A Transfer Probabilistic Collective Factorization Model to Handle Sparse Data in Collaborative Filtering

How Jing
National Taiwan University
Taipei, Taiwan
kublai.jing@gmail.com

An-Chun Liang and Shou-De Lin
National Taiwan University
Taipei, Taiwan
notmydiagnosis@gmail.com, sdlin@csie.ntu.edu.tw

Yu Tsao
Academia Sinica
Taipei, Taiwan
yu.tsao@citi.sinica.edu.tw

***Abstract*—Data Sparsity incurs serious concern in collaborative filtering (CF). This issue is especially critical for newly launched CF applications where observed ratings are too scarce to learn a good model to predict missing values. There could be, however, information from other related domains which are with relatively denser data that can be utilized. This paper proposes a transfer-learning based approach that exploits probabilistic matrix factorization model trained with variational expectation-maximization (VEM) to resolve data sparsity by using information from multiple auxiliary domains. We conduct experiments on several data combination and report significant improvements over state-of-the-art transfer-based models for collaborative filtering. The results also show that our framework is the only solution that can achieve acceptable performance when each user has only one single rating. The code of our model is available at [1].**

I. Introduction

Collaborative Filtering (CF) [8] is a popular technique used to design a recommendation engine where the goal is to predict missing values in an incomplete rating matrix. One major limitation of CF is that it cannot produce satisfactory results when the observed ratings are considerably fewer than the total entries of the matrix, which is generally regarded as data sparsity or cold-start phenomenon. Such issue happens frequently for newly launched online services where users have just begun to visit the site with insufficient amount of data collected. Data *sparsity* seriously degrades the performance of prediction due to over-fitting. It is a serious concern in practice as generally new services demand an reliable system more eagerly than mature ones.

Fortunately, in practice there are related data that one can take advantage of from other domains to enhance a cold-start CF model. They are usually called the auxiliary data, and many transfer learning algorithms have been proposed to leverage them from various perspectives [3], [4], [6], [7], [12], [13]. For example, rating data from a movie recommendation system may be helpful to alleviate the sparsity problem in a book recommendation system if we assume certain, most likely latent, correspondences are shared between these two user-item matrices. In [3] and [4], authors assume that cluster-level patterns of matrices are shared across domains, so they construct clusters of rating patterns as bridges to transfer the knowledge.

Another type of auxiliary data that exists frequently in real recommendation systems are users' implicit feedbacks, usually encoded in binary (i.e. click or unclick). It is almost always the case that users have clicked or browsed a lot more items than they have actually rated. Such click records show users' preferences of items to some degree, and they are useful if we could extract and transfer the knowledge from such 'heterogeneous' feedbacks to enhance the system. Likewise, item-side auxiliary data existing in the form of implicit feedbacks are relatively easy to obtain in practice. For instance, favored/disfavored is provided in Movieplot [2] and love/ban is provided in Last.fm [3] for users to express their preferences in a way that would not degrade users' browsing experiences and satisfaction. In [6], the authors show how a tri-factorization model that first learns latent factors from auxiliary domains with binary feedbacks, then transfers the latent factors to the target domain can boost the performance in the target domain.

We roughly categorize current state-of-the-art transfer-based methods for handling sparsity issue in CF into four categories by asking two questions. First, do we know explicitly the correspondences of users and/or items between the auxiliary and target domains? Second, do auxiliary data come from a homogeneous source (e.g. ratings) or heterogeneous source (e.g. binary like/dislike values)? Table 1 shows the state-of-the-art transferring methods that approach the problem from these two perspectives.

Unfortunately, methods proposed to handle one type of transfer scenario usually cannot handle the others. For example, to construct cluster-level rating patterns as a knowledge bridge, the auxiliary data needs to be homogeneous, meaning that they should have the same rating scale as in the target domain. Heterogeneous values are not suitable for clustering rating values as described in [6]. On the other hand, methods that use binary heterogeneous implicit feedbacks require that either user or item variables share the same latent factors across domains. Therefore, without knowing explicitly the correspondences of users and/or items between target and auxiliary domains, those methods cannot be exploited.

[1] https://github.com/Kublai-Jing/TPCF

[2] http://www.moviepilot.de/

[3] http://www.last.fm/

	homogeneous	heterogeneous
U and/or I		TCF [7],CST [6]
None	CBT [3] RMGM [4]	

TABLE I: Current state-of-the-art methods on transfer-based models for sparsity reduction in collaborative filtering. $\mathbf{U}$ and/or $\mathbf{I}$ denotes whether the correspondences of users and/or items are given in the auxiliary domains. Homo/Heterogeneous represent auxiliary data type.

The question we ask throughout this paper is: can we simultaneously transfer data from multiple domains with different properties (i.e. data that falls into different cells in Table 1)? The goal of this paper is to design a more general model not only outperforming each existing work individually, but also enjoying further performance leap by allowing the transferring of multiple types of knowledge to the target domain.

We present a unified probabilistic framework named *Transfer Probabilistic Collective Factorization* (TPCF) aiming at bringing different information from multiple CF tasks into the target domain to alleviate the sparsity problem. Specifically, we focus on three auxiliary CF data. First type is data with aligned users but not aligned items that have heterogeneous implicit feedbacks, existing in binary form. Second type is data with aligned items but not aligned users, also existing in binary form. Third type is homogeneous data that have the same rating scale as the target domain, but without the knowledge of the correspondences of users and items between target and auxiliary domains. We note that our model is general and extendable to other cases that appear in each cell of Table 1 (such as the situation considered in [7]). Without loss of generality, in this paper we will focus on the three cases for transferring, because these cases happen more frequently in practice.

The main contributions of this paper are as follows:

- We present TPCF, a general probabilistic model for transfer learning in collaborative filtering by using data from multiple domains with different statistical properties. To our knowledge, this is the first unified transfer learning model for CF that considers all the above-mentioned scenarios.
- One technical challenge in the problem is how to simultaneously transfer data from multiple domains and tackle both binary and rating values well. We present a learning schema based on variational EM algorithm to solve this issue within the probabilistic framework.
- The experiments show our model not only outperforms the state-of-the-art CF-based transfer learning models before unifying all the auxiliary sources together but also enjoys another level of performance boosting when bringing all information into the target domain. The results also show that our model can achieve reasonable outcome when there are as few as only one single rating for each user.

II. Related Work

Transfer learning algorithms have been proved to be effective in solving CF problems. Knowledge transferring is carried out by assuming there is some shared latent structure present across different domains. This section briefly introduces some popular models.

Collective Matrix Factorization (CMF) uses common latent features to jointly factorize multiple matrices with correspondences between rows and columns Usually, CMF is trained by minimizing some pre-specified Bregman divergence that does not have to be squared L2 loss, plus regularization terms to prevent over-fitting [13]. However, it was shown that in such SVD style model, complexity control of regularization parameters is a challenging problem that needs to be carefully tuned. Probabilistic models, on the other hand, have the advantage of using an adaptive prior for automatic complexity control [9]. Rating Matrix Generative Model (RMGM) takes cluster-level information to be transferred from auxiliary domains by assuming that different domains share the same cluster-level rating patterns [4]. RMGM can be viewed as an extension of mixture model that simultaneously clusters users and items to model rating patterns across domains. The drawback of RMGM comes from its inability to handle binary source data for transferring. Also, RMGM maximizes a self-defined likelihood objective which does not always penalize bad predictions. Coordinate System Transfer (CST) is a matrix tri-factorization approach that assumes latent features are *similar* but not identical across domains and allows the user to control the closeness of the latent features between different domains [6]. It first learns users' and items' factors from the auxiliary domains, then adapts them to the target domain.

III. Problem Formulation and Notations

We now describe the scenario we consider in this paper formally. As mentioned before, our model is very flexible and can be applied to problems in any quadrant of Table 1. However, for clarity purpose, from this point on we only focus on three transferring tasks with three types of auxiliary data. There are four different domains represented as matrices. One is the target matrix denoted as $\mathbf{R}$ which has sparse rating values and represents the domain we want to predict missing values in. There are two auxiliary matrices with heterogeneous binary feedback, denoted as $\mathbf{R^U}$ and $\mathbf{R^I}$, representing domains with only aligned users ($\mathbf{R^U}$) and only aligned items ($\mathbf{R^I}$) respectively. We assume that the two auxiliary matrices have binary values 0/1 indicating users' implicit feedback, though it would be clear later on that our framework can be easily adjusted to handle numerical values. Users in $\mathbf{R^U}$ are aligned to users in $\mathbf{R}$, and items in $\mathbf{R^I}$ are aligned to items in $\mathbf{R}$. Also, there is another auxiliary domain, denoted as $\mathbf{R^O}$, that represents another different but related CF task with neither users nor items aligned. To be more precise, we assume that rating values in $\mathbf{R^O}$ have the same scale as $\mathbf{R}$, but items and users in $\mathbf{R^O}$ are different to those in $\mathbf{R}$. An illustration of the scenario we consider is shown in Figure 1.

