Version-Aware Rating Prediction for Mobile App Recommendation

YUAN YAO, Nanjing University, China
WAYNE XIN ZHAO*, Renmin University of China
YAOJING WANG, Nanjing University, China
HANGHANG TONG, Arizona State University, USA
FENG XU, Nanjing University, China
JIAN LU, Nanjing University, China

With the great popularity of mobile devices, the amount of mobile apps has grown in a dramatic rate than we have ever expected. A technical challenge is how to recommend suitable apps to mobile users. In this work, we identify and focus on a unique characteristic that exists in mobile app recommendation, i.e., an app usually corresponds to multiple release versions. Based on this characteristic, we propose a fine-grained version-aware app recommendation problem. Instead of directly learning the users' preferences over the apps, we aim to infer the ratings of users on a specific version of an app. However, the user-version rating matrix will be sparser than the corresponding user-app rating matrix, making existing recommendation methods less effective. In view of this, our approach has made two major extensions. First, we leverage the review text that is associated with each rating record; more importantly, we consider two types of version-based correlations. The first type is to capture the temporal correlations between multiple versions within the same app, and the second type of correlation is to capture the aggregation correlations between similar apps. Experimental results on a large data set demonstrate the superiority of our approach over several competitive methods.

CCS Concepts: **•Information systems → Data mining;**

General Terms: Algorithm, experiment

Additional Key Words and Phrases: App rating prediction, recommender systems, version correlation

ACM Reference Format:
Yao et al., Version-Aware Rating Prediction for Mobile App Recommendation. *ACM Trans. Inf. Syst.* V, N, Article 00 (November 2016), 30 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

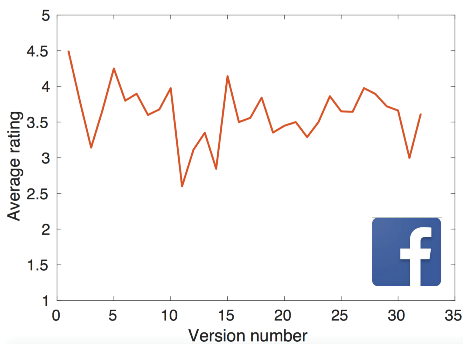
1. INTRODUCTION

In recent years, we have witnessed the great success of mobile devices, such as smartphones and tablet computers. The enormous popularity of mobile devices can be partially credited to *mobile applications* (apps for brevity). For example, apps can help users to buy products, book restaurants, watch movies, and read news simply with a mobile device connected to the Internet. Recently, the amount of mobile apps has grown in a dramatic rate than we have ever expected. For example, on Google Play (formerly known as the Android Market), there were over 1.43 million apps published and over 50 billion downloads[1]. With such a huge number of apps, a direct challenge for the app

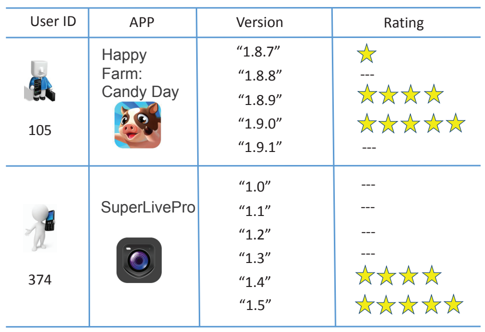
[1]https://en.wikipedia.org/wiki/Google_Play

*Xin Zhao is the corresponding author.
Author's addresses: Y. Yao, State Key Laboratory for Novel Software Technology, Nanjing University, China; email: y.yao@nju.edu.cn; W.X. Zhao, School of Information & Beijing Key Laboratory of Big Data Management and Analysis Methods, Renmin University of China; email: batmanfly@gmail.com; Y. Wang, State Key Laboratory for Novel Software Technology, Nanjing University, China; email: wyj@smail.nju.edu.cn; H. Tong, School of Computing, Informatics and Decision Systems Engineering, Arizona State University, USA; email: hanghang.tong@asu.edu; F. Xu, State Key Laboratory for Novel Software Technology, Nanjing University, China; email: xf@nju.edu.cn; J. Lu, State Key Laboratory for Novel Software Technology, Nanjing University, China; email: lj@nju.edu.cn.

DOI: http://dx.doi.org/10.1145/0000000.0000000



(a) App rating variation.



User ID	APP	Version	Rating
105	Happy Farm: Candy Day	"1.8.7"	★
		"1.8.8"	---
		"1.8.9"	★★★★
		"1.9.0"	★★★★★
		"1.9.1"	---
374	SuperLivePro	"1.0"	---
		"1.1"	---
		"1.2"	---
		"1.3"	---
		"1.4"	★★★★
		"1.5"	★★★★★

(b) User rating variation.

Fig. 1. Rating variation examples of multiple app versions.

market is how to help users find suitable apps. To this end, recommender systems can be used to understand users' preferences and predict what apps a user might be interested in. In recommender systems, rating prediction is a widely studied task, with the goal of predicting the preference degree of a user on a product/service that she has not rated before. In this work, we put our focus on such rating prediction problem for mobile app recommendation.

Various recommendation methods have been proposed for rating prediction in the past years, including collaborative filtering methods, content-based methods, and hybrid methods. Collaborative filtering methods build a model from a user's past behaviors as well as the decisions made by other similar users. Content-based methods extract a set of important features of an item in order to recommend other items with similar features. These two types of methods are often combined in practical systems to form the hybrid methods. Based on the existing recommendation methods, a first thought would be whether traditional rating prediction algorithms can be directly employed on the task of mobile app recommendation. However, several recent studies have revealed that mobile app recommendation has its own characteristics which distinguish itself from traditional recommendation settings. For example, apps could have privileges to access the user's personal information such as locations, contacts, and messages, and thus user privacy has to be considered during the recommendation process [Liu et al. 2015]; the contextual information of mobile users collected by the sensors in mobile devices can also be exploited in the recommendation task [Zhu et al. 2015]. These studies indicate that it might be suboptimal to simply reuse existing techniques for mining mobile app data.

Different from previous studies [Liu et al. 2015; Zhu et al. 2015], we identify and focus on another unique characteristic that exists in mobile app recommendation: an app usually has multiple releases, and each release corresponds to a unique version of the app. Since the first release, the maintenance of an app is a continuous development process. The developers of an app may collect users' feedback and then improve their app in the next release by fixing bugs or adding new features [Villarroel et al. 2016]. On mainstream app markets (e.g., Google Play and App Store), a user review is usually associated with a corresponding version. By collecting these version-related information, we can better analyze the version-level mobile adoption behaviors. In Fig. 1(a), we present the average ratings of multiple versions from a well-known app *Facebook*. As we can see, the app has shown a large rating variation on its different versions. Meanwhile, corresponding to apps' evolving process, a mobile user is likely to assign different ratings to multiple versions of the same app. Fig. 1(b) presents the ratings of two mobile users on multiple versions of two apps (i.e., *Happy Farm* and *SuperLivePro*). We can see that a user does not necessarily rate all versions of an app, and the ratings given by a user to multiple versions of an app can be very different. These two examples indicate that it is necessary to consider version information for app rating prediction.

In this work, we propose to distinguish between multiple versions of an app and treat them as different entities (i.e., *items* in traditional recommendation setting). Our goal is to infer the preference degree (i.e., rating) of a user on the latest version[2] of an app. We call this task as *version-aware app rating prediction*. In traditional recommendation methods, an item is typically considered to be static and it corresponds to a single entity. It would be difficult for these methods to work well in our task. For example, traditional collaborative filtering methods formulate the rating prediction task as inferring the values for the missing entries over a user-item rating matrix. Taking the user-app rating matrix as input, these methods will lose the fine-grained rating behaviors where a user may assign different ratings to different versions of an app. A straightforward solution is to replace the user-app matrix with user-version matrix. Although this solution can solve the above task to some extent, it will aggravate the data sparsity issue, making collaborative filtering methods less effective. Based on a user-version rating matrix, the main focus of this work is to alleviate the data sparsity issue and improve the prediction accuracy for the task of version-aware app rating prediction.

Our approach has made two major extensions to address the above sparsity issue. On one hand, review text associated with each rating is leveraged to enrich the traditional matrix factorization model. We aggregate the review text by both apps and versions and capture two granularities of text semantics. In our model, text semantics are characterized by jointly factorizing app-term and version-term matrices. On the other hand, we consider two types of version-based correlations. The first type is to capture the temporal correlations between multiple versions of the same app, and the second type is to capture the aggregation correlations between similar apps. These two types of correlations are modeled as regularization terms and incorporated into the matrix factorization framework.

In summary, the main contributions of this article include:

— *Problem definition and empirical analysis*. We define the version-aware rating prediction problem for mobile app recommendation. We further construct an empirical study on a real-world mobile data set, showing that a user is likely to rate multiple versions of the same app and that their preferences over different versions may be different.
— *A version-aware model for app recommendation*. We propose a fine-grained model (VAMF) for the version-aware rating prediction problem. The proposed model integrates the ratings, the review text, the temporal correlations between multiple versions of the same app, and the aggregation correlations between similar apps. Moreover, we propose an efficient algorithm to solve the proposed model, and analyze its optimality and computational complexity.
— *Experimental evaluations on a real-world data set*. In order to evaluate the effectiveness of the proposed approach, we compare it with several competitive baselines on a large data set. We construct extensive experiments, and the results demonstrate the superiority of our approach over the compared methods.

To the best of our knowledge, there are seldom studies that address the version-aware app rating prediction task. Nevertheless, version-based information is critical to the prosperity of an app or even the entire app market. For example, with version-based user preferences, the app developers can better understand users' needs and make necessary updates in upcoming versions. We believe that our work will inspire more follow-up studies in the task of mobile app recommendation.

The rest of this article is organized as follows. We first present the some empirical analysis in Section 2. Problem definition and the proposed model are described in Section 3 and Section 4, respectively. Section 5 presents the algorithm to solve the model in the previous section. In Section 6, we conduct extensive experiments and analyze the experimental results. We review the related work in Section 7. Finally, Section 8 concludes the paper and discusses the future work.

[2]Although our method can infer the ratings on any specific version, users are not allowed to download previous versions of apps in most app markets.

Table I. The basic statistics of the Google Play data set.

# of users	170,781
# of apps	16, 457
# of versions	104,061
# of reviews	3,367,435
# of categories	42
# of terms	10,569

2. EMPIRICAL ANALYSIS

In this section, we introduce the data set and perform some empirical analysis.

2.1. Data Set Description

We use the Google Play data set shared in [Chen et al. 2015] as the data collection. For each app, the data set contains the reviews for all its versions and the descriptions of its latest version. Each review is a short text posted by a user on some version of an app[3], and it is explicitly associated with a rating ranging from one to five stars. We further remove the users and apps with fewer than 10 reviews, respectively. We have done some common preprocessing steps on the review text, including tokenization, short-word removal, infrequent-word removal, stopword removal, and stemming. The basic statistics of the resulting data set are summarized in Table I. On average, an app has 6.3 versions.

2.2. Empirical Analysis

Rating Sparsity. Rating sparsity is one of the major technical challenges for collaborative filtering algorithms. When users or items have very few ratings, a recommender system may not be able to generate good recommendation results.

Here, we construct two rating matrices: a user-app matrix and a user-version matrix. In the user-app matrix, an entry will be filled in if a user has rated at least one version for an app; in the user-version matrix, an entry will be filled in if a user has rated the exact version of an app. We compare the rating sparsity of these two matrices. Intuitively, a user may not rate all the versions of an app. Thus, the user-version matrix will be sparser than the user-app matrix. As expected, for the data set in Table I, the density of the user-app matrix is 0.1126%, while the density of the user-version matrix is 0.0189%. This result indicates that rating prediction in the version level will be more difficult than that in the app level.

Distribution of Version-based Ratings. In this part, we continue to present some statistical analysis results on the distribution of version-based app ratings.

We first study how many versions an app is associated with. To see this, we plot the distribution of the number of apps *v.s.* the number of versions in Fig. 2. The y-axis in the figure is of log scale. As we can see, the distribution has a power-law like shape. Overall, we have found that 39.3% apps have more than 5 versions. It is also interesting to observe that there exist a considerable amount of apps with a large number of versions (e.g., more than 20 versions).

Next, we continue to study the multiple-version rating behaviors, i.e., a user has rated multiple versions of the same app. It is particularly important to check whether the multiple-version rating behaviors are prevalent for mobile users. In our data set, we found 359,545 multiple-version ratings. For these 359,545 ratings, we plot the distribution of the number of user-app pairs *v.s.* the number of versions in Fig. 3. Here, each user-app pair indicates that the user has rated multiple versions of the app. Given a number of versions x (i.e., x-axis), we count the number of corresponding user-app pairs (i.e., y-axis) that the user has rated exactly x versions of the app. The y-axis of the figure is in

[3]We have found that 88.1% of the reviews explicitly contain a version, and we remove the rest reviews that do not have versions.

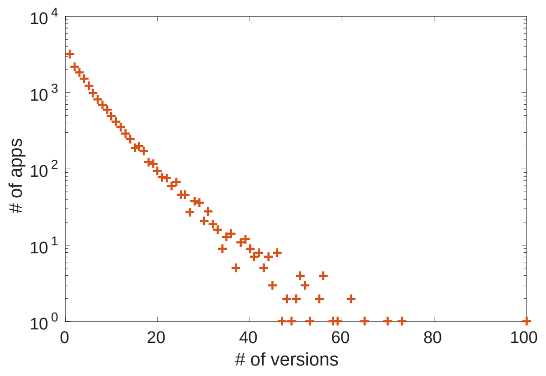


Fig. 2. The number of apps vs. the number of versions.

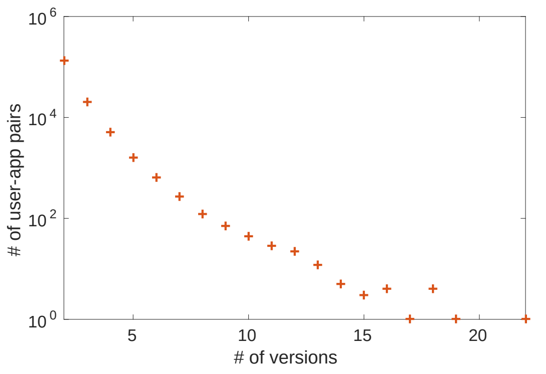


Fig. 3. The number of user-app pairs vs. the number of versions.

the log scale. As seen in Fig. 3, the distribution also shows a power-law like shape, and some user even rated more than 10 versions of an app.

Combing the results in Fig. 2 and Fig. 3, it can be seen that multiple-version rating behavior is indeed popular in our data set.

Version-level Rating Variation. Next, we check whether there are any rating variations between multiple versions of an app on both user side and app side. If the variations are small, it may be unnecessary to make version-level rating prediction.

We quantitatively measure the degree of version-level rating variations, and construct two analysis experiments here. We first focus on the rating behaviors of a user on multiple versions of an app. Given a user u, we propose to measure her Rating Variation (RV) between two different versions of an app in terms of both ratios and absolute values as follows

$$\begin{aligned} RV_{ratio}(u) &= \frac{1}{|\mathcal{A}^{(u)}|}\sum_{a\in\mathcal{A}^{(u)}}\frac{1}{0.5\times N^{(u,a)}\times(N^{(u,a)}-1)}\sum_{<k,k'>\in\mathcal{V}^{(u,a)},k<k'}\frac{|r_{u,k}-r_{u,k'}|}{r_{u,k}}, \\ RV_{value}(u) &= \frac{1}{|\mathcal{A}^{(u)}|}\sum_{a\in\mathcal{A}^{(u)}}\frac{1}{0.5\times N^{(u,a)}\times(N^{(u,a)}-1)}\sum_{<k,k'>\in\mathcal{V}^{(u,a)},k<k'}|r_{u,k}-r_{u,k'}|, \end{aligned} \quad (1)$$

where $\mathcal{A}^{(u)}$ is the set of apps that user u has rated at least two versions, $\mathcal{V}^{(u,a)}$ is the set of versions of app a that has been rated by user u, k and k' are two different version indices from $\mathcal{V}^{(u,a)}$, $r_{u,k}$ is the rating that user u has given to the k-th version of app a, and $N^{(u,a)}$ is size of $\mathcal{V}^{(u,a)}$. Larger values of $RV_{ratio}(u)$ and $RV_{value}(u)$ indicate larger rating variations of user u over multiple versions on an app.

