Top-N Recommendations by Learning User Preference Dynamics

Yongli Ren, Tianqing Zhu, Gang Li*, and Wanlei Zhou

School of Information Technology, Deakin University,
221 Burwood Highway, Vic 3125, Australia
{yongli,ztianqin,gang.li,wanlei}@deakin.edu.au

Abstract. In a recommendation system, user *preference patterns* and the *preference dynamic effect* are observed in the *user* × *item* rating matrix. However, their value has barely been exploited in previous research. In this paper, we formalize the preference pattern as a sparse matrix and propose a *Preference Pattern Subspace* to iteratively model the *personal* and the *global* preference patterns with an EM-like algorithm. Furthermore, we propose a *PrepSVD-I* algorithm by transforming the Top-N recommendation as a pairwise preference learning process. Experiment results show that the proposed *PrepSVD-I* algorithm significantly outperforms the state-of-the-art Top-N recommendation algorithms.

1 Introduction

Although recommendation system research has seen the development of techniques about *rating prediction*, the majority of commercial recommender systems aims to generate a list of recommended items, which is the task of *Top-N recommendation* [9].

Various techniques have been proposed for Top-N recommendations. Most of them are based on the modelling of user rating patterns by analysing the *user* × *item* rating matrix. These methods show improved performance, but their abilities in Top-N recommendations are still limited by the availability of user ratings. Specifically, most of the available ratings are given to a small fraction of items, and this is known as the *long tail effect* [2]. As shown in Fig. 1c, 33% of ratings are observed from only around 5.5% of items in the *MovieLens* data set. These items are referred as *popular* items, while the other 94.5% are referred as *unpopular* or *long tail* items [4]. Thus, the *long tail effect* indicates that ratings on *long tail* items are much fewer. Consequently, these analysing methods for rating patterns are naturally limited by the *long tail effect*.

In this work, we observe that each user has a *preference pattern* that is different from his/her rating pattern, and the *preference pattern* tends to change over time. For example, Fig. 1a and 1b show the user *preference patterns* and the temporal dynamics observed on a real movie recommender system, *MovieLens*. Specifically, Fig. 1a shows that fresh users tend to rate movies from a larger

* Corresponding author.

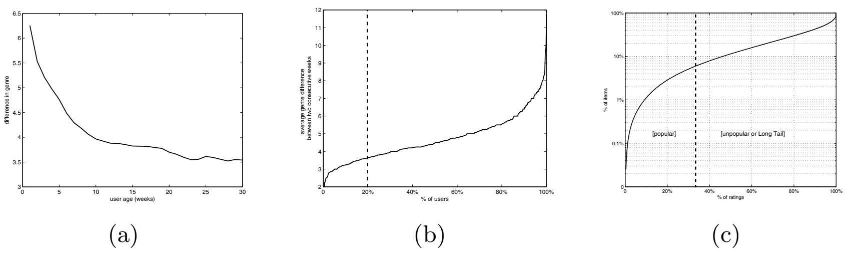


Fig. 1. Preference dynamics in *MovieLens*. a) genre difference by user age; b) user distribution over genre difference; c) rating distribution over item popularity.

range of genres than experienced users. Fig. 1b shows that, about 80% of users rated movies that spread over at least 3.5 genres on average during every two consecutive weeks. These observations indicate the existence of patterns on user preference styles, as well as their dynamics. In this paper, we name this new effect as the *preference dynamic effect.* Please note that the temporal characteristic in user *preference patterns* is different from the one observed by Koren [12], which is the temporal dynamics in user *rating patterns.*

In this paper, we focus on the modelling of the *preference dynamic effect* with the *Preference Pattern Subspace.* The basic idea is to model the *user preference styles* and their temporal dynamics by constructing a low-rank subspace. Firstly, a low-rank subspace is built to capture the *global preference patterns* for all users; then, the projection for each *personal preference pattern* on the subspace is individually refined based on his/her own preference styles. After that, the refined user projections on the subspace are used to improve the modelling of the *global preference patterns.* Iteratively, we can obtain a well-trained low-rank subspace to model both the *user preference styles* and their temporal dynamics. Based on the model, we formulate Top-N recommendation as a pairwise preference learning process, and propose a *PrepSVD-I* algorithm. Experimental results show that *PrepSVD-I* significantly outperforms the state-of-the-art Top-N recommendation techniques, especially when recommending *long tail* items. The contributions of this paper are as follows:

- For the first time, the *preference dynamic effect* is proposed to capture the personal and temporal characteristics of user preferences.
- We propose a novel *Preference Pattern* model and a subspace approach, *Preference Pattern Subspace*, to model the *preference dynamic effect.*
- Based on the *Preference Pattern Subspace*, we formulate Top-N recommendation as a pairwise preference learning process.

The rest of this paper is organized as follows. In Section 2, the *Preference Pattern* is proposed. In Section 3, we present *Preference Pattern Subspace.* We present the results of the experiment in Section 4, and the conclusion in Section 5.

2 The Preference Pattern

In this section, we propose a novel *Preference Pattern* model to capture the *preference dynamic effect*. Notations used in this paper are summarized in Table 1.

