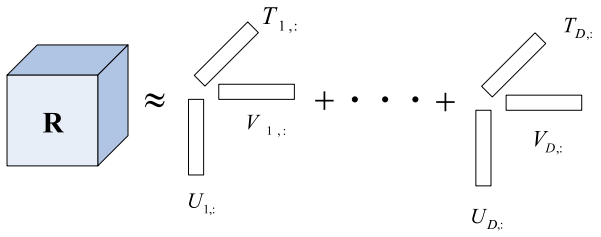

Figure 1: CP decomposition of a three-way tensor $\mathbf{R}$

made independently conditioned on latent factors:

$$\begin{aligned}
&\log p(\mathbf{U},\mathbf{V},\mathbf{T},T_0|\mathbf{R}) \\
\propto\ &\log p(\mathbf{R}|\mathbf{U},\mathbf{V},\mathbf{T},T_0)+\log p(\mathbf{U},\mathbf{V},\mathbf{T},T_0) \\
=\ &\sum_{k=1}^{K}\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}^k \log p(R_{ij}^k|U_i,V_j,T_k)+\sum_{i=1}^{N}\log p(U_i) \\
&+\sum_{j=1}^{M}\log p(V_j)+\sum_{k=1}^{K}\log p(T_k|T_{k-1})+\log p(T_0) \\
=\ &-\sum_{k=1}^{K}\sum_{i=1}^{N}\sum_{j=1}^{M}\frac{I_{ij}^k(R_{ij}^k-<U_i,V_j,T_k>)^2}{2\alpha^{-1}}+\frac{(\#\text{nz})\log\alpha}{2} \\
&-\sum_{i=1}^{N}\frac{||U_i||^2}{2\sigma_U^2}-N\log\sigma_U-\sum_{j=1}^{M}\frac{||V_j||^2}{2\sigma_V^2}-M\log\sigma_V \\
&-\sum_{k=1}^{K}\frac{||T_k-T_{k-1}||^2}{2\sigma_{dT}^2}-K\log\sigma_{dT}-\frac{||T_0-\mu_T||^2}{2\sigma_0^2} \\
&-\log\sigma_0+C,
\end{aligned}$$

where I_{ij}^k is one if R_{ij}^k is available and zero otherwise, #nz is the total number of ratings, and C is a constant. Under fixed values of $\alpha,\sigma_U,\sigma_V,\sigma_{dT}$, σ_0 and μ_T, which are usually referred to as hyper-parameters, maximizing the log-posterior with respect to $\mathbf{U},\mathbf{V},\mathbf{T}$, and T_0 is equivalent to minimizing the following regularized sum of squared errors:

$$\begin{aligned}
&\sum_{k=1}^{K}\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}^k(R_{ij}^k-<U_i,V_j,T_k>)^2+\sum_{i=1}^{N}\frac{\lambda_U||U_i||^2}{2}+ \\
&\sum_{j=1}^{M}\frac{\lambda_V||V_j||^2}{2}+\sum_{k=1}^{K}\frac{\lambda_{dT}||T_k-T_{k-1}||^2}{2}+\frac{\lambda_0||T_0-\mu_T||^2}{2},
\end{aligned} \tag{3.9}$$

where $\lambda_U=(\alpha\sigma_U^2)^{-1},\lambda_V=(\alpha\sigma_V^2)^{-1},\lambda_{dT}=(\alpha\sigma_{dT}^2)^{-1}$, and $\lambda_0=(\alpha\sigma_0^2)^{-1}$.

This objective function (3.9) is non-convex, and we may only be able to find a local minimum. To optimize it, common choices include *stochastic gradient descent* and *block coordinate descent*, both of which update the latent feature vectors iteratively. After the MAP estimates $\mathbf{U}^*,\mathbf{V}^*$, and $\mathbf{T}^*$ are obtained, we may predict an unobserved rating $\hat{R}_{ij}^k$ by the distribution (3.5) or simply $<U_i^*,V_j^*,T_k^*>$.

One issue with the aforementioned approach is the tuning of the hyper-parameters $\alpha,\sigma_U,\sigma_V,\sigma_{dT}$, σ_0 and μ_T. Since there are quite a few, the usual approach of hyper-parameter selection, such as cross-validation, is infeasible even for a modest problem size. We thus propose in the next section a fully Bayesian treatment to average out the hyper-parameters in the model, leading to an almost *parameter-free* estimation procedure.

3.2 Bayesian Probabilistic Tensor Factorization (BPTF) The performance of PTF is tied to the careful tuning of the hyper-parameters when model parameters are estimated by maximizing the posterior probability, as pointed out in [18]. Such a point estimate as obtained by MAP is often vulnerable to over-fitting when hyper-parameters are not properly tuned, and is more likely so when the data is large and sparse.

An alternative estimation scheme that may help alleviate over-fitting is a fully Bayesian treatment, which integrates out all model parameters and hyper-parameters, arriving at a *predictive distribution* of future observations given observed data. Because this predictive distribution is obtained by averaging all models in the model space specified by the priors, it is less likely to over-fit a given set of observations.

However, when integrating over parameters one often cannot obtain an analytical solution, thus we will need to apply sampling-based approximation methods, such as Markov Chain Monte Carlo (MCMC). For large-scale problems, sampling-based methods are usually not preferred due to their computational cost and convergence-related issues. Nevertheless, [19] devises an MCMC procedure for PMF that can run efficiently on large data sets like Netflix. The main trick is choosing proper distributions for hyper-parameters so that sampling can be carried out efficiently.

Inspired by the work of [19], we present in the following a fully Bayesian treatment to the Probabilistic Tensor Factorization model proposed in Section 3.1. We refer to the resulting method as BPTF for *Bayesian Probabilistic Tensor Factorization*.