Next, we analyze the average rating variation of an app on its multiple versions. Given an app a, we measure its Rating Variation (RV) between two different versions in terms of both ratios and absolute values as follows

$$\begin{aligned} RV_{ratio}(a) &= \frac{1}{0.5 \times N^{(a)} \times (N^{(a)} - 1)} \sum_{k=1}^{N^{(a)}} \sum_{k'=k+1}^{N^{(a)}} \frac{|\bar{r}_{a,k} - \bar{r}_{a,k'}|}{\bar{r}_{a,k'}}, \\ RV_{value}(a) &= \frac{1}{0.5 \times N^{(a)} \times (N^{(a)} - 1)} \sum_{k=1}^{N^{(a)}} \sum_{k'=k+1}^{N^{(a)}} |\bar{r}_{a,k} - \bar{r}_{a,k'}|, \end{aligned} \tag{2}$$

where $N^{(a)}$ is the number of versions for app a, and $\bar{r}_{u,k}$ is the average rating for the k-th version of app a. Larger values of $RV_{ratio}(a)$ and $RV_{value}(a)$ indicate larger rating variations of app a over its versions. In order to compute $RV_{ratio}(a)$ and $RV_{value}(a)$, we do not consider apps with a single version.

Table II. Statistics of average version-based rating variations for both users and apps. The standard deviations are shown in parentheses.

$RV_{value}(user)$	$RV_{ratio}(user)$	$RV_{value}(app)$	$RV_{ratio}(app)$
0.8098 (1.0391)	0.3636 (0.6814)	0.8241 (0.5223)	0.2772 (0.2551)

We further average the rating variations over all the users and apps, respectively. The standard derivations for these values are also calcuated. We present the results in Table II. As we can see, the multiple-version rating variations are indeed significant for both users and apps. For example, $RV_{value}(user) = 0.8098$ indicates users' average rating difference over different versions. A traditional version-unaware recommendation algorithm may not be able to capture users' interest variation over multiple versions.

3. PROBLEM DEFINITION

In this section, we formally define our task. The notations used throughout the paper are summarized in Table III. Following conventions, we use bold capital letters for matrices. For example, we use a $U \times V$ matrix $\mathbf{R}$ to denote the user-version rating matrix where U and V are the number of users and versions, respectively. Similar to the coding mechanism in Matlab, we denote the i-th row of matrix $\mathbf{R}$ as $\mathbf{R}(i, :)$, and the transpose of a matrix with a prime (i.e., $\mathbf{R}' \equiv \mathbf{R}^\top$).

Before presenting our problem definition, we first introduce some basic concepts. We first distinguish between the two terms of *app* and *version*[4] in the following definition.

Definition 3.1. ***App ID and Version ID.*** Assume that there are totally V versions from A apps in our data collection. We associate each app with a unique app ID, numbered from 1 to A. For versions, we organize the multiple versions of the same app consecutively and they are numbered in an ascending order according to their release dates. Similarly, we associate each version with a unique version ID, numbered from 1 to V. The app IDs and version IDs are allocated separately[5].

Furthermore, to connect app IDs with version IDs, we build an app-version association matrix $\mathbf{M} \in \mathbb{R}^{A \times V}$ as follows

$$m_{a,i} = \begin{cases} 1, & \text{if verison } i \text{ belongs to app } a; \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

[4]For convenience, we use "version" to refer to an app with a specific version number in this article.

[5]Here, app IDs and version IDs are the internal IDs mapped from external ID values. An external ID value is usually a character string generated by an app market. We map them into integers.

Table III. List of symbols and descriptions.

Symbol	Description
$\mathbf{R}$	the user-version rating matrix
$\mathbf{X}$	the version-term matrix
$\mathbf{Y}$	the app-term matrix
$\mathbf{S}$	the app similarity matrix
$\mathbf{P}$	the version temporal correlation matrix
$\mathbf{M}$	the app-version association matrix
$\mathbf{U}, \mathbf{V}$	the latent factors of users and versions for $\mathbf{R}$
$\mathbf{F}_1$	the latent topics of versions for $\mathbf{X}$
$\mathbf{F}_2$	the latent topics of apps for $\mathbf{Y}$
$\mathbf{G}$	the topic-term matrices for both apps and versions
$\mathbf{R}'$	the transpose of matrix $\mathbf{R}$
$\mathbf{R}(u, :)$	the u^{th} row of $\mathbf{R}$
$\mathbf{R}(u, i)$	the element at the u^{th} row and i^{th} column of $\mathbf{R}$
$\mathbf{I}^R$	the indicator matrix for $\mathbf{R}$
$\mathcal{A}$	the set of apps
U, V, A	the number of users, versions, and apps
T	the vocabulary size, or the number of terms
K	the number of latent factors
u	the users
i, j	the versions
l	the maximum iteration number
ξ	the threshold to terminate the iteration

With the $\mathbf{M}$ matrix, we can find the corresponding app ID given a version ID, and we can find the version IDs of a given app ID.

Definition 3.2. ***App-specific Version Number.*** Given app a, let i_a be the version ID for its first release. Assume that app a contains V_a versions. According to our numbering method, the version ID for the latest release of app a will be $i_a + V_a - 1$. Given a version indexed by the version ID i', if it belongs to app a, its corresponding version number for app a is $i' - i_a + 1$. That is to say, version i' corresponds to the $(i' - i_a + 1)$-th version of app a.

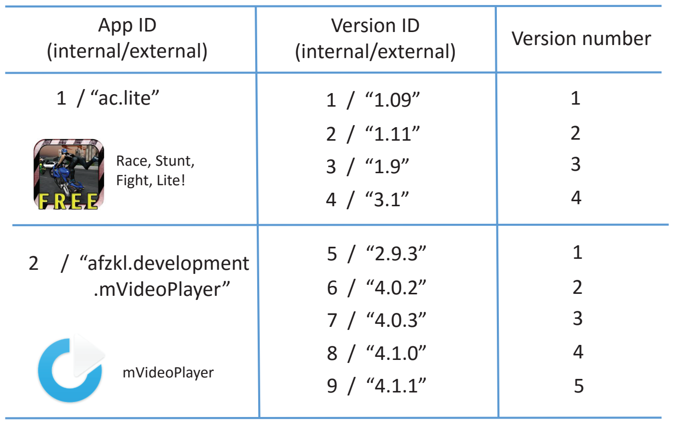
Note that the definitions of app ID, version ID, and version number are mainly for ease of our formulation in the next section. We present an illustrative example to explain the relationship between app ID, version ID, and version number in Fig. 4. We can see that both app IDs and version IDs are mapped from external ID values (e.g., "*ac.lite*"). They are allocated in a global numbering way. The version numbers are used as local ID values w.r.t. to a certain app. In what follows, the usage of i in "version i" refers to the version ID but not a version number unless specified otherwise.

Next, we present the definition for the version-based app rating matrix.

Definition 3.3. ***Version-based App Rating Matrix.*** Assume that there are U users and V versions from A apps in our data collection. The version-based app rating matrix or the user-version rating matrix $\mathbf{R}$ is a $U \times V$ matrix, where the entry $\mathbf{R}(u, i)$ indicates the rating score that user u has assigned to version i. The rating score indicates the preference degree of a user over a version of an app. As aforementioned, the user-version rating matrix is very sparse, and many entries of $\mathbf{R}$ are missing. We pair $\mathbf{R}$ with an indicator matrix $\mathbf{I}^R \in \mathbb{R}^{U \times V}$, where a value of 1 indicates the corresponding entry in $\mathbf{R}$ is not empty.

In the user-version matrix $\mathbf{R}$, each column corresponds to the ratings for a specific version. Again, we organize the columns in the follow way: the multiple versions of the same app are placed consecutively and they are sorted in an ascending order according to their release dates.

Apart from the rating data, in our setting, we assume that users' review text is also available. These textual information can be leveraged to alleviate the sparsity issue of the rating matrix. Here, we aggregate reviews both by versions and by apps. Let T be the number of terms in the vocabulary for our data collection. Following "bag-of-words" assumption, we can represent the review text



App ID (internal/external)	Version ID (internal/external)	Version number
1 / "ac.lite" Race, Stunt, Fight, Lite!	1 / "1.09"	1
	2 / "1.11"	2
	3 / "1.9"	3
	4 / "3.1"	4
2 / "afzkl.development .mVideoPlayer" mVideoPlayer	5 / "2.9.3"	1
	6 / "4.0.2"	2
	7 / "4.0.3"	3
	8 / "4.1.0"	4
	9 / "4.1.1"	5

Fig. 4. An illustrative example for app ID, version ID, and version number.

aggregated in the version level and the app level as two matrices[6]: the version-term matrix $\mathbf{X} \in \mathbb{R}^{V \times T}$ and the app-term matrix $\mathbf{Y} \in \mathbb{R}^{A \times T}$.

Definition 3.4. ***Version-Term Matrix and App-Term Matrix.*** The version-term matrix $\mathbf{X}$ is a $V \times T$ matrix, where each entry $\mathbf{X}(i, w)$ is the weight of the w-th term in the review text related to version i. Similarly, the app-term matrix $\mathbf{Y}$ is an $A \times T$ matrix, where each entry $\mathbf{Y}(a, w)$ is the weight of the w-th term in the review text related to app a.

To set the values for both $\mathbf{X}$ and $\mathbf{Y}$, we tried several term weighting methods borrowed from Information Retrieval, including term frequency, tf-idf, and binary weighting. In our work, we have found that the normalized term frequency is more suitable to the matrix factorization approach, and we adopt it as the term weighting method.

In addition to review text, we assume that the similarities between the apps can be built from the data collection. The app similarity matrix can be constructed using the information such as app categories and app descriptions. Here, app similarities are characterized in matrix $\mathbf{S} \in \mathbb{R}^{A \times A}$.

Definition 3.5. ***App Similarity Matrix.*** The app similarity matrix $\mathbf{S}$ is an $A \times A$ matrix, where each entry $\mathbf{S}(a, a')$ indicates the similarity between app a and app a'.

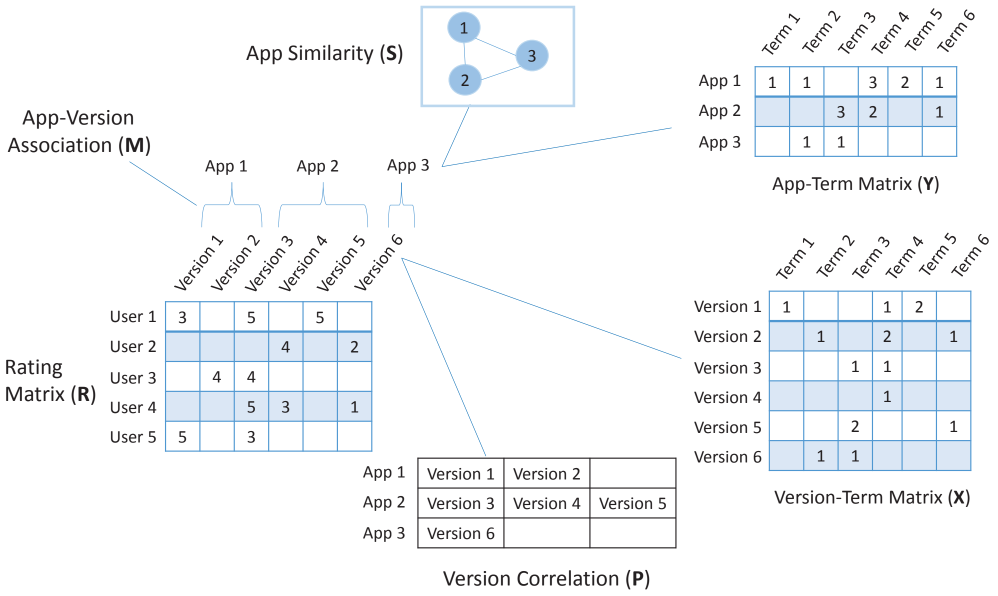
To set the values of $\mathbf{S}$, we still use the normalized term frequency vectors to represent each app, and then compute the cosine similarities between these vectors. Here, to keep the computation efficiently, we only compute the similarities of apps that belong to the same category, and the values for the other entries are set to zero.

Based on the above definitions, we now formally state our task of version-aware app rating prediction in the following definition.

Definition 3.6. ***Version-aware App Rating Prediction.*** Given the user-version rating matrix $\mathbf{R}$, the version-term $\mathbf{X}$, the app-term matrix $\mathbf{Y}$, the app-version association matrix $\mathbf{M}$, and the app similarity matrix $\mathbf{S}$, our goal is to predict the missing rating values for the latest versions in $\mathbf{R}$.

As stated in the above problem definition, our goal is to fill in the empty entries in certain columns of the rating matrix $\mathbf{R}$. These columns correspond to the apps' current/latest versions. To better understand our task, we further present an illustrative example for the task in Fig. 5. In addition to matrix $\mathbf{R}$, The input includes two term matrices (version-term $\mathbf{X}$ and app-term $\mathbf{Y}$), the app-version associations (matrix $\mathbf{M}$), as well as the similarities between the apps (matrix $\mathbf{S}$). The input also in-

[6]Note that the two matrices share the same vocabulary.


App Similarity (**S**)
1
2
3
App-Version Association (**M**)
App 1
App 2
App 3
Rating Matrix (**R**)
App-Term Matrix (**Y**)
Version-Term Matrix (**X**)
Version Correlation (**P**)

	Version 1	Version 2	Version 3	Version 4	Version 5	Version 6
User 1	3		5		5	
User 2				4		2
User 3		4	4			
User 4			5	3		1
User 5	5		3			

	Term 1	Term 2	Term 3	Term 4	Term 5	Term 6
App 1	1	1		3	2	1
App 2			3	2		1
App 3		1	1			

	Term 1	Term 2	Term 3	Term 4	Term 5	Term 6
Version 1	1			1	2	
Version 2		1		2		1
Version 3			1	1		
Version 4				1		
Version 5			2			1
Version 6		1	1			

App 1	Version 1	Version 2	
App 2	Version 3	Version 4	Version 5
App 3	Version 6		

Fig. 5. An illustrative example for version-based app rating prediction. There are five users and six versions from three apps. The task is to predict the missing values for the latest versions (i.e., columns 2, 5, 6) in the user-version rating matrix. App-level and version-level text information can be used. The correlations between apps and versions are also available.

cludes the version correlation matrix **P**, which can be derived from matrix **M**. The **P** matrix captures the temporal correlations between multiple versions of the same app, and we will further discuss the construction of matrix **P** in Section 4.2.

4. THE PROPOSED FORMULATION

In this section, we present the proposed model for the version-aware rating prediction problem defined in the previous section. We start with a base model, and then introduce the considerations for the awareness of version information.

4.1. A Base Model

In our base model, we incorporate the review text into the matrix factorization framework.

4.1.1. A Non-Negative Matrix Factorization Formulation. In this work, to model the rating behaviors, we adopt the standard non-negative matrix factorization which is defined as follows

$$\min_{\mathbf{U},\mathbf{V}\geqslant 0} \|\mathbf{I}^R \odot (\mathbf{R} - \mathbf{U}\mathbf{V}')\|_F^2 + \lambda_r(\|\mathbf{U}\|_F^2 + \|\mathbf{V}\|_F^2) \tag{4}$$

where $\mathbf{I}^R$ is the indicator matrix for the non-empty entries in **R** and λ_r is the tuning parameter for the regularization factors. The basic idea of the above formulation is to decompose the original rating matrix into two low-rank matrices $\mathbf{U} \in \mathbb{R}^{U\times K_1}$ and $\mathbf{V} \in \mathbb{R}^{V\times K_1}$ by minimizing the reconstruction error, where K_1 is the number of dimensions in the latent space and we have $K_1 \ll N$ and $K_1 \ll M$. Here, the non-negativity constraint (i.e., all the entries in **U** and **V** are non-negative) makes the resulting matrices easier to inspect.