Table 1. Symbols

Symbol	Description
u_a	the active user for whom the recommendations are generated
r_{xl}	the rating on an item t_l by user u_x
$\mathcal{R}$	the *user* $\times$ *item* rating matrix
$T_N(u_x)$	the list of Top-N items recommended to user u_x
$\mathcal{U} = \{u_1, \cdots, u_m\}$	a set of m users
$\mathcal{T} = \{t_1, \cdots, t_n\}$	a set of n items
$\mathcal{C} = \{c_1, \cdots, c_q\}$	a set of q categories
$\mathbf{p}_x$	the *preference pattern* for user u_x
$\mathcal{P} = \{\mathbf{p}_1, \cdots, \mathbf{p}_m\}$	the *preference patterns* for m users
$\mathbf{v}_x$	the projection of $\mathbf{p}_x$
$\mathcal{V} = \{\mathbf{v}_1, \cdots, \mathbf{v}_m\}$	the projection of $\mathcal{P}$ on a low-rank subspace

Definition 1. *A* preference pattern *is a sequence of personal preference styles aligned in a time order. Precisely, for user u_x, the* preference pattern *is denoted as $\mathbf{p}_x = [p_{x1}, \cdots, p_{xi}]^T$, where $p_{xi} = [p^1_{xi}, \cdots, p^q_{xi}]$ denotes the preferences of u_x at time i over category $\mathcal{C} = [c_1, \cdots, c_q]$, and p^j_{xi} denotes the preference of u_x at time i on c_j.*

The *preference pattern* has two key characteristics, *personalization* and *time*. All preference styles within a *preference pattern* come from the same user, and are sorted in a time order. For a particular user u_x, a *preference style* refers to his/her preferences over a range of categories (e.g. *genre* in movies/songs) of items at a particular time. The preference style at time i can be represented as a q-D vector p_{xi}, which is defined as a *preference pattern vector*. Its value at position j, p^j_{xi}, indicates the preference of user u_x over category j at time i. For the value of p^j_{xi}, we approximate it as a function of the implicit rating history of user u_x. Formally, p^j_{xi} is defined as follows:

$$p^j_{xi} = \sum_{l=1}^{n} b^j_{xl}, \tag{1}$$

where n is the number of items, and

$$b^j_{xl} = \begin{cases} \frac{1}{|\mathcal{C}_l|} & r_{xl} \neq \oslash, t_l \in c_j, r_{xl} \text{ is given at time i} \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

where $r_{xl} \neq \oslash$ denotes that r_{xl} is available, $\mathcal{C}_l$ denotes a set of categories to which t_l belongs, and $t_l \in c_j$ denotes that item t_l belongs to category c_j. Please note that there is an inverse relationship between b^j_{xl} and $|\mathcal{C}_l|$ in Eq. 2.

The *preference pattern* is defined for each user in a way to naturally integrate a user's various needs with the corresponding dynamics. The *preference pattern* *vectors* within a preference pattern captures the user's corresponding *preference styles*, and the differences in two consecutive preference pattern vectors imply the dynamics of a user's *preference styles.* If all values in each *preference pattern vector* are available, the preference pattern is *complete*, otherwise it is *incomplete.* In a real recommender system, as many missing values exist within the preference patterns, the modelling of preference patterns is a challenge.

3 The Preference Pattern Subspace

In this section, we build a *Preference Pattern Subspace* to model the preference patterns for each user, then propose the *PrepSVD-I* algorithm by formulating the task of Top-N recommendation as a pairwise preference learning process.

3.1 Learning the Preference Pattern Subspace

To model the user preference patterns, we propose a *Preference Pattern Subspace* by applying the *Singular Value Decomposition* (SVD). Conventionally, the SVD of the preference patterns $\mathcal{P}$ is the factorization of the form: $\mathcal{P} = \mathcal{U} \cdot \Sigma \cdot \mathcal{V}^T$, where $\mathcal{U}$ is an $m \times m$ orthogonal matrix, Σ is an $m \times n$ diagonal matrix containing the singular values of $\mathcal{P}$ on the diagonal, $\mathcal{V}$ is an $n \times n$ orthogonal matrix.

According to the *Eckart-Young theorem* [7], it is well-known that the best rank-k approximation of matrix $\mathcal{P}$ can be achieved by SVD. However, conventional SVD is defined without considering the existence of missing values. Therefore, as $\mathcal{P}$ is highly incomplete, SVD can not be directly applied to analyse preference patterns $\mathcal{P}$. To overcome this problem, we propose an EM-like learning algorithm to capture as much as possible the main variance of the highly incomplete preference patterns $\mathcal{P}$. Specifically, $\mathbf{p}_x$ can be divided into two parts, $\mathbf{p}_x^a$ and $\mathbf{p}_x^m$, representing the *available* part and the *missing* part of $\mathbf{p}_x$, respectively. We estimate $\mathcal{P}$ using SVD to construct a low rank k subspace:

$$\hat{\mathcal{P}} = \mathcal{U}_k \cdot \Sigma_k \cdot \mathcal{V}_k^T, \tag{3}$$

where $\mathcal{U}_k$ contains the first k columns of $\mathcal{U}$, Σ_k is a $k \times k$ diagonal matrix that contains the first k singular values of $\mathcal{P}$, and $\mathcal{V}_k$ contains the first k columns of $\mathcal{V}$. Consequently, the reconstruction $\hat{\mathbf{p}}_x$ of $\mathbf{p}_x$ is defined as:

$$\hat{\mathbf{p}}_x = \mathcal{U}_k \cdot \Sigma_k \cdot \mathbf{v}_x^T, \tag{4}$$

where $\mathbf{v}_x$ is the xth row of $\mathcal{V}_k$, and denotes the projection of $\mathbf{p}_x$ for user u_x on the low rank k subspace. Similarly, the reconstruction $\hat{\mathbf{p}}_x$ can also be divided into two parts $\hat{\mathbf{p}}_x^a$ and $\hat{\mathbf{p}}_x^m$, representing the reconstruction of $\mathbf{p}_x^a$ and $\mathbf{p}_x^m$, respectively. The SVD guarantees to produce the best k-rank approximation of $\mathcal{P}$ with minimal reconstruction errors. However, as $\mathcal{P}$ is highly incomplete, we change the modelling objective to minimize the reconstruction error on the available preferences in $\mathcal{P}$. This is defined as the squared distance between the original available part of $\mathcal{P}$ and their reconstructions:

$$\varepsilon^a = \frac{1}{m} \sum_{x=1}^{m} (\mathbf{p}_x^a - \hat{\mathbf{p}}_x^a)^T \cdot (\mathbf{p}_x^a - \hat{\mathbf{p}}_x^a), \tag{5}$$

where m is the number of users.

