Information Sciences 381 (2017) 161–175

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Version-sensitive mobile App recommendation

Da Cao[a], Liqiang Nie[b], Xiangnan He[c], Xiaochi Wei[d], Jialie Shen[e], Shunxiang Wu[a,*], Tat-Seng Chua[c]

[a] *Department of Automation, Xiamen University, Xiamen, Fujian 361005, PR China*
[b] *School of Computer Science and Technology, Shandong University, Jinan, Shandong 250101, PR China*
[c] *School of Computing, National University of Singapore, Singapore 117417, Singapore*
[d] *School of Computer Science, Beijing Institute of Technology, Beijing 100081, PR China*
[e] *School of Information Systems, Singapore Management University, Singapore 178902, Singapore*

A R T I C L E   I N F O

Article history:
Received 2 April 2016
Revised 31 October 2016
Accepted 26 November 2016
Available online 28 November 2016

Keywords:
Mobile App recommendation
Version progression
Data sparsity problem
Cold-start problem
Plug-in component
Online environment

A B S T R A C T

Being part and parcel of the daily life for billions of people all over the globe, the domain of mobile Applications (Apps) is the fastest growing sector of mobile market today. Users, however, are frequently overwhelmed by the vast number of released Apps and frequently updated versions. Towards this end, we propose a novel version-sensitive mobile App recommendation framework. It is able to recommend appropriate Apps to right users by jointly exploring the version progression and dual-heterogeneous data. It is helpful for alleviating the data sparsity problem caused by version division. As a byproduct, it can be utilized to solve the in-matrix and out-of-matrix cold-start problems. Considering the progression of versions within the same categories, the performance of our proposed framework can be further improved. It is worth emphasizing that our proposed version progression modeling can work as a plug-in component to be embedded into most of the existing latent factor-based algorithms. To support the online learning, we design an incremental update strategy for the framework to adapt the dynamic data in real-time. Extensive experiments on a real-world dataset have demonstrated the promising performance of our proposed approach with both offline and online protocols. Relevant data, code, and parameter settings are available at http://apprec.wixsite.com/version.

1. Introduction

Over the past decade, the proliferation of mobile devices has transformed us into an App-driven society. Mobile consumers are hence spending more time on surfing their Apps than ever before. This presents marketers with new opportunities to connect with consumers by creating more interesting and sophisticated Apps to command their attention. However, the sheer number of Apps[1] with frequently updated versions, and their diverse functions (e.g., games, sports, and shopping), make it difficult for consumers to locate their preferred Apps, which is called information overload problem. Thereby, the ability to recommend suitable Apps to right consumers becomes an urgent task.

Recommender systems [5] have been deployed to alleviate the information overload problem and help users identify relevant information on the Internet. They suggest items of interest to target users by matching the target user's interests

* Corresponding author.
E-mail addresses: dcao@stu.xmu.edu.cn (D. Cao), nieliqiang@gmail.com (L. Nie), xiangnan@comp.nus.edu.sg (X. He), wxchi@bit.edu.cn (X. Wei), jlshen@smu.edu.sg (J. Shen), sxwu@xmu.edu.cn (S. Wu), dcscts@nus.edu.sg (T.-S. Chua).

[1] As Apple reported, there had been over 1.4 million Apps available in iPhone' App store. Retrieved on Jan 8, 2015, from http://tinyurl.com/mxs8hkf.

http://dx.doi.org/10.1016/j.ins.2016.11.025
0020-0255/© 2016 Elsevier Inc. All rights reserved.

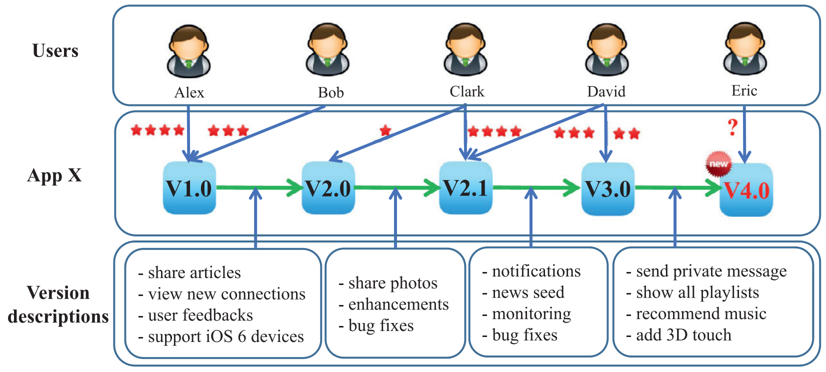


Fig. 1. The visualization of five challenges in App recommendation.

with items' content (content-based filtering) [41,44], mining other similar users' preferences (collaborative filtering) [34,48], or integrating both (hybrid filtering) [22,42]. However, the design of recommendation methods for Apps is significantly different from developing recommender systems in traditional domains (e.g., movie, music, and book), since Apps change and evolve along with version updates, which is indicated by their version numbers and version descriptions. If a recommender system suggests Apps without considering their version property, it may end up with unsuitable Apps and adversely hurt users' experience. To provide a high quality App recommendation service for users, it is inevitable to take the factor of version into account.

Despite its value and significance, recommending mobile Apps remains in its infancy due to the following challenges as revealed in Fig. 1: 1) Unlike conventional static items considered in recommendation approaches, mobile Apps evolve with every revision, indicated by an increment in its version number, which may involve substantial changes. Thus, an App that was unappealing in the past may become favorable after a version update. For instance, an improvement to Version 2.0 of App X, which was unfavorable, with new photo sharing function into Version 2.1 may re-arouse consumers' interests. However, how to capture and model the version progression is a non-trivial task. 2) The data on mobile Apps usually exhibit dual-heterogeneities: digital ratings on each version of Apps by consumers, and textual descriptions of versions by App developers. The heterogeneous data is the key evidence for understanding the rationale for App recommendation. How to effectively uncover the information embedded in the hybrid data and seamlessly sew them up remain largely unaddressed research problem. 3) Version division makes the data sparsity problem more serious. The simplest way to distinguish the difference among versions for each App is to regard each version as an App. However, this will lead to serious data sparsity problem because ratings previously belong to an App now belong to a specific version. For instance, App X has 5 versions, the sparsity of user-version ratings matrix is much serious than that of user-App ratings matrix. How to correlate the version series to overcome the data sparsity problem caused by version division is of great interests. 4) The cold-start problem [36] is much heavier in App's version modeling, since Apps often release new versions and the new versions lack enough ratings. For instance, when the latest version of App X is first released, it has zero rating even though its previous versions have many ratings. Thus how to leverage rating influence of previous version of a new version is of great interests. 5) The ability for online learning is important for mobile App recommendation, where new users, Apps, versions and ratings continuously stream in. For example, with the increasing number of ratings on the latest version of App X, the performance of recommendation could be gradually improved. Therefore, how to devise an online manner to refresh recommendations for users is a valuable research issue. In summary, to better serve users with a quality recommendation service, it is highly desirable to develop an unified framework that comprehensively considers the version progression, heterogeneous data structure, data sparsity problem, cold-start issue, and online learning.

To tackle these problems, we devise a mobile App recommendation framework, which is able to alleviate the App overload problem. It factorizes the historical ratings to discover the latent features underlying the interactions between consumers and each version of an App. High correspondence between versions and consumer factors leads to a recommending action. Besides, it models the version progression, by transforming and vectorizing topic distributions on textual descriptions of the App updates, to strengthen the recommending precision. These two parts are co-regularized and mutually reinforced. The data sparsity problem caused by version division could be alleviated by considering the associations among versions. Furthermore, the cold-start problem on newly released versions of existing Apps can be well solved by utilizing historical rating records, new version descriptions, and evolution patterns. To meet the demand of online learning, we further develop an incremental update strategy to update model parameters given new data. By conducting experiments on our constructed real-world dataset, we demonstrate that our proposed approach is able to yield significant gains as compared with several state-of-the-art competitors.

The main contributions of this work can be summarized as follows.

- To the best of our knowledge, our version-sensitive App recommendation is the first work that considers items' dynamic evolution process not only in App recommendation, but also in recommender systems. We propose a framework, which correlates version series in App recommendation.

- We use evolution progress modeling to correlate digital and textual data. The gap between topic distributions on version descriptions and latent factors on ratings is bridged by using a transfer matrix.
- The familiar data sparsity and cold-start problems of traditional recommender systems are alleviated in the domain of App recommendation. Our idea can be treated as a plug-in component to strengthen most of existing latent factor methods. Meanwhile, the performance of our framework can be further improved by considering the influence of categories.
- We devise an incremental update strategy to support real-time online learning, which is particularly helpful for new users and items.

The remainder of this article is organized as follows. In Section 2, we briefly introduce the related work on content-based filtering, collaborative filtering, and App recommendation. Section 3 formally introduces our proposed framework with both offline and online protocols. The corresponding optimization algorithms and rating prediction method are also illustrated in detail. In Section 4, we conduct experiments on a real-world dataset. The experiments include overall performance comparison, data sparsity problem, cold-start issue, plug-in property, and online learning. Finally, Section 5 concludes and discusses a few directions as future work.

2. Related work

Traditional recommender systems are typically classified into three categories: content-based filtering (CBF), collaborative filtering (CF), and hybrid solutions. To begin with, we give a brief review of the first two as well as their implementation techniques. In the second, we discuss some hybrid methods that attempt to combine ratings and textual content. At last, we introduce related works on App recommendation.