Once the two low-rank matrices have been derived, we can simply predict a missing entry by using a dot product between two corresponding row vectors in **U** and **V**

$$\hat{\mathbf{R}}(u,i) = \mathbf{U}(u,:)\mathbf{V}(i,:)'. \tag{5}$$

4.1.2. Enhancing Non-Negative Matrix Factorization with Version-level Review Text. A direct technical challenge of matrix factorization is that it usually suffers from the data sparsity issue, i.e., the model becomes less effective for versions with very few ratings. Therefore, a commonly used method is to incorporate auxiliary information to enhance matrix factorization. Here, we propose to incorporate the review text that is associated with each rating. Review text contains users' opinions towards the corresponding version of an app. The opinionated text has been shown to be effective for recommender systems [Agarwal and Chen 2009; Wang and Blei 2011]. We combine all the reviews related to a version into a single document, and model the review text as a version-term matrix $\mathbf{X}$. In this work, the review text is only used for enriching the item side.

Our idea is to decompose the version-term matrix $\mathbf{X}$ into two low-rank matrices $\mathbf{F}_1 \in \mathbb{R}^{V \times K_2}$ and $\mathbf{G} \in \mathbb{R}^{T \times K_2}$. In this way, an item (i.e., a version) would have two kinds of latent factors, either derived from rating data or review text. We can concatenate these two kinds of latent factors and derive a NMF formulation with review text as follows

$$\begin{aligned} \min_{\mathbf{U},\mathbf{H},\mathbf{F}_1,\mathbf{G} \geqslant 0} \quad & ||\mathbf{I}^R \odot (\mathbf{R} - \mathbf{U}\mathbf{V}')||_F^2 + \lambda_x ||\mathbf{X} - \mathbf{F}_1\mathbf{G}'||_F^2 + \lambda_r(||\mathbf{U}||_F^2 + ||\mathbf{H}||_F^2 + ||\mathbf{F}_1||_F^2 + ||\mathbf{G}||_F^2) \\ s.t. \quad & \mathbf{V} = [\mathbf{H}, \mathbf{F}_1] \end{aligned} \tag{6}$$

where $\mathbf{H} \in \mathbb{R}^{V \times K_1}$ and $\mathbf{F}_1 \in \mathbb{R}^{V \times K_2}$ are the rating- and text-based latent representation matrices for versions, and λ_x is used to control the importance of the review text. In order to recover $\mathbf{R}$, we set $\mathbf{V} = [\mathbf{H}, \mathbf{F}_1]$, which is a concatenation of $\mathbf{H}$ and $\mathbf{F}_1$. The resulting $\mathbf{V}$ is a $V \times (K_1 + K_2)$ matrix, and $\mathbf{U}$ is a $U \times (K_1 + K_2)$ matrix. By representing users and items in the same latent space, we can reconstruct the rating matrix $\mathbf{R}$ using the decomposed matrices as Eq. (5).

4.1.3. Improving Text Semantic Represntation with App-level Review Text. Previously, we have decomposed the version-term matrix and leveraged version-related text information to improve the rating prediction task. However, a problem is that the version-term matrix itself is very sparse. For versions with very few reviews, the derived latent factors may not be reliable. Thus, we propose to aggregate reviews by apps and decompose the app-term matrix. In other words, we consider capturing the text semantics in both the version level and the app level.

Similar to the Eq. (6), we can factorize the app-term matrix $\mathbf{Y}$ into two low-rank matrices $\mathbf{F}_2 \in \mathbb{R}^{A \times K_3}$ and $\mathbf{G} \in \mathbb{R}^{T \times K_3}$. Formally, we have

$$\begin{aligned} \min_{\mathbf{U},\mathbf{H},\mathbf{F}_1,\mathbf{F}_2,\mathbf{G} \geqslant 0} \quad & ||\mathbf{I}^R \odot (\mathbf{R} - \mathbf{U}\mathbf{V}')||_F^2 + \lambda_r(||\mathbf{U}||_F^2 + ||\mathbf{H}||_F^2 + ||\mathbf{F}_1||_F^2 + ||\mathbf{F}_2||_F^2 + ||\mathbf{G}||_F^2) \\ & + \lambda_x ||\mathbf{X} - \mathbf{F}_1\mathbf{G}'||_F^2 + \lambda_y ||\mathbf{Y} - \mathbf{F}_2\mathbf{G}'||_F^2 \\ s.t. \quad & \mathbf{V} = [\mathbf{H}, \mathbf{F}_1, \mathbf{M}'\mathbf{F}_2] \end{aligned} \tag{7}$$

where $\mathbf{F}_2 \in \mathbb{R}^{A \times K_3}$ is the review-based latent matrices for apps, and λ_y is used to control the importance of app-level review text. In our model, since the version-level and app-level text share the same vocabulary, we require that they also share the same latent matrix $\mathbf{G}$ for the terms in the vocabulary. As a result, we have $K_2 = K_3$. The sharing of $\mathbf{G}$ alleviates text sparsity in the version level by sharing app-level information.

Based on Eq. (7), a version now corresponds to three kinds of latent factors, and we can follow the above method to concatenate them as the final representation. A problem is how to transform the app-level information into version-based decomposition. Here, we assume all the versions of the same app share the same app-level review-based latent factors, and we transform $\mathbf{F}_2$ for versions by using the mapping $\mathbf{M}'\mathbf{F}_2$, where $\mathbf{M}$ is the app-version association matrix defined in Eq. (3). As a result, we can set $\mathbf{V} = [\mathbf{H}, \mathbf{F}_1, \mathbf{M}'\mathbf{F}_2]$, which is a concatenation of $\mathbf{H}$, $\mathbf{F}_1$, and $\mathbf{M}'\mathbf{F}_2$. Now, $\mathbf{V}$ is a $V \times (K_1 + 2 \times K_2)$ matrix, and $\mathbf{U}$ is a $U \times (K_1 + 2 \times K_2)$ matrix. In this work, we set $K_1 = K_2 = K$ for simplicity. By projecting users and items in the same latent space, we can reconstruct the rating matrix $\mathbf{R}$ with Eq. (5).

4.2. Improved Model with Version-based Correlations

The base model in Eq. (7) considers both rating and text information. However, the multiple versions of an app are still treated as independent items in the formulation. Now, we consider improving the model by incorporating version-based correlations. In what follows, we model two kinds of version-based correlations, i.e., temporal correlations from preceding versions of the same app and aggregate correlations between similar apps.

4.2.1. Modeling Temporal Correlations from Preceding Versions. The first type of version correlation is based on the temporal correlations from preceding versions. Given a version of an app, it is intuitive that the preceding versions will have a potential influence on the current version. For example, keeping a good app feature will continuously attract high ratings from users.

To model the temporal influence, we propose to correlate the latent factors of the current version with those of its preceding versions using a temporal decaying similarity function. The intuition is that if two versions have closer release dates, they tend to be more similar with each other. Given two versions i and j from app a ($i > j$), we measure their temporal decaying similarity $\theta_{i,j}$ as

$$\theta_{i,j} = \exp(-b \cdot \Delta_{i,j}), \tag{8}$$

where $b \geq 0$ is a tuning parameter for the exponential function and $\Delta_{i,j}$ is the temporal distance between these two versions. In practice, we normalize the values of $\theta_{i,(\cdot)}$ for each version. If we further set $\Delta_{i,j} = +\infty$ when i precedes j or when i and j belong to two different apps, we can generalize Eq. (8) to any two versions as follows

$$\theta_{i,j} = \begin{cases} \exp(-b \cdot \Delta_{i,j}), & \text{if versions } i \text{ and } j \text{ are with the same app and } t_i > t_j; \\ 0, & \text{otherwise,} \end{cases} \tag{9}$$

where t_i and t_j are the release dates for versions i and j, respectively, and $\Delta_{i,j} = |t_i - t_j|$.

In order to simulate an evolutionary process, we require that only the previous versions are influential on the current versions. After normalizing the values of $\theta_{i,(\cdot)}$ for each version, we can model the temporal correlations by minimizing the following regularizer

$$\mathcal{R}_{temporal} = \sum_{i=1}^{V} ||\mathbf{V}(i,:) - \sum_{j=1}^{i-1} \theta_{i,j}\mathbf{V}(j,:)||_2^2. \tag{10}$$

The above regularizer has an intuitive explanation. The latent factors of the current version tend to be similar to those of its preceding versions with a degree controlled by their temporal distance. Based on the Eq (10), a larger b indicates that the more recent versions are more influential on the current version. For example, if we set $b = 0$, all the preceding versions would play equal roles for the current version; if we set $b = +\infty$, only the closest version would affect the current version. Similar temporal correlations are proposed in [Tang et al. 2012; Gao et al. 2013], which can be seen as a special case by setting $b = +\infty$.

With some linear algebra manipulations, we can rewrite the above regualizer as

$$\mathcal{R}_{temporal} = ||\mathbf{PV}||_F^2, \tag{11}$$

with matrix $\mathbf{P}$ in the following diagonal form

$$\mathbf{P} = \begin{vmatrix} \mathbf{P}_1 & & & \\ & \mathbf{P}_2 & & \\ & & \cdots & \\ & & & \mathbf{P}_A \end{vmatrix} \tag{12}$$

where A is the number of apps. Each diagonal submatrix $\mathbf{P}_a$ corresponds to app a, and only the similarities for the versions from the same app are non-zero.

For app a, $\mathbf{P}_a$ is a $(V_a - 1) \times V_a$ matrix

$$\mathbf{P}_a = \begin{vmatrix} -\theta_{2,1} & 1 & & & \\ -\theta_{3,1} & -\theta_{3,2} & 1 & & \\ \cdots & \cdots & \cdots & \cdots & \\ -\theta_{V_a,1} & -\theta_{V_a,2} & \cdots & -\theta_{V_a,V_a-1} & 1 \end{vmatrix} \tag{13}$$

where V_a is the number of versions for app a. The k-th row in $\mathbf{P}_a$ stores the (negative) similarity information for the $(k+1)$-th version with all its preceding versions.

4.2.2. Modeling Aggregate Correlations from Similar Apps. The second type of version-based correlation is the aggregate correlations between similar apps. Given two functionally similar apps, it is likely that they will have similar latent factors. Here, a problem is that app similarity is not measured in the version level, and we need to transfer the app-level correlation to the version level. Our idea is that, we first aggregate the latent factors from all the versions of an app. Then, we can model the correlation over these aggregated app-level latent factors. We call such correlation as *app-level aggregate correlation*.

Here, we require the aggregated latent factors of two similar apps to be close to each other. Formally, we propose to minimize the second regularizer which is defined as

$$\mathcal{R}_{app} = \frac{1}{2} \sum_{a=1}^{A} \sum_{a'=1}^{A} \mathbf{S}(a, a') \|\bar{\mathbf{v}}_a - \bar{\mathbf{v}}_{a'}\|_2^2, \tag{14}$$

where $\mathbf{S}(a, a')$ is the similarity between two apps a and a', and $\bar{\mathbf{v}}_a$ and $\bar{\mathbf{v}}_{a'}$ are the aggregated latent vectors for app a and a', respectively. With the V_a versions of app a, we set $\bar{\mathbf{v}}_a$ using an average aggregation as

$$\bar{\mathbf{v}}_a = \frac{1}{V_a} \sum_{i=i_a}^{i_a+V_a-1} \mathbf{V}(i, :), \tag{15}$$

where i_a is the index for the first version of app a, and $(i_a + V_a - 1)$ is the index for the latest version of app a.

Further, we can rewrite $\mathcal{R}_{app}$ as follows

$$\begin{aligned} \mathcal{R}_{app} &= \frac{1}{2} \sum_{a=1}^{A} \sum_{a'=1}^{A} \sum_{k=1}^{K} \mathbf{S}(a, a')(\bar{\mathbf{v}}_a(k) - \bar{\mathbf{v}}_{a'}(k))^2 \\ &= \sum_{a=1}^{A} \sum_{a'=1}^{A} \sum_{k=1}^{K} \mathbf{S}(a, a')\bar{\mathbf{v}}_a^2(k) - \sum_{a=1}^{A} \sum_{a'=1}^{A} \sum_{k=1}^{K} \mathbf{S}(a, a')\bar{\mathbf{v}}_a(k)\bar{\mathbf{v}}_{a'}(k) \\ &= \sum_{k=1}^{K} [\overline{\mathbf{M}}\mathbf{V}](:, k)'(\mathbf{D}_S - \mathbf{S})[\overline{\mathbf{M}}\mathbf{V}](:, k) \\ &= tr[(\overline{\mathbf{M}}\mathbf{V})'(\mathbf{D}_S - \mathbf{S})(\overline{\mathbf{M}}\mathbf{V})] \end{aligned} \tag{16}$$

where K is the rank of $\mathbf{V}$, $\mathbf{D}_S$ is the degree matrix for $\mathbf{S}$ with $\mathbf{D}_S(a, a') = \sum_{a'=1}^{A} \mathbf{S}(a, a')$, and $\overline{\mathbf{M}}$ is the row-normalized $\mathbf{M}$ matrix.

4.2.3. The Final Model. By integrating all the above factors, our objective function can be written as

$$\mathcal{L}(\mathbf{R}, \mathbf{X}, \mathbf{Y}, \mathbf{M}, \mathbf{P}, \mathbf{S}) = \mathcal{E}_{rating} + \mathcal{E}_{text} + \mathcal{R}_{temporal} + \mathcal{R}_{app} + \mathcal{R}_{parameter}, \tag{17}$$

where we have the input of user-version rating matrix $\mathbf{R}$, version-term matrix $\mathbf{X}$, app-term matrix $\mathbf{Y}$, app-version association matrix $\mathbf{M}$, the temporal correlation matrix $\mathbf{P}$, and the app similarity matrix

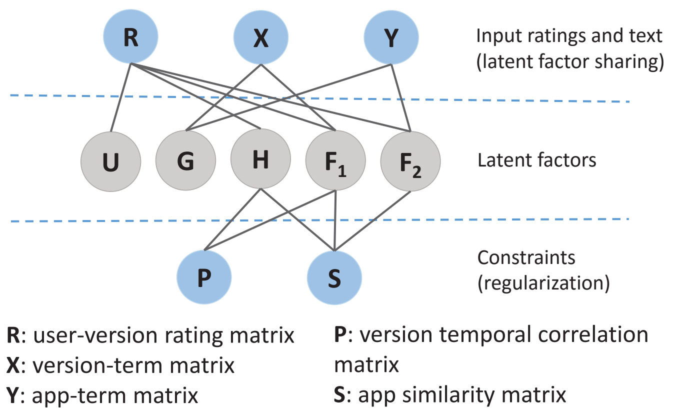

R
X
Y
Input ratings and text (latent factor sharing)
U
G
H
F_1
F_2
Latent factors
P
S
Constraints (regularization)
R: user-version rating matrix
X: version-term matrix
Y: app-term matrix
P: version temporal correlation matrix
S: app similarity matrix

Fig. 6. A decomposition graph for our model VAMF.