To build the representative subspace, an EM-like algorithm is introduced as follows: first, the missing values of $\mathcal{P}$ are replaced with their corresponding values in $\mu = \frac{1}{m}\sum_{x=1}^{m}\mathbf{p}_x$. Then, in the j-th iteration, the standard SVD algorithm is applied to calculate a low-rank subspace defined by $\mathcal{U}_k$ and Σ_k. After that, the reconstruction $\hat{\mathbf{p}}_x$ of $\mathbf{p}_x$ can be calculated with Eq. 4. However, as only a small fraction of preferences are available in $\mathbf{p}_x \in \mathcal{P}$, its projection $\mathbf{v}_x$ can not be directly estimated from Eq. 3. Please note that we can estimate $\mathbf{v}_x$ from part of $\mathbf{p}_x$, e.g, the *available* part $\mathbf{p}_x^a$, and this estimation method has been widely used in the field of multimedia research [6]. Thus, we estimate $\mathbf{v}_x$ as the least squares solution for the following equation:

$$(\mathcal{U}_k \cdot \Sigma_k) \cdot \mathbf{v}_x^T = [\mathbf{p}_x]^a, \tag{6}$$

where $[\mathbf{p}_x]^a$ denotes $\mathbf{p}_x$ in the current iteration step but only has values on the positions corresponding to $\mathbf{p}_x^a$. After $\mathbf{v}_x$ is estimated, the reconstruction $\hat{\mathbf{p}}_x$ of $\mathbf{p}_x$ can be calculated using Eq. 4. $\mathbf{p}_x^m$ will then be updated with $\hat{\mathbf{p}}_x^m$, and the new $\mu^{(j+1)}$ in the next $(j+1)$-th iteration will be calculated with the updated $\mathbf{p}_x$ accordingly. With the updated $\mathcal{P}$ and the new mean vector $\mu^{(j+1)}$, the SVD algorithm is once again applied to calculate $\mathcal{U}_k$ and Σ_k. This iterative process will continue until the reconstruction error ε^a is below a pre-defined threshold. The proof for the convergence of this training algorithm is provided as follows.

Proof 1. In the j-th iteration, the reconstruction $\hat{\mathbf{p}}_x$ of $\mathbf{p}_x$ is defined as: $\hat{\mathbf{p}}_x^j = (\mathcal{U}_k \cdot \Sigma_k)^j \cdot \mathbf{v}_x^T$, where $(\mathcal{U}_k \cdot \Sigma_k)^j$ denote $\mathcal{U}_k$ and Σ_k in the j-th iteration. We denote $\hat{\mathbf{p}}_x^j$ obtained with $(\mathcal{U}_k \cdot \Sigma_k)^j$ as $f_{\hat{\mathbf{p}}_x}^j(\mathcal{U}_k \cdot \Sigma_k)^j$. Please note that $f_{\hat{\mathbf{p}}_x}^j(\mathcal{U}_k \cdot \Sigma_k)^j$ and the data in the next iteration, $\mathbf{p}_x^{j+1}$, share values on missing part of $\mathbf{p}_x$, thus the reconstruction error on $\mathbf{p}_x^j$ is represented as:

$$(\varepsilon_x^a)^j = d\left(f_{\hat{\mathbf{p}}_x}^j(\mathcal{U}_k \cdot \Sigma_k)^j, \mathbf{p}_x^{j+1}\right), \tag{7}$$

where $d(\cdot,\cdot)$ is the *Euclidean* distance between two vectors.

If we use $(\mathcal{U}_k \cdot \Sigma_k)^j$ to calculate the reconstruction of $\mathbf{p}_x^{j+1}$, we obtain

$$d\left(f_{\hat{\mathbf{p}}_x}^{j+1}(\mathcal{U}_k \cdot \Sigma_k)^j, \mathbf{p}_x^{j+1}\right) \leq d\left(f_{\hat{\mathbf{p}}_x}^j(\mathcal{U}_k \cdot \Sigma_k)^j, \mathbf{p}_x^{j+1}\right), \tag{8}$$

because the orthogonal property of $(\mathcal{U}_k \cdot \Sigma_k)^j$ makes it sure that $f_{\hat{\mathbf{p}}_x}^{j+1}(\mathcal{U}_k \cdot \Sigma_k)^j$ and $\mathbf{p}_x^{j+1}$ have the minimum *Euclidean* distance.