For simplicity, the number of users and items in $\mathbf{R}, \mathbf{R^U}$ and $\mathbf{R^I}$ is denoted as $\mathbf{N}$ and $\mathbf{M}$ respectively. The number of users and items in $\mathbf{R^O}$ is denoted as $\mathbf{L}$ and $\mathbf{S}$. Moreover, we define a mask function $\mathbf{Y_{ab}(d)}$ which returns 1 if cell $(\mathbf{a},\mathbf{b})$

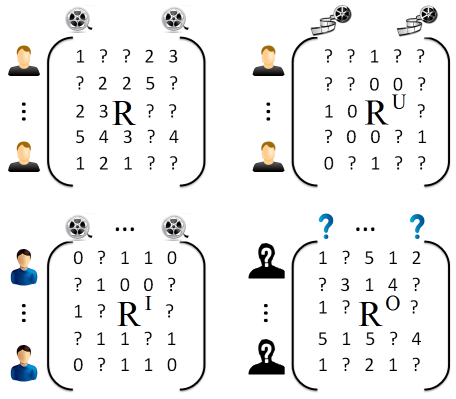

Fig. 1: Illustration of the scenario considered in this paper: users in $\mathbf{R}$ are aligned to users in $\mathbf{R^U}$; items in $\mathbf{R}$ are aligned to items in $\mathbf{R^I}$; we have no prior knowledge about the correspondences of users and items in $\mathbf{R^O}$.

is observed in domain $\mathbf{d}$, or 0 otherwise. In our setting, there are four domains, hence $\mathbf{d}$ can take on values $\{\mathbf{T}, \mathbf{U}, \mathbf{I}, \mathbf{O}\}$ where $\mathbf{T}$ denotes the target domain, and $\mathbf{U}, \mathbf{I}, \mathbf{O}$ denote the three auxiliary domains.

Our problem then can be formulated as follows: given a target rating matrix $\mathbf{R} \in \{1,2,3,4,5\}^{\mathbf{N}\times\mathbf{M}} \odot \mathbf{Y}_{**}(\mathbf{T})$ and three auxiliary matrices $\mathbf{R^U} \in \{0,1\}^{\mathbf{N}\times\mathbf{M}} \odot \mathbf{Y}_{**}(\mathbf{U})$, $\mathbf{R^I} \in \{0,1\}^{\mathbf{N}\times\mathbf{M}} \odot \mathbf{Y}_{**}(\mathbf{I})$ and $\mathbf{R^O} \in \{\mathbf{1},\mathbf{2},\mathbf{3},\mathbf{4},\mathbf{5}\}^{\mathbf{L}\times\mathbf{S}} \odot \mathbf{Y}_{**}(\mathbf{O})$, the goal is to utilize auxiliary matrices to boost missing rating prediction performance for $\mathbf{R}$. Here $\odot$ means element-wise multiplication. Note that we do not assume all three matrices $\mathbf{R^U}, \mathbf{R^I}$ and $\mathbf{R^O}$ need to exist. Our model can be conducted even when only one type of data is available.

IV. Transfer Probabilistic Collective Factorization

We now present TPCF, which falls into the category of an extension of the PMF family [9], in which we want to learn a joint probabilistic model using all four domains that can overcome the limitations of previous models mentioned in Section 2.

A. The Model

The graphical illustration of TPCF is shown in Fig 2. The intuition is that latent user factors and item factors should capture both rating and binary feedback distributions well. Moreover, no matter what domain we are in, we always assume that users are coming from the same distribution; same assumption applies for items. This assumption may seem to be too strong at first glance, but since we are working in a low-dimensional latent space, high-level shared concept such as genres of items could emerge.

The distribution of the homogeneous ratings $\mathbf{R}, \mathbf{R^O}$ are

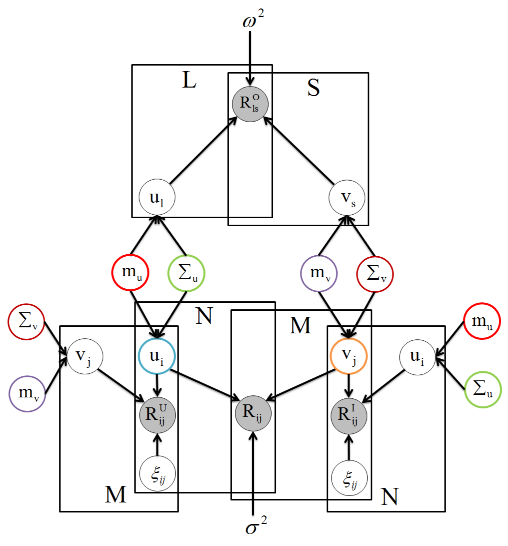

Fig. 2: The graphical representation of TPCF. Colored nodes refer to variables that are shared. ξ is an auxiliary variable for optimization that will be detailed in Section 4.2. Here we replicate prior to avoid interleaved edges to make the picture uncluttered.

assumed to be Gaussian:

$$\prod_{\mathbf{i=1}}^{\mathbf{N}} \prod_{\mathbf{j=1}}^{\mathbf{M}} \mathcal{N}(\mathbf{R_{ij}} | \mathbf{u_i^T v_j}, \sigma^{\mathbf{2}})^{\mathbf{Y_{ij}(T)}}$$

$$\prod_{\mathbf{l=1}}^{\mathbf{L}} \prod_{\mathbf{s=1}}^{\mathbf{S}} \mathcal{N}(\mathbf{R^O_{ls}} | \mathbf{u_l^T v_s}, \omega^{\mathbf{2}})^{\mathbf{Y_{ls}(O)}}$$

and the distribution of the heterogeneous binary values is modeled by Bernoulli distribution:

$$\prod_{\mathbf{i=1}}^{\mathbf{N}} \prod_{\mathbf{j=1}}^{\mathbf{M}} Bern(\mathbf{R^U_{ij}} | \sigma(\mathbf{u_i^T v_j}))^{\mathbf{Y_{ij}(U)}} Bern(\mathbf{R^I_{ij}} | \sigma(\mathbf{u_i^T v_j}))^{\mathbf{Y_{ij}(I)}}$$

where $\mathbb{U} = \{\mathbf{u_{i,i=1:N}}, \mathbf{u_{l,l=1:L}}\}$ is the set of parameters representing the preferences of the users, and $\mathbb{V} = \{\mathbf{v_{j,j=1:M}}, \mathbf{v_{s,l=1:S}}\}$ is the set of parameters representing the implicit attributes of the items. Note that from the figure, the parameters $\mathbf{u_i}, \mathbf{v_j}$ in domain $\mathbf{R}, \mathbf{R^U}$ and $\mathbf{R^I}$ are different from $\mathbf{u_l}, \mathbf{v_s}$ in domain $\mathbf{R^O}$, yet we do not distinguish them by defining different notations in order to make the equation more succinct. The distinction is implicitly embedded in domain variables $\mathbf{T}, \mathbf{U}, \mathbf{I}$ and $\mathbf{O}$. We then assume that there is a single prior distribution for *all* users' parameters, and there is a single prior distribution for *all* items' parameters that are both Gaussian. The whole generative process is as follows:

1) Domain $\mathbf{R}, \mathbf{R^U}, \mathbf{R^I}$:
   a) For each user $\mathbf{i}$, generate $\mathbf{u_i} \sim \mathcal{N}(m_u, \Sigma_u)$
   b) For each item $\mathbf{j}$, generate $\mathbf{v_j} \sim \mathcal{N}(m_v, \Sigma_v)$
   c) For each cell $(\mathbf{i}, \mathbf{j})$ in $\mathbf{R}$, generate $\mathbf{R_{ij}} \sim \mathcal{N}(\mathbf{u_i^T v_j}, \sigma^{\mathbf{2}})$
   d) For each cell $(\mathbf{i}, \mathbf{j})$ in $\mathbf{R^U}$, generate $\mathbf{R^U_{ij}} \sim \text{Bern}(\sigma(\mathbf{u_i^T v_j}))$

e) For each cell $(\mathbf{i},\mathbf{j})$ in $\mathbf{R^I}$, generate $\mathbf{R^I_{ij}} \sim \text{Bern}(\sigma(\mathbf{u_i^T v_j}))$

2) Domain $\mathbf{R^O}$:

a) For each user $\mathbf{l}$, generate $\mathbf{u_l} \sim \mathcal{N}(\mathbf{m_u}, \mathbf{\Sigma_u})$
b) For each item $\mathbf{s}$, generate $\mathbf{v_s} \sim \mathcal{N}(\mathbf{m_v}, \mathbf{\Sigma_v})$
c) For each cell $(\mathbf{l}, \mathbf{s})$ in $\mathbf{R^O}$, generate $\mathbf{R^O_{ls}} \sim \mathcal{N}(\mathbf{u_l^T v_s}, \omega^2)$,

where $\text{Bern}(\sigma(x))$ denotes the Bernoulli distribution with mean given by $\sigma(x)$ which is the sigmoid function: $\sigma(x) = \frac{1}{1+\exp(-x)}$. Note that the information of which user in $\mathbf{R^U}$ corresponds to which user in $\mathbf{R}$ and which item in $\mathbf{R^I}$ corresponds to which item in $\mathbf{R}$ are known. This makes the algorithms easier to perform knowledge transfer because the correspondence information tells us the preference of each individual user, even though this preference statistics are heterogeneous. For instance, in a movie rental application, user $\mathbf{i}$ has only a few ratings in $\mathbf{R}$, but we observe that he/she has clicked on lots of *action* movies about in $\mathbf{R^U}$. Then we can make a good guess about this user's preference. Similarly, movie $\mathbf{j}$ has only received a few ratings in $\mathbf{R}$, but we observe that this movie received a lot of clicks in $\mathbf{R^I}$. Then it might be reasonable to believe that this movie is a popular movie and will receive mostly positive ratings in $\mathbf{R}$. This correspondence/alignment information allows us to identify user's behaviors more clearly, and is the key to the success of many algorithms.