S, and we have divided the objective function into five objective terms defined as follows

$$\begin{aligned}
\mathcal{E}_{rating} &= ||\mathbf{I}^R \odot (\mathbf{R} - \mathbf{U}\mathbf{V}')||_F^2 \\
\mathcal{E}_{text} &= \lambda_x||\mathbf{X} - \mathbf{F}_1\mathbf{G}'||_F^2 + \lambda_y||\mathbf{Y} - \mathbf{F}_2\mathbf{G}'||_F^2, \\
\mathcal{R}_{temporal} &= \lambda_p||\mathbf{P}\mathbf{V}||_F^2, \\
\mathcal{R}_{app} &= \lambda_s tr[(\overline{\mathbf{M}}\mathbf{V})'(\mathbf{D}_S - \mathbf{S})(\overline{\mathbf{M}}\mathbf{V})], \\
\mathcal{R}_{parameter} &= \lambda_r(||\mathbf{U}||_F^2 + ||\mathbf{H}||_F^2 + ||\mathbf{F}_1||_F^2 + ||\mathbf{F}_2||_F^2 + ||\mathbf{G}||_F^2).
\end{aligned}$$

Note that we have incorporated corresponding tuning parameters λs to control the relative importance of each objective terms. By integrating all the objective terms into the above model, we have the final objective function

$$\begin{aligned}
\min_{\mathbf{U},\mathbf{H},\mathbf{F}_1,\mathbf{F}_2,\mathbf{G}\geqslant 0} \quad & \underbrace{||\mathbf{I}^R \odot (\mathbf{R} - \mathbf{U}\mathbf{V}')||_F^2}_{\text{reconstruction of rating matrix}} \\
& + \underbrace{\lambda_x||\mathbf{X} - \mathbf{F}_1\mathbf{G}'||_F^2 + \lambda_y||\mathbf{Y} - \mathbf{F}_2\mathbf{G}'||_F^2}_{\text{reconstruction of text matrix}} \\
& + \underbrace{\lambda_p||\mathbf{P}\mathbf{V}||_F^2}_{\text{version temporal correlation}} \\
& + \underbrace{\lambda_s tr[(\overline{\mathbf{M}}\mathbf{V})'(\mathbf{D}_S - \mathbf{S})(\overline{\mathbf{M}}\mathbf{V})]}_{\text{version aggregate correlation}} \\
& + \underbrace{\lambda_r(||\mathbf{U}||_F^2 + ||\mathbf{H}||_F^2 + ||\mathbf{F}_1||_F^2 + ||\mathbf{F}_2||_F^2 + ||\mathbf{G}||_F^2)}_{\text{parameter regularization}} \\
s.t. \quad & \mathbf{V} = [\mathbf{H}, \mathbf{F}_1, \mathbf{M}'\mathbf{F}_2]
\end{aligned} \tag{18}$$

We call the proposed model *Version-Aware Matrix Factorization (VAMF)*, which jointly considers ratings, review text, and version correlations. To better understand how VAMF works, we present a decomposition graph for it in Fig 6.

4.2.4. Discussions and Generalizations. We further present some discussions and generalizations of our VAMF model.

First, as shown in Fig 6, we adopt two common ways (i.e., latent factor sharing [Ma et al. 2008] and regularization [Yao et al. 2014]) to model the side information (i.e., review text, temporal relations, app similarities, etc.). For the review text, we constrain that the two text matrices (**X** and **Y**) share the learned latent factors with the rating matrix **R**. For the temporal relations and app similarities, we model them as regularization terms to constrain the distances between learned latent factors.

Second, in our model formulation, we encode temporal decaying with fixed an exponential factor in matrix **P**. When other information such as the version changelog is available, we can construct a more sophisticated **P** matrix. For example, if the changelog shows minor updates, we can constrain that the latent factors of the two corresponding versions are closer to each other, compared to the cases when the changelog shows major updates.

Finally, we would like to point out that our formulation is flexible and can be generalized to other settings. For instance, when users' demographical information is available, we can further use it to guide the learning of the **U** matrix by constraining that similar users tend to have similar latent factors. By modeling such information into Eq. (18), we have[7]

$$\begin{aligned}\min_{\mathbf{U},\mathbf{H},\mathbf{F}_1,\mathbf{F}_2,\mathbf{G}\geqslant 0} \quad & ||\mathbf{I}^R \odot (\mathbf{R} - \mathbf{U}\mathbf{V}')||_F^2 + \lambda_x||\mathbf{X} - \mathbf{F}_1\mathbf{G}'||_F^2 + \lambda_y||\mathbf{Y} - \mathbf{F}_2\mathbf{G}'||_F^2 + \lambda_p||\mathbf{P}\mathbf{V}||_F^2 \\ & +\lambda_s tr[(\overline{\mathbf{M}}\mathbf{V})'(\mathbf{D}_S - \mathbf{S})(\overline{\mathbf{M}}\mathbf{V})] + \lambda_r(||\mathbf{U}||_F^2 + ||\mathbf{H}||_F^2 + ||\mathbf{F}_1||_F^2 + ||\mathbf{F}_2||_F^2 + ||\mathbf{G}||_F^2) \\ & + \underbrace{\lambda_z tr[\mathbf{U}'(\mathbf{D}_Z - \mathbf{Z})\mathbf{U}]}_{\text{user side regularization}} \\ s.t. \quad & \mathbf{V} = [\mathbf{H}, \mathbf{F}_1, \mathbf{M}'\mathbf{F}_2] \end{aligned} \tag{19}$$

where **Z** contains the similarities between users derived from users' demographical information, $\mathbf{D}_Z$ is the degree matrix of **Z**, and λ_z is used to control the importance of this term. Also, we can naturally incorporate some additional constraints such as sparseness in the learned matrices. For the sake of clarity, we skip the details of such generalizations in the article.

5. THE PROPOSED ALGORITHM

In this section, we present the learning algorithm for Eq. (18) followed by some algorithm analysis.

5.1. The VAMF Algorithm

It is difficult to directly optimize Eq. (18), because the optimization problem is not jointly convex due to the coupling between **U**, **H**, $\mathbf{F}_1$, $\mathbf{F}_2$, and **G**. Therefore, instead of seeking for a global optimal solution, we aim to find a local minimum by alternatively updating one of these parameters while fixing the others.

5.1.1. Update of **U**. When **H**, $\mathbf{F}_1$, $\mathbf{F}_2$, and **G** are fixed, the optimization problem in Eq. (18) becomes the minimization problem of the following equation (by dropping some constant terms) w.r.t. the matrix **U**

$$\begin{aligned} J_U = & ||\mathbf{I}^R \odot (\mathbf{R} - \mathbf{U}\mathbf{V}')||_F^2 + \lambda_r||\mathbf{U}||_F^2 \\ & \text{s.t.} \quad \mathbf{U} \geqslant 0 \end{aligned} \tag{20}$$

For clarity, we divide **U** as $[\mathbf{U}_1, \mathbf{U}_2, \mathbf{U}_3]$ which correspond to the three entries in $[\mathbf{H}, \mathbf{F}_1, \mathbf{M}'\mathbf{F}_2]$, respectively.

[7]We do not experimentally evaluate the following model as the demographical information is unavailable in our data set.

The derivative of J_U w.r.t. $\mathbf{U}_1$, $\mathbf{U}_2$, and $\mathbf{U}_3$ can be computed as

$$\begin{aligned}
\frac{1}{2}\frac{\partial J_U}{\partial \mathbf{U}_1} &= -\mathbf{RH} + (\mathbf{I}^R \odot (\mathbf{U}_1\mathbf{H}'))\mathbf{H} + (\mathbf{I}^R \odot (\mathbf{U}_2\mathbf{F}_1'))\mathbf{H} \\
&\quad + (\mathbf{I}^R \odot (\mathbf{U}_3(\mathbf{M}'\mathbf{F}_2)'))\mathbf{H} + \lambda_r \mathbf{U}_1 n \\
\frac{1}{2}\frac{\partial J_U}{\partial \mathbf{U}_2} &= -\mathbf{RF}_1 + (\mathbf{I}^R \odot (\mathbf{U}_1\mathbf{H}'))\mathbf{F}_1 + (\mathbf{I}^R \odot (\mathbf{U}_2\mathbf{F}_1'))\mathbf{F}_1 \\
&\quad + (\mathbf{I}^R \odot (\mathbf{U}_3(\mathbf{M}'\mathbf{F}_2)'))\mathbf{F}_1 + \lambda_r \mathbf{U}_2 \\
\frac{1}{2}\frac{\partial J_U}{\partial \mathbf{U}_3} &= -\mathbf{RM}'\mathbf{F}_2 + (\mathbf{I}^R \odot (\mathbf{U}_1\mathbf{H}'))\mathbf{M}'\mathbf{F}_2 + (\mathbf{I}^R \odot (\mathbf{U}_2\mathbf{F}_1'))\mathbf{M}'\mathbf{F}_2 \\
&\quad + (\mathbf{I}^R \odot (\mathbf{U}_3(\mathbf{M}'\mathbf{F}_2)'))\mathbf{M}'\mathbf{F}_2 + \lambda_r \mathbf{U}_3 \qquad (21)
\end{aligned}$$

A fixed-point solution for each of the above equations with the non-negativity constraint leads to the following multiplicative updating rules for $\mathbf{U}_1$, $\mathbf{U}_2$, and $\mathbf{U}_3$, respectively

$$\begin{aligned}
\mathbf{U}_1(u,k) &\leftarrow \mathbf{U}_1(u,k)\sqrt{\frac{[\mathbf{RH}](u,k)}{[\mathbf{B}_1\mathbf{H} + \mathbf{B}_2\mathbf{H} + \mathbf{B}_3\mathbf{H} + \lambda_r\mathbf{U}_1](u,k)}} \\
\mathbf{U}_2(u,k) &\leftarrow \mathbf{U}_2(u,k)\sqrt{\frac{[\mathbf{RF}_1](u,k)}{[\mathbf{B}_1\mathbf{F}_1 + \mathbf{B}_2\mathbf{F}_1 + \mathbf{B}_3\mathbf{F}_1 + \lambda_r\mathbf{U}_2](u,k)}} \\
\mathbf{U}_3(u,k) &\leftarrow \mathbf{U}_3(u,k)\sqrt{\frac{[\mathbf{RM}'\mathbf{F}_2](u,k)}{[\mathbf{B}_1\mathbf{M}'\mathbf{F}_2 + \mathbf{B}_2\mathbf{M}'\mathbf{F}_2 + \mathbf{B}_3\mathbf{M}'\mathbf{F}_2 + \lambda_r\mathbf{U}_3](u,k)}} \qquad (22)
\end{aligned}$$

where

$$\begin{aligned}
\mathbf{B}_1 &= \mathbf{I}^R \odot (\mathbf{U}_1\mathbf{H}') \\
\mathbf{B}_2 &= \mathbf{I}^R \odot (\mathbf{U}_2\mathbf{F}_1') \\
\mathbf{B}_3 &= \mathbf{I}^R \odot (\mathbf{U}_3(\mathbf{M}'\mathbf{F}_2)') \qquad (23)
\end{aligned}$$

5.1.2. Update of $\mathbf{H}$. To update $\mathbf{H}$, we fix $\mathbf{U}$, $\mathbf{F}_1$, $\mathbf{F}_2$, and $\mathbf{G}$. The optimization problem in Eq. (18) becomes the minimization problem of the following equation

$$\begin{aligned}
J_H &= ||\mathbf{I}^R \odot (\mathbf{R} - \mathbf{U}_1\mathbf{H}')||_F^2 + \lambda_p||\mathbf{PH}||_F^2 + \lambda_s tr[(\overline{\mathbf{M}}\mathbf{H})'(\mathbf{D}_S - \mathbf{S})(\overline{\mathbf{M}}\mathbf{H})] + \lambda_r||\mathbf{H}||_F^2 \\
&\text{s.t.} \quad \mathbf{H} \geqslant 0 \qquad (24)
\end{aligned}$$

The derivative of J_H w.r.t. $\mathbf{H}$ is

$$\begin{aligned}
\frac{1}{2}\frac{\partial J_H}{\partial \mathbf{H}} &= -\mathbf{R}'\mathbf{U}_1 + (\mathbf{I}^R \odot (\mathbf{U}_1\mathbf{H}'))'\mathbf{U}_1 + (\mathbf{I}^R \odot (\mathbf{U}_2\mathbf{F}_1'))'\mathbf{U}_1 + (\mathbf{I}^R \odot (\mathbf{U}_3(\mathbf{M}'\mathbf{F}_2)'))'\mathbf{U}_1 \\
&\quad + \lambda_p\mathbf{P}'\mathbf{PH} + \lambda_s\overline{\mathbf{M}}'(\mathbf{D}_S - \mathbf{S})\overline{\mathbf{M}}\mathbf{H} + \lambda_r\mathbf{H} \qquad (25)
\end{aligned}$$

Notice that, there are negative elements in the $\mathbf{P}$ matrix. We divide it into two non-negative matrices as $\mathbf{P}^+ - \mathbf{P}^-$, where

$$\begin{aligned}
\mathbf{P}^+(i,j) &= (|\mathbf{P}(i,j)| + \mathbf{P}(i,j))/2 \\
\mathbf{P}^-(i,j) &= (|\mathbf{P}(i,j)| - \mathbf{P}(i,j))/2 \qquad (26)
\end{aligned}$$

This leads to the following multiplicative updating rule for $\mathbf{H}$

$$\mathbf{H}(i,k) \leftarrow \mathbf{H}(i,k)\sqrt{\frac{[\mathbf{R}'\mathbf{U}_1 + \lambda_p(\mathbf{B}_5+\mathbf{B}_6)\mathbf{H} + \lambda_s\overline{\mathbf{M}}'\mathbf{S}\overline{\mathbf{M}}\mathbf{H}](i,k)}{[\mathbf{B}_1'\mathbf{U}_1 + \mathbf{B}_2'\mathbf{U}_1 + \mathbf{B}_3'\mathbf{U}_1 + \lambda_p(\mathbf{B}_4+\mathbf{B}_7)\mathbf{H} + \lambda_s\overline{\mathbf{M}}'\mathbf{D}_S\overline{\mathbf{M}}\mathbf{H} + \lambda_r\mathbf{H}](i,k)}} \tag{27}$$

where $\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3$ are defined in Eq. (23), and $\mathbf{B}_4, \mathbf{B}_5, \mathbf{B}_6, \mathbf{B}_7$ are defined as

$$\begin{aligned} \mathbf{B}_4 &= (\mathbf{P}^+)'\mathbf{P}^+ \\ \mathbf{B}_5 &= (\mathbf{P}^+)'\mathbf{P}^- \\ \mathbf{B}_6 &= (\mathbf{P}^-)'\mathbf{P}^+ \\ \mathbf{B}_7 &= (\mathbf{P}^-)'\mathbf{P}^- \end{aligned} \tag{28}$$

In Eq. (27), $\mathbf{S}$ is a sparse non-negative symmetric matrix, and $\mathbf{D}_S$ is a diagonal non-negative matrix.

