Temporal Recommendation on Graphs via Long- and Short-term Preference Fusion

Liang Xiang [†] [*], Quan Yuan [‡], Shiwan Zhao [‡], Li Chen [§],
Xiatian Zhang [‡], Qing Yang [†] and Jimeng Sun [$]
[†] Institute of Automation Chinese Academy of Sciences
[‡] IBM Research - China
[§] Department of Computer Science, Hong Kong Baptist University
[$] IBM T.J. Watson Research Center
xlvector@gmail.com, (quanyuan,zhaosw,xiatianz)@cn.ibm.com
lichen@comp.hkbu.edu.hk,qyang@nlpr.ia.ac.cn,jimeng@us.ibm.com

ABSTRACT

Accurately capturing user preferences over time is a great practical challenge in recommender systems. Simple correlation over time is typically not meaningful, since users change their preferences due to different external events. User behavior can often be determined by individual's long-term and short-term preferences. How to represent users' long-term and short-term preferences? How to leverage them for temporal recommendation? To address these challenges, we propose Session-based Temporal Graph (STG) which simultaneously models users' long-term and short-term preferences over time. Based on the STG model framework, we propose a novel recommendation algorithm Injected Preference Fusion (IPF) and extend the personalized Random Walk for temporal recommendation. Finally, we evaluate the effectiveness of our method using two real datasets on citations and social bookmarking, in which our proposed method IPF gives 15%-34% improvement over the previous state-of-the-art.

Categories and Subject Descriptors

H.2.8.d [**Information Technology and Systems**]: Database Applications—*Data Mining*

General Terms

Algorithms, Experimentation

Keywords

Temporal Recommendation, Graph, User Preference

*This author conducted this work while he was an intern at IBM Research - China.

KDD'10, July 25–28, 2010, Washington, DC, USA.

1. INTRODUCTION

Temporal dynamics in recommender systems are of great importance for many real-world applications, such as Amazon's product recommendation, Netflix's movie recommendation, Google's news and video recommendation. The key question is how to accurately capture user preferences over time.

Unlike traditional concept drifting mining [20, 24] that focuses primarily on global pattern shifting over time, temporal recommendation focuses on local recommendation models for each user. The temporal recommendation for individual users poses great challenges due to diverse and time-sensitive user preferences. Overall behavior of a user may be determined by her long-term interest. But at any given time, a user is also affected by her short-term interest due to transient events, such as new product releases and special personal occasions such as birthdays.

Time is certainly an important aspect in capturing temporal dynamics. Many time-evolving models [21, 22] introduce time as a universal dimension shared by all users. However, we argue that the time dimension is a local effect and should *not* be compared cross all users arbitrarily. For example, users bought different items at the same time triggered by completely different external events. Correlation among those items or users on time is typically not useful because the occurrence is purely coincident. On the other hand, it is more likely due to the same external event that a user bought multiple items in a short period. In this case, the correlation of those items on time for a specific user is significant. In another word, user-specific time dimension is more likely to capture the "true" correlation over time.

Recently, on the explicit rating data of Netflix, Koren [10] modeled the time factors for each user separately in a factorization model, but generalizing factorization models to handle implicit feedback (binary) data only gains slight improvement [5]. Moreover, explicit feedback (e.g., ratings) is not always available, and in more abundant practical situations, recommender systems have to deal with implicit feedback [16], including users' purchase history, browsing history, etc.

In this paper, we focus on explicitly modeling users' long-term and short-term preferences for recommendation on implicit datasets. There are two unique challenges: 1) how to represent the two types of user preferences? 2) how to model the interaction between the two types of preferences?

To address these challenges, we propose *Session-based Temporal Graph (STG)*, and design new algorithms to make accurate top-N recommendation on STG. The uniqueness of the proposed model is the introduction of session nodes, which capture user-specific time aspect. In particular, STG captures long-term interests through user-item connections, while short-term interests through session-item connections.

To summarize, our main contributions are as follows:

- We propose a Session-based Temporal Graph (STG) which incorporates temporal information to model long-term and short-term preferences simultaneously.
- Based on STG framework, we propose a new algorithm, Injected Preference Fusion (IPF), to balance the impacts of users' long-term and short-term preferences for accurate recommendation. IPF is also a computationally efficient online method.
- To further demonstrate the model generality, we extend the personalized Random Walk [4] for temporal recommendation based on STG.
- We systematically compare our approaches with other algorithms on two real datasets. The results confirm that STG is effective for incorporating temporal data into the graph, and our proposed algorithms, especially IPF, are effective to balance users' long-term and short-term preferences for accurate recommendation.

The rest of the paper is organized as follows. Section 2 provides a brief review of related work on temporal dynamics in recommendation and graph-based recommendation respectively. Section 3 formally presents STG. Section 4 introduces our novel algorithm IPF for recommendation based on STG, along with an extension to the Random Walk model. Section 5 presents the experimental results on two real datasets. Finally, we conclude in Section 6.

Table 1 gives the main symbols used in this paper.

Symbol	Description
G	the bipartite graph STG
E	the edge set of G
U, I, S	user node set, item node set, session node set
v_u, v_i, v_{ut}	user node, item node, session node
w	the weight function defined on edges
$N(u)$	all items viewed by the user u
$N(u,t)$	all items viewed by the user u at time t
P	a path in STG, from a user or session node to an unknown item node
$\mathcal{P}(u,i)$	the set of paths from $\{v_u, v_{ut}\}$ to v_i
$\phi(P)$	the weight of the path P
$\gamma(v)$	injected preferences on the source node v
$\psi(v_k, v_{k+1})$	the propagation function from v_k to v_{k+1}
$out(v)$	the out node set of the node v
ρ	the parameter to control the impact of nodes' out degree in preference propagation
β	the parameter to adjust the ratio of injected preferences between user and session nodes
η	the parameter to adjust the edge weight from item nodes to user/session nodes

Table 1: Symbols

2. RELATED WORK

In this section, we introduce related work on fusing temporal dynamics into recommenders at first, and then describe the graph-based algorithms for recommendation and influence propagation, which related to our algorithms.