However, in $\mathbf{R^O}$ we do not have any correspondence information. In fact, we do not even assume that there is overlap of users/items between $\mathbf{R^O}$ and the other three domains. Even though there may exist some overlap between $\mathbf{R^O}$ and $(\mathbf{R}, \mathbf{R^U}, \mathbf{R^I})$, we do not assume such correspondence is available due to privacy or overhead concerns. For instance, $\mathbf{R^O}$ and $\mathbf{R}$ might be vendors that sell different products, and both companies have the obligation not to disclose the identities of its customers. This makes it harder to transfer knowledge because we cannot identify and match individual user from different domains easily.

Our basic assumption is that, in *latent* space of lower dimension, users' preferences and items' attributes, are somewhat similar across domains and come from the same prior distribution. This is different from those methods that consider cluster-level ratings as a bridge for knowledge transferring, as they assume the explicit rating patterns to be similar. We hypothesize that as rating patterns differ across domains, clustered patterns from domain A may not be beneficial to domain B. However, down to a lower dimensional latent space, we may have a better chance to capture the shared information to improve the model performance.

Back to the model itself, we write down the (marginal) log-likelihood function of our probabilistic model as follows:

$$\begin{aligned}
&\log P(\mathbf{R}, \mathbf{R^U}, \mathbf{R^I}, \mathbf{R^O}; \Theta) \\
&= \log P(\mathbf{R}, \mathbf{R^U}, \mathbf{R^I}; \Theta) + \log P(\mathbf{R^O}; \Theta) \\
&= \log \Bigg[\int_{\mathbf{u_{1:N}}} \int_{\mathbf{v_{1:M}}} \prod_{i=1}^{N} \mathcal{N}(\mathbf{u_i}; \mathbf{m_u}, \mathbf{\Sigma_u}) \prod_{\mathbf{j=1}}^{\mathbf{M}} \mathcal{N}(\mathbf{v_j}; \mathbf{m_v}, \mathbf{\Sigma_v}) \\
&\prod_{\mathbf{i=1}}^{\mathbf{N}} \prod_{\mathbf{j=1}}^{\mathbf{M}} \mathcal{N}(\mathbf{R_{ij}}; \mathbf{u_i^T v_j}, \sigma^2)^{\mathbf{Y_{ij}(T)}} \\
&\Big[\exp(\mathbf{u_i^T v_j R^U_{ij}}) \sigma(-\mathbf{u_i^T v_j}) \Big]^{\mathbf{Y_{ij}(U)}} \\
&\Big[\exp(\mathbf{u_i^T v_j R^I_{ij}}) \sigma(-\mathbf{u_i^T v_j}) \Big]^{\mathbf{Y_{ij}(I)}} \Bigg] \\
&+ \log \int_{\mathbf{u_{1:L}}} \int_{\mathbf{v_{1:S}}} \prod_{\mathbf{l=1}}^{\mathbf{L}} \mathcal{N}(\mathbf{u_l}; \mathbf{m_u}, \mathbf{\Sigma_u}) \prod_{\mathbf{s=1}}^{\mathbf{S}} \mathcal{N}(\mathbf{v_s}; \mathbf{m_v}, \mathbf{\Sigma_v}) \\
&\prod_{\mathbf{l=1}}^{\mathbf{L}} \prod_{\mathbf{s=1}}^{\mathbf{S}} \mathcal{N}(\mathbf{R^O_{ls}}; \mathbf{u_l^T v_s}, \omega^2)^{\mathbf{Y_{ls}(O)}}, \qquad (1)
\end{aligned}$$

where $\Theta = \{\mathbf{m_u}, \mathbf{\Sigma_u}, \mathbf{m_v}, \mathbf{\Sigma_v}, \sigma^2, \omega^2\}$ is a set of model parameters, and we use the equivalence: $\sigma(x)^t(1-\sigma(x))^{1-t} = \exp(xt)\sigma(-x)$ for the sigmoid function. Recall that we have defined $\mathbf{Y_{ab}(d)} = 1$ if $\mathbf{Y_{ab}}$ is a non-missing entry in domain $\mathbf{d}$ to indicate which domain we are referring to.

B. Learning

The parameters set Θ which contains the means and covariances of the Gaussian prior for users and items needs to be learned such that Equation 1 is maximized. This optimization problem is intractable due to the integration over all latent variables $\mathbf{u}$ and $\mathbf{v}$, hence we exploit the concept of variational approximation to perform learning [5]. We first use Jensen's inequality to derive a lower bound on the log-likelihood,

$$\begin{aligned}
&\log P(\mathbf{R}, \mathbf{R^U}, \mathbf{R^I}; \Theta) + \log P(\mathbf{R^O}; \Theta) \geq \\
&\mathbb{E}_Q \Big[\log P(\mathbf{R}, \mathbf{R^U}, \mathbf{R^I}, \mathbf{u_{1:N}}, \mathbf{v_{1:M}}; \Theta) \Big] + \mathcal{H}(Q(\mathbf{u_{1:N}}, \mathbf{v_{1:M}}; \psi)) \\
&+ \mathbb{E}_{Q^O} \Big[\log P(\mathbf{R^O}, \mathbf{u_{1:L}}, \mathbf{v_{1:S}}; \Theta) \Big] + \mathcal{H}(Q^O(\mathbf{u_{1:L}}, \mathbf{v_{1:S}}; \psi^O)), \qquad (2)
\end{aligned}$$

where $\mathcal{H}$ is the entropy. We have separated out domains $(\mathbf{R}, \mathbf{R^U}, \mathbf{R^I})$ and $\mathbf{R^O}$ for notational convenience. Q, Q^O are the variational posterior distributions that are governed by a set of variational parameters ψ, ψ^O respectively. The gap of this bound to the true log-likelihood is the Kullback-Leibler divergence between the approximate posterior and the true posterior, and the bound is tight if and only if Q is equal to the true posterior [1]. However, it is intractable to do exact inference on true posterior; hence we use mean-field variational distribution by assuming a fully factorized posterior,

$$\begin{aligned}
Q(\mathbf{u_{1:N}}, \mathbf{v_{1:M}}; \psi) &= \prod_{\mathbf{i=1}}^{\mathbf{N}} \mathcal{N}(\mathbf{u_i}; \lambda_{\mathbf{u_i}}, \gamma_{\mathbf{u_i}}) \prod_{\mathbf{j=1}}^{\mathbf{M}} \mathcal{N}(\mathbf{v_j}; \lambda_{\mathbf{v_j}}, \gamma_{\mathbf{v_j}}) \\
Q(\mathbf{u_{1:L}}, \mathbf{v_{1:S}}; \psi^{\mathbf{O}}) &= \prod_{\mathbf{l=1}}^{\mathbf{L}} \mathcal{N}(\mathbf{u_l}; \lambda_{\mathbf{u_l}}, \gamma_{\mathbf{u_l}}) \prod_{\mathbf{s=1}}^{\mathbf{S}} \mathcal{N}(\mathbf{v_s}; \lambda_{\mathbf{v_s}}, \gamma_{\mathbf{v_s}}), \qquad (3)
\end{aligned}$$

where λ is a set of means for variational Gaussian, and γ is a set of covariances which are set to be diagonal. We now have two types of parameters. First, a set of model parameters that control the Gaussian prior,

$$\Theta = \{\mathbf{m_u}, \mathbf{m_v}, \mathbf{\Sigma_u}, \mathbf{\Sigma_v}, \sigma^{\mathbf{2}}, \omega^{\mathbf{2}}\},$$

and two sets of variational parameters,

$$\begin{aligned}\psi &= \{\lambda_{\mathbf{u_{i=1:N}}}, \lambda_{\mathbf{v_{j=1:M}}}, \gamma_{\mathbf{u_{i=1:N}}}, \gamma_{\mathbf{v_{j=1:M}}}\},\\ \psi^{\mathbf{O}} &= \{\lambda_{\mathbf{u_{l=1:L}}}, \lambda_{\mathbf{v_{s=1:S}}}, \gamma_{\mathbf{u_{l=1:L}}}, \gamma_{\mathbf{v_{s=1:S}}}\}.\end{aligned}$$

The optimization can then be done using variational expectation-maximization (VEM) [1]. In VE-step, we fix model parameters and optimize the bound in Equation 2 w.r.t. ψ and ψ^O to make the bound as tight as possible. In VM-step, we fix variational parameters and optimize Equation 2 w.r.t. model parameters Θ to raise the bound. One thing should be mentioned is that this procedure guarantees to raise the *bound* of the log-likelihood, but not the log-likelihood itself. However, VEM has shown to work well in practice. The details of VEM steps are described below.