5.1.3. Update of $\mathbf{F}_1$. Next, we show how we update $\mathbf{F}_1$, when $\mathbf{U}$, $\mathbf{H}$, $\mathbf{F}_2$, and $\mathbf{G}$ are fixed. The corresponding optimization problem becomes to minimize the following equation

$$\begin{aligned} J_{F1} &= \|\mathbf{I}^R \odot (\mathbf{R} - \mathbf{U}_2\mathbf{F}_1')\|_F^2 + \lambda_x\|\mathbf{X} - \mathbf{F}_1\mathbf{G}'\|_F^2 + \lambda_p\|\mathbf{P}\mathbf{F}_1\|_F^2 \\ &+\lambda_s tr[(\overline{\mathbf{M}}\mathbf{F}_1)'(\mathbf{D}_S - \mathbf{S})(\overline{\mathbf{M}}\mathbf{F}_1)] + \lambda_r\|\mathbf{F}_1\|_F^2 \\ &\text{s.t.} \quad \mathbf{F}_1 \geqslant 0 \end{aligned} \tag{29}$$

The derivative of J_{F1} w.r.t. $\mathbf{F}_1$ is

$$\begin{aligned} \frac{1}{2}\frac{\partial J_{F1}}{\partial \mathbf{F}_1} &= -\mathbf{R}'\mathbf{U}_2 + (\mathbf{I}^R \odot (\mathbf{U}_1\mathbf{H}'))'\mathbf{U}_2 + (\mathbf{I}^R \odot (\mathbf{U}_2\mathbf{F}_1'))'\mathbf{U}_2 + (\mathbf{I}^R \odot (\mathbf{U}_3(\mathbf{M}'\mathbf{F}_2)'))'\mathbf{U}_2 \\ &+\lambda_x(-\mathbf{X}\mathbf{G} + \mathbf{F}_1\mathbf{G}'\mathbf{G}) + \lambda_p\mathbf{P}'\mathbf{P}\mathbf{F}_1 + \lambda_s\overline{\mathbf{M}}'(\mathbf{D}_S - \mathbf{S})\overline{\mathbf{M}}\mathbf{F}_1 + \lambda_r\mathbf{F}_1 \end{aligned} \tag{30}$$

which leads to the multiplicative updating rule for $\mathbf{F}_1$

$$\mathbf{F}_1(i,k) \leftarrow \mathbf{F}_1(i,k)\sqrt{\frac{[\mathbf{R}'\mathbf{U}_2 + \lambda_x\mathbf{X}\mathbf{G} + \lambda_p(\mathbf{B}_5+\mathbf{B}_6)\mathbf{F}_1 + \lambda_s\overline{\mathbf{M}}'\mathbf{S}\overline{\mathbf{M}}\mathbf{F}_1](i,k)}{[(\mathbf{B}_1' + \mathbf{B}_2' + \mathbf{B}_3')\mathbf{U}_2 + \lambda_x\mathbf{F}_1\mathbf{G}'\mathbf{G} + \lambda_p(\mathbf{B}_4+\mathbf{B}_7)\mathbf{F}_1 + \lambda_s\overline{\mathbf{M}}'\mathbf{D}_S\overline{\mathbf{M}}\mathbf{F}_1 + \lambda_r\mathbf{F}_1](i,k)}} \tag{31}$$

where $\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3$ are defined in Eq. (23), and $\mathbf{B}_4, \mathbf{B}_5, \mathbf{B}_6, \mathbf{B}_7$ are defined in Eq. (28).

5.1.4. Update of $\mathbf{F}_2$. Next, we show how we update $\mathbf{F}_2$. Our goal is to minimize the following equation

$$\begin{aligned} J_{F2} &= \|\mathbf{I}^R \odot (\mathbf{R} - \mathbf{U}_3(\mathbf{M}'\mathbf{F}_2)')\|_F^2 + \lambda_y\|\mathbf{Y} - \mathbf{F}_2\mathbf{G}'\|_F^2 + \\ &+\lambda_s tr[\mathbf{F}_2'(\mathbf{D}_S - \mathbf{S})\mathbf{F}_2] + \lambda_r\|\mathbf{F}_2\|_F^2 \\ &\text{s.t.} \quad \mathbf{F}_2 \geqslant 0 \end{aligned} \tag{32}$$

In the above equation, we omit the $\|\mathbf{P}(\mathbf{M}'\mathbf{F}_2)\|_F^2$ as this term will result in zero. We also simplify the $tr[\mathbf{F}_2'(\mathbf{D}_S - \mathbf{S})\mathbf{F}_2]$ term as $\overline{\mathbf{M}}\mathbf{M}' = \mathbf{I}$.

The derivative of J_{F2} w.r.t. $\mathbf{F}_2$ is

$$\begin{aligned} \frac{1}{2}\frac{\partial J_{F2}}{\partial \mathbf{F}_2} &= -\mathbf{M}\mathbf{R}'\mathbf{U}_3 + \mathbf{M}(\mathbf{I}^R \odot (\mathbf{U}_1\mathbf{H}'))'\mathbf{U}_3 + \mathbf{M}(\mathbf{I}^R \odot (\mathbf{U}_2\mathbf{F}_1'))'\mathbf{U}_3 + \mathbf{M}(\mathbf{I}^R \odot (\mathbf{U}_3(\mathbf{M}'\mathbf{F}_2)'))'\mathbf{U}_3 \\ &+\lambda_y(-\mathbf{Y}\mathbf{G} + \mathbf{F}_2\mathbf{G}'\mathbf{G}) + \lambda_s(\mathbf{D}_S - \mathbf{S})\mathbf{F}_2 + \lambda_r\mathbf{F}_2 \end{aligned} \tag{33}$$

which leads to the multiplicative updating rule

$$\mathbf{F}_2(i,k) \leftarrow \mathbf{F}_2(i,k)\sqrt{\frac{[\mathbf{M}\mathbf{R}'\mathbf{U}_3 + \lambda_y\mathbf{Y}\mathbf{G} + \lambda_s\mathbf{S}\mathbf{F}_2](i,k)}{[\mathbf{M}(\mathbf{B}_1' + \mathbf{B}_2' + \mathbf{B}_3')\mathbf{U}_3 + \lambda_y\mathbf{F}_2\mathbf{G}'\mathbf{G} + \lambda_s\mathbf{D}_S\mathbf{F}_2 + \lambda_r\mathbf{F}_2](i,k)}} \tag{34}$$

where $\mathbf{B}_1$, $\mathbf{B}_2$, $\mathbf{B}_3$ are defined in Eq. (23).

5.1.5. Update of $\mathbf{G}$. Next, we show how we update $\mathbf{G}$ when the other matrices are fixed. The equation that we aim to minimize becomes

$$\begin{aligned} J_G &= \lambda_x\|\mathbf{X} - \mathbf{F}_1\mathbf{G}'\|_F^2 + \lambda_y\|\mathbf{Y} - \mathbf{F}_2\mathbf{G}'\|_F^2 + \lambda_r\|\mathbf{G}\|_F^2 \\ &\quad \text{s.t.} \quad \mathbf{G} \geqslant 0 \end{aligned} \tag{35}$$

The derivative of J_G w.r.t. $\mathbf{G}$ is

$$\frac{1}{2}\frac{\partial J_G}{\partial \mathbf{G}} = \lambda_x(-\mathbf{X}\mathbf{G} + \mathbf{F}_1\mathbf{G}'\mathbf{G}) + \lambda_y(-\mathbf{Y}\mathbf{G} + \mathbf{F}_2\mathbf{G}'\mathbf{G}) + \lambda_r\mathbf{G} \tag{36}$$

which leads to the multiplicative updating rule for $\mathbf{G}$

$$\mathbf{G}(i,k) \leftarrow \mathbf{G}(i,k)\sqrt{\frac{[\lambda_x\mathbf{X}'\mathbf{F}_1 + \lambda_y\mathbf{Y}'\mathbf{F}_2](i,k)}{[\lambda_x\mathbf{G}\mathbf{F}_1'\mathbf{F}_1 + \lambda_y\mathbf{G}\mathbf{F}_2'\mathbf{F}_2 + \lambda_r\mathbf{G}](i,k)}} \tag{37}$$

5.1.6. The Final Learning Algorithm. Finally, we summarize the overall algorithm for solving Eq. (18) in Alg. 1. As we can see, after we initialize the matrices (Step 1), generate the matrix $\mathbf{P}$ (Step 2), and compute the matrices $\mathbf{B}_4, \mathbf{B}_5, \mathbf{B}_6, \mathbf{B}_7$ (Step 3), the algorithm begins the iteration procedure. In each iteration, the algorithm first computes the matrices $\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3$ (Step 5), and then alternatively updates $\mathbf{U}_1$, $\mathbf{U}_2$, $\mathbf{U}_3$, $\mathbf{H}$, $\mathbf{F}_1$, $\mathbf{F}_2$, and $\mathbf{G}$ (Steps 6-27). We use the following criteria to terminate the iteration procedure: either the Frobenius norm between successive estimates of both $\mathbf{U}$ and $\mathbf{V}$ is below our threshold ξ or the maximum iteration step l is reached. Finally, the algorithm outputs $\mathbf{U}$ as $[\mathbf{U}_1, \mathbf{U}_2, \mathbf{U}_3]$ and $\mathbf{V}$ as $[\mathbf{H}, \mathbf{F}_1, \mathbf{M}'\mathbf{F}_2]$. With these output results, we can predict the preference of user u on version i by $\mathbf{U}(u,:)\mathbf{V}(i,:)'$.

Remarks. Here, we further discuss the updating rules when the users' demographical information is available (i.e., Eq. (19)). When users' demographical information is added into the model, we can reuse Alg. 1 by substituting Eq. (22) with the following updating rules (i.e., Eq (38)) when updating the $\mathbf{U}$ matrix.

$$\begin{aligned} \mathbf{U}_1(u,k) &\leftarrow \mathbf{U}_1(u,k)\sqrt{\frac{[\mathbf{R}\mathbf{H} + \lambda_z\mathbf{Z}\mathbf{U}_1](u,k)}{[\mathbf{B}_1\mathbf{H} + \mathbf{B}_2\mathbf{H} + \mathbf{B}_3\mathbf{H} + \lambda_z\mathbf{D}_Z\mathbf{U}_1\lambda_r\mathbf{U}_1](u,k)}} \\ \mathbf{U}_2(u,k) &\leftarrow \mathbf{U}_2(u,k)\sqrt{\frac{[\mathbf{R}\mathbf{F}_1 + \lambda_z\mathbf{Z}\mathbf{U}_2](u,k)}{[\mathbf{B}_1\mathbf{F}_1 + \mathbf{B}_2\mathbf{F}_1 + \mathbf{B}_3\mathbf{F}_1 + \lambda_z\mathbf{D}_Z\mathbf{U}_2 + \lambda_r\mathbf{U}_2](u,k)}} \\ \mathbf{U}_3(u,k) &\leftarrow \mathbf{U}_3(u,k)\sqrt{\frac{[\mathbf{R}\mathbf{M}'\mathbf{F}_2 + \lambda_z\mathbf{Z}\mathbf{U}_3](u,k)}{[\mathbf{B}_1\mathbf{M}'\mathbf{F}_2 + \mathbf{B}_2\mathbf{M}'\mathbf{F}_2 + \mathbf{B}_3\mathbf{M}'\mathbf{F}_2 + \lambda_z\mathbf{D}_Z\mathbf{U}_3 + \lambda_r\mathbf{U}_3](u,k)}} \end{aligned} \tag{38}$$

5.2. Algorithm Analysis

Here, we briefly analyze the optimality and computational complexity of our algorithm.

We first show the correctness of Eq. (31) for updating $\mathbf{F}_1$, by proving that the fixed-point solution of Eq. (31) satisfies the KKT condition. The correctness for updating the other matrices (i.e., $\mathbf{U}$, $\mathbf{H}$ $\mathbf{F}_2$, and $\mathbf{G}$) can be proved analogously.

THEOREM 5.1. **Correctness of Eq.** (31). *The fixed-point solution of Eq.* (31) *satisfies the KKT condition.*

Algorithm 1 The VAMF Algorithm.

```
Input: the user-version rating matrix $\mathbf{R}$, the version-term matrix $\mathbf{X}$, the app-term matrix $\mathbf{Y}$, the app similarity matrix $\mathbf{S}$, and the app-version association matrix $\mathbf{M}$
Output: the latent matrix for users $\mathbf{U}$ and the latent matrix for versions $\mathbf{V}$
1: initialize $\mathbf{U}_1$, $\mathbf{U}_2$, $\mathbf{U}_3$, $\mathbf{H}$, $\mathbf{F}_1$, $\mathbf{F}_2$, and $\mathbf{G}$ randomly;
2: generate $\mathbf{P}$ as defined in Eq. (12);
3: compute $\mathbf{B}_4, \mathbf{B}_5, \mathbf{B}_6, \mathbf{B}_7$ as defined in Eq. (28);
4: while not convergent do
5:     compute $\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3$ as defined in Eq. (23);
6:     for $u = 1 : U$ do
7:         for $k = 1 : K$ do
8:             update $\mathbf{U}_1(u,k), \mathbf{U}_2(u,k), \mathbf{U}_3(u,k)$ as defined in Eq. (22);
9:         end for
10:    end for
11:    for $i = 1 : V$ do
12:        for $k = 1 : K$ do
13:            update $\mathbf{H}(i,k)$ as defined in Eq. (27);
14:            update $\mathbf{F}_1(i,k)$ as defined in Eq. (31);
15:        end for
16:    end for
17:    for $i = 1 : A$ do
18:        for $k = 1 : K$ do
19:            update $\mathbf{F}_2(i,k)$ as defined in Eq. (34);
20:        end for
21:    end for
22:    for $i = 1 : T$ do
23:        for $k = 1 : K$ do
24:            update $\mathbf{G}(i,k)$ as defined in Eq. (37);
25:        end for
26:    end for
27: end while
28: return $\mathbf{U} = [\mathbf{U}_1, \mathbf{U}_2, \mathbf{U}_3]$ and $\mathbf{V} = [\mathbf{H}, \mathbf{F}_1, \mathbf{M}'\mathbf{F}_2]$;
```

PROOF. We start with Lagrangian function of Eq. (29)

$$\begin{aligned} L_{F1} &= ||\mathbf{I}^R \odot (\mathbf{R} - \mathbf{U}_2\mathbf{F}_1')||_F^2 + \lambda_x||\mathbf{X} - \mathbf{F}_1\mathbf{G}'||_F^2 + \lambda_p||\mathbf{P}\mathbf{F}_1||_F^2 + \lambda_s tr[(\overline{\mathbf{M}}\mathbf{F}_1)'\mathbf{D}_S(\overline{\mathbf{M}}\mathbf{F}_1)] \\ &\quad -\lambda_s tr[(\overline{\mathbf{M}}\mathbf{F}_1)'\mathbf{S}(\overline{\mathbf{M}}\mathbf{F}_1)] + \lambda_r||\mathbf{F}_1||_F^2 - tr[(\Lambda'\mathbf{F})] \end{aligned}$$

where Λ is the Lagrange multiplier. Let the derivative of the above equation equal to 0, we have

$$\begin{aligned} &2(-\mathbf{R}'\mathbf{U}_2 + (\mathbf{I}^R \odot (\mathbf{U}_1\mathbf{H}'))'\mathbf{U}_2 + (\mathbf{I}^R \odot (\mathbf{U}_2\mathbf{F}_1'))'\mathbf{U}_2 + (\mathbf{I}^R \odot (\mathbf{U}_3(\mathbf{M}'\mathbf{F}_2)'))'\mathbf{U}_2 \\ &+\lambda_x(-\mathbf{X}\mathbf{G} + \mathbf{F}_1\mathbf{G}'\mathbf{G}) + \lambda_p\mathbf{P}'\mathbf{P}\mathbf{F}_1 + \lambda_s\overline{\mathbf{M}}'(\mathbf{D}_S - \mathbf{S})\overline{\mathbf{M}}\mathbf{F}_1 + \lambda_r\mathbf{F}_1) = \Lambda \end{aligned}$$

From the KKT complementary slackness condition, we have

$$\begin{aligned} &[-\mathbf{R}'\mathbf{U}_2 + (\mathbf{I}^R \odot (\mathbf{U}_1\mathbf{H}'))'\mathbf{U}_2 + (\mathbf{I}^R \odot (\mathbf{U}_2\mathbf{F}_1'))'\mathbf{U}_2 + (\mathbf{I}^R \odot (\mathbf{U}_3(\mathbf{M}'\mathbf{F}_2)'))'\mathbf{U}_2 \\ &+\lambda_x(-\mathbf{X}\mathbf{G} + \mathbf{F}_1\mathbf{G}'\mathbf{G}) + \lambda_p\mathbf{P}'\mathbf{P}\mathbf{F}_1 + \lambda_s\overline{\mathbf{M}}'(\mathbf{D}_S - \mathbf{S})\overline{\mathbf{M}}\mathbf{F}_1 + \lambda_r\mathbf{F}_1](i,k)\mathbf{F}_1(i,k) = 0 \end{aligned}$$

Clearly, a fixed point of the updating rule in Eq. (31) satisfies the above equation, which completes the proof. □

The above theorem has shown that the updating rule in Eq. (31) yields a correct solution at convergence. Next, we show that the updating rule in Eq. (31) is guaranteed to converge in the following theorem. The theorem can be proved following [Gu et al. 2010; Yao et al. 2014], and we omit it for brevity.