2.1. CBF and CF recommendation

CBF recommendation suggests items which are similar to those a given user preferred in the past. It usually relies on the content of items, such as item descriptions and manually extracted features, to analyze items' similarity or users' preference. A variety of learning algorithms have been adapted to CBF, such as nearest neighbor method [31], naive Bayes classifier [11], relevance feedback [1], and decision rule classifiers [35]. In the domain of App recommendation, digital ratings are crucial evidences for understanding users' preferences and items' features, but they are ignored in CBF techniques.

CF recommendation, on the other hand, handles the recommendation problem by analyzing the pattern of user-item pair, which is often accompanied with an integer rating. As one of the most representative realizations of latent factor models, matrix factorization (MF) [21] characterizes both items and users by factor vectors inferred from item rating patterns. Based on MF, some works like probabilistic matrix factorization (PMF) [32], singular value decomposition (SVD) [17], SVD++ model [19], and timeSVD++ model [20] have been proposed by considering probabilistic distribution, singular value decomposition, implicit information, and temporal dynamics along with time period. However, CF techniques use the past rating information and do not take the content of items into consideration. In the domain of App recommendation, rich content of Apps (e.g., App descriptions, version descriptions, and reviews) is easy to obtain and useful for interpreting both user and item features. Different from previous works, we utilized digital ratings and textual App descriptions simultaneously to improve App recommendation performance.

2.2. Semantic enhanced recommendation

There are several works that attempt to improve recommendation quality with digital ratings and textual content simultaneously, referred to as semantic enhanced recommendation. For Apps, the textual content can be App descriptions, version descriptions and reviews. As representative text processing techniques, topic models, such as probabilistic latent semantic indexing (PLSI) [15], probability latent semantic analysis (PLSA) [16], latent Dirichlet allocation (LDA) [4], and non-negative matrix factorization (NMF) [24], have been widely-accepted to interpret low-dimensional representations of documents. The approach of collaborative topic modeling (CTR) [38] integrates the merits of traditional collaborative filtering with probabilistic topic modeling to recommend scientific articles. The method of hidden factors as topics (HFT) [30] combines ratings with review texts for product recommendation, which works by aligning hidden factors in product ratings with hidden topics in product reviews. The model of TopicMF [2] recommends products by jointly considering user ratings and unstructured reviews. As an improvement of HFT, the technique of RMR (ratings meet reviews) [26] combines content-based filtering with collaborative filtering seamlessly, which exploits the information in both ratings and reviews. Most previous efforts [12,13,33] enhance recommendation performance by considering semantic information in review texts or item descriptions. However, using textual content to build the relationships among items is rarely considered in semantic enhanced recommendation. In this paper, to model the change and evolve process of Apps, we utilized a topic model to learn semantic information of version descriptions, which contain rich information of relationships among versions.

2.3. App recommendation

In fact, a wide range of recommendation approaches have been proposed and applied to recommend mobile Apps, with well theoretical underpinnings and great practical success. Some of these work focus on constructing context-aware recom-

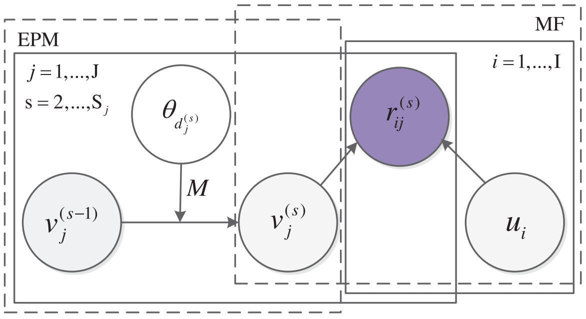


Fig. 2. The graphical representation for VER.

mender systems by collecting additional information (e.g., location, time, and activity) from the mobile device [18,29,43,45]. Some in-deep studies have been given on personalized recommendation [6,10,23,28] in the domain of App recommendation. The similarity of Apps have been discussed using kernel function [8,9] or graph [3]. On the other hand, some specific features of Apps have been utilized to improve App recommendation results. As special properties, ranking [39] and popularity [46] approaches are developed for general App recommendation. The traditional problems in recommender systems like data sparsity [37] and cold-start [25] have been well studied using some specific features of Apps (e.g., neighborhood among Apps and official twitter accounts). Mobile App recommender systems with privacy and security awareness [27,47] have been developed to avoid privacy invasion and other security concern. The work in [40] regarded App recommendation as a result of the contest between satisfaction and temptation. In this paper, we focused on developing version-sensitive App recommendation by modeling version evolution process, which is different from the preceding App recommendation tasks.

3. The proposed framework

Suppose we have I users, J Apps, and S_j versions for the jth App. Each observed data point is a five-tuple $(i, j, s, r_{ij}^{(s)}, d_j^{(s)})$, which can be interpreted as the sth version of the jth App being rated by the ith user with the rating value $r_{ij}^{(s)}$ and being described by the version description $d_j^{(s)}$. It is worthwhile to mention that a user has at most one rating for an App, which also means that the user has at most one rating for a version. (If a user has given a rating on an App, he/she cannot give a second rating on the same App, but he/she can revise the rating and the original rating would be removed.) The rating prediction task is to predict missing ratings from the observed data.

To predict missing ratings, we propose a version evolution recommendation (VER) model that tries to combine MF for latent factors representation and evolution progress modeling (EPM) for version series association. On one hand, MF is an effective way to generate user factor $\mathbf{u}_i$ and item factor $\mathbf{v}_j^{(s)}$, which are independent of each other. On the other hand, the techniques of topic model and transfer matrix are applied in EPM. Topic model is used to uncover topic distributions $\boldsymbol{\theta}$ on version description $d_j^{(s)}$. The gap between topic distributions and latent factors of items can be bridged by transforming the space of topic distributions to the space of latent factors with a transfer matrix $\mathbf{M}$. Fig. 2 shows the graphical representation of the model.

3.1. Matrix factorization for recommendation

Matrix factorization models map both users and items to a joint K-dimensional latent factor space, such that user-item interactions are estimated as the inner products in that space. Accordingly, the i-th user is represented by a latent factor $\mathbf{u}_i \in \mathbb{R}^K$, and the sth version of the jth App is represented by a latent factor $\mathbf{v}_j^{(s)} \in \mathbb{R}^K$. The predicted rating on the sth version of the jth App by the ith user is generated by the inner product of the corresponding user latent factor and item latent factor,

$$\hat{r}_{ij}^{(s)} = \mathbf{u}_i^T \mathbf{v}_j^{(s)}. \tag{1}$$

Given the known ratings $(r_{ij}^{(s)})$, we use matrix $\mathbf{U}$ and $\mathbf{V}$ to represent the latent factors of all users and all App versions respectively. Then $\mathbf{U}$ and $\mathbf{V}$ can be learned by minimizing the regularized square error loss, denoted as $\mathcal{L}_{MF}$,

$$\min_{\mathbf{U},\mathbf{V}} \frac{1}{2} \sum_{(i,j,s)\in\mathcal{R}} \left(r_{ij}^{(s)} - \mathbf{u}_i^T \mathbf{v}_j^{(s)}\right)^2 + \frac{\lambda_u}{2} \sum_{i=1}^{I} \|\mathbf{u}_i\|_F^2 + \frac{\lambda_v}{2} \sum_{j=1}^{J} \sum_{s=1}^{S_j} \left\|\mathbf{v}_j^{(s)}\right\|_F^2, \tag{2}$$

where $\mathcal{R}$ is the set of (i, j, s) triples; $r_{ij}^{(s)}$ is known in the training set; λ_u and λ_v are regularization parameters.

3.2. Evolution progress modeling

In our EPM model, we train a topic model on the version descriptions, and obtain the topic distributions $\boldsymbol{\theta}_{d_j^{(s)}} \in \mathbb{R}^T$ for $d_j^{(s)}$, where T is the number of topics. The relationships between two successive versions are presented as $tail = head + relation$ [7]. Since topic distributions and latent factors of items are in different semantic spaces, we propose to employ a transfer matrix $\mathbf{M} \in \mathbb{R}^{K \times T}$, where K is the dimensionality of latent factors in MF, to bridge the gap between the space of topic distributions and the space of latent factor. It can be formulated as

$$\mathbf{v}_j^{(s)} = \mathbf{v}_j^{(s-1)} + \mathbf{M}\boldsymbol{\theta}_{d_j^{(s)}}. \tag{3}$$

In the light of this, we can estimate the transfer matrix $\mathbf{M}$ by minimizing $\mathcal{L}_{EPM}$,

$$\min_{\mathbf{M}} \frac{1}{2} \sum_{j=1}^{J} \sum_{s=2}^{S_j} \left\| \mathbf{v}_j^{(s)} - \left(\mathbf{v}_j^{(s-1)} + \mathbf{M}\boldsymbol{\theta}_{d_j^{(s)}} \right) \right\|_F^2 + \frac{\lambda_m}{2} \|\mathbf{M}\|_F^2, \tag{4}$$

where λ_m is a regularization parameter.

In this paper, we utilize the most popular topic modeling method, LDA, to learn the topic distributions. It is a generative probabilistic model of a corpus and has been demonstrated effective in previous efforts. There are two important corpus-level parameters in LDA: 1) α is the prior of topic distributions of documents; and 2) β is the prior of word distributions of topics. The results of LDA are two distribution sets, namely, the topic distributions on each document ($\boldsymbol{\theta}$) and the word distributions on each topic ($\boldsymbol{\phi}$). We treat each version description as a document, and the generated $\boldsymbol{\theta}$ is what we want.

