Self-training Temporal Dynamic Collaborative Filtering

Cheng Luo[1], Xiongcai Cai[1,2], and Nipa Chowdhury[1]

[1] School of Computer Science and Engineering
[2] Centre of Health Informatics
University of New South Wales, Sydney NSW 2052, Australia
{luoc,xcai,nipac}@cse.unsw.edu.au

Abstract. Recommender systems (RS) based on collaborative filtering (CF) is traditionally incapable of modeling the often non-linear and non Gaussian tendency of user taste and product attractiveness leading to unsatisfied performance. Particle filtering, as a dynamic modeling method, enables tracking of such tendency. However, data are often extremely sparse in real-world RS under temporal context, resulting in less reliable tracking. Approaches to such problem seek additional information or impute all or most missing data to reduce sparsity, which then causes scalability problems for particle filtering. In this paper, we develop a novel semi-supervised method to simultaneously solve the problems of data sparsity and scalability in a particle filtering based dynamic recommender system. Specifically, it exploits the self-training principle to dynamically construct observations based on current prediction distributions. The proposed method is evaluated on two public benchmark datasets, showing significant improvement over a variety of existing methods for top-k recommendation in both accuracy and scalability.

Keywords: Temporal Recommender, Collaborative Filtering, Particle Filtering.

1 Introduction

Collaborative filtering [10] generates personalized recommendation to match users' interests and its performance can be improved by exploiting temporal information, as the tendency of user preferences and item attractiveness is not static [10,11]. There are four general approaches in temporal CF, i.e, heuristic, binning-based, online updating and dynamic-based approaches. Heuristic approach penalizes the importance of data before a pivot point[6,10], which tends to undervalue the past data. In binning-based approach, training and testing data could be from the same interval [11]. The prediction for users' interests is actually *post hoc* about what interests *would have been* in the past, rather than what interests would be in the future. Although online updating approach only uses past information to make prediction [8,12], it usually focuses on scalability and ignores the dynamics of user taste and item attractiveness. Dynamic-based approach explicitly models temporal dynamics by a stochastic state space model,

V.S. Tseng et al. (Eds.): PAKDD 2014, Part I, LNAI 8443, pp. 461–472, 2014.

which shows advantages over the other methods [13]. However, in such an approach, item attractiveness is assumed to be static with Gaussian distributions, which may be oversimplified. To overcome such problems, we first utilize particle filtering [16] as a dynamic technique to model non-Gaussian behaviors and track latent factors representing user preferences and item attractiveness.

Furthermore, under the temporal context, the data sparsity problem [19] becomes challenging as many users would be inactive for some consecutive time slots. Although matrix factorization [15] could realize sparsity reduction, the tendency tracked by particle filtering may be unreliable due to insufficient observations at every time step. Exploiting additional information, such as contextual information or common patterns [10], or imputing all or most missing data are two common approaches in CF to reduce sparsity. However, this extra information collection is usually not only infeasible in practice but also computational complexity. Therefore, we do not consider utilizing side information in this paper. Missing data are usually imputed as negative [18,14] or other heuristic values [10]. Deterministic optimization [17] and current model estimation [9] are also used to yield some reasonable imputed values. However, these methods are only developed under static context and based on some heuristic rules or point estimators. Meanwhile, all these methods impute all or most missing data, leading to an unaffordable computational complexity for succeeding recommendation algorithms, since it not only heavily reduces the scalability of algorithms but also influences the recommendation accuracy.

In this paper, we aim to solve the problems of data sparsity and scalability in particle filtering-based temporal dynamic recommender systems. We utilize self-training principle [22] to dynamically construct feedback data to enhance our particle filtering-based recommendation method. In particular, we use latent factors to compactly represent user preferences and item attractiveness respectively at each time step, whose initial settings are learned by probabilistic matrix factorization (PMF) [15]. Based on such a representation, we develop a novel self-training mechanism based on the distributions of the current personalized prediction to complement recent observations with negative items sampled from missing data. The mechanism is then cooperated with particle filtering techniques to simultaneously track the tendency of user preferences and item attractiveness. For top-k recommendation [21,4], a personalized list for each user is generated based on current user and item latent factors.

We discuss related work in Section 2. Particle filtering for PMF is described in Section 3. Self-training for the particle filtering based method is developed in Section 4. Experimental results are in Section 5. Conclusion is presented in Section 6.