2.1 Temporal Dynamics in Recommendation

Ding [2] presented a novel algorithm to compute the time weights for different items by decreasing weights to old data. This approach has a disadvantage due to latest data are not always important while old data are not trivial all the time. Zimdars [25] treated collaborative filtering (CF) as a univariate time series problem, and transforming the data to encode time order information, and following a decision-tree learning process to compare it with CF without ordering information. Lathia [11] formalized CF as a time-dependent iterative prediction problem, and found that certain prediction methods that improve prediction accuracy in the Netflix do not show similar improvement over a set of iterative train-test experiments with growing data. So they proposed a method to automatically assign and update per-user neighborhood sizes other than setting global parameters. Compared to these work which handled the temporal dynamics in different ways, we focus on explicitly modeling users' long-term and short-term preferences and their combinational impacts on recommendation.

Ricci [19] proposed a critique-based mobile recommendation methodology, which collected long-term preferences both by mining past interactions and by letting users explicitly define a set of stable preferences; and introduced a type of critique that lets users express additional session-specific preferences. Recently, Koren [10] predicted movie ratings for Netflix by modeling the temporal dynamics via a factorization model. Our focus is on using graph to explicitly model users' long-term and short-term preferences, and make accurate top-N recommendation based on implicit binary data.

2.2 Graph-based Recommendation

Graph-based methods have been introduced into recommendation to model the interaction between users and items on a graph, and compute node similarity from a global perspective, instead of local pairwise computation of neighborhood.

Baluja et al. [1] built a video co-view graph, and then adopted the label propagation method in semi-supervised learning to make video recommendation for *YouTube*. He summarized three rules for a good recommendation, i.e., the user u will have a high preference on the item i if: 1) u and i have a short path between them; 2) u and i have several paths between them and these paths are not very long; 3) u and i have paths that avoid high-degree nodes. We are inspired from these rules in developing our IPF algorithm, though IPF is based on a user-item bipartite graph, and uses a local search strategy instead of iterative global propagation.

Random walks on graphs have been extensively discussed and shown a rather good performance in recommendation. PageRank was a famous random walk model [17] being used in search engines to rank items globally, while Haveliwala [4] proposed a topic-sensitive PageRank which introduced the topic vector to make personalized search. As the topic vector can also be used as a personalized vector to bias users' pref-

erences on items, [3, 15, 12, 7] proposed their personalized random walk models respectively to make recommendation in different domains. For example, Gori et al. [3] proposed a random walk based scoring algorithm, ItemRank, which can be used to rank products according to expected user preferences. Recently, Jamali et al. [7] proposed TrustWalker, a random walk model combining the trust network and collaborative filtering for recommendation, and got the results by performing random walks on the graph. Li et al. [12] described a recommender system in the domain of grocery shopping based on basket-sensitive random walk. Besides random walk models, Huang [6] proposed a two-layered graph model for products recommendation based on the association strengths between a customer and products. However, all these work do not consider the temporal information in the random walk model.

Li et al. [13] proposed a method to incorporate temporal data on the graph for expert search by dividing the data into bins for each time window, building a graph for each window, and linking the neighboring graphs by forward and backward edges. Finally they applied random walks on this graph to rank experts. This approach introduced too many item nodes since one item appears across many windows; and it didn't address users' long-term and short-term preferences particularly.

In a word, to the best of our knowledge, we haven't seen any work on modeling users' long-term and short-term preferences on a graph, and applied this model in recommendation.

3. STG CONSTRUCTION

In this section, we illustrate how to model users' long-term and short-term preferences on the graph by using different node types, and treating edges with different weights.

Our data are in the form of <*user*, *item*, *time*> triples which are usually modeled by a tri-partite graph [8] or a tensor [22]. However, both tri-partite grpah and tensor treat time as a universal dimension shared by all users, while we argue that, in a recommender system, the time dimension is a local effect and should not be compared cross all users arbitrarily. Correlation of users/items on time is typically not useful, while correlation of items on time for a specific user is significant, i.e., items within a user session are somewhat more relevant.

Therefore, we create a new node type - session (associated with a user at specific time), to enable new linkages between items. In this way, we transform <*user*, *item*, *time*> into <*user*, *item*> and <*session*, *item*> by dividing the time into bins and binding the bins with corresponding users. Note that the session node is a combined node with a user and a specific time bin. The length of a session can last for one hour, one day or one week, etc., which depends on the domain we are facing.

Based on <*user*, *item*> and <*session*, *item*>, we create the graph and name it Session-based Temporal Graph (STG). STG is a directed bipartite graph $G(U, S, I, E, w)$, where U denotes the set of user nodes, S is the set of session nodes, and I is the set of item nodes. $w : E \to \mathbf{R}$ denotes a non-negative weight function for edges. Figure 1 is a simple example of STG containing 2 user nodes, 3 session nodes, and 4 item nodes. It shows user u_1 interacted with items i_1, i_2, i_3, and user u_2 interacted with items i_3, i_4. Furthermore, items i_1 and i_2 are also linked to the session node s_1

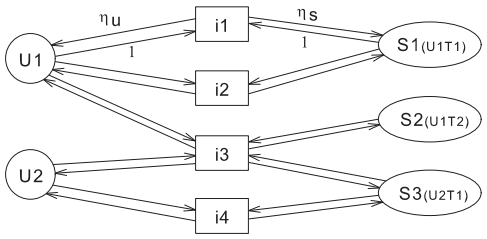

Figure 1: An example of STG.

because they were co-viewed by user u_1 at time t_1; and i_3 is connected with the session node s_2, i_3 and i_4 are connected with the session node s_3 too.