In the $(j+1)$-th iteration, after applying SVD to the updated training data $\mathbf{p}_x^{j+1}$, we observe the minimum reconstruction error by obtaining $(\mathcal{U}_k \cdot \Sigma_k)^{j+1}$:

$$d\left(f_{\hat{\mathbf{p}}_x}^{j+1}(\mathcal{U}_k \cdot \Sigma_k)^{j+1}, \mathbf{p}_x^{j+1}\right) \leq d\left(f_{\hat{\mathbf{p}}_x}^{j+1}(\mathcal{U}_k \cdot \Sigma_k)^j, \mathbf{p}_x^{j+1}\right). \tag{9}$$

For the reconstruction error in the $(j+1)$-th iteration, we obtain

$$(\varepsilon_x^a)^{j+1} = d\left(f_{\hat{\mathbf{p}}_x}^{j+1}(\mathcal{U}_k \cdot \Sigma_k)^{j+1}, \mathbf{p}_x^{j+1}\right) \leq \left(f_{\hat{\mathbf{p}}_x}^{j+1}(\mathcal{U}_k \cdot \Sigma_k)^j, \mathbf{p}_x^{j+1}\right) \leq (\varepsilon_x^a)^j. \tag{10}$$

$$(\varepsilon^a)^{j+1} = \frac{1}{m}\sum_{x=1}^{m}(\varepsilon_x^a)^{j+1} \leq \frac{1}{m}\sum_{x=1}^{m}(\varepsilon_x^a)^j = (\varepsilon^a)^j. \tag{11}$$

Thus, the algorithm will converge to minimize ε^a. □

The modelling process is an iterative refinement of the global and the personal preference patterns. One advantage of this EM-like learning algorithm is that the well-trained *Preference Pattern Subspace* can model both the personal preference patterns and the global preference patterns simultaneously.

3.2 Recommendation Generation

After learning the well-trained *Preference Pattern Subspace*, we propose a *PrepSVD-I* algorithm to generate Top-N recommendations. In this paper, we apply the latent factor model to estimate the ratings $\hat{r}_{xl}$ for user u_x on item t_l:

$$\hat{r}_{xl} = \rho_x^T \rho_l, \tag{12}$$

where ρ_x and ρ_l are the user factors and the item factors, and can be learnt with stochastic gradient descent method by looping through available ratings.

Given the user factors and the item factors, $T_N(u_x)$ can be generated by estimating ratings for un-known items with Eq. 12, then formed with the Top-N ranked ones. For $t_l \in T_N(u_x)$, t_l can be temporarily absorbed into u_x's preference pattern vector p_{xi}, and a tentatively changed preference pattern vector $\tilde{p}_{xi}$ is available. Please note that because we only want to measure the degree to which the recommendations match u_x's preference styles captured by the *Preference Pattern Subspace*, we initialize $\tilde{p}_{xi}$ as empty while keeping the other part the same as $\mathbf{p}_x$. The value at the jth position of $\tilde{p}_{xi}$ is then defined as:

$$\tilde{p}_{xi}^j = \begin{cases} \frac{1}{|\mathcal{C}_l|} & t_l \in c_j, c_j \subseteq \mathcal{C}_l \\ 0 & \text{otherwise,} \end{cases} \tag{13}$$

where $\mathcal{C}_l$ denotes a set of categories that t_l belongs to, $t_l \in c_j$ denotes that t_l belongs to category c_j. The reconstruction error for t_l to u_x at time i is defined as the squared distance between the changed preference pattern and its reconstruction:

$$\varepsilon_{t_l}^a = (\tilde{\mathbf{p}}_x^a - \hat{\mathbf{p}}_x^a)^T \cdot (\tilde{\mathbf{p}}_x^a - \hat{\mathbf{p}}_x^a), \tag{14}$$

where $\tilde{\mathbf{p}}_x^a$ is the available part of $\tilde{\mathbf{p}}_x$, $\hat{\mathbf{p}}_x$ is the reconstruction of $\tilde{\mathbf{p}}_x$ and is calculated with Eq. 6 and Eq. 4, while $\hat{\mathbf{p}}_x^a$ is the available part of $\hat{\mathbf{p}}_x$.

Moreover, the observed *preference dynamic effect* implies one constraint to the objective function of recommendation generations. It can be formulated as:

$$\forall t_l \in T_N(u_x), \varepsilon_{t_l}^a \leq \bar{\varepsilon}_{u_x}^a, \tag{15}$$

where $\bar{\varepsilon}_{u_x}^a = \frac{1}{|S(u_x)|} \sum_{t_{l'} \in S(u_x)} \varepsilon_{t_{l'}}^a$ and $S(u_x)$ is the set of items liked by u_x. Following this, we formulate the Top-N recommendation generation as a pairwise preference learning problem [8], and utilize the user average reconstruction $\bar{\varepsilon}_{u_x}^a$ as the negative preference:

$$\min_{\theta,\xi} \sum_x \xi_x + \lambda\left(\sum_{u_x \in \mathcal{U}} \|\rho_x\|^2 + \sum_{t_l \in \mathcal{T}} \|\rho_l\|^2 \right) \tag{16}$$
$$\text{s.t.:} \varepsilon_{t_l}^a - \bar{\varepsilon}_{u_x}^a \geq 1 - \xi_x \text{ and } \xi_x \geq 0$$

where ξ_x is a non-negative value measuring the degree of violating the constraint in Eq. 15, λ is the regularization weight determined by cross validation.