2 Related Work

There have been few studies [13,11,20] on exploiting temporal dynamics to improve the performance of RS. Among these studies, [13] is the most related one to our work, which uses Kalman filter as temporal priors to track user latent factors. However, item latent factors are only updated but not tracked. Meanwhile,

the usage of Kalman filter restricts the dynamic and observation functions to be linear and Gaussian. Particle filtering has been used to dynamically update a log-normal distribution that models user preferences [3] in music recommendation, assuming the staticness of item popularity. Moreover, the method is not based on latent factors, and very application-specific (otherwise, no proper features).

Conventional CFs with imputation [19] all suffer from the domination of imputed ratings. Sampling missing feedback is only used in non-temporal context and one class CF (OCCF) problem [5]. An OCCF problem can be deducted from our problem by setting, for example, relevant ratings as positive examples. User-oriented and item-oriented mechanisms, which only based on the times that items or users present, are proposed in [14] as sampling methods. In these methods, recommendation accuracy is compromised in order to boost the scalability. To obtain a more accurate recommendation, samples are selected based on pairwise estimation for OCCF to iteratively train the model parameters [21]. All of these sampling methods are developed under static context for OCCF. Unlike our proposed method, these algorithms do not aim to solve problems of scalability and sparsity at the same time.

3 Particle Filtering for Matrix Factorization

Probabilistic matrix factorization method, as a model-based approach in CF, has been widely used due to its simplicity and efficiency. In particular, assuming N users and M items, let $R \in \mathcal{R}^{N \times M}$ be a user-item preference matrix with an entry $r_{u,i}$ representing the rating given by user u to item i. Rating $r_{u,i}$ is generated with a Gaussian distribution $P(r_{u,i}|U_u, V_i)$ conditioned on K dimensional user and item latent factors $U \in \mathcal{R}^{N \times K}$ and $V \in \mathcal{R}^{M \times K}$. Prior distributions $P(U)$ and $P(V)$ are formulated to contain regularization terms [15]. These latent variables are further assumed to be marginally independent while any rating $r_{u,i}$ is assumed to be conditionally independent given latent vectors U_u and V_i for user u and item i [15]. The likelihood distribution over preference matrix R is,

$$P(R|U, V, \alpha) = \prod_{u=1}^{N} \prod_{i=1}^{M} Y_{u,i} \cdot \mathcal{N}(r_{u,i}|U_u V_i^T, \alpha^{-1}), \tag{1}$$

where $\mathcal{N}(x|\mu, \alpha^{-1})$ is a Gaussian distribution with mean μ and precision α, and $Y_{u,i}$ is an indicator variable with value 1 when rating $r_{u,i}$ is not missing and value 0 when the rating is not observed. Priors $P(U)$ and $P(V)$ are given as,

$$P(U|\alpha_U) = \textstyle\prod_{u=1}^{N} \mathcal{N}(U_u|0, \alpha_U^{-1} I) \quad p(V|\alpha_V) = \textstyle\prod_{i=1}^{M} \mathcal{N}(V_i|0, \alpha_V^{-1} I).$$

Maximizing the log-posteriors over U and V is equivalent to minimizing the sum-of-square error function with quadratic regularization terms for PMF [15], leading to the following objective function,

$$E = \frac{1}{2} \sum_{u=1}^{N} \sum_{i=1}^{M} Y_{u,i} (r_{u,i} - U_u V_i^T)^2 + \frac{\lambda_U}{2} \sum_{u=1}^{N} ||U_u||_{Fro}^2 + \frac{\lambda_V}{2} \sum_{i=1}^{M} ||V_i||_{Fro}^2, \tag{2}$$

where $\lambda_U = \alpha_U/\alpha$, $\lambda_V = \alpha_V/\alpha$, and $||\cdot||_{\text{Fro}}$ denotes the Frobenius norm.

We use a state space approach [16] to track the tendency of user preferences and item attractiveness. With linear and Gaussian assumption, it is straightforward to define the state to be a joint vector of user and item latent factors, due to the existence of analytical and tractable solution. However, it is shown [15] that empirical distributions of the posterior Hence, we use particle filtering to simultaneously track these latent factors. Particle filtering iteratively approximates regions of high density as discrete sample points. As the number of particles goes to infinity, the approximation converges to the true distribution [16]. In practice, given d-dimensional state space, the number of required particles should be $O(2^d)$ to achieve a satisfiable result [16]. To make a compromise between the accuracy of user/item representation and tractability of particle filtering, we separately track latent factors for each user and item.