Variational E-step: First we start with the VE-step of learning to obtain the mean and covariance for each variational Gaussian. Let $q_{\mathbf{u_i}}$ be a single variational Gaussian for user i with mean $\lambda_{\mathbf{u_i}}$ and (diagonal) covariance $\gamma_{\mathbf{u_i}}$, the optimal $q^*_{\mathbf{u_i}}$ can be derived by using the general principle of mean-field inference [14]:

$$\log q^*_{\mathbf{u_i}} \propto \mathbb{E}_{Q_{\text{-}q_{\mathbf{u_i}}}}\Big[\log p(\mathbf{R}, \mathbf{R^U}, \mathbf{R^I}, \mathbf{u_{1:N}}, \mathbf{v_{1:M}}; \Theta)\Big], \quad (4)$$

Making use of the independence relations in the graphical model and drop constants that are not dependent to $q_{\mathbf{u_i}}$, we obtain the following form for $q^*_{\mathbf{u_i}}$,

$$\begin{aligned}\log q^*_{\mathbf{u_i}} = \mathbb{E}_{\text{-}q_{\mathbf{u_i}}}\Big[\sum_{j=1}^{M} \mathbf{Y_{ij}(T)} \log \mathcal{N}(\mathbf{R_{ij}}; \mathbf{u_i^T v_j})\Big] +\\ \alpha \mathbb{E}_{\text{-}q_{\mathbf{u_i}}}\Big[\sum_{j=1}^{M} \mathbf{Y_{ij}(U)}\Big(\mathbf{u_i^T v_j R^U_{ij}} + \log \sigma(-\mathbf{u_i^T v_j})\Big)\Big] +\\ \alpha \mathbb{E}_{\text{-}q_{\mathbf{u_i}}}\Big[\sum_{j=1}^{M} \mathbf{Y_{ij}(I)}\Big(\mathbf{u_i^T v_j R^I_{ij}} + \log \sigma(-\mathbf{u_i^T v_j})\Big)\Big] \quad (5)\end{aligned}$$

where the expectation is taken w.r.t. other variational Gaussians, -$q_{\mathbf{u_i}}$. We have introduced a parameter α which is a trade-off parameter that controls the 'mixing' weight between $\mathbf{R}$ and $(\mathbf{R^U}, \mathbf{R^I})$. In general, the bigger the α, the more we rely on the information from $\mathbf{R^U}$ and $\mathbf{R^I}$.

In Equation 5, the first term involves the integration of a Gaussian and a bunch of independent 'log-Gaussian' over all items set, which can be done analytically,

$$\begin{aligned}&\int_{\mathbf{v}_{1:M}} \prod_{j=1}^{M} \mathcal{N}(\mathbf{v_j}; \lambda_{\mathbf{v_j}}, \gamma_{\mathbf{v_j}}) \sum_{\mathbf{j=1}}^{\mathbf{M}} \mathbf{Y_{ij}(T)} \log \mathcal{N}(\mathbf{R_{ij}}; \mathbf{u_i^T v_j}, \sigma^{\mathbf{2}}) \mathbf{dv}_{1:M}\\ &= \sum_{j=1}^{M} \frac{1}{2\sigma^2}(2\mathbf{R_{ij} u_i}^{\mathbf{T}} \lambda_{\mathbf{v_j}} - \mathbf{u_i^T}(\lambda_{\mathbf{v_j}} \lambda_{\mathbf{v_j}}^{\mathbf{T}} + \gamma_{\mathbf{v_j}})\mathbf{u_i}) + \mathbf{const},\end{aligned}$$

where $\mathbf{const}$ denotes factors that are not dependent on $\mathbf{u_i}$.

The second and the third terms in Equation 5, however, require integration of a Gaussian and a log-logistic function,

$$\int_{\mathbf{v_j}} \mathcal{N}(\mathbf{v_j}; \lambda_{\mathbf{v_j}}, \gamma_{\mathbf{v_j}}) \log \sigma(-\mathbf{u_i^T v_j}) \mathbf{dv_j}$$

that are not analytically tractable. We hence introduce another lower-bound on the log-logistic function by using first-order Taylor's approximation for convex functions [2],

$$\begin{aligned}\log \sigma(-\mathbf{u_i^T v_j}) \geq \log(\xi_{\mathbf{ij}}) +\\ \Big(\frac{-\mathbf{u_i^T v_j} - \xi_{\mathbf{ij}}}{2} - \phi(\xi_{\mathbf{ij}})((\mathbf{u_i^T v_j})^2 - \xi_{\mathbf{ij}}{}^2)\Big),\end{aligned} \quad (6)$$

where $\phi(x) = \frac{1}{2x}\Big(\sigma(x) - \frac{1}{2}\Big)$. The cost of this approximation is that we have introduced an additional set of variational parameters ξ_{ij} for each (user, item) pair in domains $\mathbf{R^U}$ and $\mathbf{R^I}$. With Equation 6, the integration of a Gaussian and the bound w.r.t. $\mathbf{v_j}$ can be derived analytically, as shown below.

$$\begin{aligned}&\int_{\mathbf{v_j}} \mathcal{N}(\mathbf{v_j}; \lambda_{\mathbf{v_j}}, \gamma_{\mathbf{v_j}}) \log \sigma(-\mathbf{u_i^T v_j}) \mathbf{dv_j}\\ &= \Big(\log(\xi_{\mathbf{ij}}) + \frac{-\mathbf{u_i^T} \lambda_{\mathbf{v_j}} - \xi_{\mathbf{ij}}}{2} - \phi(\xi_{\mathbf{ij}})(\mathbf{u_i^T}(\lambda_{\mathbf{v_j}} \lambda_{\mathbf{v_j}}^{\mathbf{T}} + \gamma_{\mathbf{v_j}})\mathbf{u_i} - \xi^{\mathbf{2}}_{\mathbf{ij}})\Big)\end{aligned}$$

To proceed, we have specified how to compute all three terms in Equation 5 that are needed to derive optimal $q^*_{\mathbf{u_i}}$. The parameters for $q_{\mathbf{u_i}}$, i.e. mean $\lambda_{\mathbf{u_i}}$ and covariance $\gamma_{\mathbf{u_i}}$, can then be determined by observing the first order and second order terms and 'complete the square' for the Gaussian density. Also, the optimal variational parameters $\xi_{\mathbf{ij}}$ can be determined by replacing $\log \sigma(-\mathbf{u_i^T v_j})$ in Equation 5 with the lower bound in Equation 6 and taking the derivative w.r.t. $\xi_{\mathbf{ij}}$. The update rules for variational parameters are shown as follows:

$$\begin{aligned}\lambda_{\mathbf{u_i}} = &\Bigg[\mathbf{\Sigma_u^{-1}} + \sum_{\mathbf{j=1}}^{\mathbf{M}}(\lambda_{\mathbf{v_j}} \lambda_{\mathbf{v_j}}^{\mathbf{T}} + \gamma_{\mathbf{v_j}})\\ &\Big(\mathbf{Y_{ij}(T)} + \alpha \mathbf{Y_{ij}(U)} \xi_{\mathbf{ij}} + \alpha \mathbf{Y_{ij}(I)} \xi_{\mathbf{ij}}\Big)\Bigg]^{-1}\\ &\Bigg[\mathbf{\Sigma_u^{-1} m_u} + \sum_{\mathbf{j=1}}^{\mathbf{M}}\\ &\lambda_{\mathbf{v_j}}\Big(\mathbf{Y_{ij}(T) R_{ij}} + \alpha \mathbf{Y_{ij}(U)}(\mathbf{R^U_{ij}} - \frac{1}{2}) + \alpha \mathbf{Y_{ij}(I)}(\mathbf{R^I_{ij}} - \frac{1}{2})\Big)\Bigg]\\ \gamma_{\mathbf{u_i,dd}} = &\Bigg[\mathbf{\Sigma^{-1}_{u,dd}} + \sum_{\mathbf{j=1}}^{\mathbf{M}}(\lambda^{\mathbf{2}}_{\mathbf{v_j,dd}} + \gamma_{\mathbf{v_j,dd}})\\ &\Big(\mathbf{Y_{ij}(T)} \frac{\mathbf{1}}{\sigma^{\mathbf{2}}} + \alpha \mathbf{Y_{ij}(U)} 2\phi(\xi_{ij}) + \alpha \mathbf{Y_{ij}(I)} 2\phi(\xi_{\mathbf{ij}})\Big)\Bigg]^{\mathbf{-1}}\\ \xi_{\mathbf{ij}} = &\,\mathbf{Y_{ij}(U) Tr}\Big((\lambda_{\mathbf{u_i}} \lambda_{\mathbf{u_i}}^{\mathbf{T}} + \gamma_{\mathbf{u_i}})(\lambda_{\mathbf{v_j}} \lambda_{\mathbf{v_j}}^{\mathbf{T}} + \gamma_{\mathbf{v_j}})\Big) +\\ &\mathbf{Y_{ij}(I) Tr}\Big((\lambda_{\mathbf{u_i}} \lambda_{\mathbf{u_i}}^{\mathbf{T}} + \gamma_{\mathbf{u_i}})(\lambda_{\mathbf{v_j}} \lambda_{\mathbf{v_j}}^{\mathbf{T}} + \gamma_{\mathbf{v_j}})\Big),\end{aligned}$$

where $\gamma_{\mathbf{u_i,dd}}$ denotes the cell in dth row and dth column of $\gamma_{\mathbf{u_i}}$. The update rules for $\lambda_{\mathbf{v_j}}$ and $\gamma_{\mathbf{v_j}}$ have the same form. The VE-step then proceeds by alternately updating λ, γ and ξ until convergence. Note that the update rules for domain $\mathbf{R^O}$ is exactly the same except that there is no ξ in $\mathbf{R^O}$ because the ratings in $\mathbf{R^O}$ are homogeneous to the ratings in $\mathbf{R}$ and hence are modeled by Gaussian distribution.