3.2.1 Model Specification for BPTF A graphical overview of our entire model is in Figure 2, and each component is described below. The model for generating ratings is the same as Eq. (3.5):

$$R_{ij}^k|\mathbf{U},\mathbf{V},\mathbf{T}\sim\mathcal{N}(<U_i,V_j,T_k>,\alpha^{-1}). \tag{3.10}$$

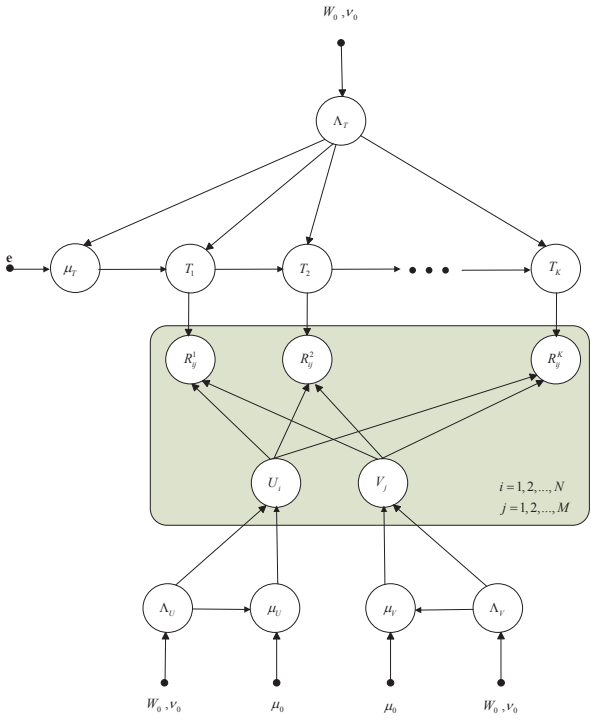

Figure 2: The graphical model for BPTF

As before, the prior distributions for the user and the item feature vectors are assumed to be Gaussian, but the mean and the precision matrix (inverse of the covariance matrix) may take arbitrary values:

$$U_i \sim \mathcal{N}(\mu_U, \Lambda_U^{-1}), \qquad i = 1 \ldots N, \tag{3.11}$$

$$V_j \sim \mathcal{N}(\mu_V, \Lambda_V^{-1}), \qquad j = 1 \ldots M. \tag{3.12}$$

For the time feature vectors, we make the same Markovian assumption as in Section 3.1 and consider the priors:

$$T_k \sim \mathcal{N}(T_{k-1}, \Lambda_T^{-1}), \quad k = 2, \ldots K, \tag{3.13}$$

$$T_1 \sim \mathcal{N}(\mu_T, \Lambda_T^{-1}). \tag{3.14}$$

The key ingredient of our fully Bayesian treatment is to view the hyper-parameters $\alpha, \Theta_U \equiv \{\mu_U, \Lambda_U\}, \Theta_V \equiv \{\mu_V, \Lambda_V\}$, and $\Theta_T \equiv \{\mu_T, \Lambda_T\}$ also as *random variables*, leading to a *predictive distribution* for an unobserved rating $\hat{R}_{ij}^k$,

$$p(\hat{R}_{ij}^k|\mathbf{R}) = \int p(\hat{R}_{ij}^k|U_i, V_j, T_k, \alpha) p(\mathbf{U}, \mathbf{V}, \mathbf{T}, \alpha, \Theta_U, \Theta_V, \Theta_T|\mathbf{R}) \, \mathrm{d}\{\mathbf{U}, \mathbf{V}, \mathbf{T}, \alpha, \Theta_U, \Theta_V, \Theta_T\} \tag{3.15}$$

that integrates over both the model parameters and the hyper-parameters, as opposed to the prediction scheme in Section 3.1 that simply plugs the MAP estimates into Eq. (3.5).

We then need to choose prior distributions for the hyper-parameters (the so-called *hyper-priors*). For the Gaussian parameters, we choose the conjugate distributions as priors that facilitate subsequent computations:

$$\begin{aligned}
p(\alpha) \quad &= \mathcal{W}(\alpha|\tilde{W}_0, \tilde{\nu}_0), \\
p(\Theta_U) &= p(\mu_U|\Lambda_U) p(\Lambda_U) = \mathcal{N}(\mu_0, (\beta_0 \Lambda_U)^{-1}) \mathcal{W}(\Lambda_U|W_0, \nu_0), \\
p(\Theta_V) &= p(\mu_V|\Lambda_V) p(\Lambda_V) = \mathcal{N}(\mu_0, (\beta_0 \Lambda_V)^{-1}) \mathcal{W}(\Lambda_V|W_0, \nu_0), \\
p(\Theta_T) &= p(\mu_T|\Lambda_T) p(\Lambda_T) = \mathcal{N}(\rho_0, (\beta_0 \Lambda_T)^{-1}) \mathcal{W}(\Lambda_T|W_0, \nu_0).
\end{aligned}$$

Here $\mathcal{W}$ is the Wishart distribution[2] of a $D \times D$ random matrix Λ with ν_0 degrees of freedom and a $D \times D$ scale matrix W_0:

$$\mathcal{W}(\Lambda|W_0, \nu_0) = \frac{|\Lambda|^{(\nu_0 - D - 1)/2}}{C} \exp(-\frac{\mathrm{Tr}(W_0^{-1}\Lambda)}{2}), \tag{3.16}$$

where C is a normalizing constant. There are several parameters in the hyper-priors: $\mu_0, \rho_0, \beta_0, W_0, \nu_0, \tilde{W}_0$, and $\tilde{\nu}_0$; These parameters should reflect our prior knowledge about the specific problem and are treated as constants during training. In fact, Bayesian learning is able to adjust them according to the training data, and varying their values (within in a reasonably large range) has little impact on the final prediction, as often observed in Bayesian estimation procedures.