We assume that the tendency of user u's preference and item i's properties follows a first-order random walk driven by multivariate normal noise, due to the lack of prior knowledge. The transition functions at time t are as follows,

$$U_t^u = U_{t-1}^u + c_t^u, V_t^i = V_{t-1}^i + d_t^i, \tag{3}$$

where $c_t^u \sim \mathcal{N}(0, \sigma_U I)$ and $d_t^i \sim \mathcal{N}(0, \sigma_V I)$ are defined as unrelated Gaussian process noises. The posterior distribution of U_t^u is approximated by particle filtering with S particles as $P(U_t^u|R_{1:t}) = \sum_{s=1}^{S} w_{U,t}^{u,(s)} \delta(U_t^u - U_t^{u,(s)})$ where $w_{U,t}^{u,(s)}$ is a weight of s-th particle at time t, and $U_t^{u,(s)}$ is obtained by propagating $U_{t-1}^{u,(s)}$ using dynamics in Eq (3). The estimation of item v's latent factors at time t is obtained in a similar way. Using the transition prior in Eq (3) as the proposal distribution, the weight at time t for all the particles is evaluated recursively as $w_t = w_{t-1} \cdot P(R_t|U_t, V_t)$ where $P(R_t|U_t, V_t)$ is the observation function.

The observation function should reflect the ability of a particle to reconstruct given ratings. The objective function in Eq (2) is an immediate candidate in which $P(R|U, V, \theta) \propto e^{-E}$. However, this candidate function is sub-optimality for a top-k recommendation task, because an algorithm attempting to minimize the root-mean-squared-error in prediction does not have a satisfiable performance for a top-k recommendation task [4]. Moreover, the objective function in Eq (2) assumes that unobserved data in both training and testing cases are missing at random. That is, the probability that a rating to be missing is independent of its value. Nevertheless, it is shown [18] that feedback in RS is generally not missing at random (NMAR). Low ratings are much more likely to be missing than high ratings [18] because users are free to choose items to give feedback.

To design a suitable observation function, the key idea is to consider the ranking of *all* the items, no matter whether they are observed or not. By treating all missing data as negative with weights (wAMAN), the observation function over imputed ratings $\bar{R}_t^u$ for s-th particle of user u is as follows,

$$P(\bar{R}_t^u|U_t^{u,(s)}, \{V_t^{i,(s')}\}) = \sum_{s'=1}^{S'} exp\{-\sum_{i=1}^{M} w_{V,t}^{i,(s')} W_{u,i} (\bar{r}_t^{u,i} - U_t^{u,(s)} (V_t^{i,(s')})^T)^2\}, \tag{4}$$

where $\bar{r}_t^{u,i} = r_t^{u,i}$ if $Y^{u,i} = 1$ and $\bar{r}_t^{u,i} = r_m$ if $Y^{u,i} = 0$. The r_m is an imputed value for all the missing data, which is regarded as the average value of ratings in the complete but unknown data. Weight $W_{u,i}$ is defined to reflect the confidence over imputation and set as a global constant w_m for simplicity [18]. Latent factors $V_t^{i,(s')}$ and their weights $w_{V,t}^{i,(s')}$ represent s'-th particle for item i. The observation function over s'-th particle of item i is defined similarly. The distributions are no longer Gaussian after the introduction of w_m and an imputed value for all the missing data. Meanwhile, no regularization terms exist in Eq (4) because we obtain point mass approximation of posterior distributions via particle filtering attempting to avoid overfitting. This method is named as PFUV hereafter.

4 Personalized Self-training Method

In practice, feedback is usually unavailable before recommendation is made, which implies observation R_t at the current period is not available to estimate the tendency of user preferences and item attractiveness before recommendation. It is straightforward to use all the historic observations $R_{1:t-1}$ to approximate the estimation. However, the ratings would be dominated by the past information and cannot represent the recent tendency. An alternative approximation uses the most recent observation R_{t-1} instead. However, under temporal context, the ratings are too sparse for each user or item to track the current tendency. The data sparsity can be reduced by imputing all the missing data as shown in Eq (4). The observation function can be approximated by $P(\bar{R}_{t-1}|U_t^{u,(s)}, \{V_t^{i,(s')}\})$. However, the dynamics in this approximation will drift away from the true tendency due to the domination of imputed ratings in $\bar{R}_{t-1}$. Meanwhile, this approximation does not have a satisfiable scalability due to the usage of all the missing data.

Therefore, we exploit self-training principle [22] to solve the above mentioned problems. Instead of treating wAMAN, for each user at every time step, we will dynamically select a subset of missing items as negative samples to complement the user's most recent observation. This personalized and self-training procedure not only distinguishes the past and recent information but also avoids to dominate recent observation with imputed data. These samples are the most confident negative samples w.r.t the current prediction distribution for each user.