We apply a simple gradient descent algorithm to optimize the objective function defined in Eq. 16. Moreover, as there is an inverse relationship between b_{xl}^{j} and $|\mathcal{C}_l|$ in the approximation of preference values in Eq. 2, we name this proposed algorithm as the *PrepSVD-I* algorithm. It loops on all $T_N(u_x), \forall u_x \in \mathcal{U}$, and updates the user factors and the item factors by following the negative gradient:

$$\rho_l = \rho_l - \gamma(h'(\rho_x^T \rho_l)\rho_x + \lambda\rho_l) \tag{17}$$

$$\rho_x = \rho_x - \gamma(\sum_{t_l \in T_N(u_x)} h'(\rho_x^T \rho_l)\rho_l + \lambda\rho_x), \tag{18}$$

where γ is the learning rate, $h' = -\frac{|T_N(u_x)|\varDelta - 1}{|T_N(u_x)| - 1} H(1 - \varepsilon_{t_l}^a + \bar{\varepsilon}_{u_x}^a)$, $\varDelta = |\varepsilon_{t_l}^a - \bar{\varepsilon}_{u_x}^a|$, and $H(z) = 1$ if $z > 0$ and 0 otherwise, denoting the *Heaviside* function [1]. When the training process is completed, we can calculate the predicted rating $\hat{r}_{xl}$ for each unknown item t_l to user u_x, then recommend the top ranked N items to u_x with Eq. 12. Here the proposed *PrepSVD-I* algorithm takes both the personal preference patterns and the global preference patterns into consideration.

4 Experiment and Analysis

The datasets we experimented with were the popular *MovieLens* dataset and *Netflix* dataset. *MovieLens* includes around 1 million ratings collected from 6,040 users on 3,900 movies. Following literature [13], the *Netflix* dataset is a subset extracted from the Netflix Prize dataset, in which each user rated at least 20 movies, and each movie was rated by 20 − 250 users. For each dataset, we split it into two subsets, the *training set* and the *test set*. Following the work of [3, 4, 11], we reasonably assume that 5-star rated items are relevant to the active user, and adopt a similar strategy to conduct experiments. Specifically, we randomly select 2% of ratings and use all 5-star selected ratings to form the *test set*, and make sure that at least one 5-star rating exists for each individual user. The remaining ratings in the data set form the *training set*. After training the model on the *training set*, we randomly select 1000 additional items that are not rated by the active user, then predict ratings on the test item and additional 1000 selected items. These items are then ranked and the top ranked N items are selected as Top-N recommendations for the active user. This testing strategy is common for Top-N recommendations research and has been adopted by [3,4,11]. To examine the algorithm performance thoroughly, we set up a series of configurations with different data sparsity levels. Specifically, on *MovieLens* data set, we keep the *test set* the same, but vary the percentage of observed ratings for each user in the *training set*, from 10% to 100% with a 10% step. These configurations are called as *Given*10% to *Given*100% accordingly. Moreover, as recommending popular items is trivial [4], we will focus on recommending *long tail* items and *all items* that include both popular and unpopular items.

4.1 Comparison and Evaluation

We examine the performance of the proposed *PrepSVD-I* algorithm by comparing it with 7 other Top-N recommendation algorithms, including *PureSVD* [4], *SLIM* [13], BPTF [14], *itemKNN* [5], *NNcosNgbr* [4], *Top Popular* (*TopPop*) [3, 4] and *Movie Average* (*MovieAvg*) [11]. Please note BPTF considers the *time* information [14]. In *PureSVD*, the number of factors is set to 50. In *SLIM*, we set $\beta = 0.1$ and $\lambda = 0.1$. The number of the nearest neighbors in *NNcosNgbr* is set to 200, and the number of neighbors in *itemKNN* is set to 20. For *BPTF*, we set *lrate* $= 0.001$, $D = 200$ and the number of samples to 50. For our method, to train the *Preference Pattern Subspace*, we set $k = 50$, $\beta = -1$, and set the max iteration of training to 50, the error threshold to 10^{-6}. For *PrepSVD-I*, $\gamma = 0.0001$, $\lambda = 0.03$, and the factors for both users and items are set to 50.

The quality of Top-N recommendations is measured by the *recall* (or Hit Rate), the *precision* and the *fall-out* [4,5,10]. For the active user, if the Top-N recommendation list contains the test item, we call this a *hit.* Therefore, *recall*, *precision* and *fall-out* are defined as follows:

$$recall = \frac{\#hits}{|X|}, \quad precision = \frac{\#hits}{N \cdot |X|}, \quad fall\text{-}out = \frac{|X| \cdot N - \#hits}{|irrelevant|},$$

where X is the *test set* and $|irrelevant|$ is the number of all non-relevant items. A higher *recall* or *precision* value indicates better Top-N recommendations, while a lower *fall-out* value means better recommendations.

4.2 Performance on Different Datasets

To fully examine the performance of the proposed model, we conduct experiments on two well-known data sets, *MovieLens* and *Netflix.* Table 2 shows the results on these datasets when $N = 20$. It is observed that *PrepSVD-I* outperforms all the compared algorithms on both data sets for both *long tail* and *all items* recommendations in all the measurement metrics. Specifically, on *MovieLens* for *long tail* item recommendations, when measuring in *recall*, *PrepSVD-I* obtains a recall at 0.5389, which outperforms the best result 0.4987 (from *PureSVD*) by 8.06%; when measuring in *precision*, *PrepSVD-I* achieves a precision at 0.0269 that also outperforms all the other compared algorithms; for *all items* recommendations, *PrepSVD-I* also achieves better performance in *recall* and *precision.* On *Netflix*, when measuring in *recall*, *PrepSVD-I* achieves a better recall at 0.6361 and 0.7526 for *long tail* and *all items* recommendations, respectively. When measuring in *fall-out*, it seems that, *PrepSVD-I*, *PureSVD* and *SLIM* show similar performance. The main reason behind this is that, according to the definition of *fall-out*, when the number of irrelevant items is large, the *fall-out* value tends to be small, and this diminishes the difference between the performance of compared algorithms. Nevertheless, *PrepSVD-I* still achieves comparable performance to the compared algorithms. This indicates that the proposed *Preference Pattern* model can benefit Top-N recommendations for both *long tail* and *all items* recommendations. This is mainly because