3.3. Version evolution recommendation

In order to model App evolution progress and improve rating prediction performance, our VER model will optimize the parameters associated with MF (i.e., **U** and **V**) and the parameter associated with EPM (i.e., **M**) simultaneously. We thus combine MF in Eq. (2) and EPM in Eq. (4) using parameter λ_e, and represented as $\mathcal{L}_{VER} = \mathcal{L}_{MF} + \lambda_e \mathcal{L}_{EPM}$. The minimizing process is conducted as follows:

$$\begin{aligned} \min_{\mathbf{U},\mathbf{V},\mathbf{M}} \frac{1}{2} \sum_{(i,j,s)\in\mathcal{R}} \left(r_{ij}^{(s)} - \mathbf{u}_i^T \mathbf{v}_j^{(s)} \right)^2 + \frac{\lambda_e}{2} \sum_{j=1}^{J} \sum_{s=2}^{S_j} \left\| \mathbf{v}_j^{(s)} - \left(\mathbf{v}_j^{(s-1)} + \mathbf{M}\boldsymbol{\theta}_{d_j^{(s)}} \right) \right\|_F^2 \\ + \frac{\lambda_u}{2} \sum_{i=1}^{I} \|\mathbf{u}_i\|_F^2 + \frac{\lambda_v}{2} \sum_{j=1}^{J} \sum_{s=1}^{S_j} \left\| \mathbf{v}_j^{(s)} \right\|_F^2 + \frac{\lambda_m}{2} \|\mathbf{M}\|_F^2. \end{aligned} \tag{5}$$

Overall, our proposed model has four parameters as shown in Eq. (5). Parameter λ_e balances the performance of MF and EPM, while the remaining three parameters are the regularization parameters to control the complexity and avoid overfitting. We will detail the parameters tuning procedure in the experiments.

3.3.1. Optimization

To minimize the objective function, the standard solution is applying gradient descent on the model parameters [21]. Since the prediction model is in a linear form, another solution is coordinate descent [14], also named as alternating least squares (ALS). The ALS solution is difficult to obtain since it requires an exact optimization solution for each parameter in each update. Thus, we resort to the stochastic gradient descent (SGD) algorithm to optimize **U, V** and **M**. In particular, we optimize one variable while fixing the others in each iteration. Algorithm 1 summarizes the detailed procedure.

Algorithm 1: SGD optimization.

```
Input: Ratings $r_{ij}^{(s)}$, topic distributions $\boldsymbol{\theta}_{d_j^{(s)}}$, and learning rate $\eta$
Output: $\{\mathbf{U}, \mathbf{V}, \mathbf{M}\}$
1: Initialize $\{\mathbf{U}, \mathbf{V}, \mathbf{M}\}$
2: while not converge do
3:   for $r_{ij}^{(s)} \in \mathcal{R}$ do
4:     Update U according to $\mathbf{u}_i \leftarrow \mathbf{u}_i - \eta \nabla_{\mathbf{u}_i} \mathcal{L}_{VER}$
5:     Update V according to $\mathbf{v}_j^{(s)} \leftarrow \mathbf{v}_j^{(s)} - \eta \nabla_{\mathbf{v}_j^{(s)}} \mathcal{L}_{VER}$
6:     Update M according to $\mathbf{M} \leftarrow \mathbf{M} - \eta \nabla_{\mathbf{M}} \mathcal{L}_{VER}$
7:   end for
8: end while
```

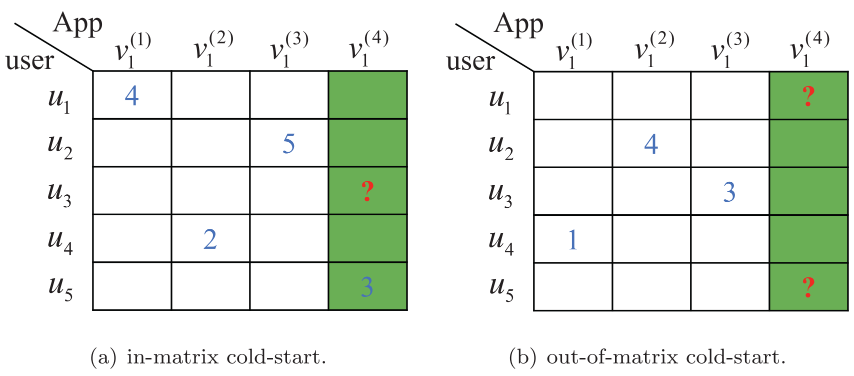


Fig. 3. Illustration of "in-matrix" and "out-of-matrix" cold-start problems. The question mark "?" stands for the desired ratings. The green color represents the latest version. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

We first fix **M** and **V**, and take derivative of our objective function with respect to $\mathbf{u}_i$. The gradient to $\mathbf{u}_i$ is calculated as

$$\nabla_{\mathbf{u}_i}\mathcal{L}_{VER} = \left(r_{ij}^{(s)} - \mathbf{u}_i^T\mathbf{v}_j^{(s)}\right)\left(-\mathbf{v}_j^{(s)}\right) + \lambda_u\mathbf{u}_i. \tag{6}$$

We then fix **M** and **U** to optimize **V**. It is slightly distinguished from the optimization of **U**. Because the first version of an App has no priors, we do not need to consider the influence of previous versions. However, regarding the subsequent versions after the first version of an App, we need to take into account the influence of version evolution, which is characterized in Eq. (3). The gradient to $\mathbf{v}_j^{(s)}$ is calculated as

$$\nabla_{\mathbf{v}_j^{(s)}}\mathcal{L}_{VER} = \begin{cases} \left(r_{ij}^{(s)} - \mathbf{u}_i^T\mathbf{v}_j^{(s)}\right)(-\mathbf{u}_i) + \lambda_v\mathbf{v}_j^{(s)}, \text{if } s = 1; \\ \left(r_{ij}^{(s)} - \mathbf{u}_i^T\mathbf{v}_j^{(s)}\right)(-\mathbf{u}_i) \\ + \lambda_e\left(\mathbf{v}_j^{(s)} - \left(\mathbf{v}_j^{s-1} + \mathbf{M}\boldsymbol{\theta}_{d_j^{(s)}}\right)\right) + \lambda_v\mathbf{v}_j^{(s)}, \text{if } s > 1. \end{cases} \tag{7}$$

Following that, we optimize **M** by fixing **U** and **V**. The gradient to **M** is calculated as

$$\nabla_{\mathbf{M}}\mathcal{L}_{VER} = -\lambda_e\left(\mathbf{v}_j^{(s)} - \mathbf{v}_j^{(s-1)} - \mathbf{M}\boldsymbol{\theta}_{d_j^{(s)}}\right)\boldsymbol{\theta}_{d_j^{(s)}}{}^T + \lambda_m\mathbf{M}, \text{ifs} > 1. \tag{8}$$

Since a constant learning rate may result in fluctuations in later iterations (close to the local minimum), we apply a decay strategy to update the learning rate. Specifically, the learning rate is initially set to 0.01, and we punish it by a ratio of 0.5 when the loss function increases. We alternatively update **U, V** and **M** until convergence.

3.3.2. Cold-start problems

Once we obtain the optimal solution of our model, we can solve the cold-start problems, including in-matrix and out-of-matrix cold-start problems.

Fig. 3(a) illustrates a case of in-matrix cold-start problem. In this case, each version of the given App has at least one rating. Such problem can be well-addressed by the traditional latent factor methods. In particular, we can approximate the rating by using Eq. (1).

Fig. 3(b) explains the situation in out-of-matrix cold-start problem, where newly released versions are not yet rated. Traditional latent factor methods cannot make prediction in this scenario as the latent factor of new App version is not available. We can simulate the latent factor of the new version based on the assumption of EPM (Eq. (3)). The rating prediction for the new version is stated as

$$r_{ij}^{(s)} = \mathbf{u}_i^T\left(\mathbf{v}_j^{(s-1)} + \mathbf{M}\boldsymbol{\theta}_{d_j^{(s)}}\right). \tag{9}$$

3.4. Category-aware extension

In addition to digital ratings and textual version descriptions, categories are crucial evidences for developing an App recommendation system. Intuitively, Apps of a same category (e.g., with the same functionality) compete with each other, which can directly impact users' decisions. Therefore, we model categories, which are predefined by App developers, to enhance recommendation performance. Fig. 4 illustrates the predefined categories in Apple's iTunes App Store and an App can be assigned to multiple categories (e.g., the App "LinkedIn"[2] is assigned to the categories of "Social Networking" and

[2] https://itunes.apple.com/us/app/linkedin/id288429040?mt=8.

Overall:
Books, Business, Catalogs, Education, Entertainment, Finance, Food & Drink, Health & Fitness, Lifestyle, Medical, Music, Navigation, News, Photo & Video, Productivity, Reference, Shopping, Social Networking, Sports, Stickers, Travel, Utilities, Weather

Games:
Action, Adventure, Arcade, Board, Card, Casino, Dice, Educational, Family, Music, Puzzle, Racing, Role Playing, Simulation, Sports, Strategy, Trivia, Word

Magazines & Newspapers:
Arts & Photography, Automotive, Brides & Weddings, Business & Investing, Children's Magazines, Computers & Internet, Cooking, Food & Drink, Crafts & Hobbies, Electronics & Audio, Entertainment, Fashion & Style, Health, Mind & Body, History, Home & Garden, Literary Magazines & Journals, Men's Interest, Movies & Music, News & Politics, Outdoors & Nature, Parenting & Family, Pets, Professional & Trade, Regional News, Science, Sports & Leisure, Teens, Travel & Regional, Women's Interest

Fig. 4. The 70 predefined categories on Apple' iTunes App Store. They are gathered into three groups — Overall, Games, and Magazines & Newspapers.