4.1 Self-training Sampling Method

Given user u and its current unobserved items $\mathcal{I}_t^{m,u}$, a set of $N_t^{n,u}$ items $\mathcal{I}_t^{n,u} \subseteq \mathcal{I}_t^{m,u}$ is selected by a multi-nominal distribution. The distribution is

$$P(x_1, \cdots, x_{N_t^{m,u}} | N_t^{n,u}, \theta_1, \ldots, \theta_{N_t^{m,u}}) \propto \theta_1^{x_1} \cdots \theta_{N_t^{m,u}}^{x_{N_t^{m,u}}}, \tag{5}$$

where $N_t^{m,u}$ is the number of unobserved items for user u until time t, $\{x_i | i \in \{1, \ldots, N_t^{m,u}\}\}$ represents the times that unobserved item i would be selected as negative, and $\{\theta_i | i \in \{1, \ldots, N_t^{m,u}\}\}$ is the probability that unobserved item i is disliked by user u. Without restricting x_i's to binary variables, this personalized selection is an adaptive mechanism. An unseen item with a high probability will

be selected more frequently than those with lower probability. As the accumulation of w_m for the same negative sample in Eq (4), more emphasis will be placed on the sample. By imposing such restriction, $N_t^{n,u}$ different items will be chosen. We will adopt this restriction for simplicity.

Confidence Estimation. A candidate negative sample should have a small prediction error and a small estimation variance if the sample was negative. This criterion resembles the bias and variance decomposition of generalized errors [10]. Given the historic data $R_{1:t-1}$, we define θ_i as $P(\hat{r}_{u,i} = r_m \wedge var(\hat{r}_{u,i})|R_{1:t-1})$ where $\hat{r}_{u,i} = r_m$ represents the event that the predicted rating equal to the imputed value and $var(\hat{r}_{u,i})$ represents the variance of the prediction. Assuming prediction error and variance are conditionally independent given U_t and V_t,

$$\theta_i = \int P(\hat{r}_{u,i} = r_m|U_t, V_t)P(var(\hat{r}_{u,i})|U_t, V_t)P(U_t, V_t|R_{1:t-1})dU_t dV_t$$
$$\sim \sum_{s=1}^{S}\sum_{s'=1}^{S'} w_{U,t}^{u,(s)} w_{V,t}^{i,(s')} P(\hat{r}_{u,i}|U_t^{u,(s)}, V_t^{i,(s')})P(var(\hat{r}_{u,i})|U_t^{u,(s)}, V_t^{i,(s')}), \quad (6)$$

where the predicted joint distribution of latent factors is estimated using particle filtering described below, S and S' are the number of particles used to track user u's latent factors and item i's latent factors, respectively.

Predication with Canonical Particles. To estimate a particle's weight for U_t in Eq (6), we need a weight of a particle for V_t. Likewise, we use a weight of a particle for U_t to reweight a particle for item latent factors. As the computation over all possible pairs of user and item particles is too expensive, we resort to canonical particles [7] $\hat{U}_t$ and $\hat{V}_t$ to respectively represent the total effect of particles on the estimation for each user and item latent factors at time t.

In general, $\hat{U}_t$ and $\hat{V}_t$ can be any proper function taking as input $\{(w_{U,t}^s, U_t^s)|s \in 1 \ldots S\}$ and $\{(w_{V,t}^{s'}, V_t^{s'})|s' \in 1 \ldots S'\}$. To avoid the degeneracy problem [16] in particle filtering, we will resample particles proportional to their weights After resampling, we use the expectation of posterior distributions of U_t and V_t as canonical particles, which are estimated as $\hat{U}_t^u = \sum_{n=1}^{S} w_{U,t}^{u,(n)} U_t^{u,(n)}$ and $\hat{V}_t^i = \sum_{m=1}^{S} w_{V,t}^{i,(m)} V_t^{i,(m)}$ for user and item latent factors, respectively.

Combining with canonical particles $\hat{V}_t$ and Eq 6, the prediction distribution of user u's preference over item i is estimated as follows,

$$\theta_i \sim \sum_{s=1}^{S} w_{U,t}^{u,(s)} P(\hat{r}_{u,i} = r_m|U_t^{u,(s)}, \hat{V}_t^i)P(var(\hat{r}_{u,i})|U_t^{u,(s)}, \hat{V}_t^i), \quad (7)$$

where $U_t^{u,(s)}$ are obtained by propagated $U_{t-1}^{u,(s)}$ as in Eq (3). A small distance between the imputed value and the predicated rating usually means a high confidence that the item should be negative. Thus, the probability of prediction error is defined as $P(\hat{r}_{u,i} = r_m|U_t^{u,(s)}, \hat{V}_t^i) = exp\{-|U_t^{u,(s)}(\hat{V}_t^i)^T - r_m|\}$. In terms of variance estimation, the probability of prediction variance can be estimated by $P(var(\hat{r}_t^{u,i}|U_t^{u,(1)}, \ldots, U_t^{u,(S)}, \hat{V}_t^i))$ as $exp\{\frac{1}{S-1}\sum_{s=1}^{S}(U_t^{u,(s)}\hat{V}_t^i - \hat{U}_t^u\hat{V}_t^i)\}$.