Variational M-step: Next we move on to the VM-step. In VM-step we fix all three sets of variational parameters that we have obtained from VE-step, and optimize the bound in Equation 2 w.r.t. the model parameters $\Theta = \{\mathbf{m_u}, \mathbf{\Sigma_u}, \mathbf{m_v}, \mathbf{\Sigma_v}, \sigma^2, \omega^2\}$. This is done by taking derivatives of Θ in Equation 2 where all expectations can be done in the same way as what we have described in VE-step. Similar to VE-step, we introduce the trade-off parameters β that controls the mixing weights between $(\mathbf{R}, \mathbf{R^U}, \mathbf{R^I})$ and $\mathbf{R^O}$ to update the parameters for prior distributions.

The optimal mean and covariance for Gaussian prior in VM-step take the following forms,

$$\mathbf{m_u} = \frac{1}{\mathbf{N} + \beta\mathbf{L}}\Big(\sum_{\mathbf{i=1}}^{\mathbf{N}}\lambda_{\mathbf{u_i}} + \beta\sum_{\mathbf{l=1}}^{\mathbf{L}}\lambda_{\mathbf{u_l}}\Big)$$

$$\mathbf{m_v} = \frac{1}{\mathbf{M} + \beta\mathbf{S}}\Big(\sum_{\mathbf{j=1}}^{\mathbf{M}}\lambda_{\mathbf{v_j}} + \beta\sum_{\mathbf{s=1}}^{\mathbf{S}}\lambda_{\mathbf{v_s}}\Big)$$

$$\mathbf{\Sigma_u} = \frac{1}{\mathbf{N} + \beta\mathbf{L}}\Bigg[\sum_{\mathbf{i=1}}^{\mathbf{N}}\Big(\gamma_{\mathbf{u_i}} + (\lambda_{\mathbf{u_i}} - \mathbf{m_u})(\lambda_{\mathbf{u_i}} - \mathbf{m_u})^{\mathbf{T}}\Big) + \beta\sum_{\mathbf{l=1}}^{\mathbf{L}}\Big(\gamma_{\mathbf{u_l}} + (\lambda_{\mathbf{u_l}} - \mathbf{m_u})(\lambda_{\mathbf{u_l}} - \mathbf{m_u})^{\mathbf{T}}\Big)\Bigg]$$

$$\mathbf{\Sigma_v} = \frac{1}{\mathbf{M} + \beta\mathbf{S}}\Bigg[\sum_{\mathbf{j=1}}^{\mathbf{M}}\Big(\gamma_{\mathbf{v_j}} + (\lambda_{\mathbf{v_j}} - \mathbf{m_v})(\lambda_{\mathbf{v_j}} - \mathbf{m_v})^{\mathbf{T}}\Big) + \beta\sum_{\mathbf{s=1}}^{\mathbf{S}}\Big(\gamma_{\mathbf{v_s}} + (\lambda_{\mathbf{v_s}} - \mathbf{m_v})(\lambda_{\mathbf{v_s}} - \mathbf{m_v})^{\mathbf{T}}\Big)\Bigg]$$

$$\sigma^2 = \frac{1}{A}\sum_{\mathbf{i=1}}^{\mathbf{N}}\sum_{\mathbf{j=1}}^{\mathbf{M}}\mathbf{Y_{ij}(T)}\Big(\mathbf{R_{ij}^2} + \lambda_{\mathbf{u_i}}^{\mathbf{T}}\gamma_{\mathbf{v_j}}\lambda_{\mathbf{u_i}} + \lambda_{\mathbf{v_j}}^{\mathbf{T}}\gamma_{\mathbf{u_i}}\lambda_{\mathbf{v_j}} - \mathbf{2R_{ij}}\lambda_{\mathbf{u_i}}^{\mathbf{T}}\lambda_{\mathbf{v_j}} + (\lambda_{\mathbf{u_i}}^{\mathbf{T}}\lambda_{\mathbf{v_j}})^{\mathbf{2}} + \mathbf{Tr}(\gamma_{\mathbf{u_i}}\gamma_{\mathbf{v_j}})\Big)$$

$$\omega^2 = \frac{1}{W}\sum_{\mathbf{l=1}}^{\mathbf{L}}\sum_{\mathbf{s=1}}^{\mathbf{S}}\mathbf{Y_{ls}(O)}\Big(\mathbf{R_{ls}^{O^2}} + \lambda_{\mathbf{u_l}}^{\mathbf{T}}\gamma_{\mathbf{v_s}}\lambda_{\mathbf{u_l}} + \lambda_{\mathbf{v_s}}^{\mathbf{T}}\gamma_{\mathbf{u_l}}\lambda_{\mathbf{v_s}} - \mathbf{2R_{ls}^O}\lambda_{\mathbf{u_l}}^{\mathbf{T}}\lambda_{\mathbf{v_s}} + (\lambda_{\mathbf{u_l}}^{\mathbf{T}}\lambda_{\mathbf{v_s}})^{\mathbf{2}} + \mathbf{Tr}(\gamma_{\mathbf{u_l}}\gamma_{\mathbf{v_s}})\Big)$$

where A and W are the total number of observed ratings in $\mathbf{R}$ and $\mathbf{R^O}$ respectively.

The complete optimization algorithm then proceeds by first initializing variational parameters, and alternating between VE-step and VM-step until some stopping criterion is met. Usually the stopping criterion would be the relative improvement on the *bound* of the log-likelihood. However, evaluating that bound requires non-trivial computation; hence in our experiments we set a fixed number for VE-step as well as the whole VEM algorithm, but we stop when RMSE on the validation set starts to increase. More details about the experiments will be given in Section 5.

C. Prediction

To predict missing ratings in $\mathbf{R}$, we find the optimal $\mathbf{R_{ij}}$ which maximizes the bound in Equation 2 by taking the derivatives w.r.t. $\mathbf{R_{ij}}$ for a given user i and item j. It turns out that the prediction has exactly the same form as in matrix factorization: $\mathbf{\hat{R}_{ij}^T} = \lambda_{\mathbf{u_i}}^{\mathbf{T}}\lambda_{\mathbf{v_j}}$, although the optimization objective is very different.

D. Automatic Complexity Control

In standard matrix factorization model such as SVD, choosing regularization parameters is critical to prevent over-fitting. The usual way is to choose a fixed regularization value by doing an exhaustive search in some pre-specified range using a validation set. Such method is very time consuming as we need to train multiple models in order to select the best one. Also, past experiences showed that such parameters are quite sensitive that using a regularizer that is too strong prevents the model from learning meaningful things, while using a weak regularizer causes the model to overfit [9].

In our model, the complexity is controlled automatically via the shared Gaussian prior. If we fix the prior and do not do any learning on that, we will go back to the same issue as how to choose the mean and covariance of the prior. However, because our prior is adaptive to the training data, we do not need to fix it beforehand, instead we can learn it through training. Moreover, since in many practical situations we have denser auxiliary data, the prior parameters will automatically balance between modeling the target data and auxiliary data that we do not need to worry about complexity control in our model. Also, update rules in variational EM are all closed forms with no need to tune the learning rate or perform line search.

V. Experiments

A. Data Sets and Evaluation Metrics

There are in total of four datasets used in our experiments, namely, *Netflix* [4], *Movielens* [5], *Book-Crossing* [6] and *EachMovie* [7], that are all popular datasets for collaborative filtering. The *Netflix* dataset contains about 10^8 integer rating values in the range $\{1, 2, 3, 4, 5\}$ given by about 7×10^4 users on around 1.7×10^4 movies. The *Movielens* dataset contains about 10^7 rating values 1-5, rated by about 7×10^4 users on around 10^4 movies. *Book-Crossing* is a dataset for book recommendation with about 2.8×10^5 users and 2.7×10^5 books. Finally, *EachMovie* contains approximately 2.8×10^6 ratings given by 7.2×10^4 users on 1628 movies.

For *EachMovie* and *Book-Crossing*, we first normalize their rating values to be in the range $\{1, 2, 3, 4, 5\}$. The evaluation metrics used in this paper is Root Mean Square Error (RMSE),

$$RMSE = \sqrt{\frac{\sum_{\mathbf{i}}^{\mathbf{N}}\sum_{\mathbf{j}}^{\mathbf{M}}(\mathbf{R_{ij}^{test}} - \mathbf{\hat{R}_{ij}^{test}})^{\mathbf{2}} \odot \mathbf{Y_{ij}(T)}}{W}},$$

where W is the number of observed ratings in the testing set $\mathbf{R^{test}}$ and $\mathbf{\hat{R}}$ is the predicted ratings given by the algorithms.