In STG, user node v_u connects to all items viewed by user u, denoted as $N(u)$, representing u's long-term preferences; session node v_{ut} only connects to items user u viewed at time t, denoted as $N(u, t)$, representing u's short-term preferences at time t. Therefore, if we start walking from the user node v_u, we will pass through $N(u)$ and then reach unknown items which are similar to items in $N(u)$; and if we walk from the session node v_{ut}, we will reach unknown items similar to items in $N(u, t)$. In this way, we say that the user node is a representation of the users' long-term preferences and the session node is a representation of short-term preferences.

The edge weights of G are defined as:

$$w(v, v') = \begin{cases} 1 & v \in U \cup S, v' \in I \\ \eta_u & v \in I, v' \in U \\ \eta_s & v \in I, v' \in S \end{cases} \quad (1)$$

This definition means, given an edge $e(v, v')$ starting from user or session nodes, its weight will be 1, since both user and session nodes are only connected to item nodes and hence arbitrary weight assignment will have the same effect after normalization based on their out-degrees. For the edge that starts from item nodes, if it connects to a user node, its weight is η_u; if it connects to a session node, its weight is η_s. Here we use different weights to model the influence of long-term and short-term preferences more precisely. If two items are connected by user nodes, their item-item similarity is contributed by long-term preferences; if two items are connected by session nodes, their item-item similarity is contributed by short-term preferences.

4. RECOMMENDATION ON STG

Users' long-term and short-term preferences are modeled by user nodes and session nodes respectively in STG, and now we will uncover the interaction of the two preferences, and their impacts on recommendation by a novel algorithm called IPF. Furthermore, based on STG, we extend the personalized Random Walk to model the impacts of long-term and short-term preferences and make temporal recommendation.

4.1 Injected Preference Fusion

When modeling the interaction of long-term and short-term preferences for user u and making recommendation at time t, our basic idea is to consider the user node v_u and its related session node v_{ut} as the sources to be injected with user preferences. Preferences injected into the user node will be propagated to items $N(u)$ viewed by the user at all time,

and then tend to propagate to unknown items approximate to u's long-term preferences; while preferences injected into the session node will propagate to items $N(u,t)$ viewed by the user at time t, and then tend to propagate to unknown items approximate to u's short-term preferences. In STG, there will be a bunch of propagation paths from the source nodes to the unknown item nodes. The user preferences to an unknown item node is the sum of the weights of all incoming paths from both user and session nodes, and the one which gets the highest preferences will be considered as the best one to be recommended to the user. The preferences propagated to this node is the consensus of the long-term and short-term preferences, balancing the two types of impacts.

Given a user u and the time t, both user node v_u and session node v_{ut} will be chosen as source nodes. Note that only the current session node v_{ut} is selected from among all session nodes, because we adopt an appropriate bin size to include most short-term influence in the current session node. Then, if $P\{v_0, v_1, ..., v_n\}$ is the path from the source node $v_0 \in \{v_u, v_{ut}\}$ to an unknown item node v_n, the final preference value propagated to v_n is defined as:

$$\phi(P) = \prod_{v_k \in P, 0 \leq k < n} \psi(v_k, v_{k+1}) \gamma(v_0) \tag{2}$$

where $\gamma(v_0)$ is the value of injected preferences on the source node, and its value depends on the node type:

$$\gamma(v_0) = \begin{cases} \beta & v_0 = v_u \\ 1 - \beta & v_0 = u_{ut} \end{cases} \tag{3}$$

Here, β is a parameter used to tune the ratio of injected preferences on the user node against the session node. $\beta = 0$ means no preferences are injected into the user node; while $\beta = 1$ means no preferences are injected into the session node.

$\psi(v_k, v_{k+1})$ is a propagation function that measures how many preferences of v_k is propagated to its succeed node v_{k+1}. It is defined as:

$$\psi(v_k, v_{k+1}) = \left(\frac{w(v_k, v_{k+1})}{\sum_{v' \in out(v_k)} w(v_k, v')}\right)^{\rho} \tag{4}$$

where $out(v_k) = \{v' \in V : e(v_k, v') \in E\}$ and $\rho \in [0, 1]$ is a parameter to tune the impact of the out-degree in the propagation process. This definition means node v_{k+1} will get lower incoming preference if node v_k has larger out-degree. By applying edge weight definition (see Equation 1), Equation 4 can be refined as:

$$\psi(v, v') = \begin{cases} \frac{1}{|out(v)|^{\rho}} & v \in U \cup S, v' \in I \\ \left(\frac{\eta}{\eta |out(v) \cap U| + |out(v) \cap S|}\right)^{\rho} & v \in I, v' \in U \\ \left(\frac{1}{\eta |out(v) \cap U| + |out(v) \cap S|}\right)^{\rho} & v \in I, v' \in S \end{cases} \tag{5}$$

where $\eta = \eta_u / \eta_s$ is a parameter which controls the ratio of preferences (from an item node) to a user node against to a session node, thus affects the importance of long-term and short-term factors respectively on measuring item-item similarity. If $\eta = \infty$, two items are only connected via user nodes and it means only users' long-term preferences can contribute to item-item similarity. If $\eta = 0$, two items are only connected via session nodes and thus only short-term preferences will contribute to item-item similarity.

As STG is a bipartite graph, the distance from user/session nodes to unknown item nodes should be odd numbers equal to or larger than 3. Given a user and an unknown item, there will be many paths (distance >= 3) between these two nodes in STG. But we only consider the shortest paths to measure the user's preferences on the item for two reasons: one is that long paths contribute less weights than short paths to the final item nodes and may even bring in noise; the other is that the shortest paths can be obtained effectively by Bread-First-Search.