THEOREM 5.2. **Convergence of Eq.** (31). *Under the updating rule of Eq.* (31)*, Eq.* (29) *decreases monotonically.*

PROOF. Omitted for brevity. □

Next, the effectiveness of Alg. 1 is shown in the following corollary, which states that the algorithm finds a local optimum for Eq. (18). Given that the original optimization problem is not jointly convex w.r.t. the parameters, such a local minimum is acceptable in practice.

COROLLARY 5.3. **Effectiveness of Alg. 1**. *Alg. 1 finds a local minimum for the optimization problem in Eq.* (18)*.*

PROOF SKETCH: Combining Theorem 5.1 and Theorem 5.2, and based on the alternating procedure in Alg. 1, we have that Alg. 1 finds a local minimum for the optimization problem in Eq. (18). □

The time complexity and space complexity of the proposed algorithm are summarized in the following lemma, which basically states that Alg. 1 enjoys linear scalability w.r.t. the data size in both time and space. Notice that all the input matrices (**R**, **X**, **Y**, **M**, **S**, and **P**) are often very sparse (e.g., $|\mathbf{R}| \ll UV$, $|\mathbf{X}| \ll VT$, etc).

LEMMA 5.4. **Complexity of Alg. 1**. *The time complexity of Alg. 1 is* $O((|\mathbf{R}|+|\mathbf{X}|+|\mathbf{Y}|+|\mathbf{S}|)Kl+(VK^2+AK^2+TK^2+UK)l)$*; the space complexity of Alg. 1 is* $O(|\mathbf{R}|+|\mathbf{X}|+|\mathbf{Y}|+|\mathbf{S}|+(U+V+A+T)K)$*.*

PROOF. In the algorithm, the first step requires $O(UK + VK + AK + TK)$ time where K is the size of latent factors. For Step 2, since one app usually has a relatively small number of versions, generating **P** would need $O(V)$ time. Similarly, Step 3 needs $O(V)$ time as both $\mathbf{P}^+$ and $\mathbf{P}^-$ are very sparse. By computing the predicted values only on the observed examples, Step 5 requires $O(|\mathbf{R}|K)$ time where $|\mathbf{R}|$ indicates the number of observed entries in **R**. For Steps 6-10, we need $O(|\mathbf{R}|K + UK + VK)$ time. Similarly, Steps 11-16 require $O(|\mathbf{R}|K + |\mathbf{S}|K + |\mathbf{X}|K + VK^2 + TK^2)$ time. Note that $|\mathbf{M}| = V$. Next, Steps 17-21 need $O(|\mathbf{R}|K + |\mathbf{S}|K + |\mathbf{Y}|K + VK + AK^2 + TK^2)$ time. We need $O(|\mathbf{X}|K + |\mathbf{Y}|K + VK^2 + AK^2 + TK^2)$ time for Steps 22-26. Overall, time complexity of Alg. 1 is $O((|\mathbf{R}| + |\mathbf{X}| + |\mathbf{Y}| + |\mathbf{S}|)Kl + (VK^2 + AK^2 + TK^2 + UK)l)$.

For the space complexity, we need $O(|\mathbf{R}|+|\mathbf{X}|+|\mathbf{Y}|+|\mathbf{S}|+V)$ space for input. We need $O((U+V+A+T)K)$ space for Step 1. We need $O(V)$ space for both Step 2 and Step 3. Step 5 requires $O(|\mathbf{R}|)$ space. We need $O(UK)$ space for Steps 6-10, $O(VK)$ space for Steps 11-16, $O(AK)$ space for Steps 17-21, and $O(TK)$ space for Steps 22-26. The overall space complexity is $O(|\mathbf{R}| + |\mathbf{X}| + |\mathbf{Y}| + |\mathbf{S}| + (U + V + A + T)K)$. □

6. EXPERIMENTAL EVALUATION

In this section, we construct the evaluation experiments and present the analysis for the results. The experiments are designed to answer the following questions.

— How accurate is the proposed model for version-based mobile app rating prediction?
— How does it perform in the cold-start setting?
— How does the performance vary for apps from different categories?
— How scalable is the proposed algorithm?

6.1. Experimental Setup

Construction of the Evaluation Set. We use the Google Play data set in Table I as the evaluation set. The data contains 3,367,435 ratings from 170,781 users over 104,061 versions from 16,457 apps. As mentioned before, for the app similarities, we represent the app descriptions as a document for each app, and then compute the cosine similarity between the term frequency vectors of these documents. To keep the similarity graph sparse, we assign $\mathbf{S}(a, a') = 1$ if the cosine similarity between app a and app a' is larger than 0.4.

For rating prediction, a common evaluation scheme is to split the rating data set into a training set and a test set. We learn the models on the training set and subsequently examine the performance of the learned models on the test set. We consider the following two splitting ways.

— New-version splitting. In the task of app rating prediction, the ratings are usually attached with the version information. Since the versions are updated with time, our goal is to predict the ratings on latest versions using historical data. We treat the latest version of an app as a new version. To implement such an evaluation, we randomly take $(100 - \gamma)\%$ of the ratings from the latest version for each app as the test set, and use the rest ratings as the training data. This splitting method measures the predictive ability of a recommender system on apps with evolving versions. When $\gamma = 0$, it essentially becomes the cold-start prediction task, where we do not have any rating information about the new version of an app. In this work, we choose four values for γ in the new-version splitting, i.e., $\gamma = 0, \gamma = 30, \gamma = 60$, and $\gamma = 90$.
— Random splitting. Users' ratings on older versions may be helpful to understand users' interest points. Therefore, we also check the rating prediction accuracy on older versions. Here, we adopt a random splitting, where we randomly sample 10% ratings as the test set and use the rest ratings as the training data.

We assume that a review is explicitly associated with a rating. When we split the ratings, the corresponding reviews are naturally re-organized into training and test sets.

Methods to Compare. We consider the following methods for performance comparison:

— *NMF [Lee and Seung 2000].* Non-negative matrix factorization (NMF) refers to the algorithm where a matrix $\mathbf{R}$ is factorized into two matrices $\mathbf{U}$ and $\mathbf{V}$, with the property that all three matrices have no negative elements. This non-negativity makes the resulting matrices easier to inspect.
— *KBV [Koren et al. 2009].* This method resembles NMF in that it also factorizes $\mathbf{R}$ into two matrices $\mathbf{U}$ and $\mathbf{V}$. KBV does not have the non-negativity constraint, and it can flexibly incorporate bias in order to improve the prediction performance.
— *HFT [McAuley and Leskovec 2013].* HFT combines ratings with review text for rating prediction. It aligns hidden factors in product ratings with hidden topics in product reviews. HFT is developed in a probabilistic topic model framework.
— *CTR [Wang and Blei 2011].* CTR combines text content of items with collaborative rating prediction. Topic modeling is used to provide a representation of the items in terms of latent themes discovered from the text content. Overall, CTR and HFT can be understood as a hybrid of collaborative filtering and content-based approaches.
— *VAMF.* It is our proposed model in Eq. (18). It integrates ratings, review text, and version-based correlations.

The comparison methods have involved the matrix factorization component, i.e., $||\mathbf{I}^R \odot (\mathbf{R} - \mathbf{U}\mathbf{V}')||_F^2$. In practice, different initializations of $\mathbf{U}$ and $\mathbf{V}$ might lead to varying performance. Thus, we use the same initialization values for $\mathbf{U}$ and $\mathbf{V}$ with a random assignment. For HFT and CTR, we follow the original setting by combing all the reviews related to an item into a single document. We preform the same preprocessing steps on review text for CTR, HFT, and VAMF. The number of latent dimensions is set to 50 for all the methods (i.e., $K = 50$ for VAMF). We set $b = 1$, $\lambda_x = \lambda_y = \lambda_r = 0.1$, $\lambda_v = 10$, and $\lambda_s = 1$ via standard cross validation. We will discuss the parameter sensitivity in the following experiments. For other parameters, we set the maximum iteration number $l = 20$ and the termination threshold $\xi_1 = 10^{-4}$.

Table IV. Performance comparisons on the evaluation set with random splitting. VAMF outperforms all the compared methods w.r.t. both RMSE and MAE.

Methods	RMSE	MAE
NMF	1.2609	0.9167
KBV	1.2495	0.9047
HFT	1.2437	0.9198
CTR	1.2438	0.8964
VAMF	1.1889	0.8767

Evaluation Metrics. For the effectiveness evaluation of rating prediction, we adopt two commonly used metrics, namely RMSE and MAE

$$RMSE = \sqrt{\frac{\sum_{(u,i)\in\mathcal{T}_{test}}(\mathbf{R}(u,i)-\hat{\mathbf{R}}(u,i))^2}{|\mathcal{T}_{test}|}},$$

$$MAE = \frac{\sum_{(u,i)\in\mathcal{T}_{test}}|\mathbf{R}(u,i)-\hat{\mathbf{R}}(u,i)|}{|\mathcal{T}_{test}|},$$

where $\mathcal{T}_{test}$ is the test set and $\hat{\mathbf{R}}(u,i)$ is the predicted value for the entry (u,i). A smaller value for both RMSE and MAE indicates better performance.

For efficiency, we report the wall-clock time as the evaluation metric. All the experiments were run on a machine with eight 3.5GHz Intel Cores and 64GB memory.

6.2. Experimental Results

In this part, we present and analyze the experimental results.

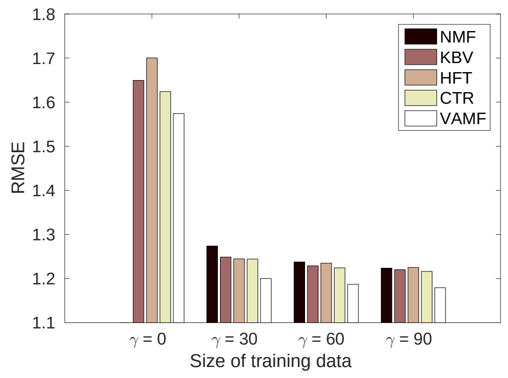
Effectiveness Results with Random Splitting. We first compare the performance of different methods on the evaluation set with randoms splitting[8]. The results are presented in Table IV. Among the baselines, CTR achieves the best performance in terms of MAE, while HFT achieves the best performance in terms of RMSE. Both methods have used review text information, which improves over the methods using the rating data alone including NMF and KBV. CTR and HFT adopt two different ways to combine the information of topics and ratings. CTR characterizes the item latent vector by combing the corresponding topic distribution with an item-specific bias vector, while HFT links the topic distribution with the rating latent vector by using an exponential transformation. The above results indicate that the review text information is important to consider in app rating prediction. In addition to review text, our model VAMF incorporates the version-based correlations through two regularization terms, and it performs best among all the methods.

Effectiveness Results with New-Version Splitting. Next, since our goal is to predict the ratings for the latest versions, we compare the performance of different methods on the evaluation set with new-version splitting. Recall that $(100-\gamma)\%$ of the ratings from the latest version of each app are used as the test data. We study how the performance of different methods vary with different values of γ, and the results[9] are presented in Fig 7.

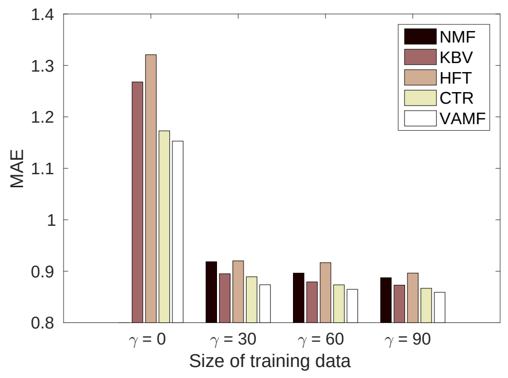
There are several observations from the figures. First of all, the proposed VAMF outperforms all the compared methods w.r.t. both RMSE and MAE in all cases. For example, when $\gamma = 30$, VAMF improves NMF, KBV, HFT, and CTR by 5.8%, 3.9%, 3.6%, and 3.6% w.r.t. RMSE, respectively. Second, all the methods perform relatively poor when $\gamma = 0$ (i.e., the full cold-start setting). For KBV, HFT, and CTR, such a pure cold-start problem is handled by modeling users' rating bias. In

[8]All the remaining experiments are based on the new-version splitting.

[9]The results of NMF for $\gamma = 0$ are ignored in the figures as it performs much worse than the other methods.

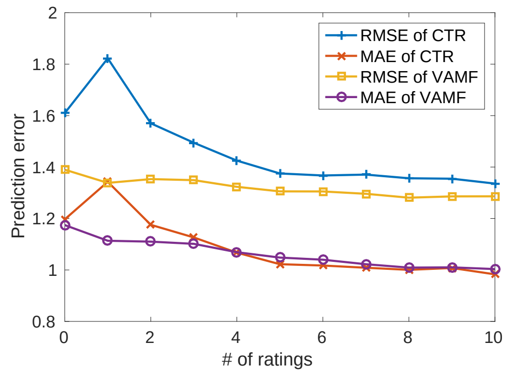


(a) RMSE

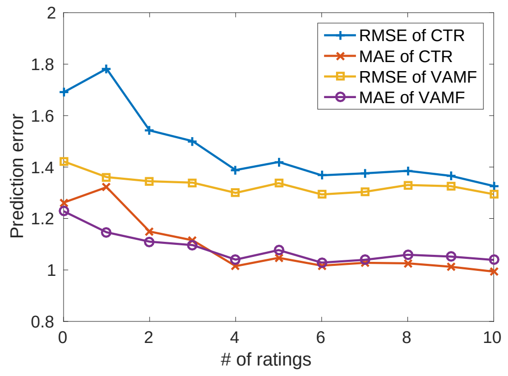


(b) MAE

Fig. 7. Performance comparisons on the evaluation set with the version-based splitting. VAMF outperforms all the compared methods w.r.t. both RMSE and MAE in all cases.



(a) $\gamma = 30$



(b) $\gamma = 60$

Fig. 8. Detailed performance comparisons on cold-start versions. VAMF outperforms CTR especially for severe cold-start cases.

contrast, we handle the problem by employing the latent factors from previous versions. This indicates the importance of version correlations for cold-start scenarios. Third, overall, the performance of all the methods improves with more training data on the new versions. As for the compared methods, CTR performs better than HFT in terms of both metrics. A possible reason is that HFT makes a direct mapping between topic distribution and latent vectors for items; in contrast, CTR combines the topic distribution with an item-specific bias vector as the item latent representation, potentially having more flexibility.

Detailed Analysis in the Cold-Start Setting. The above experimental results have shown the overall effectiveness of our method in the full cold-start setting (i.e., $\gamma = 0$), where all the apps do not have any ratings for their latest versions. Next, we consider the cold-start setting when the apps may have a few ratings for their latest versions. Since we have found that CTR is the most competitive baseline for new-version rating prediction, we adopt it as the comparison method here. The results[10] are presented in Fig. 8. The x-axis of the figure indicates the number of ratings received by the cold-start versions. Here, we report the results with $\gamma = 30$ and $\gamma = 60$.

[10]We treat a version as cold-start version if it has fewer than ten ratings in the training set. We categorize the results into different groups according to the number of ratings that an app has received for its latest version.

Table V. Effects of different components of VAMF. All components are useful to reduce the prediction error.

Methods	γ = 30		γ = 60		γ = 90	
	RMSE	MAE	RMSE	MAE	RMSE	MAE
NMF	1.2739	0.9158	1.2242	0.8897	1.2111	0.8816
NMF + **X**	1.2578	0.9028	1.2099	0.8793	1.1959	0.8713
NMF + **Y**	1.2419	0.9044	1.2082	0.8777	1.1965	0.8702
NMF + **S**	1.2487	0.9020	1.2051	0.8845	1.1969	0.8785
NMF + **P**	1.2176	0.8925	1.2026	0.8799	1. 1969	0.8751
NMF + *All*	1.2001	0.8737	1.1870	0.8648	1.1793	0.8590

As shown in Fig. 8, our method VAMF performs consistently better than CTR in terms of RMSE especially when the cold-start degree is severe (e.g., when the number of ratings of a given app version is less than four). As to MAE, VAMF also outperforms CTR when the number of available ratings of a given app version is less than four. Note that the original motivation of CTR is to make meaningful recommendations about items before anyone has rated them, and it is commonly used as a strong baseline for cold-start recommendation task. Overall, it can be concluded that our method VAMF produces competitive or even better performance than CTR in the cold-start setting.