"Business"). As each version is a unique status of an App, all versions of a specific App belong to the same categories. The rating of a version of an App is not only related to the App and its versions, but also some other Apps in the same category and their versions. Therefore, we regard the mixture of categories as implicit feedbacks and incorporated them into collaborative filtering method as is utilized in [19,27]. Each data point is restructured as $\left(i, j, s, r_{ij}^{(s)}, d_j^{(s)}, C(j)\right)$, which is an increment of the original data point by adding a new element $C(j)$. $C(j)$ represents the set of categories the jth App belongs to. The loss function defined in Eq. (5) is restructured as

$$\min_{\mathbf{U},\mathbf{V},\mathbf{Y},\mathbf{M}} \frac{1}{2} \sum_{(i,j,s)\in\mathcal{R}} \left(r_{ij}^{(s)} - \mathbf{u}_i^T \left(\mathbf{v}_j^{(s)} + |C(j)|^{-\frac{1}{2}} \sum_{c\in C(j)} \mathbf{y}_c \right) \right)^2$$
$$+ \frac{\lambda_e}{2} \sum_{j=1}^{J} \sum_{s=2}^{S_j} \left\| \mathbf{v}_j^{(s)} - \left(\mathbf{v}_j^{(s-1)} + \mathbf{M}\boldsymbol{\theta}_{d_j^{(s)}} \right) \right\|_F^2 + \frac{\lambda_u}{2} \sum_{i=1}^{I} \|\mathbf{u}_i\|_F^2$$
$$+ \frac{\lambda_v}{2} \sum_{j=1}^{J} \sum_{s=1}^{S_j} \left\| \mathbf{v}_j^{(s)} \right\|_F^2 + \frac{\lambda_y}{2} \sum_{c=1}^{C} \|\mathbf{y}_c\|_F^2 + \frac{\lambda_m}{2} \|\mathbf{M}\|_F^2, \tag{10}$$

where **U, V**, and **M** have the same meanings as presented in Eq. (5); $\mathbf{y}_c \in \mathbb{R}^K$ denotes the latent factor for the cth category; **Y** represents the latent factors of all categories; and λ_y is the regularization parameter. The latent factor $\mathbf{v}_j^{(s)}$ is complemented by the sum $|C(j)|^{-\frac{1}{2}} \sum_{c\in C(j)} \mathbf{y}_c$, which means that the version is influenced by the mixture of various categories. For convenience, we use VER-C (short for "VER includes categories") to represent this extensional model. The gradients to $\mathbf{u}_i$, $\mathbf{v}_j^{(s)}$, and **M** are almost the same as presented in Section 3.3.1, which just need to replace $\mathbf{v}_j^{(s)}$ with $\mathbf{v}_j^{(s)} + |C(j)|^{-\frac{1}{2}} \sum_{c\in C(j)} \mathbf{y}_c$. In addition, the gradient to $\mathbf{y}_c$ is $\nabla_{\mathbf{y}_c}\mathcal{L}_{VER-C}$, and it is calculated as

$$\left(r_{ij}^{(s)} - \mathbf{u}_i^T \left(\mathbf{v}_j^{(s)} + |C(j)|^{-\frac{1}{2}} \sum_{c\in C(j)} \mathbf{y}_c \right) \right) (-\mathbf{u}_i)|C(j)|^{-\frac{1}{2}} + \lambda_y \mathbf{y}_c. \tag{11}$$

3.5. Online updates

VER assumes that user ratings are static, which makes it unsuitable for real-world scenarios. To capture the information embedded in a newly received rating and its corresponding version description, the model has to be retrained for all existing data. Obviously, the training process is time-consuming, and it is better to adjust the model in an online manner. Therefore, we propose an online update which assumes that an existing factorization (i.e., **U, V**, and **M**) is given (offline learning) and then a new rating with its corresponding version description comes in. In the following, we present our online algorithm for VER to adjust the model to new data.

Let $\hat{\mathbf{U}}$, $\hat{\mathbf{V}}$, and $\hat{\mathbf{M}}$ denote the model parameters learnt from offline learning, $r_{ij}^{(s)}$ denotes the new rating streamed in, accompanied with its version description $d_j^{(s)}$. Since the online update progress is performed on the new rating, we only conduct optimization operations for $\mathbf{u}_i$, $\mathbf{v}_j^{(s)}$, and **M**. The underlying assumption is that, given a new rating $r_{ij}^{(s)}$, the features for $\mathbf{u}_i$, $\mathbf{v}_j^{(s)}$, and **M** are influenced by the rating significantly, while other variables in $\hat{\mathbf{U}}$ and $\hat{\mathbf{V}}$ still keep stable. Algorithm 2 sum-

Algorithm 2: Online Updates for VER.

```
Input: $\hat{\mathbf{U}}, \hat{\mathbf{V}}, \hat{\mathbf{M}}$, new ratings $r_{ij}^{(s)}$, topic descriptions $d_j^{(s)}$, and learning rate $\eta$
Output: $\{\mathbf{U}, \mathbf{V}, \mathbf{M}\}$
 1: Initialize $\mathbf{U} \leftarrow \hat{\mathbf{U}}$, $\mathbf{V} \leftarrow \hat{\mathbf{V}}$, and $\mathbf{M} \leftarrow \hat{\mathbf{M}}$
 2: Initialize $\mathbf{u}_i$ randomly if $i$ is a new user
 3: Initialize $\mathbf{v}_\mathbf{j}^{(\mathbf{s})}$ randomly if $j$ is a new App
 4: Initialize $\mathbf{v}_\mathbf{j}^{(\mathbf{s})} \leftarrow \mathbf{v}_j^{(s-1)} + \mathbf{M}\boldsymbol{\theta}_{d_j^{(s)}}$ if $s$ is a new version for an existing App $j$
 5: while not converge do
 6:   for $r_{ij}^{(s)} \in \mathcal{R}$ do
 7:     Update $\mathbf{U}$ according to $\mathbf{u}_i \leftarrow \mathbf{u}_i - \eta \nabla_{\mathbf{u}_i} \mathcal{L}_{VER}$
 8:     Update $\mathbf{V}$ according to$\mathbf{v}_j^{(s)} \leftarrow \mathbf{v}_j^{(s)} - \eta \nabla_{\mathbf{v}_j^{(s)}} \mathcal{L}_{VER}$
 9:     Update $\mathbf{M}$ according to$\mathbf{M} \leftarrow \mathbf{M} - \eta \nabla_{\mathbf{M}} \mathcal{L}_{VER}$
10:   end for
11: end while
```

marizes the incremental learning strategy for VER. Regarding the deviation functions of $\nabla_{\mathbf{u}_i}\mathcal{L}_{VER}$, $\nabla_{\mathbf{v}_j^{(s)}}\mathcal{L}_{VER}$, and $\nabla_{\mathbf{M}}\mathcal{L}_{VER}$, their computing methods were illustrated in Eq. (6), Eq. (7), and Eq. (8). The learning rate decay strategy we have mentioned in Section 3.3.1 is also utilized. Through this way, the algorithm can steadily converge to a optimal solution.

3.6. Time complexity and memory cost

In order to analyze the complexity of VER in Algorithm 1, we need to solve the time complexity in terms of constructing **U, V**, and **M** in Eq. (6), Eq. (7), and Eq. (8). The computational complexity of the training process is $O(N \times (O_1 + O_2 + O_3))$, where O_1, O_2, and O_3 are equal to $|\mathcal{R}|K$, $|\mathcal{R}_{s=1}|K + |\mathcal{R}_{s>1}|KT$, and $|\mathcal{R}_{s>1}|KT$ respectively. N is the iteration times of the optimization process, $\mathcal{R}_{s=1} = \{(i, j, s)|(i, j, s) \in \mathcal{R}, s = 1\}$, and $\mathcal{R}_{s>1} = \{(i, j, s)|(i, j, s) \in \mathcal{R}, s > 1\}$. $|\mathcal{R}_{s=1}|$, $|\mathcal{R}_{s>1}|$, K and T respectively refer to the number of ratings in $\mathcal{R}_{s=1}$, the number of ratings in $\mathcal{R}_{s>1}$, the dimensionality of latent factors, and the number of topics in topic model. The complexity computing method of online updates for VER in Algorithm 2 is almost the same as in Algorithm 1, the main difference is the scale of training data, the computing process is omitted here.

The memory cost for VER includes the user matrix **U**, version matrix **V**, transfer matrix **M**, and gradients of $\nabla_{\mathbf{u}_i}\mathcal{L}_{VER}$, $\nabla_{\mathbf{v}_j^s}\mathcal{L}_{VER}$, $\nabla_{\mathbf{M}}\mathcal{L}_{VER}$. The total memory cost is $O((I + J_{s\geq 1} + T)K)$, where I, $J_{s\,\geq\,1}$, T, and K respectively refer to the number of users, the number of versions, the number of topics in topic model, and the dimensionality of latent factors. The similar computing process of memory cost for VER in Algorithm 2 is omitted here.

4. Experiments

In this section, we conduct extensive experiments on our dataset and aim to answer the following five research questions:

- **RQ1**. How do our designed VER and VER-C approaches perform as compared with other state-of-the-art recommendation algorithms?
- **RQ2**. The version division leads to serious data sparsity problem. How is the performance of VER in alleviating this problem?
- **RQ3**. How is the effectiveness of VER in handling the out-of-matrix cold-start problem (new version cold-start problem)?
- **RQ4**. The key component of VER is the version progression. Can it be applied to other latent factor-based techniques?
- **RQ5**. Is VER suitable for online environment? How is VER's stability in online learning?