3.2.2 Learning by Markov Chain Monte Carlo

The predictive distribution (3.15) involves a multi-dimensional integral, which cannot be computed analytically. We thus resort to numerical approximation techniques. The main idea is to view Eq. (3.15) as an *expectation* of $p(\hat{R}_{ij}^k|U_i, V_j, T_k, \alpha)$ over the posterior distribution $p(\mathbf{U}, \mathbf{V}, \mathbf{T}, \alpha, \Theta_U, \Theta_V, \Theta_T|\mathbf{R})$, and approximate the expectation by an average on samples drawn from the posterior distribution. Since the posterior is too complex to directly sample from, we apply a widely-used indirect sampling technique, Markov Chain Monte Carlo (MCMC) [14, 13, 8]. The method works by drawing a sequence of samples from some *proposal distribution* such that each sample depends only on the previous one, thus forming a Markov chain. When the sampling step obeys certain properties, the most notably being *detailed balance*, the chain converges to the desired distribution. Then we collect a number of samples and approximate the integral in Eq. (3.15) by

$$p(\hat{R}_{ij}^k|\mathbf{R}) \approx \sum_{l=1}^{L} p(\hat{R}_{ij}^k|U_i^{(l)}, V_j^{(l)}, T_k^{(l)}, \alpha^{(l)}), \tag{3.17}$$

[2]The Wishart distribution is usually used as the *conjugate prior* for the precision matrix in a Gaussian distribution.

where L denotes the number of samples collected and $U_i^{(l)}, V_j^{(l)}, T_k^{(l)}$, and $\alpha^{(l)}$ are from the lth sample. A detailed treatment of MCMC can be found in [17].

There are quite a few different flavors of MCMC. Here we choose to use the Gibbs sampling paradigm [7]. In Gibbs sampling, the target random variables are decomposed into several disjoint subsets or blocks, and at each iteration a block of random variables is sampled while all the others are fixed. All the blocks are iteratively sampled until convergence. Such a scheme is very similar to the *nonlinear Gauss-Seidel method* (Chapter 2.7, [3]) for nonlinear optimization, which optimizes iteratively over blocks of variables.

As indicated by its parametrization, our target distribution $p(\mathbf{U}, \mathbf{V}, \mathbf{T}, \alpha, \Theta_U, \Theta_V, \Theta_T|\mathbf{R})$ bears an inherent block structure of random variables. In Appendix A we show that such a block structure, together with our choice of model components in Section 3.2.1, gives rise to conditional distributions that are easy to sample from, leading to a simple and efficient Gibbs sampling procedure as outlined in Algorithm 3.1. It has two notable features: 1) the only distributions that need to be sampled are multivariate Gaussian distributions and the Wishart distribution; 2) individual user feature vectors can be sampled in parallel, so can individual item vectors.

3.3 Scalability and Practical Issues In our implementation, the PTF model is optimized using *alternating least squares*, which is actually a block coordinate descent algorithm that optimizes one user or one item at each time. And the learning process of the BPTF model is implemented using Gibbs sampling as described in algorithm 3.1. Both of them are efficient and scalable for large data sets.

Let #nz be the number of observed relations in the training data. For each iteration, the time complexity for both PTF and BPTF is $O(\#\text{nz} \times D^2 + (N + M + K) \times D^3)$. In typical cases, the term $(\#\text{nz} \times D^2)$ is much larger than the rest so we can consider this complexity to be linear with respect to the number of observations. For the choice of D, in our experience using tens of latent features usually achieves a good balance between speed and accuracy. Inevitably, the running time of BPTF is slower than the non-Bayesian PMF, which has a complexity of $O(\#\text{nz} \times D)$ for each iteration using stochastic gradient descent. But using PMF involves a model selection problem. Typically parameters λ_U and λ_V have to be tuned along with the early-stopping strategy. This process can be prohibitive for large data sets. On the other hand, BPTF eliminates the existence of hyper-parameters by introducing priors for them. Therefore, we can set the priors according to our knowledge and let the algorithm adapt them to the data. Empirically, impressive results can be obtained without any tuning.

When using MCMC, a typical issue is the convergence of sampling. Theoretically, the results generated are only accurate when the chain has reached its equilibrium. This however would usually take a long time and there is no effective way to diagnose the convergence. To alleviate this, we use the MAP result from PMF to initialize the sampling. Then the chain usually converges within a few hundreds samples from our experience. Moreover, we found that the accuracy increases mono-

ALGORITHM 3.1. Gibbs sampling for BPTF

Initialize model parameters $\{\mathbf{U}^{(1)}, \mathbf{V}^{(1)}, \mathbf{T}^{(1)}\}$.
For l=1, ..., L,

- Sample the hyperparameters according to (A.2), (A.3), (A.4) and (A.5), respectively:

$$\begin{aligned}
\alpha^{(l)} &\sim p(\alpha^{(l)}|\mathbf{U}^{(l)}, \mathbf{V}^{(l)}, \mathbf{T}^{(l)}, \mathbf{R}), \\
\Theta_U^{(l)} &\sim p(\Theta_U^{(l)}|\mathbf{U}^{(l)}), \\
\Theta_V^{(l)} &\sim p(\Theta_V^{(l)}|\mathbf{V}^{(l)}), \\
\Theta_T^{(l)} &\sim p(\Theta_T^{(l)}|\mathbf{T}^{(l)}).
\end{aligned}$$

- For $i = 1, \ldots, N$, sample the user features (in parallel) according to (A.6):

$$U_i^{(l+1)} \sim p(U_i|\mathbf{V}^{(l)}, \mathbf{T}^{(l)}, {\Theta_U}^{(l)}, \alpha^{(l)}, \mathbf{R}).$$

- For $j = 1, \ldots, M$, sample the item features (in parallel) according to (A.7):

$$V_j^{(l+1)} \sim p(V_j|\mathbf{U}^{(l+1)}, \mathbf{T}^{(l)}, {\Theta_V}^{(l)}, \alpha^{(l)}, \mathbf{R}).$$

- Sample the time features according to (A.8):
  For $k = 1$,

$$T_1^{(l+1)} \sim p(T_1|\mathbf{U}^{(l+1)}, \mathbf{V}^{(l+1)}, T_2^{(l)}, {\Theta_T}^{(l)}, \alpha^{(l)}, \mathbf{R}).$$

  For $k = 2, \ldots, K-1$,

$$T_k^{(l+1)} \sim p(T_k|\mathbf{U}^{(l+1)}, \mathbf{V}^{(l+1)}, T_{k-1}^{(l+1)}, T_{k+1}^{(l)}, {\Theta_T}^{(l)}, \alpha^{(l)}, \mathbf{R}).$$

  For $k = K$,

$$T_K^{(l+1)} \sim p(T_K|\mathbf{U}^{(l+1)}, \mathbf{V}^{(l+1)}, T_{K-1}^{(l+1)}, {\Theta_T}^{(l)}, \alpha^{(l)}, \mathbf{R}).$$

tonically as the number of samples increases. Therefore in practice we can just monitor the performance on validation sets and stop sampling when the improvement from more samples is diminishing.