Table 2. Performance on *MovieLens* and *Netflix* when $N = 20$

items	Algorithm	*MovieLens*			*Netflix*		
		recall	*precision*	*fall-out*	*recall*	*precision*	*fall-out*
long tail	PrepSVD-I	**0.5389**	**0.0269**	**0.0195**	**0.6361**	**0.0318**	**0.0194**
	PureSVD	0.4987	0.0249	**0.0195**	0.6165	0.0308	**0.0194**
	SLIM	0.4527	0.0226	**0.0195**	0.5987	0.0299	**0.0194**
	NNcosNgbr	0.4518	0.0226	**0.0195**	0.4988	0.0249	0.0195
	itemKNN	0.3273	0.0164	0.0197	0.4393	0.0220	0.0196
	BPTF	0.1992	0.0100	0.0198	0.2960	0.0148	0.0197
	TopPop	0.0096	0.0005	0.0200	0.2041	0.0102	0.0198
	MovieAvg	0.0818	0.0041	0.0199	0.0312	0.0016	0.0200
		MovieLens			*Netflix*		
items	Algorithm	*recall*	*precision*	*fall-out*	*recall*	*precision*	*fall-out*
all items	PrepSVD-I	**0.6928**	**0.0346**	**0.0193**	**0.7526**	**0.0376**	**0.0192**
	PureSVD	0.6709	0.0335	**0.0193**	0.7193	0.0360	0.0193
	SLIM	0.6794	0.0340	**0.0193**	0.7295	0.0365	0.0193
	NNcosNgbr	0.4962	0.0248	0.0195	0.5258	0.0263	0.0195
	itemKNN	0.5625	0.0281	0.0194	0.6436	0.0322	0.0194
	BPTF	0.3389	0.0169	0.0197	0.3837	0.0192	0.0196
	TopPop	0.3857	0.0193	0.0196	0.5000	0.0250	0.0195
	MovieAvg	0.1734	0.0087	0.0198	0.0589	0.0029	0.0199

the *Preference Pattern* is based on users' personal preference styles, and also because it takes the global preference patterns into consideration. Therefore, it can lead to better recommendations regardless of whether the target item is popular or not.

4.3 Performance on *long tail* Item Recommendations

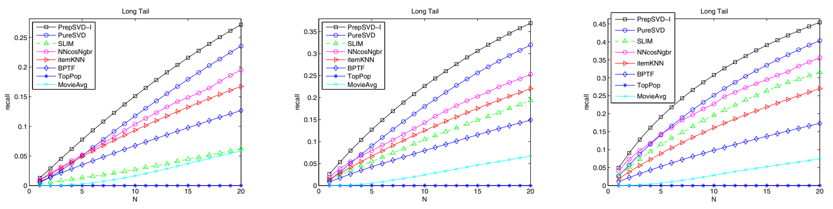
As recommending popular items is trivial [4], here we examine the proposed *PrepSVD-I* by comparing it with 7 other state-of-the-art recommendation algorithms under various data sparsity levels on *long tail* recommendations.

Table 3 shows the *recall* performance of the examined algorithms when N equals 10 and 20. It is clear that the proposed *PrepSVD-I* algorithm significantly outperforms all of the compared algorithms under all sparsity conditions except on *Given*10% when $N = 10$ with *BPTF* obtaining a slightly higher *recall*. However, *BPTF* performs badly on all other sparsity levels. For example, on *Given*90% when $N = 20$, *BPTF* only obtains a *recall* at 0.1824, which is much worse than all the other personalized algorithms, e.g. *PrepSVD-I*, *itemKNN*, *NNcosNgbr* and *SLIM*. Moreover, *PrepSVD-I* performs steadily through various sparsity levels, and always achieves better performance. Specifically, on *Given*50%, when $N = 20$, *PrepSVD-I* achieves a *recall* of 0.3147, which outperforms the best compared result of 0.2751 (from *PureSVD*) by 14.39%. This is, as expected, because the proposed *Preference Pattern Subspace* is capable of capturing user preference styles and the corresponding dynamics, and is not affected by whether the item is popular or not. Moreover, it is also observed that the sparser the training data set, the larger the improvements. For example, when $N = 10$, the improvement on *Given*100% is 12.66%, and increases to 29.76% on *Given*40% data set. The reason behind this is that when the training set is sparser, the *long tail* effect indicates that available ratings on *long tail*

Table 3. *recall* when $N = 10$ and $N = 20$ on *long tail* Items

Algorithm	*Given*10% $N = 10$	*Given*10% $N = 20$	*Given*20% $N = 10$	*Given*20% $N = 20$	*Given*30% $N = 10$	*Given*30% $N = 20$	*Given*40% $N = 10$	*Given*40% $N = 20$	*Given*50% $N = 10$	*Given*50% $N = 20$
PrepSVD-I	0.0388	**0.0903**	**0.0677**	**0.1490**	**0.1092**	**0.2112**	**0.1526**	**0.2728**	**0.1851**	**0.3147**
PureSVD	0.0299	0.0818	0.0525	0.1311	0.0848	0.1815	0.1176	0.2353	0.1474	0.2751
SLIM	0.0000	0.0000	0.0000	0.0004	0.0058	0.0167	0.0270	0.0623	0.0587	0.1188
NNcosNgbr	0.0230	0.0598	0.0371	0.0785	0.0552	0.1141	0.1037	0.1949	0.1156	0.2257
itemKNN	0.0267	0.0501	0.0517	0.0901	0.0726	0.1321	0.0935	0.1673	0.1112	0.1970
BPTF	**0.0491**	0.0873	0.0484	0.0910	0.0549	0.1064	0.0673	0.1265	0.0738	0.1359
TopPop	0.0000	0.0000	0.0000	0.0001	0.0000	0.0001	0.0000	0.0001	0.0000	0.0001
MovieAvg	0.0036	0.0232	0.0073	0.0367	0.0095	0.0431	0.0165	0.0586	0.0169	0.0560