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

[5]http://www.grouplens.org/node/73

[6]http://www.informatik.uni-freiburg.de/ cziegler/BX/

[7]http://www.cs.cmu.edu/ lebanon/IR-lab/data.html

B. Implementation Details

For the implementation of TPCF, there are few tunable parameters: α, β and k which is the rank of matrix factors. Throughout our experiments, we keep k at 10 unless otherwise specified. For parameter α, we perform a grid search in $\{0.1, 0.3, 0.5, 1\}$, and for parameter β, $\{0.1, 0.5, 1, 3, 5\}$ are tried. More experiments on the effects of α and β will be presented later.

For VEM optimization algorithm, the number of VE-steps is set to 20, and the total number of iterations of VEM is set to 20 as well. For each iteration of VEM, we record the RMSE on the validation set and we stop when this number starts to increase. The final parameters for α and β were selected by choosing the one that performs the best on the validation set in terms of RMSE. Furthermore, we preprocessed the data on $\mathbf{R}$ and $\mathbf{R^O}$ by subtracting each rating with the mean of all ratings. This step is necessary to achieve good results when using Gaussian distribution to model rating values.

C. Compared Algorithms

We compare our model with three transfer learning algorithms introduced in Section 2: *Collective Matrix Factorization with L2 regularization* (CMF-L2) [13], *Coordinate System Transfer* (CST) [6], and *Rating-Matrix Generative Model* (RMGM) [4]. We have implemented CMF-L2 and CST in MATLAB. For RMGM, the authors have provided example code [8] and we modify it for the purpose of this paper. Note that CMF-L2 and CST cannot use data in $\mathbf{R^O}$ because they require knowing explicitly the correspondences for either users or items in order to transfer knowledge. Also, RMGM cannot use data in $\mathbf{R^U}$ and $\mathbf{R^I}$ as it requires the matrix to have homogeneous ratings. Nevertheless, we can still compare our model with these algorithms assuming our model only observes partial data used in the corresponding scenario.

In the following section, we will first show some analysis of TPCF alone, and comparisons of previous state-of-the-art will be detailed later.

D. Results

1) Learning Netflix with MovieLens: First, we take *Netflix* and *MovieLens* datasets to conduct this part of the experiment. Here the goal is to predict missing values in *Netflix* (target domain) and *MovieLens* is only used to construct $\mathbf{R^O}$. In order to evaluate the effectiveness of using auxiliary data, we perform the following pre-processing steps:

1) Randomly extract a 4000×4000 dense rating matrix X from *Netflix data*, and take a sub-matrix $X_{1:2000,1:2000}$ as the target matrix $\mathbf{R}$.
2) Take a sub-matrix $X_{1:2000,2001:4000}$ as the auxiliary matrix $\mathbf{R^U}$ with shared users. Clearly, $\mathbf{R^U}$ and $\mathbf{R}$ only share common users but not common items.
3) Take a sub-matrix $X_{2001:4000,1:2000}$ as the auxiliary matrix $\mathbf{R^I}$ with shared items. Clearly, $\mathbf{R^I}$ and $\mathbf{R}$ only share common items but not common users.
4) Split the target matrix $\mathbf{R}$ into disjoint sets $\mathbf{R^{train}}, \mathbf{R^{val}}$, and $\mathbf{R^{test}}$ as training set, validation set and testing set respectively.
5) To simulate implicit user feedbacks, we pre-process $\mathbf{R^U}$ and $\mathbf{R^I}$ by relabeling ratings in the range $\{1, 2, 3\}$ as 0 and ratings in the range $\{4, 5\}$ as 1 [11] .
6) Finally, we randomly select 2000×2000 dense sub-matrix from *Movielens* data as $\mathbf{R^O}$ to simulate what happens when we have other CF task without making any assumption on the correspondence of users/items between $(\mathbf{R}, \mathbf{R^U}, \mathbf{R^I})$ and $\mathbf{R^O}$.

We make our training data in target domain very scarce. The sparsity levels for each of the matrices are listed in Table 2. For training set $\mathbf{R^{train}}$, we start from an extremely sparse case with each user having only 1 rating, and gradually add observed ratings.

Data	Data Source	Domain	Form	Sparsity
$\mathbf{R^{train}}$	*Netflix*	target	1-5	$\leq$ 1.00%
$\mathbf{R^{val}}$	*Netflix*	target	1-5	1.00%
$\mathbf{R^{test}}$	*Netflix*	target	1-5	3.48%
$\mathbf{R^U}$	*Netflix*	aux	0/1	6.00%
$\mathbf{R^I}$	*Netflix*	aux	0/1	6.00%
$\mathbf{R^O}$	*MovieLens*	aux	1-5	6.00%

TABLE II: Statistics of matrices considered in *Netflix-MovieLens* task. The sparsity levels of $\mathbf{R^U}, \mathbf{R^I}, \mathbf{R^O}$ are purposely set to be identical so that we can make fair observation about which type of auxiliary data is most useful.

For this part of the experiment, we focus on our method alone. In particular, we want to know which auxiliary domain contributes the most to the prediction improvement. Hence we start from the single-task version of TPCF without using any data other than $\mathbf{R}$ (i.e. VBMF [5], [10]), and gradually add auxiliary information from $\mathbf{R^O}, \mathbf{R^U}$ and $\mathbf{R^I}$ to examine the performance gain.

We run the tests with a set of $\mathbf{R^{train}}$ having various levels of sparsity. We start with the case where each user in $\mathbf{R^{train}}$ has rated only *one* item. Then we gradually increase the number of ratings for each user. The result in terms of RMSE is shown in Fig 3 where we denote $|\mathbf{R}|_\mathbf{u}$ as the number of ratings that each user has provided in $\mathbf{R^{train}}$. The trend is very clear

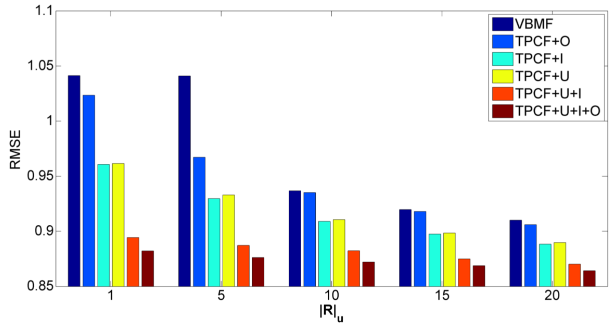

Fig. 3: RMSE on *Netflix* dataset with different types of information from auxiliary domains and different $|\mathbf{R}|_\mathbf{u}$, that is, the number of observed ratings for each user in $\mathbf{R^{train}}$. The matrix $\mathbf{R^O}$ is sampled from *MovieLens* dataset

that at the beginning, when $|\mathbf{R}|_\mathbf{u}$ is small, VBMF does poorly because there is not enough information to learn the model. For

[8]https://sites.google.com/site/libin82cn/

$\mathbf{TPCF + O}$, i.e. using data from *MovieLens* to learn a better prior distribution for users and items, the improvements are significant when $|\mathbf{R}|_\mathbf{u} \leq \mathbf{5}$, albeit the amount of improvement gradually decreases when $|\mathbf{R}|_\mathbf{u} \geq \mathbf{10}$. When we use $\mathbf{R^U}$ or $\mathbf{R^I}$ alone, the RMSE reduces greatly as expected. When using both $\mathbf{R^U}$ and $\mathbf{R^I}$, a huge reduction in RMSE is further obtained. Interestingly, even though much information from $\mathbf{R^U}$ and $\mathbf{R^I}$ has already been brought into the model, there is still some improvement when we include $\mathbf{R^O}$, which essentially just affects the prior distribution. This result shows that the parameters of prior are critical in having a good prediction results, and our model can automatically learn the priors even though users and items are not well-aligned.

2) Learning EachMovie with MovieLens: Second, similar to the previous experiment, we take *EachMovie* and *MovieLens* datasets and the goal is to predict missing values in *EachMovie* (target domain). All preprocessing steps for this experiment is the same as in *Netflix-MovieLens* experiment. The dimensions of $\mathbf{R}, \mathbf{R^U}$ and $\mathbf{R^I}$ are all 1000×800; and we sample *MovieLens* randomly to select 1000 users and 800 items to form $\mathbf{R^O}$. Statistics are shown in Table 3:

Data	Data Source	Domain	Form	Sparsity
$\mathbf{R^{train}}$	*EachMovie*	target	1-5	$\leq$ 2.50%
$\mathbf{R^{val}}$	*EachMovie*	target	1-5	1.25%
$\mathbf{R^{test}}$	*EachMovie*	target	1-5	4.73%
$\mathbf{R^U}$	*EachMovie*	aux	0/1	5.00%
$\mathbf{R^I}$	*EachMovie*	aux	0/1	5.00%
$\mathbf{R^O}$	*MovieLens*	aux	1-5	5.00%

TABLE III: Statistics of matrices considered in *EachMovie-MovieLens*.