4 Related Work

There is a lot of work on factorization methods for collaborative filtering, among which the most well-known one is *Singular Value Decomposition* (SVD), which is also called *Latent Semantic Analysis* (LSA) in the language and information retrieval communities. Based on the LSA, *probabilistic LSA* [9] was proposed to provide the probabilistic modeling, and further *latent Dirichlet allocation* (LDA) [4] provides a Bayesian treatment of the generative process. Along another direction, PMF improves the SVD by introducing regularization and handling missing values, which became one of the most effective algorithms in the Netflix Prize.

Bayesian PMF (BPMF) [19] provides a Bayesian treatment for PMF to achieve automatic model complexity control. It demonstrates the effectiveness and efficiency of Bayesian methods and MCMC in real-world large-scale data mining tasks, and inspired our research. However, as mentioned before, BPMF is still a static model that cannot handle evolving data. BPTF enhance it by adapting the latent features to include the time information. From the algorithmic perspective, BPTF extends BPMF so that it can deal with multi-dimensional tensor data and the time dimension is specially taken care of. Although BPTF gives more flexibility over BPMF, the increase of parameters is negligible considering that the number of time slices are often much smaller than the number of entities. Another difference is that BPMF leaves the observation precision α as a tuning parameter while our Bayesian treatment covers all the parameters. There are also other tensor factorizations such as Multi-HDP [15], Probabilistic Non-negative Tensor Factorization [21], and Probabilistic polyadic factorization [**?**]. Yet they are neither designed for prediction purpose nor modeling temporal effects.

Temporal modeling has been largely neglected in the collaborative filtering community until Koren [12] proposed their award winning algorithm timeSVD++. The timeSVD++ method assumes that the latent features consist of some components that are evolving over time and some others that are dedicated bias for each user at each specific time point. This model can effectively capture local changes of user preference which the authors claim to be vital for improving the performance. On the other hand, BPTF tries to capture the overall effect of time that are shared among all users and items. For our sales prediction purpose we argue that modeling the evolution of the overall market would be more effective since the behavior of retailers are not very localized and the data is very sparse.

Real data sets are rarely stationary. Recently, several algorithms aimed at learning the evolution of relational data were proposed. Tong *et al.* [22] proposed an online algorithm to efficiently compute the proximity in a series of evolving bipartite graphs. Ahmed and Xing [1] added dynamic components to the LDA to track the evolution of topics in a text corpus. Sarkar *et al.* [20] considers the dynamic graph embedding problem and uses *Kalman Filter* to track the embedding coordinates through time. All these works reveal the dynamic nature of various problems.

5 Experiments

We conducted several experiments on three real world data sets to test the effectiveness of BPTF. In these data sets, a timestamp is available for each relational instance, which can thus be denoted by the tuple $(u_i, v_j, t_k, R^k_{ij})$. The experimental domains include sales prediction and online movie recommendation.

For comparison, we also implemented and report the performance of PMF and BPMF. When training the non-temporal models, the time information is dropped so the actual tuple used is (u_i, v_j, R^k_{ij}). For PMF model, plain stochastic gradient descent with a fixed learning rate ($lrate$) is adopted for training, and its parameters are obtained by hand tuning to achieve the best accuracy. For BPMF and BPTF, Gibbs sampling is used for training and the results from PMF are used to initialize the sampling. Similar to [19], parameters for Bayesian methods are set according to prior knowledge without tuning. Unless indicated otherwise, parameters used for priors are $\mu_0 = 0, \nu_0 = D, \beta_0 = 1, W_0 = \mathbf{I}, \rho_0 = \mathbf{e}, \tilde{\nu}_0 = 1$, where $\mathbf{e}$ is an $D \times 1$ column vector of 1s.

The algorithms are implemented in MATLAB with some embedded C functions.

5.1 Sales Prediction In this section we evaluate the performance of BPTF on a sales prediction task for ECCO®, a shoe company selling thousands of kinds of shoes to thousands of retailer customers from all over the world. For the consistency of expression we still use "user" to represent "customer" and "item" to represent ECCO's product: shoes.

ECCO sells its shoes in two seasons each year. Here we use "2008.1" to denote the spring season of 2008 and "2006.2" for the fall season of 2006. For each season there is a period for accepting orders. Suppose we are in the middle of current ordering period, our problem is: in the rest of this season, how many orders of an item can be expected from a particular user?

The data we have is only the existing sales record. No attributes for the items or users are available. As mentioned in section 1, this is a relational data set characterized by changing preferences and the fast emergence and disappearance of entities. On average we have thousands of items and users with only 2% of the possible relations observed. Moreover, in each season $75-80\%$ of the items and around 20% of the users are new arrivals compared to the last corresponding season. All these characteristics render it a particular challenging problem for collaborative filtering.

The data specification is as follows. We have the sales record from years 2005 to 2008 so there are 4 spring seasons and 4 fall seasons, which are handled separately. For each season, we select a week as the cut-off point so that orders before this week will be used for training and the rest are for testing. For example, if we want to predict for orders of season 2008.1 after week 40 of 2007 (the cut-off point), then the training data will be orders in seasons {2005.1, 2006.1, 2007.1, 2008.1} that happened before the cut-off and the testing data are orders of 2008.1 after the cut-off. We use a single cut-off point for all spring seasons and another one for fall seasons. The resulting test set contains $15-20\%$ of the orders. Note that this choice is arbitrary in the sense that the progress of the sales vary from season to season. We measure the performance of algorithms using *mean absolute error* (MAE) for each order since it is the most relevant quantity for ECCO.

We observed that the within-season variability of data is much larger than the cross-season one. This means that trends like "Customers tend to order formal shoes early and golf shoes late" are strong. Therefore, we assign the timestamp of each order according to the cut-off week so that the latent factors can evolve within seasons. Concretely, every season is divided into *early season* and *late season* by the cut-off week. Then we have two time slices and the orders are assigned accordingly. Note that now the data are not grouped by seasons, and all the test data are in the *late season* slice.