4.1. Experimental settings

4.1.1. Dataset construction

We constructed a benchmark dataset based upon Apple's iTunes App Store[3], which contains both ratings and version descriptions. We further processed the dataset by retaining Apps with at least 10 ratings and 5 versions, and users who rated at least 10 Apps in the past. Based upon these criteria, we ultimately obtained 47,440 users, 8,403 Apps, 89,456 versions, 55 categories, and 898,213 ratings. The user-App ratings matrix has a sparsity of 99.77%, while the user-version ratings matrix has a sparsity of 99.98%. Detailed statistics of the dataset are provided in Table 1.

[3] https://itunes.apple.com/us/genre/ios/id36?mt=8.

Table 1
Statistics of the dataset.

	Number	Min.#ratings	Max.#ratings	Avg.#ratings
User	47,440	10	195	18.93
App	8,403	10	5,359	106.89
Version	89,456	1	4,026	10.04

In our experiment, we used three different methods to generate training data and testing data for different testing scenarios. 1) In the overall performance comparison, data sparsity problem handling, and plug-in property study, the original dataset was divided into two disjoint sets, with 80% randomly selected ratings used for training and the remaining 20% for testing. We used 5-fold cross validation. 2) For the cold-start problem handling, we first removed the ratings on latest versions of all Apps. We then used the ratings of earlier versions for training, and the latest version with zero rating for testing. As a result, the training data occupies 92.31% of the whole dataset, while the percentage for testing data is only 7.69%. We conducted the experiment for 5 times. 3) In the experiment of online updates, we used the training data in the cold-start problem handling for offline learning. Meanwhile, we split the testing data in the cold-start problem to online training data and online testing data with different proportions (i.e., 0%/100%, 10%/90%, ... , 90%/10%) to illustrate the effectiveness of incremental updates.

What we need to mention here is that, in all of these experiments, the training data was further divided into training set and validation set by a ratio of 8 to 2, the parameters were carefully tuned on the validation set.

4.1.2. Evaluation metrics

To measure the performance of different methods from multiple aspects, we employed three commonly used metrics. First of all, we adopted the widely used metric, mean absolute error (MAE), which compute the average of the absolute difference between the predictions and true ratings. It is calculated as

$$MAE = \frac{1}{|\mathcal{T}|} \sum_{(i,j,s)\in\mathcal{T}} |r_{ij}^{(s)} - \hat{r}_{ij}^{(s)}|, \tag{12}$$

where $\mathcal{T}$ is the test set, $|\mathcal{T}|$ is the number of ratings in $\mathcal{T}$, $\hat{r}_{ij}^{(s)}$ is the predicted rating of the ith user on the sth version of the jth item, and $r_{ij}^{(s)}$ is the ground truth. Meanwhile, as a popular metric used in movie recommendation [20], the root mean squared error (RMSE) amplifies the contributions of the absolute errors between the predictions and the true values. It is defined as

$$RMSE = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(i,j,s)\in\mathcal{T}} \left(r_{ij}^{(s)} - \hat{r}_{ij}^{(s)}\right)^2}. \tag{13}$$

Besides, we also utilized the mean signed difference (MSD) as our another metric. It is useful for detecting prediction bias and it is expressed as

$$MSD = \frac{1}{|\mathcal{T}|} \sum_{(i,j,s)\in\mathcal{T}} \left(r_{ij}^{(s)} - \hat{r}_{ij}^{(s)}\right). \tag{14}$$

We can see that smaller MAE, RMSE, and MSD values normally indicate better performance for rating predictions. MAE assigns equal weight to the data whereas RMSE emphasizes the extremes. This is why RMSE will always be larger or equal to MAE. Meanwhile, MAE focuses on absolute difference whereas MSD considers signed difference. Hence, the difference can be measured from different angles.

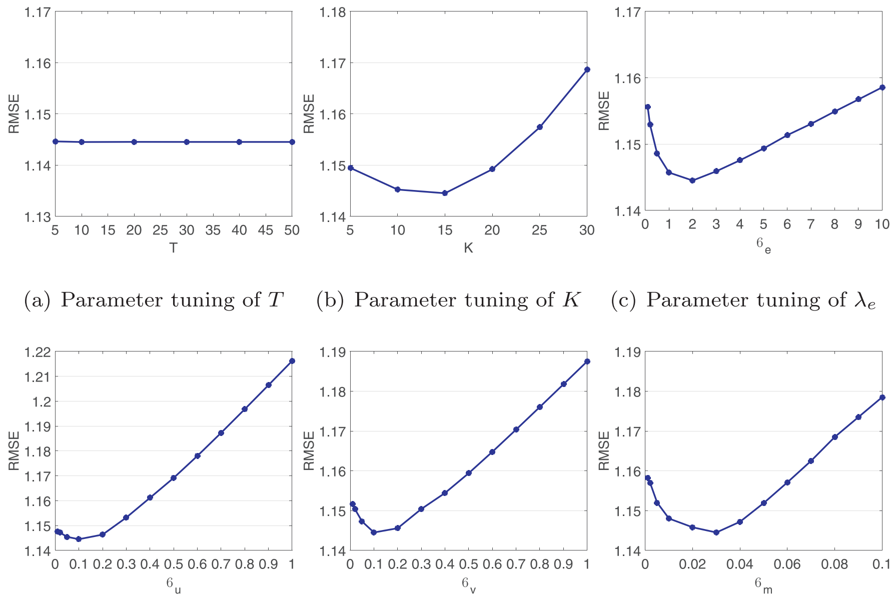
4.1.3. Hardware and software platform

All the experiments were conducted over a server equipped with Intel(R) Core(TM) i7-4790 CPU at 3.60 GHz on 32G RAM, 8 cores and 64-bit Windows 10 operation system. VER is implemented in Python programming language using NumPy[4]. The training time for our VER model in the overall performance comparison is 41 minutes and 29 seconds, and the peak memory cost is 752.9M.

4.2. Parameter tuning and convergence analysis

There are six important parameters in our VER model: 1) the number of topics T in topic modeling; 2) the dimensionality of latent factors K in matrix factorization; and 3) the parameter λ_e, λ_u, λ_v, and λ_m that balance the terms in our proposed model. Evaluation results were computed on the validation set. Grid search with a small but adaptive step size

[4] NumPy: http://www.numpy.org/.



(a) Parameter tuning of T (b) Parameter tuning of K (c) Parameter tuning of λ_e

(d) Parameter tuning of λ_u (e) Parameter tuning of λ_v (f) Parameter tuning of λ_m

Fig. 5. Parameter tuning in terms of RMSE.

was conducted to locate the optimal parameter settings. Here we revealed the tuning results in the overall performance comparison, and observed that when $T = 10$, $K = 15$, $\lambda_e = 2$, $\lambda_u = 0.1$, $\lambda_v = 0.1$, and $\lambda_m = 0.03$, our model achieved the best performance regarding RMSE. We then investigated the sensitivity of our VER to these parameters by varying one and fixing the others. We also conducted experiments to study the convergence with the increasing number of iterations.

We first fixed $K = 15$, $\lambda_e = 2$, $\lambda_u = 0.1$, $\lambda_v = 0.1$, $\lambda_m = 0.03$, and varied T. As shown in Fig. 5(a), VER is relatively stable and not sensitive to T. We utilized a transfer matrix to bridge the gap between the space of topic distributions and the space of latent factors. Even though the number of topics varies, no topic information is lost and the topics can be finely mapped to the latent factor space by using the transfer matrix.

We then fixed $T = 10$, $\lambda_e = 2$, $\lambda_u = 0.1$, $\lambda_v = 0.1$, $\lambda_m = 0.03$, and varied K. As shown in Fig. 5(b), RMSE decreases first and then increases along the increasing of K. It reaches its minimum when $K = 15$. Our finding is different from traditional latent factor methods, which are inclined to use more factors [21]. This is mainly because latent factors are also influenced by topic distributions on version descriptions in our model.

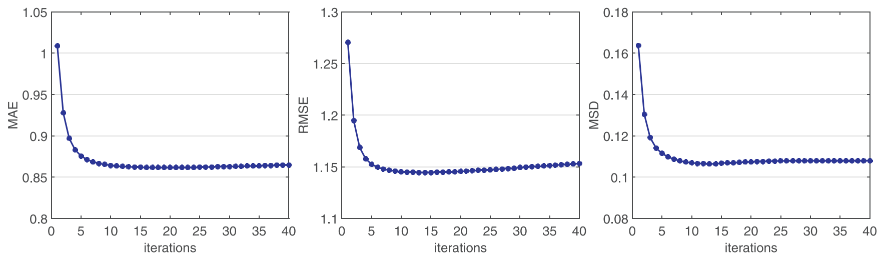
We next fixed $T = 10$, $K = 15$, $\lambda_u = 0.1$, $\lambda_v = 0.1$, $\lambda_m = 0.03$, and varied parameter λ_e. The results are shown in Fig. 5(c). As can been seen, RMSE decreases first and then increases, and reaches its minimum when $\lambda_e = 2$. λ_e balances the influence of MF and EPM. With the increasing of λ_e, our model puts more emphasis on EPM. However, over emphasis on EPM is unreasonable. This is why the value of RMSE increases after $\lambda_e = 2$. We chose $\lambda_e = 2$ as our default value.