Consequently, we use $\mathcal{P}(u, i)$ to represent the set of shortest paths from source nodes to an unknown item node v_i for the user u, and the estimated preference p_{ui} of user u on item i is then measured as:

$$p_{ui} = \sum_{P \in \mathcal{P}(u,i)} \phi(P) \tag{6}$$

where $\phi(P)$ is the weight of the path P defined in Equation 2.

Finally, we sort p_{ui} for all unknown items, and then return top-N unknown items to user u.

IPF is implemented by Bread-First-Search on STG. The pseudo code of IPF is shown in Algorithm 1.

In a word, given a user u, there are four types of shortest paths starting from source nodes v_u, v_{ut} to unknown item nodes with three steps.

- user-item-user-item (P1): this type of paths start from v_u, and then jump to all items in $N(u)$ viewed by u, and at last jump to u's unknown items through other users who also viewed items in $N(u)$;
- user-item-session-item (P2): similar to P1, but connect viewed items with unknown items by session nodes instead of user nodes;
- session-item-user-item (P3): this type of paths start from a session node v_{ut}, and jump to all items in $N(u,t)$ viewed by u at t, then reach similar unknown items;
- session-item-session-item (P4): similar to P3, but connect viewed items with unknown items by session nodes instead of user nodes.

Thus in summary, P1 and P2 start from user node v_u, and at last reach unknown item nodes similar to $N(u)$, reflecting u's long-term preferences. P3 and P4 start from session node v_{ut}, and at last reach unknown item nodes similar to $N(u,t)$, reflecting u's short-term preferences. P1 and P3 connect items by user nodes, measuring the item relevance mainly from users' long-term preferences. P2 and P4 connect items by session nodes, measuring the item similarity mainly from users' short-term preferences.

For the special case without temporal data (no session nodes), only P1 is available in the user-item bipartite graph. IPF can also be used for non-temporal recommendation by only injecting preferences on the source user node. We name this type of IPF Single Source IPF (SS-IPF) because the preference propagation only starts from user nodes and follows P1 paths. IPF in STG with temporal data is called Multi Source IPF (MS-IPF) for its use of both user and session nodes as injected source nodes.

Algorithm 1: Pseudo code of IPF to make recommendation for active user u at time t.

```
Data: STG G, user u, time t
Result: Recommendation for user u at time t
Queue Q;
NodeSet V;
Q.append(v_u);
Q.append(v_ut);
distance[v_u] = distance[v_ut] = 0;
rank[v_u] = β;
rank[v_ut] = 1 − β;
while Q is not empty do
    Node v = Q.top();
    if V.contains(v) then
        continue;
    if distance[v] > 3 then
        break;
    V.insert(v);
    foreach v' ∈ out(v) do
        if !V.contains(v') then
            distance[v'] = distance[v] + 1;
            Q.append(v');
        if distance[v] < distance[v'] then
            rank[v'] = rank[v'] + rank[v] ·ψ(v, v');
rank.sort();
return top-N unknown items;
```

4.2 Temporal Personalized Random Walk

Recommendation via personalized random walks on graphs has been well studied [3, 15, 12, 7]. Here, we extend PageRank, one of the most typical random walk algorithms, as temporal personalized random walk (TPRW) to support temporal recommendation.

The following formulation is used by PageRank [4] to rank nodes in a graph:

$$\mathrm{PR} = \alpha \cdot M \cdot \mathrm{PR} + (1-\alpha) \cdot d \tag{7}$$

where α is the damping factor, M is a transition matrix and vector d defined below is a user-specific personalized vector indicating which nodes the random walker will jump to after a restart:

$$d(v) = \begin{cases} 1 & v = v_u \\ 0 & otherwise \end{cases} \tag{8}$$

The basic idea of TPRW is similar to personalized PageRank. When making recommendation for the active user u at time t, vector d should bias both user node v_u and related session node v_{ut}, which means that the active user would like to jump to his own user node and corresponding session node after a restart:

$$\widetilde{d}(v) = \begin{cases} \beta & v = v_u \\ 1-\beta & v = v_{u,t} \\ 0 & otherwise \end{cases} \tag{9}$$

d is $\widetilde{d}$ after L-1 normalization. This formula also contains the idea of fusing users' long-term and short-term preferences by using a temporal personalized vector.

In order to make recommendation for user u at time t, we need to construct d for each user firstly, and run Equation 7 to rank all nodes on STG. Finally, we only keep the top-N item nodes for recommendation.

4.3 Complexity Analysis

This section discusses the complexity of all the algorithms. IPF is based on Breadth-First-Search (BFS) on the graph and no iteration is involved. In IPF, the injected preferences propagate on a subgraph of STG which only includes nodes whose distance is less than or equal to 3-step from source node(s). Like BFS, the major computational cost of IPF comes from the final step of preference propagation, i.e., from length-2 nodes to length-3 nodes. For example, in SS-IPF, the length-2 nodes are user nodes, and length-3 nodes are item nodes. In the worst case, the length-2 nodes contain all the user nodes, and then the preferences are propagated to all the item nodes, in which all the edges are involved in the propagation, so the complexity for all users is $O(e \cdot |U|)$, where e is the total number of edges, and $|U|$ is the number of user nodes. For MS-IPF, due to the introduction of session nodes and corresponding session-item edges, we assume that its number of edges is k times larger than SS-IPF, so its complexity is $O(k \cdot e \cdot |U|)$.

For TPRW, it walks on the global graph, and is implemented in an iterative method, so for all users its complexity is $O(m \cdot (e) \cdot |U|)$, where m is the iteration times, and e is the number of edges. For userKNN, because it is based on the pair-wise similarity computation of users, and the average items viewed by each user is $e/|U|$, its complexity is $O(|U|^2 \cdot (e/|U|))$, which equals to $O(e \cdot |U|)$. In the same way, the complexity for itemKNN is $O(e \cdot |I|)$, where $|I|$ is the total number of item nodes.