We test the performance of three algorithms on all the seasons except 2005.1 and 2005.2 since they do not have previous seasons. The parameters are $\lambda_U = \lambda_V = 0.1, lrate = 1 \times 10^{-5}$ for PMF, $\alpha = 0.04$ for BPMF, and $\tilde{W}_0 = 0.04$ for BPTF. BPMF and BPTF both use the same initialization from PMF. 50 samples are generated in sampling when the accuracy stabilizes.

The prediction accuracies are reported in Figure 3. We conclude that our prediction has an average error of 20 pairs for each order, and the accuracy for spring seasons are much lower than fall. For all the seasons BPTF consistently outperforms the static methods by a fairly large margin. Note that PMF appears better than BPMF here. The reason may be that for this moderately sized data set we are able fine tune the parameters of PMF to gain the best results, while for BPMF (and also BPTF) we assign the parameters by prior knowledge. The results verify that we can enhance the prediction by modeling the temporal effects of data using BPTF.

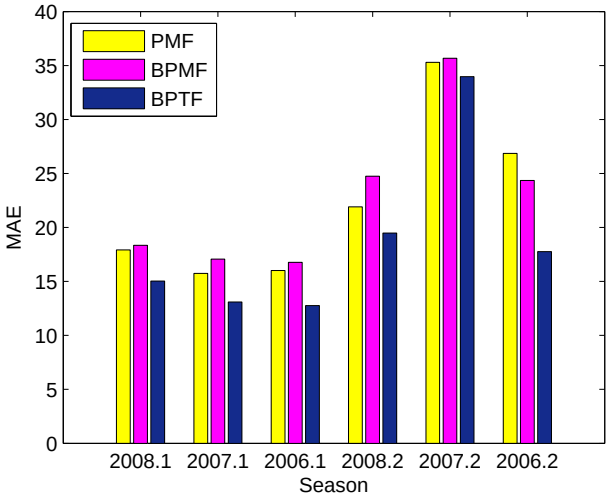

Figure 3: Performance comparison of PMF, BPMF, and BPTF on 6 seasons of ECCO data. BPTF outperforms others by a large margin. See text for details.

5.2 Movie Rating Prediction To make the comparison more transparent, we also did experiments on benchmark movie rating problems: Netflix[3] and MovieLens[4]. These large-scale data sets consist of users' ratings to various movies on a 5-star scale, and our task is to predict the rating for new user-movie pairs.

To measure the accuracy, we adopt the *root mean squared error* (RMSE) criterion as commonly used in collaborative filtering literature and the Netflix Prize. For all models, raw user ratings are used as the input. Prediction results are clipped to fit between $[1, 5]$.

5.2.1 Netflix The Netflix data set contains $100,480,507$ ratings from $N = 480,189$ users to $M = 17,770$ movies between 1999 and 2005. Among these ratings, $1,408,395$ are selected uniformly over the users as the *probe* set for validation. Time information is provided in days. The ratio of observed ratings to all entries of the rating matrix is 1.16%. As a baseline, the test score of Netflix's *Cinematch* system is RMSE

[3] http://archive.ics.uci.edu/ml/datasets/Netflix+Prize
[4] http://www.grouplens.org/node/73

	PMF	BPMF	BPTF
RMSE	0.9166	0.9083	0.9044

Table 1: RMSE of PMF, BPMF and BPTF on Netflix data.

$= 0.9514$.

Basically the timestamps we used for BPTF correspond to calendar months. However, since the ratings in the early months are much more scarce than that in the later months, we aggregated several earlier months together so that every time slice contains an approximately equal number of ratings. In practice we found that in a fairly large range, the slicing of time does not affect the performance much. In the end, we have 27 time slices for the entire data set.

Following the settings in the BPMF paper [19], we use $D = 30$ latent features to model each entity and set $\lambda_U = \lambda_V = 0.015, lrate = 0.001$ for PMF, $\alpha = 2$ for BPMF, and $\tilde{W}_0 = 2$ for BPTF. These parameters for Bayesian methods are set as constant based on prior knowledge and not tuned for best accuracy. 100 samples are used to generate the final prediction.

The prediction accuracies of PMF, BPMF, and BPTF on the probe set are presented in Table 1. Figure 4 shows the change of accuracies as the number of sample increases. BPMF shows a large improvement over its non-Bayesian ancestor PMF, and BPTF further provides a steady increment in accuracy. However, BPTF does not beat the RMSE $= 0.8891$ result of 20-dimensional timeSVD++ (quoted from their paper), which is the state-of-the-art temporal model for the Netflix. As pointed out by the authors of timeSVD++, the most important trait of the Netflix data is that there are many local changes of preference which could just affect one user in one day. BPTF on the other hand aims at learning the global evolution thus cannot capture these changes. However, modeling the global changes still gives us improved performance. Another difference is that BPTF has almost no parameters to tune, while timeSVD++ still has several of them that need to be set by cross-validation.

To generate one sample, BPTF with $D = 30$ latent features took about 9 minutes using about 5GB RAM. For comparison, BPMF uses 6 minutes for one sample. We ran our experiments in a single-threaded MATLAB process on a 2.4 GHz AMD Opteron CPU with 64 GB RAM. We did not use the parallel implementation because it involves distributing a large amount of data and the computational model provided by MATLAB does not handle it well. However, since each user and movie latent feature vector can be sampled independently, we believe that on more sophisticated platforms, BPTF can work nicely with *MapReduce*-style parallel processing.