The experiment result is shown in Fig 4, and we observe similar trends as in *Netflix-MovieLens* experiment.

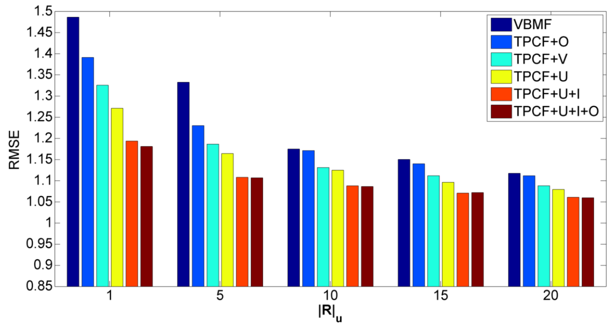

Fig. 4: RMSE on *EachMovie* dataset with different types of information from auxiliary domains and different $|\mathbf{R}|_\mathbf{u}$. The matrix $\mathbf{R^O}$ is sampled from *MovieLens* dataset.

3) Comparison with Related Algorithms: In this part of experiment, we compare TPCF with several algorithms to show its effectiveness on *Netflix-MovieLens* and *EachMovie-MovieLens* tasks with all preprocessing steps being the same as described in last two subsections. We compare against methods introduced in Section 5.3, i.e. CMF-L2, CST, and RMGM. CST is proven to be more powerful than CMF-L2 in the original paper because CST is a tri-factorization method [6], hence we consider CST as the state-of-the-art benchmark model in the scenario where we have $\mathbf{R^U}$ and $\mathbf{R^I}$ at hand. RMGM, on the other hand is the state-of-the-art cross-domain learning algorithm considering situation that there are no correspondence between users/items in the target domain and auxiliary domains. All parameters in these three models are also selected using the validation set.

We report the results with latent dimension $k = \{10, 50\}$ shown in Table 4 and 5. Here the experiments are repeated five times with random initialization of the parameters. Comparing TPCF+O to RMGM, using exactly the same data for both models, our model outperforms RMGM with a big margin. Also, comparing TPCF+U+I to CMF-L2 and CST with the same set of data, our model shows significant improvements over the other methods for all the chosen sparsity levels. Moreover, as mentioned before, our probabilistic model can effectively ensemble all three sources of auxiliary data, shown as TPCF+U+I+O, to achieve further improvement.

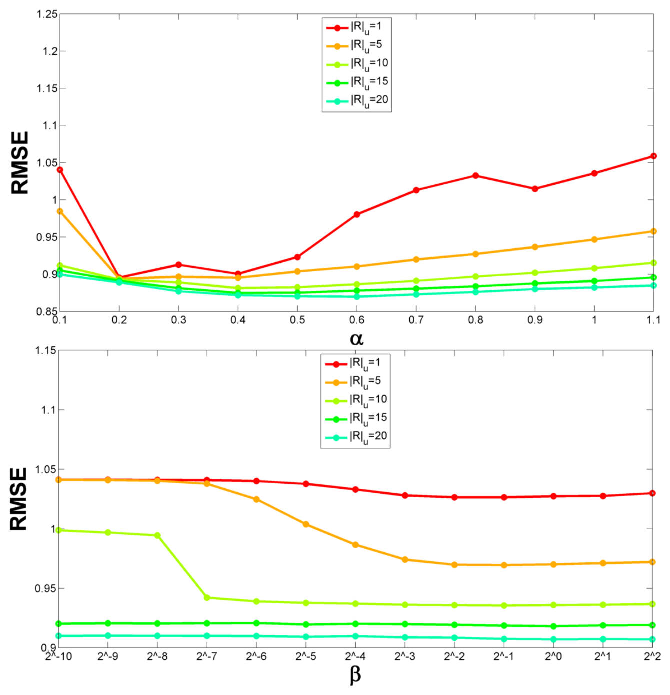

Fig. 5: RMSE on *Netflix* dataset with $\mathbf{TPCF}$. **Top**: varying value of α; **Bottom**: varying value of β.

4) The Effects of α and β: Here we test how changes in α and β affect the performance. α is the mixing weight that controls relative importance of domains $\mathbf{R}$ and $(\mathbf{R^U}, \mathbf{R^I})$, whereas β is the trade-off parameter between $(\mathbf{R}, \mathbf{R^U}, \mathbf{R^I})$ and $\mathbf{R^O}$. To reveal the effect of choosing the right α and β, we fix the value of one of them and vary the other to see the change in performance. We show the plots for *Netflix-MovieLens* and *EachMovie-MovieLens* tasks in Fig 5 and 6.

5) Learning Books from Movies: Previous experiments are conducted on domains with the same kind of items (movies). It seems reasonable to assume that we can place the same prior distribution over the latent space for items because they indeed belong to the same class of items. In this part, we take *Book-Crossing* as target matrix $\mathbf{R}$ and *Netflix* as auxiliary matrix $\mathbf{R^O}$

Netflix-MovieLens					
k = 10	$\mathbf{\|R\|_u = 1}$	$\mathbf{\|R\|_u = 5}$	$\mathbf{\|R\|_u = 10}$	$\mathbf{\|R\|_u = 15}$	$\mathbf{\|R\|_u = 20}$
RMGM	1.0435±0.0013	1.0097±0.0011	0.9638±0.0026	0.9389±0.0010	0.9269±0.0035
TPCF+O	**1.0263±0.0004**	**0.9695±0.0007**	**0.9356±0.0004**	**0.9182±0.0012**	**0.9070±0.0009**
CMF-L2	0.9894±0.0027	0.9514±0.0031	0.9264±0.0045	0.9107±0.0040	0.9023±0.0029
CST	0.9352±0.0009	0.9057±0.0011	0.8975±0.0010	0.8925±0.0009	0.8792±0.0017
TPCF+U+I	**0.9126±0.0010**	**0.8965±0.0007**	**0.8824±0.0005**	**0.8750±0.0008**	**0.8703±0.0010**
TPCF+U+I+O	**0.8823±0.0002**	**0.8763±0.0003**	**0.8723±0.0002**	**0.8689±0.0001**	**0.8643±0.0003**
k = 50	$\mathbf{\|R\|_u = 1}$	$\mathbf{\|R\|_u = 5}$	$\mathbf{\|R\|_u = 10}$	$\mathbf{\|R\|_u = 15}$	$\mathbf{\|R\|_u = 20}$
RMGM	1.0443±0.0019	1.0104±0.0012	0.9613±0.0014	0.9394±0.0020	0.9284±0.0017
TPCF+O	**1.0265±0.0013**	**0.9705±0.0015**	**0.9360±0.0015**	**0.9182±0.0012**	**0.9059±0.0014**
CMF-L2	1.0427±0.0019	1.0015±0.0025	0.9719±0.0020	0.9545±0.0038	0.9442±0.0017
CST	0.9610±0.0022	0.9453±0.0024	0.9274±0.0020	0.9016±0.0020	0.8987±0.0031
TPCF+U+I	**0.9338±0.0012**	**0.9044±0.0011**	**0.8929±0.0011**	**0.8843±0.0015**	**0.8766±0.0018**
TPCF+U+I+O	**0.8919±0.0016**	**0.8860±0.0014**	**0.8829±0.0014**	**0.8737±0.0016**	**0.8671±0.0016**

TABLE IV: RMSE on *Netflix*: auxiliary domains include $\mathbf{R^U}, \mathbf{R^I}$ and $\mathbf{R^O}$ as described in section 5.4.1. Each algorithm utilizes all applicable auxiliary data.

EachMovie-MovieLens					
k = 10	$\mathbf{\|R\|_u = 1}$	$\mathbf{\|R\|_u = 5}$	$\mathbf{\|R\|_u = 10}$	$\mathbf{\|R\|_u = 15}$	$\mathbf{\|R\|_u = 20}$
RMGM	1.4554±0.0025	1.2963±0.0022	1.2210±0.0028	1.1984±0.0018	1.1827±0.0031
TPCF+O	**1.4004±0.0011**	**1.2310±0.0013**	**1.1716±0.0013**	**1.1408±0.0016**	**1.1140±0.0012**
CMF-MAP	1.3424±0.0014	1.2361±0.0026	1.1696±0.0010	1.1406±0.0011	1.1180±0.0020
CST	1.2621±0.0016	1.1707±0.0026	1.1483±0.0020	1.1371±0.0011	1.1159±0.0013
TPCF+U+I	**1.1936±0.0012**	**1.1079±0.0014**	**1.0877±0.0014**	**1.0706±0.0013**	**1.0609±0.0012**
TPCF+U+I+O	**1.1807±0.0012**	**1.1067±0.0017**	**1.0862±0.0018**	**1.0720±0.0015**	**1.0599±0.0014**
k = 50	$\mathbf{\|R\|_u = 1}$	$\mathbf{\|R\|_u = 5}$	$\mathbf{\|R\|_u = 10}$	$\mathbf{\|R\|_u = 15}$	$\mathbf{\|R\|_u = 20}$
RMGM	1.4418±0.0020	1.2812±0.0053	1.2175±0.0033	1.1901±0.0037	1.1750±0.0035
TPCF+O	**1.4016±0.0010**	**1.2526±0.0009**	**1.1759±0.0014**	**1.1439±0.0012**	**1.1195±0.0008**
CMF-MAP	1.3735±0.0020	1.2655±0.0018	1.1962±0.0033	1.1568±0.0024	1.1302±0.0019
CST	1.2838±0.0009	1.1912±0.0012	1.1557±0.0019	1.1408±0.0021	1.1299±0.0022
TPCF+U+I	**1.1705±0.0007**	**1.1113±0.0009**	**1.0863±0.0012**	**1.0688±0.0014**	**1.0594±0.0009**
TPCF+U+I+O	**1.1660±0.0010**	**1.1080±0.0009**	**1.0847±0.0018**	**1.0686±0.0015**	**1.0548±0.0011**