From above complexity analysis, IPF is rather efficient compared to TPRW because it is a local search algorithm and does not need iterations. When comparing the complexity between IPF and UserKNN/ItemKNN, we can see that the complexity of SS-IPF equals to userKNN, and may be slower or faster than itemKNN depending on the ratio between the number of users and items. MS-IPF is slower than SS-IPF due to the growth of edges, but k is a constant, and from our experience on the two datasets, its value approximately equals to 4.

5. EXPERIMENTS

5.1 Data Description

As our goal is to make accurate top-N recommendation on implicit binary data abundant on the web, we adopt two real binary datasets, CiteULike[1] and Delicious[2], instead of explicit rating data like Netflix. CiteULike is a free online service to organize academic publications. It allows users to tag or bookmark research papers. CiteULike releases a "who bookmarked what" data which contains 52,689 users, 1,793,954 items and 2,119,200 user-item pairs from Nov. 2004 to Sep. 2009. This dataset is very sparse and many papers are only tagged once. Our dataset is a dense subset which contains 4,607 users, 16,054 items and 109,346 user-item pairs. The dataset is made by firstly removing papers which were posted by less than 5 users and then removing

[1]The CiteULike data can be downloaded from the website of CiteULike (`http://www.citeulike.org`).
[2]The Delicious data can be downloaded from the website of DAI-Labor (`http://www.dai-labor.de`).

users who posted less than 5 papers. The sparsity of the resulting dataset is 99.85%.

Delicious is a social bookmarking web service for storing, sharing, and discovering web bookmarks. Users can tag web pages from different web sites in Delicious. In the experiment, only web pages from Wikipedia and users' preferences on Wikipedia pages are extracted. Then, items which were tagged by less than 10 users and users who tagged less than 4 items were deleted. The resulting dataset contains 8,861 users, 3,257 items and 59,694 user-item pairs from Dec. 2005 to Dec. 2007. The sparsity of this dataset is 99.79%.

Furthermore, we are also interested in the average activity level of users and the average lifecycle of items in these two datasets. The activity level of a user is the number of days this user is active in the system while the lifecycle of an item is the number of days this item is viewed/tagged by some users. In CiteUlike dataset, the average user activity level is 313 days and the average item lifecycle is 701 days. In Delicious dataset, the average user activity is 138 days and average item lifecycle is 309 days.

5.2 Evaluation Metric

We adopt the All-But-One evaluation method and use Hit Ratio [9] as the metric for the Top-N recommendation. Our datasets were splitted into training part and testing part: for every user, the latest item she likes is selected as test data and other items are selected as training data.

When make recommendation, we generate a list of N (N=10) items named $R(u,t)$ for each user u at time t. If the test item appears in the recommendation list, we call it a hit. The Hit Ratio is calculated in the following way:

$$\text{Hit Ratio} = \frac{\sum_u I(T_u \in R(u,t))}{|U|}$$

where $I(\cdot)$ is an indicator function, $R(u,t)$ is a set of top-N items recommended to user u at time t, T_u is the hold-off item that user u accessed at time t.

5.3 Compared Methods

In last two decades, many algorithms have been proposed for top-N recommendation on binary data. Two most famous algorithms of them are user-based collaborative filtering [18] (UserKNN) and item-based collaborative filtering [14] (ItemKNN). These two algorithms, UserKNN and ItemKNN, together with personalized PageRank [4] (PersonalPR, which is extended to TPRW by us to support temporal data), a graph-based top-N recommendation algorithm, will be used to compare with SS-IPF in the experiment of non-temporal top-N recommendation. Note that we do not compare our algorithm with SVD-like factorization methods, because they only achieve slight improvement on implicit binary data [5].

Research on temporal top-N recommendation on binary data began in recent years, but only got a little attention. Ding et al. [2] and Andreas [23] both proposed a time-weighted item-based collaborative filtering method (TItemKNN) by reducing the influence of old data when predicting users' further behavior. TItemKNN was designed for rating prediction task but it can be easily extended to Top-N recommendation for binary data. Similarly, UserKNN can also be revised to support temporal recommendation (TUserKNN). Therefore, TItemKNN and TUserKNN are selected to compare with two STG-based temporal algorithms, TPRW and MS-IPF.

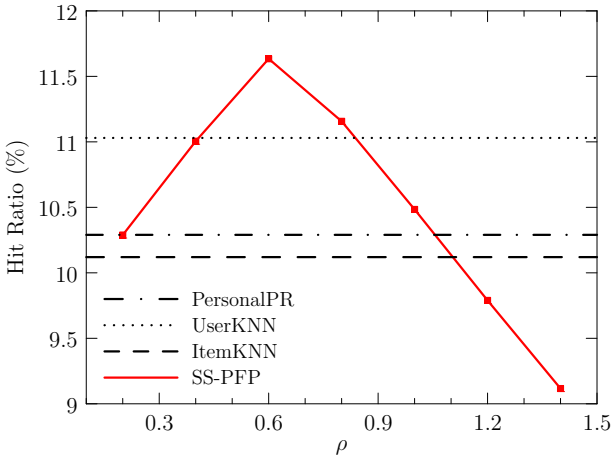

Figure 2: The parameter ρ's impact of SS-IPF on CiteULike dataset.

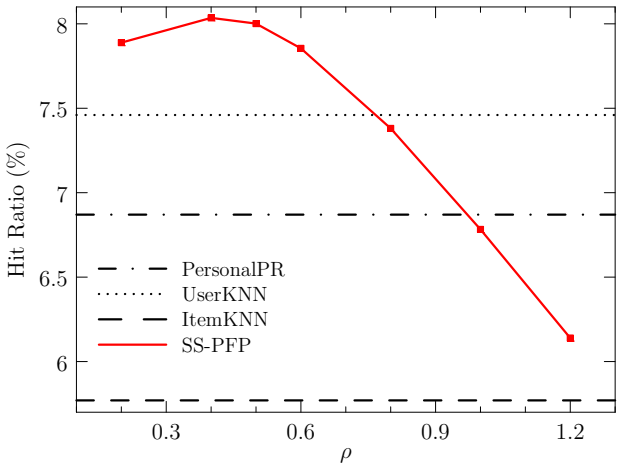

Figure 3: The parameter ρ's impact of SS-IPF on Delicious dataset.