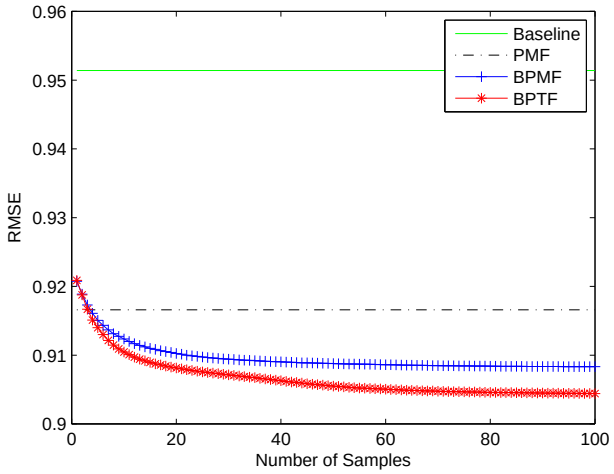

Figure 4: Convergence curves of BPTF and BPMF on the full Netflix data. As the number of samples increase, the RMSE of Bayesian methods drop monotonically. The RMSE of the Netflix's baseline and PMF are also presented.

We also did a group of experiments on a subset of the Netflix data constructed by randomly selecting 20% of the users and 20% of the movies. It consists of $N = 95,992$ users, $M = 3,565$ movies, and $4,167,600$ ratings. This subset is further divided into training and testing sets by randomly selecting 10 ratings (or 1/3 of the total ratings, whichever is smaller) from each user as the testing set. This sampling strategy is similar to the way that the Netflix Prize did it. Finally the new data set contains about 4% of the original data set and is thus suitable for detailed experimental analysis. In the training process, parameters are $\lambda_U = \lambda_V = 0.03, lrate = 0.001$ for PMF, and for Bayesian methods the same parameters as for full data are adopted.

Firstly, we investigate the performance of algorithms as the number of factors varies. For dimensions 10, 20, 50, and 100, the curves of convergence are shown in Figure 5. The RMSE steadily decreases as the number of factors increase, and no over-fitting is observed. When using 100 factors, there are on average two parameters for a single rating. This clearly shows the effect of model averaging using Bayesian technique. Also by comparing the curves of BPTF and BPMF, we see that BPTF with 20 factors performs similarly to BPMF with 100 factors. This demonstrates the advantage of temporal modeling considering that the number of parameters in BPMF is about 5 times more than BPTF.

We further examine the significance of the improvement of BPTF over the BPMF by repeating the prediction tasks 20 times using different random test sets.

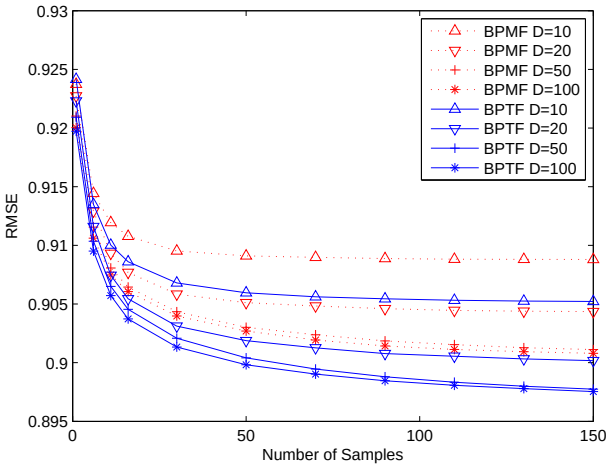

Figure 5: Convergence curves of BPMF and BPTF with different number of factors. The accuracy increases when more factors are used, and no over-fitting is observed. Also, BPTF with 20 factors achieves similar performance as BPMF with 100 factors.

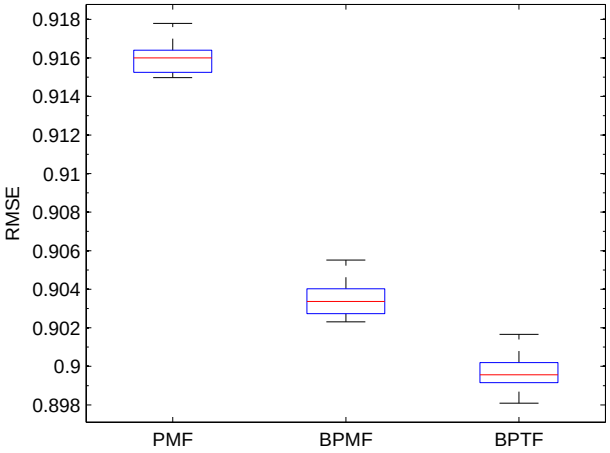

Figure 6: Box plot of accuracies from PMF, BPMF, and BPTF on Netflix. See text.

The resulting *box plot* of RMSEs are shown in figure 6. The *p-value* of *paired t-test* between the results of BPMF and BPTF is 1.3×10^{-22}. In fact, in all runs, BPTF always produce better results than BPMF.

5.2.2 MovieLens The MovieLens data set contains $1,000,209$ ratings from $N = 6,040$ users and $M = 3,706$ movies between April, 2000 and February, 2003, with the restriction that each user has at least 20 ratings. The ratio of observed ratings is round 4.5%. Time information is provided in seconds. We randomly select 10 ratings from each user as the test set, which is roughly 6.5% as large as the training set. The timestamp used for BPTF corresponds to calendar months. We also use $D = 30$ latent features here. The parameters for PMF are $\lambda_U = \lambda_V = 0.05, lrate = 0.001$ as in [6], and the parameters for Bayesian methods are the same as for Netflix.

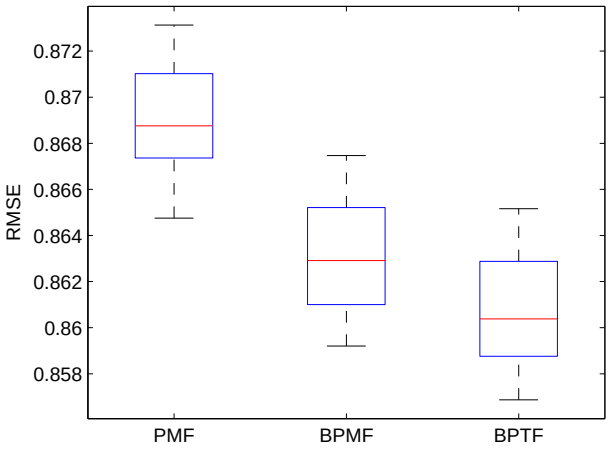

Figure 7: Box plot of the accuracies from PMF, BPMF, and BPTF on MovieLens data.