Parameters λ_u, λ_v, and λ_m are used to avoid overfitting. Using the similar method mentioned above to adjust these parameter, we can see from Fig. 5(d), Fig. 5(e), and Fig. 5(f) that the value of RMSE reaches its minimum when $\lambda_u = 0.1$, $\lambda_v = 0.1$, and $\lambda_m = 0.03$, respectively.

At last, we recorded the value of MAE and RMSE along with each iteration using the settings of $T = 10$, $K = 15$, $\lambda_e = 2$, $\lambda_u = 0.1$, $\lambda_v = 0.1$, and $\lambda_m = 0.03$. Fig. 6(a), (b), and (c) shows the convergence study with increasing number of iterations. We can see that our model converges within less than 20 iterations. This demonstrates its efficiency.

4.3. Overall performance comparison (RQ1)

To demonstrate the overall effectiveness of our proposed VER and VER-C models, we compared them with several state-of-the-art recommendation approaches: (i) LDA [4] (content-based filtering). This method first used LDA on the textual content (i.e., version description) to obtain the item representation, and then applied the item nearest neighbor method [31] to generate recommendation. Cosine similarity was used to measure the distance. (ii) MF [21] (collaborative filtering method). This baseline only used digital ratings as data source. We regarded each version of an App as an item. (iii) MF-A. We also implemented a MF baseline, which treats each App as an item regardless of versions, denoted as MF-A. (iv) CTR [38] (semantic enhanced method). It utilized both digital ratings and textual content. The topic distributions are generated from version descriptions. What we need to mention here is that some related methods are not suitable for comparison. 1) App recommendation related algorithms. As far as we know, there are no other works that try to model the version evolution



(a) Convergence of MAE (b) Convergence of RMSE (c) Convergence of MSD

Fig. 6. Convergence analysis in terms of MAE, RMSE, and MSD.

Table 2
Performance comparison of various methods.

Methods	MAE	p-value	RMSE	p-value	MSD	p-value
LDA	0.9458 ± 0.003	$2.79e-12$	1.5982 ± 0.004	$9.53e-11$	0.0347 ± 0.002	$5.68e-12$
MF	0.9268 ± 0.002	$7.62e-12$	1.1756 ± 0.003	$1.07e-07$	0.3091 ± 0.003	$8.89e-14$
MF-A	0.8961 ± 0.004	$9.03e-11$	1.1601 ± 0.004	$3.42e-06$	0.2170 ± 0.002	$1.00e-12$
CTR	0.8982 ± 0.003	$7.19e-11$	1.1624 ± 0.003	$5.07e-06$	0.2402 ± 0.003	$4.68e-13$
VER	**0.8602 ± 0.003**	–	**1.1445 ± 0.003**	–	**0.1064 ± 0.002**	–
VER-C	**0.8471 ± 0.003**	–	**1.1380 ± 0.002**	–	**0.0431 ± 0.003**	–

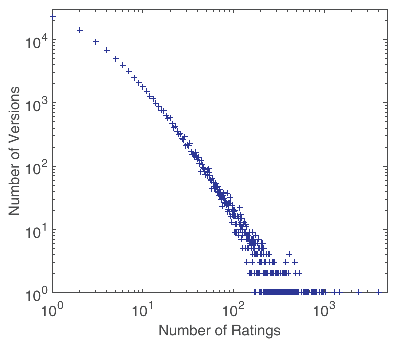
progress to recommend Apps to be accurate to version. At the same time, the data used here only includes users' ratings and versions' textual descriptions. No other App recommendation algorithm utilizes these material only. 2) Context-aware recommender systems. Each App has its own version division, and Apps do not share any common versions. So version cannot be regarded as context. 3) Time-sensitive recommender systems. According to our statistics, we do not have any findings about time. That is why the time-sensitive methods are ignored here.

For each method mentioned above, the involved parameters are carefully tuned on the validation set, and the best performance of various methods is used to report the final comparison results. Table 2 summarizes the results. We have the following observations: 1) Our VER model achieves the MAE of 0.8602, and the RMSE of 1.1445, which shows improvements over LDA, MF, MF-A, CTR of 11.84%, 7.19%, 4.01%, 4.24% in MAE, and 28.39%, 2.65%, 1.35%, 1.55% in RMSE respectively. Meanwhile, VER obtains the MSD of 0.1064, which gains improvements over MF, MF-A, CTR of 65.58%, 50.97%, 55.70%. It is worth noting that LDA outperforms VER in MSD. It is mainly because LDA utilizes item nearest neighbor method to generate recommendation which is average computing to some extent, while VER ignores the universal bias, item bias, and user bias which are crucial to avoid prediction bias. In the following experiment of plug-in portability (Section 4.6), we incorporate biases into MF, the performance of VER in MSD is significantly improved. All the p-values of the significance test (paired two-sample t-test) between our model and each of the baselines are much smaller than 0.05, which indicates that the improvements are also statistically significant. This is mainly because VER not only considers the ratings to be accurate to version, but also associates the version series. 2) The experimental results of MF-A are superior to MF in MAE, RMSE, and MSD. Since App has 10.6 versions on average in our dataset. The sparsity of MF is 10.6 times as that of MF-A, which causes the worse performance of MF. Considering the version progression, VER overcomes the problem caused by sparsity and improves the performance significantly. 3) LDA and CTR do not perform well compared with VER. This is because the version descriptions always depict version progression and they are insufficient to depict the features of App version. On the other hand, VER makes full use of the textual corpus to model the version progression to improve recommendation quality. 4) VER-C outperforms VER with relative improvements of 1.52%, 0.57% and 59.49% in terms of MAE, RMSE and MSD, respectively. This admits the significance of modeling categories for mobile App recommendation. Users' preferences on an App are not only related to the App itself, but also other similar Apps of the same category. VER-C considers the influence of category mixture to correlate the Apps of the same categories, thus yields notable improvements.

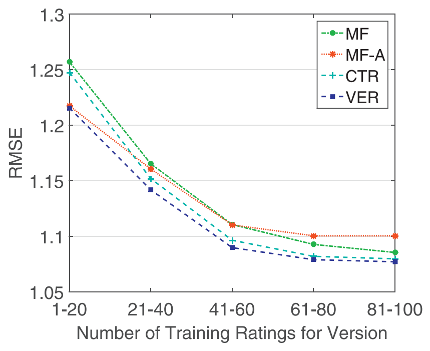
4.4. Capacity of alleviating data sparsity problem (RQ2)

According to the statistics in Table 1, the sparsity of user-version ratings matrix is much serious than that of user-App ratings matrix (Each version has 10.04 ratings on average, while it is 106.89 for App). Furthermore, as we can see from Fig. 7(a), versions show a power-law distribution — most versions only have very few ratings, and only a small proportion of versions have many ratings. In order to illustrate the capacity of VER in alleviating the data sparsity problem, we compared it with the baselines used in Section 4.3, namely, (i) MF, (ii) MF-A, and (iii) CTR (LDA is removed here, because it does not consider ratings and does not have the data sparsity problem).

We used the results obtained in Section 4.3, and further disposed the results by selecting versions whose number of training ratings are located in specific ranges (i.e., 1 – 20, ... , 81 – 100). The performance of various methods is shown in



(a) Data distribution



(b) Comparison in terms of RMSE

Fig. 7. Version distribution with respect to the number of ratings and the performance of various methods in handling data sparsity problem.

Table 3
Performance comparison of various methods in handling the cold-start problem.

Methods	MAE	*p*-value	RMSE	*p*-value	MSD	*p*-value
LDA	0.9465 ± 0.001	2.36*e* – 11	1.6067 ± 0.001	3.85*e* – 11	0.0507 ± 0.002	3.42*e* – 06
MF-A	0.9361 ± 0.001	5.98*e* – 11	1.2177 ± 0.002	1.25*e* – 05	0.2203 ± 0.003	1.03*e* – 09
CTR	0.9729 ± 0.001	4.36*e* – 12	1.3194 ± 0.002	1.70*e* – 08	0.0687 ± 0.002	4.11*e* – 07
VER	**0.8963 ± 0.003**	–	**1.1953 ± 0.002**	–	0.0250 ± 0.001	–

Fig. 7(b). We have the following observations: 1) VER consistently outperforms MF, MF-A and CTR, and the improvement over MF and CTR gradually decreases with the growth of the number of training ratings for version. VER considers the version progression, which has a great effect on alleviating the data sparsity problem when the version has only a few ratings. With the increasing of the number of training ratings for version, the latent factors of versions can be better learnt and the performances of MF, CTR, and VER are gradually getting closer to each other. 2) The performance of MF-A is better than that of MF when the version has a few training ratings (less than 40), while MF outperforms MF-A when the version has sufficient training ratings (more than 61). MF-A regards the App as an item, and does not consider the differences among versions. Even though the version does not have enough ratings, ratings on other versions of the App can alleviate the sparsity problem, that is why MF-A performs well when the version has a few training ratings. However, MF performs well when enough ratings are accumulated on the version compared with MF-A. When enough ratings are obtained for the version, the latent factor of version could be well learnt for MF.

4.5. Robustness of handling cold-start problem (RQ3)

As to the capability to handle cold-start problem, we only verified the effectiveness of our model on out-of-matrix cold-start problem, since the in-matrix one is trivial and can be addressed by most existing methods. MF cannot make prediction for out-of-matrix items (new versions) as it heavily relies on existing ratings of items. We thus coped with the cold-start problem in comparison with the following methods: (i) LDA. As a content-based method, LDA makes prediction based on the version descriptions regardless of the ratings. (ii) MF-A. MF-A makes prediction on Apps without considering version, so it can make prediction on the new versions of existing Apps. (iii) CTR. CTR addresses the out-of-matrix problem by using the dot product of user vector and topic distributions on version descriptions. While VER depends on latent factor of previous version, topic distributions on version descriptions and transfer matrix as shown in Eq. (9).