5.4 Results without Temporal Data

We first present the results on user-item matrix (without temporal data) for two reasons: one is that we want to study the importance of the parameter ρ (Equation 4) which controls the impact of out-degree in the propagation process, and then fix it for following experiments; the other is that the results can be used as a baseline to see whether temporal data can improve recommendation accuracy.

For the user-item data, the propagation factor ψ defined in Equation 5 will become

$$\psi(v,v') = \frac{1}{|out(v)|^{\rho}}$$

We compare the Hit Ratio of our SS-IPF with other three algorithms, ItemKNN, UserKNN, and PersonalPR [4], on CiteULike and Delicious datasets. Results are shown in Figure 2 and Figure 3.

The results show that ρ is important in determining the Hit Ratio, which means in the propagation process, the

element	CiteULike	Delicious
#user	4607	8861
#item	16054	3257
#session	39939	45463
#edge	437384	238776

Table 2: Basic information of STG constructed on CiteULike/Delicious datasets

propagated preferences are not linearly proportional to the node's out-degree. In CiteULike, the optimal ρ is about 0.6, and in Delicious, it is about 0.4. Both of them are less than 1, so it means the preferences will be amplified when propagated to next nodes, and we do not ensure the total sum of the preferences on final unknown nodes equals to the sum of source nodes. The Hit Ratios of other algorithms are under these optimal parameters: in CiteULike dataset, the neighborhood size of ItemKNN is 100, the neighborhood size of UserKNN is 60 and damping factor α of PersonalPR is 0.5. In Delicious dataset, the neighborhood size of ItemKNN is 300, the neighborhood size of UserKNN is 60 and damping factor α of PersonalPR is 0.5.

In the following experiments, we fix ρ to 0.6 for CiteULike dataset and 0.4 for Delicious dataset.

5.5 Results with Temporal Data

In this section, we illustrate the results of temporal top-N recommendation. Firstly, we split the time into bins for the construction of session nodes. According to our experiments, the optimal time unit in CiteULike is one week while in Delicious is one day. This is owing to the difference between papers and web pages. Then we build the STG on $<user, item>$ and $<session, item>$ data. Detail information of the STGs based on two datasets is shown in Table 2.

In the following, we investigate the impacts of the parameters β and η respectively, and then compare the Hit Ratios of the four temporal algorithms: MS-IPF, TPRW, TItemKNN, and TUserKNN.

5.5.1 Long-term versus short-term influence by β

This section focuses on analyzing the parameter β (Equation 3 and 9), which governs the influence of long-term and short-term preferences in temporal recommendation. MS-IPF and TPRW are STG-based algorithms and above equations (Equation 3 and 9) show that they are both related to β. However, the meaning of β varies for each algorithm. In MS-IPF, β controls the ratio of injected preferences into user nodes against session nodes. If β equals to 0, no preferences are injected into the user node; if β equals to 1, no preferences are injected into the session node. In TPRW, β adjusts the probability of jumping to the user node and session node after a restart in the form of personalized vector. Although they are in different forms, β balances the long-term and short-term influence in both algorithms, the bigger the β is, the bigger the influence of long-term preferences.

When tuning β, we set the parameter η to 0.5, which means we balance the influence of long-term and short-term preferences on measuring item-item similarity. The results of how Hit Ratios change against β of both algorithms are shown in Figure 4 and Figure 5. TUserKNN and TItemKNN do not have parameter β, so their Hit Ratios are drawn as a straight line. Firstly, the results show that MS-IPF outperforms other algorithms when $\beta \in [0.1, 1]$ in both datasets. Secondly, the results also show that ignoring long-term preferences ($\beta = 0$) can not generate good results while ignoring short-term preferences ($\beta = 1$) can not generate good results either. Optimal results can be got by combining long-term and short-term factors together. Moreover, one advantage of our algorithms is that β is rather stable: arbitrary $\beta \in [0.1, 0.5]$ can generate results with similar accuracy. Therefore, we simply fix β to 0.5 in the following experiments.

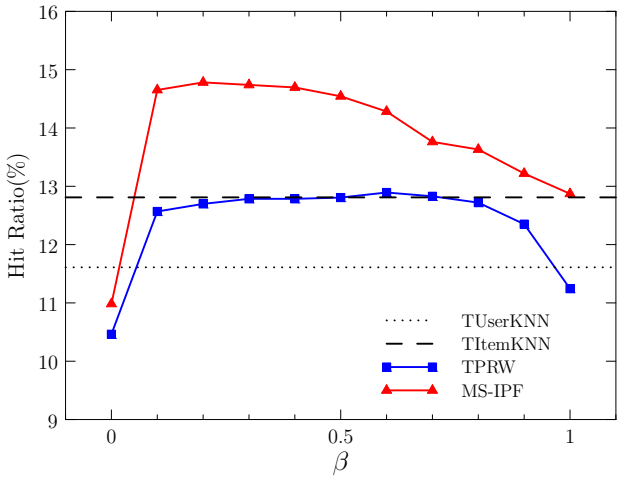

Figure 4: Hit Ratios of all algorithms with different β on CiteULike dataset ($\eta = 0.5, \rho = 0.6$).