Algorithm	*Given*60% $N = 10$	*Given*60% $N = 20$	*Given*70% $N = 10$	*Given*70% $N = 20$	*Given*80% $N = 10$	*Given*80% $N = 20$	*Given*90% $N = 10$	*Given*90% $N = 20$	*Given*100% $N = 10$	*Given*100% $N = 20$
PrepSVD-I	**0.2275**	**0.3701**	**0.2614**	**0.4088**	**0.3100**	**0.4559**	**0.3479**	**0.4957**	**0.3995**	**0.5389**
PureSVD	0.1801	0.3198	0.2091	0.3568	0.2515	0.4035	0.2988	0.4502	0.3546	0.4987
SLIM	0.1048	0.1938	0.1532	0.2679	0.1959	0.3151	0.2451	0.3741	0.3229	0.4527
NNcosNgbr	0.1429	0.2531	0.1872	0.3034	0.2272	0.3562	0.2666	0.3889	0.3293	0.4518
itemKNN	0.1256	0.2204	0.1414	0.2417	0.1616	0.2702	0.1816	0.2954	0.2086	0.3273
BPTF	0.0796	0.1490	0.0873	0.1600	0.0984	0.1733	0.1032	0.1824	0.1196	0.1992
TopPop	0.0000	0.0002	0.0000	0.0002	0.0001	0.0003	0.0001	0.0029	0.0013	0.0096
MovieAvg	0.0245	0.0673	0.0281	0.0716	0.0289	0.0745	0.0303	0.0755	0.0318	0.0818



(a) *Given*40% (b) *Given*60% (c) *Given*80%

Fig. 2. *recall* at N on *long tail*

items will be much more limited. Consequently, the user rating patterns will be extremely incomplete, and solely modelling them does not lead to good recommendations. In this case, the *preference dynamic effect* becomes more valuable to predict users' preferences on items. Therefore, the proposed *Preference Pattern Subspace* will show high effectiveness for recommendation purposes.

To thoroughly examine the performance of *PrepSVD-I*, we vary the N value from 1 to 20, and report the results on *Given*40%, *Given*60% and *Given*80% as shown in Fig. 2. We observe that *PrepSVD-I* outperforms all compared algorithms at all N values on all the data sets. This indicates that *PrepSVD-I* is effective in recommending the desired items at the top of the recommendation list. The experiment results show that, when recommending *long tail* items, *PrepSVD-I* is robust to the data sparsity issue, and can significantly outperforms state-of-the-art Top-N recommendation algorithms in terms of accuracy.

4.4 Performance on *all items* Recommendations

we also conduct experiments on recommending *all items*, including both popular and *long tail* items. Table 4 shows the *recall* performance of all compared algo-

Table 4. *recall* when $N = 10$ and $N = 20$ on *All Items*

	*Given*10%		*Given*20%		*Given*30%		*Given*40%		*Given*50%	
Algorithm	$N = 10$	$N = 20$	$N = 10$	$N = 20$	$N = 10$	$N = 20$	$N = 10$	$N = 20$	$N = 10$	$N = 20$
PrepSVD-I	0.1421	0.2300	0.1926	0.3007	0.2363	**0.3585**	**0.2826**	**0.4165**	**0.3267**	**0.4637**
PureSVD	0.1310	0.2102	0.1716	0.2701	0.2115	0.3205	0.2471	0.3698	0.2840	0.4131
SLIM	0.2032	0.3059	0.2221	**0.3238**	**0.2392**	0.3472	0.2811	0.3960	0.3211	0.4462
NNcosNgbr	0.2027	0.2990	0.2052	0.3095	0.2096	0.3248	0.2532	0.3709	0.2617	0.3818
itemKNN	0.0290	0.0548	0.1383	0.2032	0.2252	0.3223	0.2748	0.3855	0.3042	0.4265
BPTF	0.1014	0.1756	0.1285	0.2065	0.1475	0.2332	0.1653	0.2549	0.1803	0.2736
TopPop	**0.2088**	**0.3080**	**0.2247**	0.3222	0.2256	0.3251	0.2332	0.3320	0.2413	0.3410
MovieAvg	0.0036	0.0443	0.0152	0.0812	0.0235	0.1003	0.0353	0.1198	0.0451	0.1253