4.2 Two-Phase Self-training Method

Considering the large size and high sparsity of user-item preference matrix, the previous sampling mechanism considering all the unobserved items is infeasible in practice. We use a two-phase approach to reduce the computational complexity.

In phase I, for each user u, we sample a subset $\mathcal{I}'^{n,u}_t$ from the unobserved items $\mathcal{I}^{m,u}_t$. Generally, these sampling schemes can be implemented in terms of any distribution that properly represents NMAR. It is shown in [18] that arbitrary data missing mechanism is NMAR as long as they are missing with a higher probability than relevant ratings do. We use a uniform distribution of $\mathcal{I}^{m,u}_t$ to avoid interfering prediction distribution, which has been extensively used to handle some large datasets [21,14]. This simple and efficient distribution is a rational choice as long as we set $N^{n,u}_t$ to be a reasonably large value, which will be discussed in Section 5. For simplicity, we set $|\mathcal{I}'^{n,u}_t| = 2 * N^{n,u}_t$.

In phase II, personalized probability θ_i will be computed only for candidates $\mathcal{I}'^{n,u}_t$. Based on Eq (5), negative samples will be selected and then combined with observed data R_{t-1} to construct a sparsity reduced data $\bar{R}_t$ at time t. User and item latent factors are thus tracked by using this dynamically built data. We name this two-phase self-training method as ST-PFUV hereafter. Let $N^{n,u}_t = \hat{N}$ for simplicity. For computational complexity, at each time step, PFUV takes $O(KSMN)$. As sampling size $\hat{N}$ is usually comparable with $K \ll min(N, M)$, ST-PFUV takes $O(KS(M+N)\hat{N}) \approx O(K^2S(M+N))$, which scales efficiently as a linear function of user and item size.

5 Experiments

The performance of the proposed algorithm is tested on the popular Movielens 100K [1] and HetRec MovieLens datasets [2]. Both are public benchmark datasets. MovieLens 100K contains 530 users and 1493 items with sparsity 93.3% (at the 16-th week in 1998). HetRec contains 1775 users and 9228 items with sparsity 95.1% (in December 2008). MovieLens spans 8 months with integer scale 1 - 5 while HetRec spans 12 years with half mark scales from 1 to 5.

Protocol. In our experiments, ratings are grouped into time frames based on their timestamps. All the ratings before a predefined time instance t_{test} are training data, and ratings after it are testing data. This setting is preferred over a random split over all the data as it is infeasible to make prediction utilizing information in the future in a real-world deployment. The training periods for MovieLens 100K and HetRec are Sept. $\sim$ Dec. 1997 and Sept. 1997 $\sim$ Dec. 2007, respectively. Their testing periods are the 1st $\sim$ 16th weeks in 1998 and Jan. $\sim$ Dec. 2008, respectively. Different units of time frame are selected to ensure that ratings for each user in a time slot are not too sparse. Therefore, the task in our experiments is to predict individual's ranking over all the items in next time frame t based on all the information upto $t-1$. All the algorithms are repeated 10 times and the average results are reported. They are all implemented in Matlab with 3.3G Hz CPU and 8G memory.

We use precision@k, recall@k [4] to measure recommendation relevance, which directly assess the quality of top-k recommendations. To measure user satisfaction in recommendation, we use top-k hitrate [4,18]. As top-k hitrate recall is proportional to top-k hitrate precision [18], we only test top-k hitrate recall and name it top@k. In order to test temporal performance, the temporal extensions of those metrics are defined. Conventional accuracy metrics adopted to RS can be found in [4,10], which are omitted here due to space limitation.