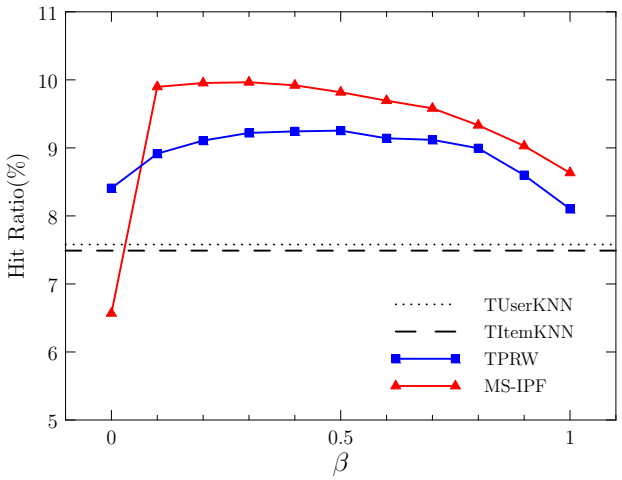

Figure 5: Hit Ratios of all algorithms with different β on Delicious dataset ($\eta = 0.5, \rho = 0.4$).

5.5.2 Long-term versus short-term linkage by η

In this section, we look into the parameter η (Equation 5), which is used to balance the influence of long-term and short-term preferences from items' angle. $\eta = 0$ means items are only connected by user nodes and item-item similarity only depends on users' long-term preference while $\eta = \infty$ means items are only connected by session nodes and item-item similarity only depends on users' short-term preferences. Both Figure 6 and Figure 7 describe the trends of

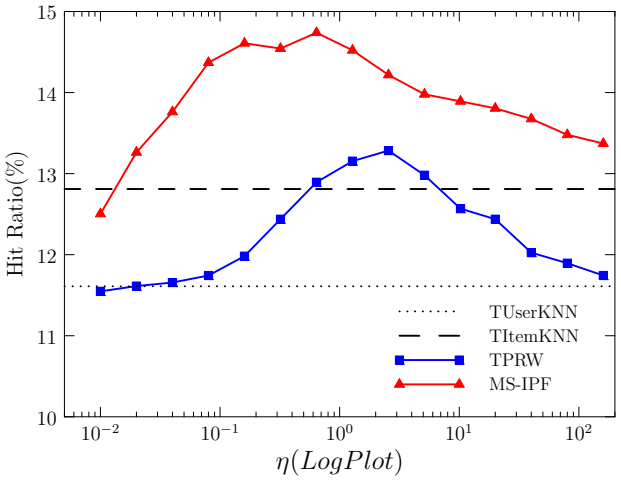

Figure 6: Performance of all algorithms with different η on CiteULike dataset ($\beta = 0.5, \rho = 0.6$).

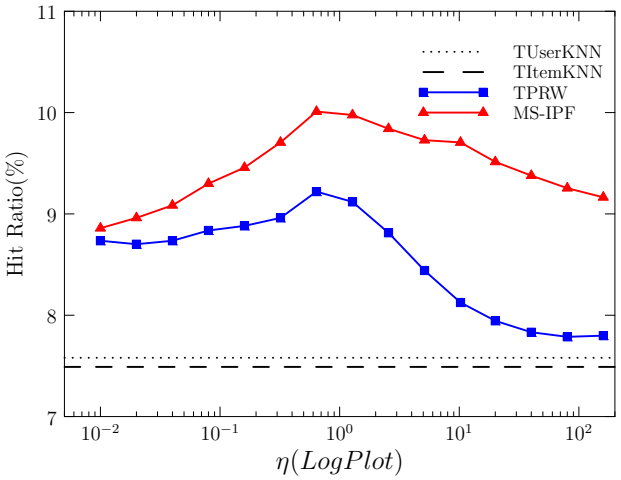

Figure 7: Performance of all algorithms with different η on Delicious dataset ($\beta = 0.5, \rho = 0.4$).

accuracy against η. As the X-axis is the logarithmic value of η, we can see in both datasets, η being close to 1 can produce good results, which means in both datasets, both users' long-term and short-term interests are important to measure item similarity. Furthermore, results show that η is not a sensitive parameter either.

5.5.3 Selection of time window size

Before creating session nodes, we split time into bins. When determining the window size, two criteria were considered. One is recommendation accuracy which means selecting of a proper time window to generate a more accurate recommendation. The other is computational complexity. As the smaller the window size is, the larger the number of session nodes and edges would be, which in turn increases computational complexity, so when recommendations from two time window sizes are with similar accuracy, we prefer to choose the one with larger window size. In order to see the impact of the window size on recommendation accuracy, we did experiments on both datasets, and results are shown in Table 3.

time window (days)	CiteULike	Delicious
1	13.85%	9.83%
2	13.70%	9.72%
3	13.70%	9.72%
4	13.76%	9.72%
5	13.81%	9.74%
6	13.87%	9.68%
7	13.85%	9.69%
15	13.76%	9.59%
30	13.81%	9.48%
45	13.35%	9.24%
60	13.24%	9.20%
90	13.19%	8.83%

Table 3: Time window size's impact on Hit Ratio of MS-IPF ($\beta = 0.5, \eta = 0.5, \rho = 0.5$)

We observed that in CiteULike, recommendations with time window size less than one month have similar accuracy, and the optimal size is 6 to 7 days (one week or so), which is our setting in previous experiments. For Delicious data, recommendations with window size less than 7 days have similar accuracy, while the optimal size is one day. The above results indicate that users' interests on research topics (CiteULike) drift more slowly than interests on browsing web pages (Delicious), which is consistent with our real life experience. Moreover, it also shows that IPF's accuracy is not sensitive to the selection of window size. As long as we set the window size in a proper range, it will generate similar results.

5.5.4 Overall accuracy comparison

In this section, we compare the overall accuracy of all algorithms in temporal Top-N recommendation.