Effects of Different Components in VAMF. As shown in Eq. (17), VAMF is a combination of multiple components. Previous experiments have shown the overall effectiveness of VAMF. Here we study the effects of each individual component on the final performance. Our model VAMF is developed based on a standard NMF formulation, and it further incorporates the version-term matrix decomposition (corresponding to **X**), app-term matrix decomposition (corresponding to **Y**), version temporal correlation (corresponding to **P**), and app-level aggregate correlation (corresponding to **S**). Table V presents the results when each of the above four components is added. We report the cases when γ is set to 30, 60, and 90. The results of NMF and our final mode VAMF (referred to as "NMF + All" in the table) are also added for comparison.

It can observed from the table that all the components are indeed useful to reduce the prediction error (for both RMSE and MAE) in all the three cases. For example, when γ = 30, the four components (i.e., **X**, **Y**, **S**, and **P**) improve the NMF baseline by 1.3%, 2.5%, 2.01%, and 4.4% w.r.t. RMSE, respectively. Further, we can observe from the table that the version-temporal correlation (i.e., NMF + **P**) contributes most to the final performance. This indicates the importance of the version-based correlation.

Detailed Performance Comparisons By Category. We have shown that VAMF is more effective than the other methods by having an overall better performance. Next, we check the detailed performance of VAMF in all the 42 categories of the data collection. We also adopt the best baseline CTR as the comparison method, and the RMSE results are shown in Table VI. As we can see from the table, for the three cases ($\gamma = 30, 60, 90$), VAMF is better than CTR in all the categories w.r.t. RMSE. For certain categories (e.g., "Comics"), VAMF can achieve up to 17.5% improvement over CTR.

Parameter Sensitivity. Next, we present a parameter study of the proposed method. In our model VAMF, we have several parameters to set, including the component coefficients λs, the decaying factor b, and the number of latent dimensions K. For all these parameter, we tune one of them and fix the others. The results are shown in Fig. 9. Here, we use the training data with $\gamma = 60$. The results of CTR are also incorporated for comparison. Overall, we can observe from the figures that the performance is relatively stable with the varying of all the parameters in a relatively wide range. In this work, we set $\lambda_x = \lambda_y = 0.1$, $\lambda_s = 1$, $\lambda_p = 10$, $b = 1$, and $K = 50$.

Efficiency Analysis. Finally, we evaluate the efficiency of our method by reporting the wall-clock time of VAMF in the training stage (i.e., Alg. 1). We use the subsets of the data set to test its scalability. For example, when we test the scalability of VAMF w.r.t. the number of versions, we tune the size of versions while fix the size of others (e.g., users, terms, etc.). The results are shown in

Table VI. The RMSE results of VAMF on different categories. VAMF outperforms CTR in all the categories.

Category	γ = 30			γ = 60			γ = 90		
	CTR	VAMF	Improve	CTR	VAMF	Improve	CTR	VAMF	Improve
Communication	1.3393	1.3069	2.42%	1.3348	1.2991	2.67%	1.3250	1.2862	2.93%
Game Racing	1.2186	1.1773	3.39%	1.2140	1.1660	3.95%	1.2171	1.1751	3.45%
Productivity	1.1603	1.1105	4.29%	1.1269	1.0879	3.46%	1.1538	1.1087	3.91%
Personalization	1.1447	1.1069	3.31%	1.1173	1.0922	2.25%	1.0657	1.0641	0.15%
Media and Video	1.2875	1.2316	4.34%	1.2842	1.2416	3.32%	1.2497	1.2147	2.80%
Travel and Local	1.2295	1.1810	3.94%	1.2339	1.1987	2.85%	1.1872	1.1637	1.98%
Entertainment	1.3793	1.3194	4.34%	1.3532	1.3075	3.38%	1.3609	1.3121	3.59%
Finance	1.3068	1.2458	4.67%	1.2772	1.2251	4.08%	1.2838	1.2513	2.53%
Tools	1.2320	1.1779	4.39%	1.2047	1.1637	3.40%	1.1758	1.1378	3.23%
Game Strategy	1.2552	1.2103	3.57%	1.2256	1.1739	4.22%	1.2208	1.1900	2.52%
Game Card	1.2021	1.1500	4.33%	1.1649	1.1362	2.47%	1.1219	1.0924	2.63%
Game Casual	1.2747	1.2273	3.71%	1.2585	1.2175	3.26%	1.2705	1.2244	3.63%
Weather	1.2025	1.1672	2.93%	1.1456	1.1155	2.63%	1.0934	1.0490	4.06%
Game Arcade	1.2123	1.2007	0.95%	1.1934	1.1802	1.11%	1.2048	1.1819	1.90%
Game Sports	1.1987	1.1665	2.68%	1.1861	1.1523	2.85%	1.1674	1.1346	2.81%
Photography	1.2375	1.1759	4.97%	1.2041	1.1603	3.64%	1.2000	1.1508	4.10%
Education	1.1546	1.0816	6.32%	1.1160	1.0629	4.76%	1.1478	1.0932	4.76%
Game Puzzle	1.1898	1.1416	4.05%	1.1602	1.1247	3.06%	1.1417	1.1106	2.72%
Game Educational	1.3089	1.2475	4.69%	1.2736	1.2342	3.09%	1.2474	1.2254	1.76%
Game Role Playing	1.3223	1.2551	5.08%	1.2745	1.2176	4.47%	1.2632	1.2297	2.65%
Business	1.2466	1.1900	4.54%	1.2568	1.2021	4.35%	1.2077	1.1515	4.65%
Game Family	1.2644	1.2062	4.61%	1.2399	1.1974	3.43%	1.2214	1.1734	3.93%
Game Casino	1.2011	1.1648	3.02%	1.1990	1.1682	2.57%	1.1926	1.1683	2.04%
Game Action	1.2162	1.1871	2.39%	1.2016	1.1692	2.70%	1.1926	1.1532	3.31%
Health and Fitness	1.2166	1.1569	4.90%	1.1726	1.1270	3.89%	1.1431	1.1116	2.75%
Game Word	1.2325	1.1969	2.89%	1.2197	1.1839	2.93%	1.2080	1.1836	2.02%
Game Trivia	1.2290	1.1609	5.54%	1.2357	1.1978	3.06%	1.2729	1.2518	1.66%
Game Music	1.1722	1.1257	3.97%	1.1423	1.0962	4.04%	1.0490	1.0376	1.08%
Game Adventure	1.2596	1.2132	3.69%	1.2592	1.2171	3.35%	1.2177	1.1643	4.39%
Music and Audio	1.2648	1.2001	5.12%	1.2520	1.1901	4.94%	1.2137	1.1845	2.40%
Game Simulation	1.3577	1.3074	3.70%	1.3346	1.2851	3.71%	1.3294	1.2707	4.42%
Lifestyle	1.2739	1.1880	6.74%	1.2367	1.1960	3.29%	1.1885	1.1464	3.55%
Social	1.4076	1.3408	4.74%	1.3881	1.3393	3.51%	1.3864	1.3286	4.17%
Transportation	1.3210	1.2067	8.65%	1.3427	1.2945	3.59%	1.2752	1.2506	1.93%
Books and Reference	1.1020	1.0604	3.78%	1.0925	1.0666	2.37%	1.0339	1.0179	1.55%
Sports	1.4659	1.3743	6.25%	1.4015	1.3434	4.14%	1.4497	1.3357	7.86%
Medical	1.3089	1.1647	11.02%	1.2285	1.1729	4.53%	1.3324	1.2101	9.17%
News and Magazines	1.3429	1.2711	5.34%	1.2778	1.2201	4.52%	1.2933	1.2375	4.31%
Shopping	1.1885	1.1385	4.21%	1.1963	1.1397	4.73%	1.2850	1.2313	4.18%
Comics	1.4318	1.3217	7.69%	1.3294	1.2492	6.03%	1.4840	1.2246	17.48%
Game Board	1.2296	1.1906	3.17%	1.2166	1.1944	1.83%	1.2312	1.2097	1.74%
Libraries and Demo	1.3286	1.2926	2.71%	1.3928	1.3401	3.79%	1.3770	1.3445	2.36%

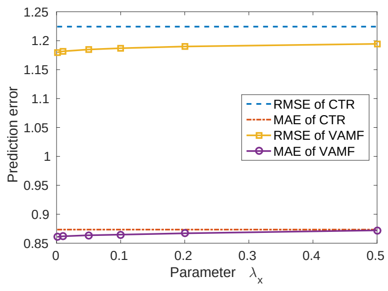
Fig. 10. In the figures, we present the results in terms of users, versions, and ratings. As we can see, the proposed algorithm scales linearly w.r.t. the number of users, the number of versions, and the number of ratings (i.e., $|\mathbf{R}|$). This result is also consistent with our algorithm analysis in Section 5.

7. RELATED WORK

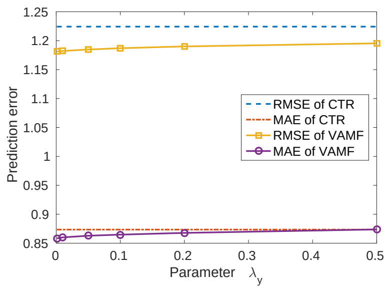
In the section, we briefly review the related work, including recommender systems, online review mining, and mobile app mining.

7.1. Recommender Systems

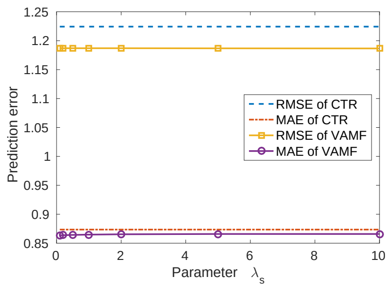
In the literature, two major tasks have been widely studied in recommender systems, namely *rating prediction* and *top-N item recommendation*. Rating prediction aims to predict the ratings from users to items, while top-*N* item recommendation aims to generate a short list of recommendations for users [Jiang et al. 2012; Deshpande and Karypis 2004; Gao et al. 2015].



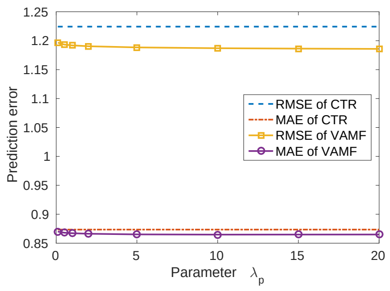
(a) Parameter λ_x



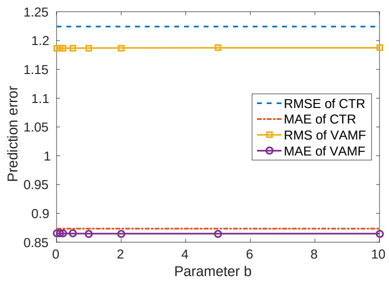
(b) Parameter λ_y



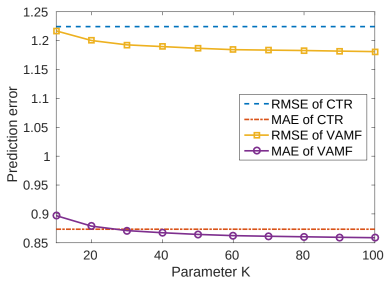
(c) Parameter λ_s



(d) Parameter λ_p

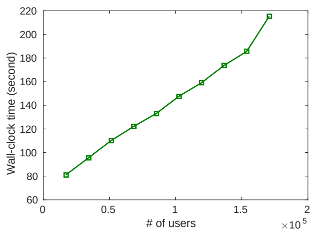


(e) Parameter b

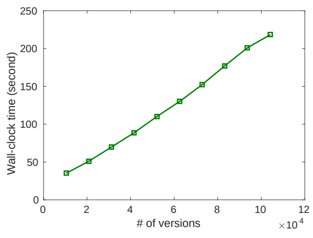


(f) Parameter K

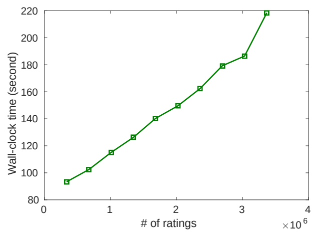
Fig. 9. Parameter sensitivity analysis. The proposed VAMF is robust with all the parameters in a relatively large range.



(a) Wall-clock time vs. # of users



(b) Wall-clock time vs. # of versions



(c) Wall-clock time vs. # of ratings

Fig. 10. Scalability of the VAMF algorithm. It scales linearly w.r.t. the data size.

Typical collaborative filtering methods for rating prediction include two major categories, namely memory-based methods and model-based methods [Jamali and Ester 2010; Jiang et al. 2012]. A combination of memory-based method and model-based method has also been explored [Koren 2008; Ma 2013]. To improve the recommendation accuracy, social links have been widely used in collaborative filtering [Ma et al. 2008, 2011; Yang et al. 2011; Shen and Jin 2012; Tang et al. 2013]. The key idea of these methods is that the linked users tend to have similar latent preferences for items.

As to top-*N* item recommendation, we divide the existing works into three categories. In the first category, top-*N* recommendation is connected with the rating prediction problem, and the predicted ratings are used to generate a ranking list [Ning and Karypis 2011; Yang et al. 2012]. In the second category, special treatment is applied to the rating prediction framework. The works in this category are usually proposed for implicit feedback where direct rating prediction methods may become ineffective [Yao et al. 2014]. For example, Hu et al. [2008] proposed a weighting-based method to weight the missing entries; Sindhwani et al. [2010] directly optimized over the missing entries; Paquet and Koenigstein [2013] sampled some negative observations from missing entries. In the third category, the loss function in the optimization objective is changed. For example, AUC-based loss function [Rendle et al. 2009; Kabbur et al. 2013] and MAP-based loss function [Shi et al. 2012] are used to substitute the squared loss in rating prediction.

The cold-start problem is one of the key challenges for both tasks. We summarize the existing solutions for the cold-start problem into three major categories: *interview based*, *adjustment based*, and *side-information based*. The interview-based methods are designed for cold-start users. In these methods, an additional set of items is usually provided for new users to rate [Harpale and Yang 2008; Zhou et al. 2011; Sun et al. 2013]. The adjustment-based methods pay special attention to the small amount of ratings from new users or for new items [Hacker and Von Ahn 2009; Xu et al. 2015]. The side-information based methods have been mostly adopted to alleviate the cold-start problem. Many kinds of auxiliary information have been exploited, including item attributes [Schein et al. 2002], demographical information [Zhang et al. 2014] and social links [Ma et al. 2008; Shen and Jin 2012].

Our work is inspired by the aforementioned recommendation algorithms, and we focus on the rating prediction problem for app recommendation. The key observation of our work is to formulate the version-level recommendation task, which is often more difficult than the app-level task. We extend existing approaches by leveraging review text and incorporating version-based correlations. We formally characterized each of the above two extensions, and carefully verified their effectiveness through extensive experiments.

7.2. Online Review Mining

Online review mining has been a hot research topic in the past decade. Pioneered by the seminar work of Hu and Liu [2004], an important task is to generate review summarization for entities such as restaurants, hotels, and products. To fulfill this task, two major steps have been involved, namely aspect/feature identification and opinion extraction. Hu and Liu [2004] applied frequent itemset mining to identify product features without supervision and considered adjectives collocated with feature words as opinion words. Following this work, some researchers including Jin and Ho [2009], Jin et al. [2009], and Wu et al. [2009] used supervised learning that requires human-labeled training sentences to identify both aspects and opinions. To develop more effective aspect extraction methods, topic modeling has been widely used as an unsupervised and knowledge-lean approach to opinion mining. These works (e.g., [Titov and McDonald 2008; Lin and He 2009; Brody and Elhadad 2010]) mainly extend standard topic models such as LDA [Blei et al. 2003] to derive better aspects from review text. As to the identification of opinion words, typical ways include using opinion lexicons [Mei et al. 2007] and using supervised information [Zhao et al. 2010].