For user u, precision@k over month t is denoted as $prec(k, u, t)$. During the prediction at time $t-1$, instead of using all the items in testing data as conventional precision@k does, *only* items in month t are scanned to determined their relevance to a user. The temporal precision is defined as, $prec_{temp}(k) = \frac{1}{T*N}\sum_{t=1}^{T}\sum_{u=1}^{N} prec(k, u, t)$, which is the average on all users over all the time frames T. The temporal recall $recall_{temp}(k)$ and temporal hitrate $top_{temp}(k)$ are defined in a similar fashion. As a common practice, we treat items with rating 5 as relevant items, and measure $k = 10$ for precision. For hitrate, we set $k = 10$ and each relevant item is mixed with 500 randomly sampled unobserved items to avoid spending too much computational power on evaluation. We set $k = 100$ for recall as we are ranking all the items in temporal context. We tune all the model parameters under temporal recall and use the identical setting to test the performance of algorithms under temporal recall and hitrate.

Baseline Methods. In order to test the performance of the proposed algorithms (ST-PFUV) that balances imputation and the recent observations, we compare it with the following five algorithms as part of baseline methods: PFUV, ST-PFUV-User, TopPopular, PureSVD [4] and AllRank [18]. PFUV is used to verify the efficiency and scalability of our sampling methods on incorporating temporal dynamics. It is empirically shown [14] that user-oriented sampling based on user preference has better performance than uniform sampling, we therefore hybrid the PFUV method with user-oriented sampling, and name it ST-PFUV-User. As TopPopular is a non-personalized algorithm that ranks item higher when the item is more often rated as relevant, which is included to verify the benefits of considering personalized recommendations in temporal context. PureSVD and AllRank are state-of-art algorithms developed to pursuit the top-k recommendation task. They are selected to illustrate the ability of our sampling methods to cope with non-Gaussian behaviors. These baseline algorithms are designed without exploiting any temporal information. In our experiment, to make these algorithms dynamic, we retrain all the learned models at each time step. This simple extension, which is a common practice in real-world deployment, is important to make the comparison fair. To the best of our knowledge, most of developed algorithms in temporal RS are compared with static versions of some baseline algorithms. We name these dynamic extensions as DynTopPopular, DynPureSVD and DynAllRank. To confirm the necessity of exploiting temporal information, we also adopt PMF as the only static baseline method, which always predicts the ranking without updating model parameters. To balance the accuracy, variance and scalability, we set $S = S' = 1000$ and $K = 8$ for all the PFUV-based methods. The imputed value is set to $r_m = 2$ which is

Table 1. Results on MovieLens 100K under temporal accuracy metrics. The best performance is in italic font.

Method	$prec_{temp}(k)$	$recall_{temp}(k)$	$top_{temp}(k)$
PMF	0.0310	0.3301	0.1425
DynTopPop	0.0465	0.2910	0.1782
DynPureSVD	0.0233	0.3410	0.2806
DynAllRank	0.0546	0.4140	0.3162
PFUV	0.0401	0.3626	0.2909
ST-PFUV-User	0.0408	0.3630	0.2339
ST-PFU	0.0498	0.3794	0.3115
ST-PFUV	*0.0613*	*0.4403*	*0.3543*

(a) Temporal accuracy.

Method	$prec_{temp}(k)$	$recall_{temp}(k)$	$top_{temp}(k)$
PFUV-Rect-wAMAN	0.0423	0.3310	0.2400
PFUV-Hist-User	*0.0447*	*0.3648*	*0.2439*
DynAllRank-User	0.0164	0.2358	0.1151

(b) Effects of recent and historic data.

a lower value than the average observed ratings in MovieLens 100K and Hetrec datasets. All the PFUV-based methods are initialized by AllRank.

MovieLens 100K. Table 1a shows results of above methods under temporal accuracy metrics. By cross validation method [10], we set weight $w_m = 0.05$, regularization constant λ to 0.05 and $K = 12$ for DynAllRank, and $K = 10$ for DynPureSVD and PMF. To reduce variance in particle filtering, we set $\sigma_U = \sigma_V = 0.05$. By cross validation, we set $\hat{N} = 30$ for any method involving sampling. The low values in the table are due to the fact that few relevant items exist for each user in a time frame. Compared with these baseline methods, ST-PFUV has the best performance. All the improvement is statistically significant under paired t-tests [10] with $p < 0.01$. This result verifies that the efficiency of proposed self-training sampling mechanism on modeling temporal dynamics in data and systematically selecting informative negative samples for each user. To verify the benefit of tracking item latent factors, we set $V_t = V_0$ in ST-PFUV, and name it as ST-PFU. Table 1a also shows the benefit of tracking the tendency for both users and items.