For the accuracy comparison, table 4 shows the best Hit Ratios of all algorithms on CiteULike data under optimal parameters. Results show that TUserKNN has the lowest Hit Ratio while MS-IPF has the highest Hit Ratio. By using TItemKNN [2] as the benchmark, TPRW outperforms TItemKNN 4.75%, and MS-IPF improves it up to 15.02%. Table 5 shows the best Hit Ratios on Delicious dataset. In this dataset, TItemKNN and TUserKNN have close results, and there is a gain of 25.37% over TItemKNN from TPRW, and 34.46% from MS-IPF. It is worth noting that, on STG, TPRW outperforms TUserKNN on both datasets, while on the user-item graph without temporal data, UserKNN outperforms personalPR instead, this strongly proves that STG is effective for incorporating temporal data into the graph. Moreover, MS-IPF outperforms all the other temporal algorithms proves that IPF is effective to balance users' long-term and short-term preferences for accurate recommendation.

Finally, comparing MS-IPF with SS-IPF, MS-IPF achieves 26.87% improvement over SS-IPF on CiteULike dataset, and achieves 25.88% improvement on Delicious dataset. This proves that time factors play an important role in improving recommendation accuracy, and our STG-based algorithm, MS-IPF, is effective in leveraging temporal dynamics for recommendation.

Method	Hit Ratio	Improvement
TItemKNN	12.85%	–
TUserKNN	11.63%	-9.49%
TPRW	13.46%	4.75%
MS-IPF	14.78%	15.02%

Table 4: Best results of different temporal recommendation algorithms on CiteULike dataset

Method	Hit Ratio	Improvement
TItemKNN	7.49%	–
TUserKNN	7.58%	1.2%
TPRW	9.39%	25.37%
MS-IPF	10.07%	34.45%

Table 5: Best results of different temporal recommendation algorithms on Delicious dataset

6. CONCLUSION

User preferences often exhibit long-term and short-term factors. Tracking and leveraging these factors for temporal recommendation poses great challenges. In this paper, we propose a simple graph-based model named Session-based Temporal Graph (STG) to efficiently capture the long-term and short-term factors over time. Based on the STG framework, we propose an online method, Injected Preference Fusion (IPF) for temporal recommendation. In addition, we extend the personalized Random Walk for temporal recommendation based on STG. The extensive experiments on real datasets confirm the effectiveness of the proposed method over other existing ones. Future work includes the extension of STG using user clusters and multiple time windows, as well as a more flexible approach in determining time window size and selecting relevant session nodes. We also plan to explore other applications for STG beyond recommendation, such as user characterization.

7. REFERENCES

[1] S. Baluja, R. Seth, D. Sivakumar, Y. Jing, J. Yagnik, S. Kumar, D. Ravichandran, and M. Aly. Video suggestion and discovery for youtube: taking random walks through the view graph. In *WWW '08*, pages 895–904, 2008.

[2] Y. Ding and X. Li. Time weight collaborative filtering. In *CIKM '05*, pages 485–492, 2005.

[3] M. Gori and A. Pucci. Research paper recommender systems: A random-walk based approach. In *WI '06*, pages 778–781, 2006.

[4] T. H. Haveliwala. Topic-sensitive pagerank. In *WWW '02*, pages 517–526, 2002.

[5] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM '08*, pages 263–272, 2008.

[6] Z. Huang, W. Chung, and H. Chen. A graph model for e-commerce recommender systems. *J. Am. Soc. Inf. Sci. Technol.*, 55(3):259–274, 2004.

[7] M. Jamali and M. Ester. Trustwalker: a random walk model for combining trust-based and item-based recommendation. In *KDD '09*, pages 397–406, 2009.

[8] R. Jäschke, L. Marinho, A. Hotho, L. Schmidt-Thieme, and G. Stumme. Tag recommendations in social bookmarking systems. *AI Commun.*, 21(4):231–247, 2008.

[9] G. Karypis. Evaluation of item-based top-n recommendation algorithms. In *CIKM '01*, pages 247–254, 2001.

[10] Y. Koren. Collaborative filtering with temporal dynamics. In *KDD '09*, pages 447–456, 2009.

[11] N. Lathia, S. Hailes, and L. Capra. Temporal collaborative filtering with adaptive neighbourhoods. In *SIGIR '09*, pages 796–797, 2009.

[12] M. Li, B. M. Dias, I. Jarman, W. El-Deredy, and P. J. Lisboa. Grocery shopping recommendations based on basket-sensitive random walk. In *KDD '09*, pages 1215–1224, 2009.

[13] Y. Li and J. Tang. Expertise search in a time-varying social network. In *WAIM '08*, pages 293–300, 2008.

[14] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[15] N. N. Liu and Q. Yang. Eigenrank: a ranking-oriented approach to collaborative filtering. In *SIGIR '08*, pages 83–90, 2008.

[16] D. Oard and J. Kim. Implicit feedback for recommender systems. In *AAAI Workshop on Recommender Systems, 1998*.

[17] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, 1998.

[18] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *CSCW '94*, pages 175–186, 1994.

[19] F. Ricci and Q. N. Nguyen. Acquiring and revising preferences in a critique-based mobile recommender system. *IEEE Intelligent Systems*, 22(3):22–29, 2007.

[20] J. Schlimmer and R. Granger. Beyond incremental processing: Tracking concept drift. *Proc. 5th National Conference on Artificial Intelligence*, pages 502–507, 1986.

[21] J. Sun, C. Faloutsos, S. Papadimitriou, and P. S. Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *KDD '07*, pages 687–696, 2007.

[22] J. Sun, D. Tao, and C. Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *KDD '06*, pages 374–383, 2006.

[23] A. Töscher and M. Jahrer. The bigchaos solution to the netflix prize 2008. Technical report, 2008.

[24] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Mach. Learn.*, 23(1):69–101, 1996.

[25] A. Zimdars, D. M. Chickering, and C. Meek. Using temporal data for making recommendations. In *UAI '01*, pages 580–588, 2001.