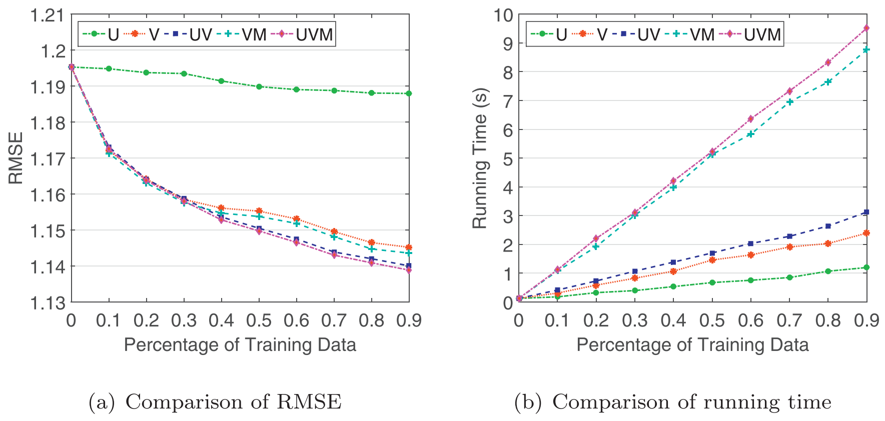
Results are displayed in Table 3. From this table, we observed that VER achieves the MAE of 0.8963, the RMSE of 1.1953, and the MSD of 0.0250, which gains improvements over LDA, MF-A, CTR at 5.30%, 4.25%, 7.87% in MAE, 25.60%, 1.84%, 9.41% in RMSE, and 50.69%, 88.65%, 63.60% in MSD respectively. The paired two-sample *t*-test results also support the conclusion. It is noted that MF-A ignores the specific features of latest versions, LDA and CTR ignore the association among version series; on the other hand, VER considers the relations among version series by utilizing textual corpus. This is why VER performs the best.

4.6. Plug-in portability (RQ4)

Latent factor methods represent users and items with latent vectors. The ratings are the inner products of corresponding user latent vector and item latent vector. As a complement, our EPM model associates version series and can be treated as a plug-in component to be embedded into traditional recommendation approaches, especially latent factor methods, including: (i) MF. It is illustrated in Section 4.3, so we will not expand here. (ii) MF with biases. As reported in [21],

Table 4
Performance comparison between latent factor methods and their enhanced version with our proposed EPM.

Methods	MAE	*p*-value	RMSE	*p*-value	MSD	*p*-value
MF	0.9268 ± 0.002	7.62*e* – 12	1.1756 ± 0.003	1.07*e* – 07	0.3091 ± 0.003	8.89*e* – 14
MF + EPM	**0.8602** ± **0.003**	–	**1.1445** ± **0.003**	–	**0.1064** ± **0.002**	–
BMF	0.8667 ± 0.003	9.76*e* – 10	1.1424 ± 0.003	2.62*e* – 06	0.0076 ± 0.001	1.14*e* – 05
BMF + EPM	**0.8469** ± **0.003**	–	**1.1273** ± **0.003**	–	**0.0057** ± **0.001**	–
SVD++	0.8506 ± 0.003	2.73*e* – 09	1.1379 ± 0.003	5.61*e* – 07	0.0030 ± 0.001	1.45*e* – 04
SVD++ + EPM	**0.8353** ± **0.003**	–	**1.1215** ± **0.003**	–	**0.0020** ± **0.001**	–



(a) Comparison of RMSE (b) Comparison of running time

Fig. 8. Performance comparison of incremental updates for online environment with different proportions of training data and different update protocols.

integrating with global average, user bias, and item bias, the performance of MF could be improved a lot. Inspired by this, we incorporated biases into MF, and denoted it as BMF. (iii) SVD++. SVD++ achieved further accuracy improvements by incorporating implicit feedbacks, which are rated items in our experiments. In our experiment, SVD++ is also enhanced with biases. We incorporated our EPM model to these three latent factor-based recommendation methods and referred to them as MF + EPM, BMF + EPM, and SVD++ + EPM, respectively.

The settings and experimental results of MF and MF + EPM (VER) are the same as that stated in Section 4.3. Regarding the rest of methods in Table 4, we also employed grid search to find their optimal parameter settings on the validation set. The comparison results are summarized in Table 4. The results show that by incorporating our EPM model, the performance of MF, BMF, and SVD++ have improved by 7.19%, 2.27%, 1.80% in MAE, 2.65%, 1.32%, 1.44% in RMSE, and 65.58%, 25.00%, 33.33% in MSD respectively. The paired two-sample *t*-test results are also very significant. The experimental results indicate that, as a plug-in component, our EPM model can further improve most of the existing latent factor methods. This again demonstrates the importance of version progression in App recommendation.

4.7. Incremental updates for online environment (RQ5)

According to Algorithm 2, the online updates for VER are conducted on user matrix **U**, item matrix **V**, and transfer matrix **M**. In order to distinguish the influence of different parameters, we conducted the experiments on different combinations of parameters: (i) U. We used U to represent the online update which only updates the latent factors of users. If the user is a new one, we first randomly initialized the latent factor for the user and then updated it. If the user has rating records in the offline learning, the latent factor of the user is updated based on the optimal latent factor learned in the offline process. (ii) V. V is used to represent the online update which only considers the impact of item. If the item is the first version of a newly released App, we randomly initialized the latent factor for the item. If the item is a new version of an existing App, we initialized the latent factor by using the combination of the latent factor of previous version and the topic distributions of current version' textual description, which is illustrated in Eq. (3). (iii) UV. UV represents the online update that takes into account the effects of both user and item. (iv) VM. Transfer matrix is used to correlate the latent factor of item and the topic distributions of version description, which also has a direct influence on the correlation of version series. So we used VM to represent the online update for both item and transfer matrix. (v). UVM. Similar to VM, UVM is used to represent the online update for user, item, and transfer matrix.

The offline learning is carefully tuned using the same parameter settings in overall performance comparison. In order to illustrate the effectiveness and efficiency of online updates for VER, we recorded RMSE of the final result and running time of one iteration with different ratio of training data. Fig. 8(a) summarizes the results in terms of RMSE. We have the following observations: 1) V notably and consistently outperforms U. The latent vectors of new versions could be significantly improved due to lacking of enough historical ratings, while the improvement of users' latent vectors is not significant because they have been carefully learnt from historical rating records. It is reasonable to see that UV is superior to both U and V. 2) VM does not yield obvious improvement over V, which is also the same to the pair of UVM and UV. Transfer matrix bridges the gap between latent factor space and topic distribution space, which also correlates the version series and influences the performance of V. The improvement is not obvious enough since transfer matrix has been carefully learnt in the offline learning. 3) As we can see from V, UV, VM, UVM, the performance is dramatically improved when small proportion of training data is accumulated, which illustrates the remarkable effect of the online learning on newly appeared items. Fig. 8(b) presents the results in terms of running time of one iteration. We can see that: 1) The running time of VM is dramatically longer than that of V, which is also the same to the pair of UVM and UV. This indicates that the update progress for transfer matrix is time-consuming compared with other parameters. 2) The running time of V is longer than that of U. This is mainly because the update progress for item is more complex compared with that of user.

5. Conclusion and future work

In this paper, we presented a novel version-sensitive mobile App recommendation framework, which is, as far as we know, the first work that tries to model the dynamic changes of an item not only in App recommendation, but also in recommender systems. It is able to boost the rating prediction accuracy by jointly considering the version progression, textual descriptions and historical digital ratings. The data sparsity problem caused by version division can be alleviated, which is especially beneficial to versions which have only a few ratings. As a byproduct, the in-matrix and out-of-matrix cold-start problems could be solved. Accounting for the influence of category mixture, the performance of our proposed framework can be further improved. Furthermore, the version progress modeling can be viewed as a plug-in component to conveniently enhance traditional latent factor-based recommendation algorithms. For online environment, we devised an incremental update strategy for the framework to adapt dynamic data in real-time. To validate the effectiveness of our proposed approach, we constructed a benchmark dataset. Extensive experiments on this real-world dataset have demonstrated its efficacy, efficient convergence, robustness, portability, and expandability. Meanwhile, we have released our self-collected dataset and our implementation codes to convenient the research community.

In the future, we plan to expand our work in the following three aspects: 1) Modeling the dynamics of consumers' preferences, since consumers' preferences are often shifting over time. 2) Training topic distributions and latent factors in an unified framework. The performance of MF and EPM can be mutually enforced. 3) Modeling the dynamic of topics associated with the progression of versions. The newly appeared topics in new versions could be obtained and impact on ratings.

Acknowledgments

The authors are highly grateful to the anonymous referees for their careful reading and insightful comments. The work is supported by the NUS-Tsing Extreme Search (NExT) project (R-252-300-001-490) and the National Natural Science Foundation of China under Grant No. 61673327. NExT research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its IRC@SG Funding Initiative.