More relevant to our work, joint modeling of reviews and ratings has received much attention from the research community. On one hand, some studies focus on generating multiple-aspect review summarization by leveraging rating information. For example, Wang et al. [2010] proposed to analyze opinions expressed about an entity in an online review at the level of topical aspects. They originally incorporated keywords to guide the learning process of aspects, and later proposed a generative model that did not need pre-specified aspect keywords [Wang et al. 2011]. On the other hand, some recent works proposed to improve the performance of rating prediction and enhance the interpretability of the generated recommendations with review text. As our two main baselines, HFT [McAuley and Leskovec 2013] aligned hidden factors in product ratings with hidden topics in product reviews, and two parts were jointly modeled in a probabilistic topic model framework; CTR [Wang and Blei 2011] set the item latent vector using the topic distribution from the corresponding "document" by adding an item-specific bias vector. Both approaches can be understood as a hybrid of collaborative filtering and content-based approaches.

Our work resembles these studies [Wang and Blei 2011; McAuley and Leskovec 2013] in that the review text is used to enhance the factorization of the rating matrix. Their focus lies in how to bridge topics and ratings, while we focus on how to solve version-aware app rating prediction. In addition, a major novelty of our work is to incorporate version-based correlations, distinguishing itself from previous studies in both task definition and model formulation. Currently, our model was developed in a matrix factorization framework due to its effectiveness for rating prediction and its flexibility for incorporating version-based correlations. Alternatively, a topic modeling approach can also be considered.

7.3. Mobile App Mining

With the great popularity of mobile devices in recent years, mining mobile app data has become an important research topic and a variety of studies have been conducted to improve the app service for both users and developers. Zhu et al. [2014] proposed to leverage both Web knowledge and real-world context for enriching the contextual information of apps, which can be exploited to improve the performance of mobile app classification. Chen et al. [2015] explored multi-modal heterogeneous data in app markets for detecting similar apps using machine learning methods. Chen et al. [2014] proposed to mine informative reviews for developers from mobile app markets. Park et al. [2015] used user reviews to improve the accuracy for mobile app retrieval. Xu and Chen [2016] studied the A/B testing framework for evaluating mobile app releases.

To provide better personalized service, several recent studies focus on improving the performance of mobile app recommendation. For example, Zhu et al. [2015] proposed a framework for personalized context-aware recommendation, and the framework can integrate both context independency and dependency assumptions. Baeza-Yates et al. [2015] studied how to improve homescreen apps' usage experience through a prediction mechanism, allowing to recommend which app a user is going to adopt in the immediate future. To alleviate the cold-start problem in app recommendation, Lin et al. [2013] described a method that accounts for nascent information culled from Twitter to provide relevant recommendations in cold-start situations. Shi and Ali [2012] focused on the heavy long-tail distribution in app market, and developed a recommendation engine for their website Getjar. In addition to the functional usage, Zhu et al. [2014] and Liu et al. [2015] incorporated users' privacy preferences to perform personalized app recommendations. Yin et al. [2013] assumed that the app adoption is a contest between the old apps' satisfaction values and the candidate apps' temptation value, and they modeled the factors that invoked a user to replace an old app with a new app. Lin et al. [2014] also studied the version-level app recommendation. However, their focus is on the meta information of each version, and they require the changelog of each version update, which is usually unavailable in open data sets[11]. In contrast, our focus is to predict users' ratings on different versions of apps by incorporating review text and version-based correlations.

8. CONCLUSION

In this article, we have proposed a unified model VAMF for the version-aware mobile app recommendation problem. In VAMF, we address the data sparsity issue by (1) incorporating review text from both the version level and the app level, and (2) modeling version-based correlations of version-level temporal correlation and app-level aggregate correlation. We propose an efficient algorithm to solve the model, and analyze its optimality and complexity. Extensive experiments on a

[11]They culled the version changelog from a paid third-party service. Our current data set does not contain such meta information, and thus we do not directly compare with this method.

large data set show that (1) the proposed method outperforms the compared methods in prediction accuracy, (2) the version-based correlation contributes most to the final performance, and (3) the algorithm enjoys linear scalability.

Currently, the evolutionary process of an app has been modeled in a simplified way: only temporal decaying similarities between two versions have been exploited. We plan to develop a more principled approach to characterize the evolving updates of an app, where the users' feedback and developers' update can be jointly and interactively characterized. We can also incorporate users' demographic information and social relationships into the rating prediction model, as such information could be useful to the app recommendation task.

ACKNOWLEDGMENTS

We would like to thank the reviewers for their constructive comments.

This work is supported by the National Key Research and Development Program of China (No. 2016YFB1000802), National 863 Program of China (No. 2015AA01A203), National Natural Science Foundation of China (No. 61672274, 61690204), the Collaborative Innovation Center of Novel Software Technology and Industrialization, and Program A for Outstanding PhD candidate of Nanjing University. Xin Zhao is partially supported by National Natural Science Foundation of China under Grant No. 61502502 and Beijing Natural Science Foundation under Grant No. 4162032. Hanghang Tong is partially supported by DTRA under the grant number HDTRA1-16-0017, by Army Research Office under the contract number W911NF-16-1-0168, by National Institutes of Health under the grant number R01LM011986, Region II University Transportation Center under the project number 49997-33 25 and a Baidu gift.

REFERENCES

Deepak Agarwal and Bee-Chung Chen. 2009. Regression-based latent factor models. In *KDD*. 19–28.

Ricardo Baeza-Yates, Di Jiang, Fabrizio Silvestri, and Beverly Harrison. 2015. Predicting the next app that you are going to use. In *WSDM*. ACM, 285–294.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet Allocation. *Journal of Machine Learning Research* 3 (2003).

Samuel Brody and Noemie Elhadad. 2010. An Unsupervised Aspect-Sentiment Model for Online Reviews. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

Ning Chen, Steven CH Hoi, Shaohua Li, and Xiaokui Xiao. 2015. Simapp: a framework for detecting similar mobile applications by online kernel learning. In *WSDM*. ACM, 305–314.

Ning Chen, Jialiu Lin, Steven CH Hoi, Xiaokui Xiao, and Boshen Zhang. 2014. AR-Miner: mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, 767–778.

Mukund Deshpande and George Karypis. 2004. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)* 22, 1 (2004), 143–177.

Huiji Gao, Jiliang Tang, Xia Hu, and Huan Liu. 2013. Exploring temporal effects for location recommendation on location-based social networks. In *RecSys*. ACM, 93–100.

Huiji Gao, Jiliang Tang, Xia Hu, and Huan Liu. 2015. Content-Aware Point of Interest Recommendation on Location-Based Social Networks.. In *AAAI*. 1721–1727.

Quanquan Gu, Jie Zhou, and Chris HQ Ding. 2010. Collaborative Filtering: Weighted Nonnegative Matrix Factorization Incorporating User and Item Graphs.. In *SDM*. SIAM, 199–210.

Severin Hacker and Luis Von Ahn. 2009. Matchin: eliciting user preferences with an online game. In *CHI*. ACM, 1207–1216.

Abhay S Harpale and Yiming Yang. 2008. Personalized active learning for collaborative filtering. In *SIGIR*. ACM, 91–98.

Minqing Hu and Bing Liu. 2004. Mining and summarizing customer reviews. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *ICDM*. 263–272.

Mohsen Jamali and Martin Ester. 2010. A matrix factorization technique with trust propagation for recommendation in social networks. In *RecSys*. ACM, 135–142.

Meng Jiang, Peng Cui, Rui Liu, Qiang Yang, Fei Wang, Wenwu Zhu, and Shiqiang Yang. 2012. Social contextual recommendation. In *CIKM*. ACM, 45–54.

Wei Jin and Hung Hay Ho. 2009. A novel lexicalized HMM-based learning framework for web opinion mining. In *Proceedings of the 26th International Conference on Machine Learning*.

Wei Jin, Hung Hay Ho, and Rohini K. Srihari. 2009. OpinionMiner: A novel machine learning system for web opinion mining and extraction. In *Proceedings of the 15th ACM SIGKDD*.

Santosh Kabbur, Xia Ning, and George Karypis. 2013. Fism: factored item similarity models for top-n recommender systems. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 659–667.

Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*. ACM, 426–434.

Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

Daniel D Lee and H Sebastian Seung. 2000. Algorithms for non-negative matrix factorization. In *NIPS*. 556–562.

Chenghua Lin and Yulan He. 2009. Joint Sentiment/Topic Model for Sentiment Analysis. In *Proceeding of the Eighteenth ACM Conference on Information and Knowledge Management*.

Jovian Lin, Kazunari Sugiyama, Min-Yen Kan, and Tat-Seng Chua. 2013. Addressing cold-start in app recommendation: latent user models constructed from twitter followers. In *The 36th International ACM SIGIR conference on research and development in Information Retrieval, SIGIR '13, Dublin, Ireland - July 28 - August 01, 2013*. 283–292.

Jovian Lin, Kazunari Sugiyama, Min-Yen Kan, and Tat-Seng Chua. 2014. New and improved: modeling versions to improve app recommendation. In *SIGIR*. ACM, 647–656.

Bin Liu, Deguang Kong, Lei Cen, Neil Zhenqiang Gong, Hongxia Jin, and Hui Xiong. 2015. Personalized mobile app recommendation: Reconciling app functionality and user privacy preference. In *WSDM*. ACM, 315–324.

Hao Ma. 2013. An experimental study on implicit social recommendation. In *SIGIR*. ACM, 73–82.

Hao Ma, Haixuan Yang, Michael R Lyu, and Irwin King. 2008. Sorec: social recommendation using probabilistic matrix factorization. In *CIKM*. 931–940.

Hao Ma, Dengyong Zhou, Chao Liu, Michael R Lyu, and Irwin King. 2011. Recommender systems with social regularization. In *WSDM*. ACM, 287–296.

Julian McAuley and Jure Leskovec. 2013. Hidden factors and hidden topics: understanding rating dimensions with review text. In *RecSys*. ACM, 165–172.

Qiaozhu Mei, Xu Ling, Matthew Wondra, Hang Su, and ChengXiang Zhai. 2007. Topic sentiment mixture: Modeling facets and opinions in weblogs. In *Proceedings of the 16th International Conference on World Wide Web*.

Xia Ning and George Karypis. 2011. Slim: Sparse linear methods for top-n recommender systems. In *2011 IEEE 11th International Conference on Data Mining*. IEEE, 497–506.

Ulrich Paquet and Noam Koenigstein. 2013. One-class collaborative filtering with random graphs. In *WWW*. 999–1008.

Dae Hoon Park, Mengwen Liu, ChengXiang Zhai, and Haohong Wang. 2015. Leveraging User Reviews to Improve Accuracy for Mobile App Retrieval. In *SIGIR*. ACM, 533–542.

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UAI*. 452–461.

Andrew I Schein, Alexandrin Popescul, Lyle H Ungar, and David M Pennock. 2002. Methods and metrics for cold-start recommendations. In *SIGIR*. ACM, 253–260.

Yelong Shen and Ruoming Jin. 2012. Learning personal+ social latent factor model for social recommendation. In *KDD*. 1303–1311.

Kent Shi and Kamal Ali. 2012. GetJar mobile application recommendations with very sparse datasets. In *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, Beijing, China, August 12-16, 2012*. 204–212.

Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Alan Hanjalic, and Nuria Oliver. 2012. TFMAP: optimizing MAP for top-n context-aware recommendation. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*. ACM, 155–164.

Vikas Sindhwani, Serhat Selcuk Bucak, Jianying Hu, and Aleksandra Mojsilovic. 2010. One-class matrix completion with low-density factorizations. In *ICDM*. 1055–1060.

Mingxuan Sun, Fuxin Li, Joonseok Lee, Ke Zhou, Guy Lebanon, and Hongyuan Zha. 2013. Learning multiple-question decision trees for cold-start recommendation. In *WSDM*. ACM, 445–454.

Jiliang Tang, Xia Hu, Huiji Gao, and Huan Liu. 2013. Exploiting local and global social context for recommendation. In *IJCAI*. 2712–2718.

Jiliang Tang, Huan Liu, Huiji Gao, and Das Das Sarmas. 2012. eTrust: understanding trust evolution in an online world. In *KDD*. ACM, 253–261.

Ivan Titov and Ryan McDonald. 2008. Modeling Online Reviews with Multi-grain Topic Models. In *Proceeding of the 17th International Conference on World Wide Web*.

Lorenzo Villarroel, Gabriele Bavota, Barbara Russo, Rocco Oliveto, and Massimiliano Di Penta. 2016. Release planning of mobile apps based on user reviews. In *Proceedings of the 38th International Conference on Software Engineering*. ACM, 14–24.

Chong Wang and David M Blei. 2011. Collaborative topic modeling for recommending scientific articles. In *KDD*. ACM, 448–456.

Hongning Wang, Yue Lu, and Chengxiang Zhai. 2010. Latent aspect rating analysis on review text data: a rating regression approach. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*. 783–792.

Hongning Wang, Yue Lu, and ChengXiang Zhai. 2011. Latent aspect rating analysis without aspect keyword supervision. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*. 618–626.

Yuanbin Wu, Qi Zhang, Xuangjing Huang, and Lide Wu. 2009. Phrase Dependency Parsing for Opinion Mining. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*.

Jingwei Xu, Yuan Yao, Hanghang Tong, Xianping Tao, and Jian Lu. 2015. Ice-breaking: mitigating cold-start recommendation problem by rating comparison. In *IJCAI*. 3981–3987.

Ya Xu and Nanyu Chen. 2016. Evaluating Mobile Apps with A/B and Quasi A/B Tests. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 313–322.

Shuang-Hong Yang, Bo Long, Alex Smola, Narayanan Sadagopan, Zhaohui Zheng, and Hongyuan Zha. 2011. Like like alike: joint friendship and interest propagation in social networks. In *WWW*. ACM, 537–546.

Xiwang Yang, Harald Steck, Yang Guo, and Yong Liu. 2012. On top-k recommendation using social networks. In *Proceedings of the sixth ACM conference on Recommender systems*. ACM, 67–74.

Yuan Yao, HangHang Tong, Guo Yan, Feng Xu, Xiang Zhang, Boleslaw Szymanski, and Lu Jian. 2014. Dual-regularized one-class collaborative filtering. In *CIKM*. ACM, 759–768.

Peifeng Yin, Ping Luo, Wang-Chien Lee, and Min Wang. 2013. App recommendation: a contest between satisfaction and temptation. In *WSDM*. ACM, 395–404.

Mi Zhang, Jie Tang, Xuchen Zhang, and Xiangyang Xue. 2014. Addressing Cold Start in Recommender Systems: A Semi-supervised Co-training Algorithm. In *SIGIR*.

Wayne Xin Zhao, Jing Jiang, Hongfei Yan, and Xiaoming Li. 2010. Jointly Modeling Aspects and Opinions with a MaxEnt-LDA Hybrid. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, EMNLP 2010, 9-11 October 2010, MIT Stata Center, Massachusetts, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*. 56–65.

Ke Zhou, Shuang-Hong Yang, and Hongyuan Zha. 2011. Functional matrix factorizations for cold-start recommendation. In *SIGIR*. 315–324.

Hengshu Zhu, Enhong Chen, Hui Xiong, Huanhuan Cao, and Jilei Tian. 2014. Mobile app classification with enriched contextual information. *IEEE Transactions on Mobile Computing* 13, 7 (2014), 1550–1563.

Hengshu Zhu, Enhong Chen, Hui Xiong, Kuifei Yu, Huanhuan Cao, and Jilei Tian. 2015. Mining mobile user preferences for personalized context-aware recommendation. *ACM Transactions on Intelligent Systems and Technology (TIST)* 5, 4 (2015), 58.

Hengshu Zhu, Hui Xiong, Yong Ge, and Enhong Chen. 2014. Mobile app recommendations with security and privacy awareness. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 951–960.