To further illustrate the power of self-training and the importance of distinguishing recent and historic ratings, we test the performance of PFUV under two extra settings, where ratings consist of 1) the recent observation and wAMAN, 2) all the historic data and missing data sampled by user-oriented distribution. We name these methods as PFUV-Rect-wAMAN and PFUV-Hist-User. We also extend the best baseline method DynAllRank by replacing wAMAN with missing data sampled by user-oriented distribution. We name it DynAllRank-User. For easy comparison, DynAllRank-User has the same setting as DynAllRank. Table 1b shows

Table 2. Results on HetRec dataset under temporal recall metric. The best performance is in italic font.

Method	PMF	DynAllRank	ST-PFUV-User	ST-PFUV
$recall_{temp}(k)$	0.1718	0.2263	0.2020	*0.3036*
time (second)	432.9	947.9	*179.3*	198.7

(a) Temporal recall and scalability. $\hat{N} = 50$ for ST-PFUV and ST-PFUV-User.

$\hat{N}$	10	20	30	40	50	60	70
ST-PFUV-User	0.1754	0.1882	0.1977	0.2026	0.2020	0.2071	0.2041
ST-PFUV	0.2536	0.2826	0.2980	0.3016	*0.3036*	0.3013	0.3032

(b) Effect of different number of samples

the performance of these methods. Combining with the results in Table 1a, results in Table 1b further confirm the ability of our methods to balance information intrinsic in recent observations and the sparsity reduction introduced by imputation to better incorporate temporal and dynamic information.

HetRec MovieLens. In following experiments, we focus on the study of accuracy and scalability of ST-PFUV over a longer period, and the influence of the number of selected samples. We compare ST-PFUV with the best baseline method DynAllRank and the static PMF method. Similar to previous experiments, ST-PFUV has the best performance in accuracy. Due to space limitation, only temporal recall is shown here. By cross validation, we set $w_m = 0.08$, $K = 40$ and $\lambda = 0.12$ for DynAllRank, and $\lambda_U = \lambda_V = 0.1$ for ST-PFUV. For PMF, we set $K = 40$, step size for gradient descent as 0.005 and $\lambda = 0.05$.

Table 2a shows the results of these methods under temporal recall, where $\hat{N} = 50$. The results show that ST-PFUV significantly outperforms other methods in terms of recommendation accuracy. All the improvement is statistically significant under paired t-tests with $p < 0.01$. Compared with results on MovieLens 100K, ST-PFUV has much greater accuracy improvement over baseline methods, verifying its much better exploiting temporal and dynamic information, especially over a longer period. As a sequential approach, our proposed algorithms do not require retraining stages. Thus, we average the total running time (both retraining time and testing time) to compare the scalability as shown in Table 2a. These empirical results confirm that the proposed sampling methods are much faster than baseline methods, and the two-phase method ST-PFUV is comparable with its one-phase counterpart. Note that PMF has been among the fastest state-of-art CF methods.

The results with different number of samples are shown in Table 2b. Upto $\hat{N} = 50$, the performance of ST-PFUV is constantly being improved as the effect of sparsity reduction. The performance is not improved significantly when $\hat{N}$ is larger, as the built observations are cluttered by negative samples.

To further evaluate temporal behaviors of ST-PFUV, we define the average of accumulated improvement (AAI) over time. Let the performance of any two

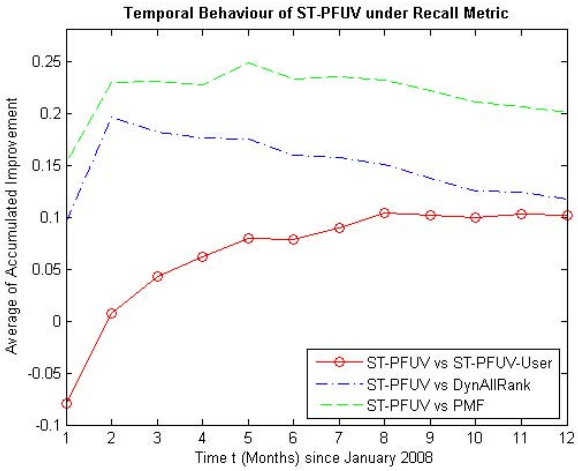

Fig. 1. Comparison of temporal behavior on HetRec dataset by AAI over time

methods under temporal recall in month t be $Rec_1(t)$ and $Rec_2(t)$, respectively. The AAI in month t_1 is $\frac{1}{t_1}\sum_{t=1}^{t_1}(Rec_1(t) - Rec_2(t))$. Figure 1 plots the AAI among ST-PFUV, ST-PFUV-User, DynAllRank and PMF methods. Except in the first month of red curve (ST-PFUV vs ST-PFUV-User), all the curves are above zero, showing that our method constantly outperforms baseline methods over time by selecting negative items via personalized self-training sampling scheme. Meanwhile, compared with DynPMF and DynAllRank, the tendency of dash and dot dash curves demonstrates that ST-PFUV is more efficient at exploiting the underlying temporal patterns. While baseline methods require longer training period, ST-PFUV performs well even if the training period is short (within 5 months) and accumulated amount of ratings is few.