References

[1] J.-w. Ahn, P. Brusilovsky, J. Grady, D. He, S.Y. Syn, Open user profiles for adaptive news systems: help or harm? in: Proceedings of the 16th International Conference on World Wide Web, ACM, 2007, pp. 11–20.
[2] Y. Bao, H. Fang, J. Zhang, Topicmf: simultaneously exploiting ratings and reviews for recommendation, in: Proceedings of the 28th AAAI Conference on Artificial Intelligence, AAAI Press, 2014, pp. 2–8.
[3] U. Bhandari, K. Sugiyama, A. Datta, R. Jindal, Serendipitous recommendation for mobile apps using item-item similarity graph, in: Information Retrieval Technology, Springer, 2013, pp. 440–451.
[4] D.M. Blei, A.Y. Ng, M.I. Jordan, Latent dirichlet allocation, J. Mach. Learn. Res. 3 (2003) 993–1022.
[5] J. Bobadilla, F. Ortega, A. Hernando, A. Gutiérrez, Recommender systems survey, Knowl. Based Syst. 46 (2013) 109–132.
[6] M. Böhmer, L. Ganev, A. Krüger, Appfunnel: a framework for usage-centric evaluation of recommender systems that suggest mobile applications, in: Proceedings of the 2013 International Conference on Intelligent User Interfaces, ACM, 2013, pp. 267–276.
[7] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, O. Yakhnenko, Translating embeddings for modeling multi-relational data, in: Proceedings of the 27st Annual Conference on Neural Information Processing Systems, 2013, pp. 2787–2795.
[8] N. Chen, S.C. Hoi, S. Li, X. Xiao, Simapp: a framework for detecting similar mobile applications by online kernel learning, in: Proceedings of the 8th ACM International Conference on Web Search and Data Mining, ACM, 2015, pp. 305–314.
[9] N. Chen, S.C. Hoi, S. Li, X. Xiao, Mobile app tagging, in: Proceedings of the 9th ACM International Conference on Web Search and Data Mining, ACM, 2016, pp. 63–72.
[10] E. Costa-Montenegro, A.B. Barragáns-Martínez, M. Rey-López, Which app? a recommender system of applications in markets: implementation of the service for monitoring users' interaction, Expert Syst. Appl. 39 (10) (2012) 9367–9375.
[11] M. Degemmis, P. Lops, G. Semeraro, A content-collaborative recommender that exploits wordnet-based user profiles for neighborhood formation, User Model User-adapt Interact. 17 (3) (2007) 217–255.
[12] X. He, T. Chen, M.-Y. Kan, X. Chen, Trirank: review-aware explainable recommendation by modeling aspects, in: Proceedings of the 24th ACM International Conference on Information and Knowledge Management, ACM, 2015, pp. 1661–1670.
[13] X. He, M. Gao, M.-Y. Kan, Y. Liu, K. Sugiyama, Predicting the popularity of web 2.0 items based on user comments, in: Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 2014, pp. 233–242.
[14] X. He, H. Zhang, M.-Y. Kan, T.-S. Chua, Fast matrix factorization for online recommendation with implicit feedback, in: Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 2016.
[15] T. Hofmann, Probabilistic latent semantic indexing, in: Proceedings of the 22nd International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 1999, pp. 50–57.

[16] T. Hofmann, Unsupervised learning by probabilistic latent semantic analysis, Mach. Learn. 42 (1–2) (2001) 177–196.
[17] Y. Hu, Y. Koren, C. Volinsky, Collaborative filtering for implicit feedback datasets, in: Proceeding of the 8th IEEE International Conference on Data Mining, IEEE, 2008, pp. 263–272.
[18] A. Karatzoglou, L. Baltrunas, K. Church, M. Böhmer, Climbing the app wall: enabling mobile app discovery through context-aware recommendations, in: Proceedings of the 21st ACM International Conference on Information and Knowledge Management, ACM, 2012, pp. 2527–2530.
[19] Y. Koren, Factorization meets the neighborhood: a multifaceted collaborative filtering model, in: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2008, pp. 426–434.
[20] Y. Koren, Collaborative filtering with temporal dynamics, Commun. ACM 53 (4) (2010) 89–97.
[21] Y. Koren, R. Bell, C. Volinsky, et al., Matrix factorization techniques for recommender systems, Comput. (Long Beach Calif) 42 (8) (2009) 30–37.
[22] C.-H. Lai, D.-R. Liu, C.-S. Lin, Novel personal and group-based trust models in collaborative filtering for document recommendation, Inf. Sci. (Ny) 239 (2013) 31–49.
[23] C. Lin, R. Xie, X. Guan, L. Li, T. Li, Personalized news recommendation via implicit social experts, Inf. Sci. (Ny) 254 (2014) 1–18.
[24] C.-J. Lin, Projected gradient methods for nonnegative matrix factorization, Neural Comput. 19 (10) (2007) 2756–2779.
[25] J. Lin, K. Sugiyama, M.-Y. Kan, T.-S. Chua, Addressing cold-start in app recommendation: latent user models constructed from twitter followers, in: Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 2013, pp. 283–292.
[26] G. Ling, M.R. Lyu, I. King, Ratings meet reviews, a combined approach to recommend, in: Proceedings of the 8th ACM Conference on Recommender Systems, ACM, 2014, pp. 105–112.
[27] B. Liu, D. Kong, L. Cen, N.Z. Gong, H. Jin, H. Xiong, Personalized mobile app recommendation: Reconciling app functionality and user privacy preference, in: Proceedings of the 8th ACM International Conference on Web Search and Data Mining, ACM, 2015, pp. 315–324.
[28] D.-R. Liu, P.-Y. Tsai, P.-H. Chiu, Personalized recommendation of popular blog articles for mobile applications, Inf. Sci. (Ny) 181 (9) (2011) 1552–1572.
[29] Q. Liu, H. Ma, E. Chen, H. Xiong, A survey of context-aware mobile recommendations, Int. J. Inf. Technol. Decis Mak 12 (01) (2013) 139–172.
[30] J. McAuley, J. Leskovec, Hidden factors and hidden topics: understanding rating dimensions with review text, in: Proceedings of the 7th ACM Conference on Recommender Systems, ACM, 2013, pp. 165–172.
[31] S.E. Middleton, N.R. Shadbolt, D.C. De Roure, Ontological user profiling in recommender systems, ACM Trans. Inf. Syst. 22 (1) (2004) 54–88.
[32] A. Mnih, R. Salakhutdinov, Probabilistic matrix factorization, in: Proceedings of the 21st Annual Conference on Neural Information Processing Systems, 2007, pp. 1257–1264.
[33] L. Nie, Y.-L. Zhao, X. Wang, J. Shen, T.-S. Chua, Learning to recommend descriptive tags for questions in social forums, ACM Trans. Inf. Syst. 32 (1) (2014) 5.
[34] W. Pan, S. Xia, Z. Liu, X. Peng, Z. Ming, Mixed factorization for collaborative recommendation with heterogeneous explicit feedbacks, Inf. Sci. (Ny) 332 (2016) 84–93.
[35] M.J. Pazzani, D. Billsus, Content-based Recommendation Systems, in: The adaptive web, Springer, 2007, pp. 325–341.
[36] A.I. Schein, A. Popescul, L.H. Ungar, D.M. Pennock, Methods and metrics for cold-start recommendations, in: Proceedings of the 25th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 2002, pp. 253–260.
[37] K. Shi, K. Ali, Getjar mobile application recommendations with very sparse datasets, in: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2012, pp. 204–212.
[38] C. Wang, D.M. Blei, Collaborative topic modeling for recommending scientific articles, in: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2011, pp. 448–456.
[39] D. Yankov, P. Berkhin, R. Subba, Interoperability ranking for mobile applications, in: Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 2013, pp. 857–860.
[40] P. Yin, P. Luo, W.-C. Lee, M. Wang, App recommendation: a contest between satisfaction and temptation, in: Proceedings of the 6th ACM International Conference on Web Search and Data Mining, ACM, 2013, pp. 395–404.
[41] Z. Yu, C. Wang, J. Bu, X. Wang, Y. Wu, C. Chen, Friend recommendation with content spread enhancement in social networks, Inf. Sci. (Ny) 309 (2015) 102–118.
[42] Z. Zhang, H. Lin, K. Liu, D. Wu, G. Zhang, J. Lu, A hybrid fuzzy-based personalized recommender system for telecom products/services, Inf. Sci. (Ny) 235 (2013) 117–129.
[43] V.W. Zheng, B. Cao, Y. Zheng, X. Xie, Q. Yang, Collaborative filtering meets mobile recommendation: a user-centered approach, in: Proceedings of the 24th AAAI Conference on Artificial Intelligence, AAAI Press, 2010, pp. 236–241.
[44] X. Zhou, S. Wu, C. Chen, G. Chen, S. Ying, Real-time recommendation for microblogs, Inf. Sci. (Ny) 279 (2014) 301–325.
[45] H. Zhu, E. Chen, K. Yu, H. Cao, H. Xiong, J. Tian, Mining personal context-aware preferences for mobile users, in: Proceeding of the 12th IEEE International Conference on Data Mining, IEEE, 2012, pp. 1212–1217.
[46] H. Zhu, C. Liu, Y. Ge, H. Xiong, E. Chen, Popularity modeling for mobile apps: a sequential approach, IEEE Trans. Cybern. 45 (7) (2015) 1303–1314.
[47] H. Zhu, H. Xiong, Y. Ge, E. Chen, Mobile app recommendations with security and privacy awareness, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2014, pp. 951–960.
[48] B. Zou, C. Li, L. Tan, H. Chen, Gputensor: efficient tensor factorization for context-aware recommendations, Inf. Sci. (Ny) 299 (2015) 159–177.