Figure 7 shows the performance of three algorithms from 20 random runs. This result is similar to what we have for the Netflix data. BPTF still consistently outperforms BPMF, and the *p-value* of *paired t-test* between them is 8.9×10^{-15}.

6 Conclusions

We present the Bayesian Probabilistic Tensor Factorization algorithm for modeling evolving relational data. By introducing a set of additional time features to traditional factor-based collaborative filtering algorithms, and imposing a smoothness constraint on those factors, BPTF is able to learn the global evolution of latent features. An efficient MCMC procedure is proposed to realize automatic model averaging and largely eliminates the need for tuning parameters on large-scale data. We show extensive empirical results on several real-world data sets to illustrate the advantage of temporal model over static models.

There are several possible directions worth exploring in the future. First, we may adopt other types of observational models other than Gaussian, such as the *exponential family* distributions. Then this more general approach can accommodate various kinds of data. However, this may lead to a more complicated posterior distribution for which Gibbs sampling is not applicable. We may then consider the more general *Metropolis-Hastings* sampling techniques such as [16]. Another direction is that we can learn a *Kalman Filtering*-style

dynamic system by estimating the covariance matrix between successive time factors, so that predictions about the future can be made.

References

[1] A. Ahmed and E. P. Xing. Dynamic non-parametric mixture models and the recurrent chinese restaurant process. In *Proceedings of SDM 2008*, 2008.

[2] R. Bell, Y. Koren, and C. Volinsky. The bellkor 2008 solution to the netflix prize, 2008. Available at www.research.att.com/~volinsky/netflix/.

[3] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA 02178-9998, second edition, 1999.

[4] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[5] G. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis: Forecasting and Control*. Pretice-Hall, 1994.

[6] M. Chih-Chao. Large-scale collaborative filtering algorithms. Master's thesis, National Taiwan University, 2008.

[7] A. E. Gelfand and A. F. M. Smith. Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association*, 85(410):398–409, 1990.

[8] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57:97–109, 1970.

[9] T. Hofmann. Probabilistic latent semantic analysis. In *In Proc. of Uncertainty in Artificial Intelligence, UAI99*, pages 289–296, 1999.

[10] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3), September 2009 (to appear).

[11] Y. Koren. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2008.

[12] Y. Koren. Collaborative filtering with temporal dynamics. In *KDD-09*, 2009.

[13] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1091, 1953.

[14] N. Metropolis and S. Ulam. The monte carlo method. *Journal of the American Statistical Association*, 44(247):335–341, 1949.

[15] I. Porteous, E. Bart, and M. Welling. Multi-hdp: A non-parametric bayesian model for tensor factorization. In *AAAI*, 2008.

[16] Y. Qi and T. P. Minka. Hessian-based markov chain monte-carlo algorithms. In *First Cape Cod Workshop on Monte Carlo Methods*, 2002.

[17] C. P. Robert and G. Casella. *Monte Carlo Statistical Methods*. Springer-Verlag New York, LLC, 2nd edition, 2004.

[18] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems (NIPS)*, volume 20, 2007.

[19] R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML 2008), Helsinki, Finland*, 2008.

[20] P. Sarkar, S. M. Siddiqi, and G. J. Gordon. A latent space approach to dynamic embedding of co-occurrence data. In *AISTAT-07*, 2007.

[21] M. N. Schmidt and S. Mohamed. Probabilistic non-negative tensor factorization using markov chain monte carlo. In *European Signal Processing Conference (EUSIPCO)*, 2009.

[22] H. Tong, S. Papadimitriou, P. s. Yu, and C. Faloutsos. Proximity tracking on time-evolving bipartite graphs. In *SDM-08*, 2008.

A Conditional distributions in Gibbs sampling

In this section we give explicit forms for the conditional distributions used in Algorithm 3.1. According to our model assumption in Figure 2, the joint posterior distribution can be factorized as

$$\begin{aligned} & p(\mathbf{U},\mathbf{V},\mathbf{T},\alpha,\Theta_U,\Theta_V,\Theta_T|R) \\ \propto\ & p(\mathbf{R}|\mathbf{U},\mathbf{V},\mathbf{T},\alpha)p(\mathbf{U}|\Theta_U)p(\mathbf{V}|\Theta_V)p(\mathbf{T}|\Theta_T) \\ & p(\Theta_U)p(\Theta_V)p(\Theta_T)p(\alpha). \end{aligned} \tag{A.1}$$

By plugging into Eq. (A.1) all the model components described in Section 3.2.1 and carrying out proper marginalization, we derive the desired conditional distributions in the following two subsections.

A.1 Hyper-parameters By using the conjugate prior for the rating precision α, we have that the conditional distribution of α given $\mathbf{R}$, $\mathbf{U}$, $\mathbf{V}$ and $\mathbf{T}$ follows the Wishart distribution:

$$p(\alpha|\mathbf{R},\mathbf{U},\mathbf{V},\mathbf{T}) = \mathcal{W}(\alpha|W_0^*,\nu_0^*), \tag{A.2}$$

$$\nu_0^* = \tilde{\nu}_0 + \sum_{k=1}^{K}\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}^k,$$

$$(\tilde{W}_0^*)^{-1} = \tilde{W}_0^{-1} + \sum_{k=1}^{K}\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}^k (R_{ij}^k - < U_i, V_j, T_k >)^2.$$

For $\Theta_U \equiv \{\mu_U, \Lambda_U\}$, our graphical model assumption in Figure 2 suggests that it is conditionally independent of all the other parameters given $\mathbf{U}$. We thus integrate out all the random variables in Eq. (A.1) except $\mathbf{U}$ and

obtain the Gaussian-Wishart distribution:

(A.3)
$$p(\Theta_U|\mathbf{U}) = \mathcal{N}(\mu_U|\mu_0^*, (\beta_0^*\Lambda_U)^{-1})\mathcal{W}(\Lambda_U|W_0^*, \nu_0^*),$$
$$\mu_0^* = \frac{\beta_0\mu_0 + N\bar{U}}{\beta_0 + N}, \qquad \beta_0^* = \beta_0 + N, \qquad \nu_0^* = \nu_0 + N;$$
$$(W_0^*)^{-1} = W_0^{-1} + N\bar{S} + \frac{\beta_0 N}{\beta_0 + N}(\mu_0 - \bar{U})(\mu_0 - \bar{U})',$$
$$\bar{U} = \frac{1}{N}\sum_{i=1}^{N} U_i, \qquad \bar{S} = \frac{1}{N}\sum_{i=1}^{N}(U_i - \bar{U})(U_i - \bar{U})'.$$