	*Given*60%		*Given*70%		*Given*80%		*Given*90%		*Given*100%	
Algorithm	$N = 10$	$N = 20$	$N = 10$	$N = 20$	$N = 10$	$N = 20$	$N = 10$	$N = 20$	$N = 10$	$N = 20$
PrepSVD-I	**0.3821**	**0.5242**	**0.4322**	**0.5746**	**0.4819**	**0.6213**	**0.5229**	**0.6588**	**0.5678**	**0.6928**
PureSVD	0.3267	0.4592	0.3721	0.5090	0.4275	0.5640	0.4939	0.6226	0.5534	0.6709
SLIM	0.3785	0.5090	0.4277	0.5640	0.4588	0.5897	0.5017	0.6273	0.5668	0.6794
NNcosNgbr	0.2796	0.3905	0.3112	0.4156	0.3374	0.4426	0.3558	0.4553	0.4036	0.4962
itemKNN	0.3360	0.4594	0.3595	0.4832	0.3836	0.5094	0.4128	0.5360	0.4423	0.5625
BPTF	0.1930	0.2884	0.2026	0.3003	0.2127	0.3118	0.2196	0.3221	0.2381	0.3389
TopPop	0.2484	0.3480	0.2559	0.3564	0.2661	0.3658	0.2717	0.3724	0.2860	0.3857
MovieAvg	0.0585	0.1456	0.0710	0.1631	0.0691	0.1629	0.0691	0.1613	0.0742	0.1734

rithms across various sparsity levels on *all item* recommendations. We can observe that *PrepSVD-I* achieves the best *recall* on all data sets, except *Given*10%, *Given*20% and *Given*30% (on $N = 10$). It is unexpected that *TopPop* achieves the highest *recall* values on *Given*10% and *Given*20% (on $N = 10$). This is mainly because when the training set is very sparse, the majority of available ratings are given for popular items. Therefore, the popularity of items will bias the performance of algorithms. This is consistent with the findings in [4]. *SLIM* achieves the best *recall* on *Given*20% (on $N = 20$) and *Given*30% (on $N = 10$). However, both *TopPop* and *SLIM* become almost useless in recommending *long tail* items on the same data sets, as shown in Table 3, which confirms the popularity-related bias.

On the other hand, it is also clear that *PrepSVD-I* outperforms all compared algorithms on 7 out of 10 data sets, including *TopPop* and *SLIM*. Specifically, when $N = 20$ on *Given*40% data set, *PrepSVD-I* achieves a *recall* at 0.4165 which outperforms the best compared results of 0.3960 (from *SLIM*); on *Given*100% data set, *PrepSVD-I* obtains a *recall* at 0.6928 that outperforms the best compared results of 0.6794 (from *SLIM*). This indicates that the *Preference Pattern Subspace* can also benefit *all items* recommendations, including popular items.

In terms of *accuracy* on recommending both *all items* and *long tail* items, it is clear that *PrepSVD-I* performs better than all compared state-of-the-art Top-N recommendation algorithms. Although *TopPop* and *SLIM* achieve a slightly better *recall* values on *Given*10%, *Given*20% and *Given*30% when recommending *all items*, they perform badly on the same data set when recommending *long tail* items, as shown in Table 3. This will limit their recommendation abilities. However, *PrepSVD-I* can perform very well on both *all items* and *long tail* item recommendations. This means *PrepSVD-I* possesses a better recommendation ability than other rating-pattern-based techniques.

5 Conclusion

This paper introduces a novel *preference dynamic effect* in the context of recommender systems, and proposes a *Preference Pattern Subspace* approach to model this effect. The basic idea is to build a low-rank subspace to capture the *personal preference patterns* together with their temporal dynamics by refining the *global* and the *personal* preference patterns iteratively with an EM-like algorithm. Experiment results show that the proposed *PrepSVD-I* significantly outperforms the other state-of-the-art Top-N recommendation techniques in terms of *accuracy*, and is robust to challenge the data sparsity issue.

References

1. Abramowitz, M., Stegun, I.A.: Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables. Dover, New York (1972)
2. Anderson, C.: The Long Tail: Why the Future of Business Is Selling Less of More, vol. 33. Hyperion (2006)
3. Cremonesi, P., Garzotto, F., Negro, S., Papadopoulos, A.: Comparative Evaluation of Recommender System Quality. In: CHI Extended Abstracts, pp. 1927–1932 (2011)
4. Cremonesi, P., Koren, Y., Turrin, R.: Performance of Recommender Algorithms on Top-N Recommendation Tasks. In: Recsys, pp. 39–46. ACM (2010)
5. Deshpande, M., Karypis, G.: Item-based top- N recommendation algorithms. ACM TOIS 22(1), 143–177 (2004)
6. Geng, X., Zhou, Z.-H., Smith-Miles, K.: Automatic age estimation based on facial aging patterns. IEEE TPAMI 29(12), 2234–2240 (2007)
7. Golub, G.H., Van Loan, C.F.: Introduction to Matrix, 3rd edn., vol. 1. The Johns Hopkins University Press, Baltimore (1996)
8. Herbrich, R., Graepel, T., Obermayer, K.: Support Vector Learning for Ordinal Regression. In: ICANN 1999, vol. 470, pp. 97–102 (1999)
9. Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating Collaborative Filtering Recommender Systems. ACM Transactions on Information Systems 22(1), 5–53 (2004)
10. Karypis, G.: Evaluation of item-based top-n recommendation algorithms. In: CIKM 2001, pp. 247–254. ACM (2001)
11. Koren, Y.: Factorization meets the neighborhood: a multifaceted collaborative filtering model. In: SIGKDD 2008, pp. 426–434. ACM (2008)
12. Koren, Y.: Collaborative filtering with temporal dynamics. In: SIGKDD 2009, pp. 447–456. ACM (2009)
13. Ning, X., Karypis, G.: SLIM: Sparse Linear Methods for Top-N Recommender Systems. In: ICDM 2011, pp. 497 – 506 (2011)
14. Xiong, L., Chen, X., Huang, T.K., Schneider, J., Carbonell, J.G.: Temporal collaborative filtering with bayesian probabilistic tensor factorization. In: Proceedings of SIAM Data Mining (2010)