6 Conclusion

In order to simultaneously solve the problems of data sparsity and scalability for temporal dynamic recommender systems, we have developed a novel two-phase self-training mechanism to dynamically construct a small but delicate set of observations from missing data. Cooperating with a particle filtering-based dynamic model, this work facilitates to track temporal dynamic user preference and item attractiveness in recommender systems.

The proposed algorithms are evaluated on two public benchmark datasets under the temporal accuracy metrics. The empirical results show that the proposed methods significantly improve recommendation performance over a variety of state-of-art algorithms. The experiments also illustrate the efficiency and scalability of the developed self-training temporal recommendation algorithms.

In future, we would like to investigate more sophisticated techniques to even better represent and learn the underlying dynamics of user preferences and item

characteristics. It is also worth exploring likelihood functions for other recommendation tasks considering multiple criterion.

References

1. MovieLens 100K dataset (2003), http://www.grouplens.org/data/
2. HetRec MovieLens dataset (2011), http://www.grouplens.org/node/462
3. Chung, T.S., Rust, R.T., Wedel, M.: My mobile music: An adaptive personalization system for digital audio players. Marketing Science 28(1), 52–68 (2009)
4. Cremonesi, P., Koren, Y., Turrin, R.: Performance of recommender algorithms on top-n recommendation tasks. In: RecSys 2010, pp. 39–46 (2010)
5. Diaz-Aviles, E., Drumond, L., Gantner, Z., Schmidt-Thieme, L., Nejdl, W.: What is happening right now ... that interests me?: online topic discovery and recommendation in twitter. In: CIKM 2012, pp. 1592–1596 (2012)
6. Ding, Y., Li, X.: Time weight collaborative filtering. In: CIKM 2005, pp. 485–492 (2005)
7. Grimes, D.B., Shon, A.P., Rao, R.P.N.: Probabilistic bilinear models for appearance-based vision. In: ICCV 2003, pp. 1478–1485 (2003)
8. Hong, W., Li, L., Li, T.: Product recommendation with temporal dynamics. Expert Systems with Applications 39(16), 12398–12406 (2012)
9. Jeong, B., Lee, J., Cho, H.: An iterative semi-explicit rating method for building collaborative recommender systems. Expert Systems Applications 36(3), 6181–6186
10. Kantor, P.B.: Recommender systems handbook (2009)
11. Koren, Y.: Collaborative filtering with temporal dynamics. In: SIGKDD 2009, pp. 447–456 (2009)
12. Liu, N.N., Zhao, M., Xiang, E., Yang, Q.: Online evolutionary collaborative filtering. In: RecSys 2010, pp. 95–102 (2010)
13. Lu, Z., Agarwal, D., Dhillon, I.S.: A spatio-temporal approach to collaborative filtering. In: RecSys 2009, pp. 13–20 (2009)
14. Rong, P., Yunhong, Z., Bin, C., Liu, N.N., Lukose, R., Scholz, M., Qiang, Y.: One-class collaborative filtering. In: ICDM 2008, pp. 502–511 (2008)
15. Salakhutdinov, R., Mnih, A.: Bayesian probabilistic matrix factorization using markov chain monte carlo. In: ICML 2008, vol. 25, pp. 880–887 (2008)
16. Sanjeev Arulampalam, M., Maskell, S., Gordon, N., Clapp, T.: A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. IEEE Transactions on Signal Processing 50(2), 174–188 (2002)
17. Sindhwani, V., Bucak, S.S., Hu, J., Mojsilovic, A.: One-class matrix completion with low-density factorizations. In: ICDM 2010, pp. 1055–1060 (2010)
18. Steck, H.: Training and testing of recommender systems on data missing not at random. In: SIGKDD 2010, pp. 713–722 (2010)
19. Su, X., Khoshgoftaar, T.M.: A survey of collaborative filtering techniques. Advances in Artificial Intelligent 2009, 4:2 (2009)
20. 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)
21. Zhang, W., Chen, T., Wang, J., Yu, Y.: Optimizing top-n collaborative filtering via dynamic negative item sampling. In: SIGIR 2013, pp. 785–788 (2013)
22. Zhu, X.: Semi-supervised learning literature survey (2006)