Similarly, $\Theta_V \equiv \{\mu_V, \Lambda_V\}$ is conditionally independent of all the other parameters given $\mathbf{V}$, and its conditional distribution has the same form:

(A.4)
$$p(\Theta_V|\mathbf{V}) = \mathcal{N}(\mu_V|\mu_0^*, (\beta_0^*\Lambda_V)^{-1})\mathcal{W}(\Lambda_V|W_0^*, \nu_0^*),$$
$$\mu_0^* = \frac{\beta_0\mu_0 + M\bar{V}}{\beta_0 + M}, \qquad \beta_0^* = \beta_0 + M, \qquad \nu_0^* = \nu_0 + M;$$
$$(W_0^*)^{-1} = W_0^{-1} + M\bar{S} + \frac{\beta_0 M}{\beta_0 + M}(\mu_0 - \bar{V})(\mu_0 - \bar{V})',$$
$$\bar{V} = \frac{1}{M}\sum_{j=1}^{M} V_j, \qquad \bar{S} = \frac{1}{M}\sum_{j=1}^{M}(V_j - \bar{V})(V_j - \bar{V})'.$$

Finally, $\Theta_T \equiv \{\mu_T, \Lambda_T\}$ is conditionally independent of all other parameters given $\mathbf{T}$, and its conditional distribution also follows a Gaussian-Wishart distribution:

(A.5)
$$p(\Theta_T|\mathbf{T}) = \mathcal{N}(\mu_T|\mu_0^*, (\beta_0^*\Lambda_U)^{-1})\mathcal{W}(\Lambda_T|W_0^*, \nu_0^*),$$
$$\mu_0^* = \frac{T_1 + \beta_0\rho_0}{\beta_0 + 1}, \qquad \beta_0^* = \beta_0 + 1, \qquad \nu_0^* = \nu_0 + K;$$
$$(W_0^*)^{-1} = W_0^{-1} + \sum_{k=2}^{K}(T_k - T_{k-1})(T_k - T_{k-1})' + \frac{\beta_0}{1 + \beta_0}(T_1 - \rho_0)(T_1 - \rho_0)'.$$

A.2 Model parameters We first consider the user features $\mathbf{U}$. According to the graphical model in Figure 2, its conditional distribution factorizes with respect to individual users:

$$p(\mathbf{U}|\mathbf{R}, \mathbf{V}, \mathbf{T}, \alpha, \Theta) = \prod_{i=1}^{N} p(U_i|\mathbf{R}, \mathbf{V}, \mathbf{T}, \alpha, \Theta_U).$$

We then have, for each user feature vector,

(A.6)
$$p(U_i|\mathbf{R}, \mathbf{V}, \mathbf{T}, \alpha, \Theta_U) = \mathcal{N}(U_i|\mu_i^*, (\Lambda_i^*)^{-1}),$$
$$\mu_i^* \equiv (\Lambda_i^*)^{-1}\Big(\Lambda_U\mu_U + \alpha\sum_{k=1}^{K}\sum_{j=1}^{M} I_{ij}^k R_{ij}^k Q_{jk}\Big),$$
$$\Lambda_i^* \equiv \Lambda_U + \alpha\sum_{k=1}^{K}\sum_{j=1}^{M} I_{ij}^k Q_{jk}Q_{jk}',$$

where $Q_{jk} \equiv V_j \cdot T_k$ is the element-wise product of V_j and T_k. For the item features $\mathbf{V}$ the conditional distribution factorizes with respect to individual items, and for each item feature vector we have

(A.7)
$$p(V_j|\mathbf{R}, \mathbf{U}, \mathbf{T}, \alpha, \Theta_V) = \mathcal{N}(V_j|\mu_j^*, (\ \Lambda_j^*)^{-1}),$$
$$\mu_j^* \equiv (\Lambda_j^*)^{-1}\Big(\Lambda_V\mu_V + \alpha\sum_{k=1}^{K}\sum_{i=1}^{N} I_{ij}^k R_{ij}^k P_{ik}\Big),$$
$$\Lambda_j^* \equiv \Lambda_V + \alpha\sum_{k=1}^{K}\sum_{i=1}^{N} I_{ij}^k P_{ik}P_{ik}',$$

where $P_{ik} \equiv U_i \cdot T_k$.

Regarding the time features, the conditional distribution of T_k is also a Gaussian distribution:

(A.8) $$p(T_k|\mathbf{R}, \mathbf{U}, \mathbf{V}, \mathbf{T}_{-k}, \alpha, \Theta_T) = \mathcal{N}(T_k|\mu_k^*, (\Lambda_k^*)^{-1}),$$

where $\mathbf{T}_{-k}$ denotes all the time feature vectors except T_k. The mean vectors and the precision matrices depend on k in the following way:
For $k = 1$,

$$\mu_1^* = \frac{T_2 + \mu_T}{2}, \quad \Lambda_1^* = 2\Lambda_T + \alpha\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}^1 X_{ij}X_{ij}',$$

where $X_{ij} \equiv U_i \cdot V_j$ (the same for the following).
For $2 \leqslant k \leqslant K - 1$,

$$\mu_k^* = (\Lambda_k^*)^{-1}\Big(\Lambda_T(T_{k-1} + T_{k+1}) + \alpha\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}^k R_{ij}^k X_{ij}\Big),$$
$$\Lambda_k^* = 2\Lambda_T + \alpha\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}^k X_{ij}X_{ij}'.$$

For $k = K$,

$$\mu_K^* = (\Lambda_K^*)^{-1}\Big(\Lambda_T T_{K-1} + \alpha\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}^K R_{ij}^K X_{ij}\Big),$$
$$\Lambda_K^* = \Lambda_T + \alpha\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}^K X_{ij}X_{ij}'.$$