TABLE V: RMSE on *EachMovie*: auxiliary domains include $\mathbf{R^U}, \mathbf{R^I}$ and $\mathbf{R^O}$ as described in Section 5.4.2. Each algorithm utilizes all applicable auxiliary data.

to see what will happen when item sets do not belong to the same kinds. RMGM was proposed to solve this specific kind of cross-domain collaborative filtering problem by capturing similar cluster patterns. Our model, on the other hand, does not use cluster-level patterns. Instead, we use auxiliary data to learn a better prior distribution for users and items in the target domain without knowing any correspondences of users and items from different domains. We believe that down to a lower-dimensional space (such as 10-D or 50-D), we can still capture the semantic relations between books and movies (such as genre) to perform predictions well in the target domain.

To see whether we can learn a better model for book recommendation using auxiliary data from a set of ratings on movies, we first randomly select 500 users and 500 books with the most ratings from *Book-Crossing* dataset as target data. Then we normalize the ratings to be in the range $\{1, 2, 3, 4, 5\}$. Next, we sample a 2000×2000 sub-matrix from *Netflix* dataset as auxiliary data with density 6%.

In this part of experiment, we only compare with RMGM because CST and CMF-L2 could not handle data without knowing correspondence information explicitly. The dimension of k is fixed at 10. We report the RMSE with varying β for our model and the RMSE of RMGM in Fig 8. We also compare to a non-transfer version of RMGM that only uses target domain as training data. We see that a transfer version of RMGM learning with data from *Netflix* actually degrades the model performance [9]. To investigate the cause of this result, we study the rating patterns in these two datasets. We treat the observed ratings from *Book-Crossing* and *Netflix* as two categorical distribution with values in the range $\{1, 2, 3, 4, 5\}$, then we compute the Kullback–Leibler divergence between the two datasets. The result is 0.3251. Computing the KL divergence between training and testing ratings from *Book-Crossing* dataset gives us a value of 0.0016. This shows a clear distribution mismatch, indicating inconsistent rating behaviors between users in *Book-Crossing* and *Netflix* datasets. We believe this is the core reason why a transfer version of RMGM performs poorly. Because rating patterns across domains are

[9]In the original implementation of RMGM provided by the authors, they use categorical distribution for generating data. We have therefore also implemented a real-value version of RMGM using Gaussian distribution to generate data. However, we still could not get a better result than non-transfer version of RMGM.

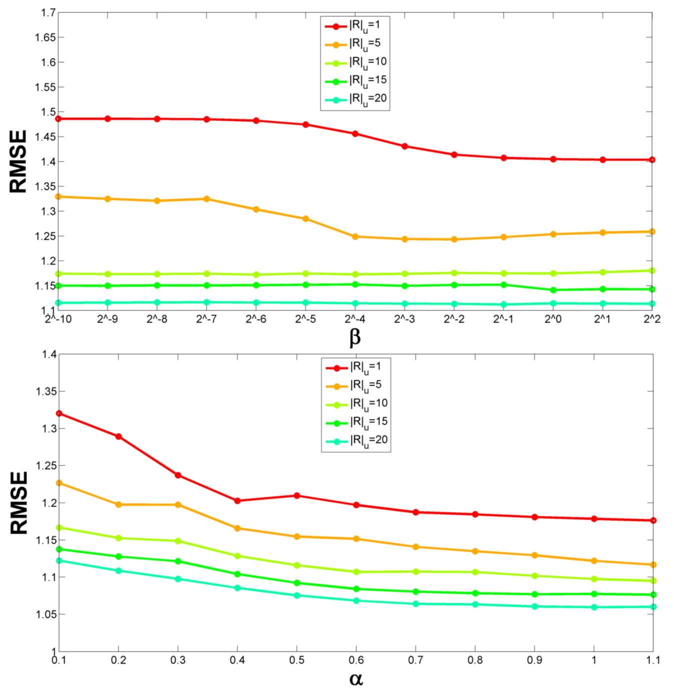

Fig. 6: RMSE on *EachMovie* dataset with **TPCF**. **Top**: varying value of α; **Bottom**: varying value of β

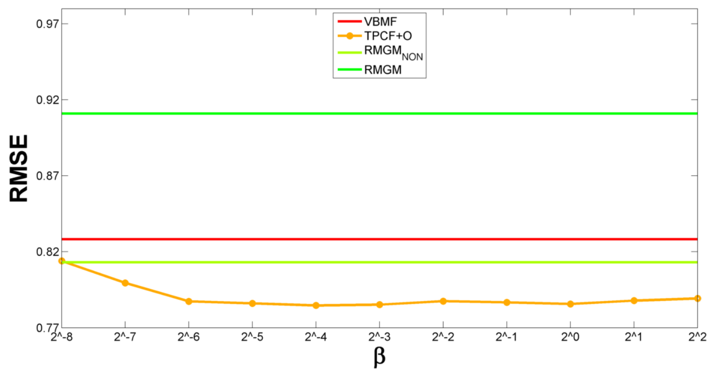

Fig. 7: Testing RMSE on *Book-Crossing* of **TPCF** with varying β and RMGM. RMGM$_{NON}$ means learning RMGM model without using auxiliary data from *Netflix*.

very different, cluster-level patterns from *Netflix* do not help alleviate sparsity issue in *Book-Crossing* at all. Our model, on the other hand, does not rely on explicit rating patterns. We instead use an adaptive shared prior to automatically leverage the data from the two domains. From the plot we see that it indeed helps a lot as the RMSE reduces from 0.8283, with non-transfer VBMF to **0.7861**. For β greater than 2^{-6}, we get about the same performance, showing the insensitivity of choosing exact value of β.

VI. Conclusion

A newly launched CF application usually contains scarce rating values, and such cold-start phenomenon prevents non-transfer learning algorithm from accurately capturing users/items preferences. In this paper, we propose a robust method to alleviate such problem with TPCF to transfer knowledge from related domains in a unified sense. As shown in the experiments, our method is particularly useful when there are extremely scarce data in the target domain, as the improvements are significant compared to other benchmark algorithms.

VII. Acknowledgement

We thank Chung-Yi Li for the implementation of real-value version of RMGM. This study is also under the "Online and Offline integrated Smart Commerce Platform(1/4)" project of the Institute for Information Industry which is subsidized by the Ministry of Economy Affairs of the Republic of China.

References

[1] J. M. Bernardo, M. J. Bayarri, J. O. Berger, A. P. Dawid, D. Heckerman, A. F. M. Smith, M. West (eds, Matthew J. Beal, and Zoubin Ghahramani. The variational bayesian em algorithm for incomplete data: with application to scoring graphical model structures, 2003.

[2] T.S. Jaakkola and M.I. Jordan. A variational approach to bayesian logistic regression models and their extensions, 1996.

[3] B. Li, Q. Yang, and X. Xue. Can movies and books collaborate?: Cross-domain collaborative filtering for sparsity reduction. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, pages 2052–2057, 2009.

[4] B. Li, Q. Yang, and X. Xue. Transfer learning for collaborative filtering via a rating-matrix generative model. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 617–624, 2009.

[5] Y. J. Lim and Y. W. Teh. Variational Bayesian approach to movie rating prediction. In *Proceedings of KDD Cup and Workshop*, 2007.

[6] W. Pan, and N. Liu E. Xiang, and Q. Yang. Transfer learning in collaborative filtering for sparsity reduction. In *Proceedings of the 24th Association for the Advancement of Artificial Intelligence*, pages 617–624, 2010.

[7] W. Pan, N. Liu, E. Xiang, and Q. Yang. Transfer learning to predict missing ratings via heterogeneous user feedbacks. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 2318–2323, 2011.

[8] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, pages 175–186, 1994.

[9] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems*, volume 20, 2008.

[10] H. Shan and A. Banerjee. Generalized probabilistic matrix factorizations for collaborative filtering. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, pages 1025–1030, 2010.

[11] V. Sindhwani, S. S. Bucak, J. Hu, and A. Mojsilovic. A family of non-negative matrix factorizations for one-class collaborative filtering problems, 2009.

[12] A. Singh and G.J. Gordon. A bayesian matrix factorization model for relational data. In *UAI*, pages 556–563, 2010.

[13] A.P. Singh and G.J. Gordon. Relational learning via collective matrix factorization. In *Proceedings of the 14th ACM SIGKDD*, pages 650–658, 2008.

[14] E.P. Xing, M.I. Jordan, and S. Russell. A generalized mean field algorithm for variational inference in exponential families. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, pages 583–591, 2003.