This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2803171, IEEE Transactions on Services Computing

Mashup Recommendation by Regularizing Matrix Factorization with API Co-Invocations

Lina Yao, Xianzhi Wang, Quan Z. Sheng, Boualem Benatallah and Chaoran Huang

Abstract—Mashups are a dominant approach for building data-centric applications, especially mobile applications, in recent years. Since mashups are predominantly based on public data sources and existing APIs, it requires no sophisticated programming knowledge of people to develop mashup applications. The recent prevalence of open APIs and open data sources in the Big Data era has provided new opportunities for mashup development, but at the same time increase the difficulty of selecting the right services for a given mashup task. The API recommendation for mashup differs from traditional service recommendation tasks in lacking the specific QoS information and formal semantic specification of the APIs, which limits the adoption of many existing methods. Although there are a significant number of service recommendation approaches, most of them focus on improving the recommendation accuracy and work pays attention to the diversity of the recommendation results. Another challenge comes from the existence of both explicit and implicit correlations among the different APIs, which are generally neglected by existing recommendation methods. In this paper, we address the above deficiencies of existing approaches by exploring API recommendation for mashups in the reusable composition context, with the goal of helping developers identify the most appropriate APIs for their composition tasks. In particular, we propose a probabilistic matrix factorization approach with implicit correlation regularization to solve the recommendation problem and enhance the recommendation diversity. We conjecture that the co-invocation of APIs in real-world mashups is driven by both the explicit textual similarity and implicit correlations of APIs such as the similarity or the complementary relationship of APIs. We develop a latent variable model to uncover the latent correlations between APIs by analyzing their co-invocation patterns. We further explore the relationships of topics/categories to the proposed approach. We demonstrate the effectiveness of our approach by conducting extensive experiments on a real dataset crawled from ProgrammableWeb.

Index Terms—Recommendation; matrix factorization; mashup; latent variable model

1 Introduction

Web services are a major set of technology enabling automated interactions among distributed and heterogeneous applications and connecting business processes [1], [2]. As the fundamental computing paradigm of Web services, service-oriented computing (SOC) presents computing resources in the form of platform-independent Web services through the Internet. These services can then be discovered, selected, and composed together to construct new applications [3]. Such a service composition approach promises improved development efficiency of Web applications through the reuse of existing services and achieves novel value-added by satisfying the complex application requirements that cannot be satisfied by the previous single services. All those advantages lead to a large number of applications that are developed based on consuming existing Web services [4] [5].

Among these applications, mashups represent a type of lightweight Web applications that compose existing Web services in an agile manner, which helps shorten the development period and enhance the scalability of applications [6], [7]. Different from traditional service composition approaches, mashups are generally data-centric applications that provide user-friendly functionalities and typically designed and developed in an easy-to-accomplish manner via graphical user interfaces. By virtue of the above advantages, mashup services have become popular and gained enormous support from multiple platforms, such as Google Mashup Editor[1], IBM Mashup Center[2], and Yahoo pipes[3]. Until Feb. 2015, there are already been over 12,839 publically accessible Web APIs (i.e., Web services) and over 6,168 Mashups on ProgrammableWeb[4], and their numbers are still increasing [8], [9].

To gain a more intuitive perception of mashup applications, Fig. 1 shows the mashup scenario of a mobile application named *WunderWalk*[5]. Basically, *WunderWalk* enables users to search for places of interest and explore places based on their online social relations in urban settings. It features the combination of the Google Maps API[6] and the Foursquare API[7] to facilitate an effective map-based search. In particular, the Google Maps API provides the basic functionalities related to the maps, such as searching and marking specific locations, while the Foursquare API enables the social capabilities such as check-in services and the sharing of comments and pictures among friends. By involving both the two types of APIs in the same application, *WunderWalk* is able to provide some interesting and powerful utilities to users, such as reviewing the marked locations, viewing friends' check-in records, and sharing pictures associated with some locations with friends.

Despite the huge potentials of combining different APIs, the

- *Lina Yao, Boualem Benatallah, and Chaoran Huang are with the School of Computer Science and Engineering, The University of New South Wales, Sydney, Australia. E-mail: {lina.yao, boualem benatallah, chaoran.huang}@unsw.edu.au.*
- *Xianzhi Wang is with the School of Information Systems, Singapore Management University, Singapore. E-mail: xzwang@smu.edu.sg.*
- *Quan Z. Sheng is with the Department of Computing, Macquarie University, Sydney, Australia. E-mail: michael.sheng@mq.edu.au*

1. https://developers.google.com/mashup-editor/
2. http://www-10.lotus.com/ldd/mashupswiki.nsf
3. https://pipes.yahoo.com/pipes/
4. http://www.programmableweb.com/
5. http://www.wunderwalk.com/
6. https://maps.googleapis.com/maps/api/js
7. https://api.foursquare.com/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2803171, IEEE Transactions on Services Computing

Fig. 1. An example mashup: the *WunderWalk* application integrates the Google Maps API and the Foursquare API to provide mobile social services

large number and diversity of available services on the Web pose critical challenges to developing mashup services, as it is can be extremely difficult in finding appropriate APIs to construct a mashup service given the unprecedentedly large scope of choices in selecting the services. The issue is further deteriorated by the heterogeneous relationships among Web services. For example, some APIs are similar to each other while some others may be dissimilar with the others with respect to both functionalities and Quality of Service (QoS). As a result of the above challenges, it becomes a challenging issue as for how to effectively recommend mashup developers with a set of most relevant and coherent services to accelerate the mashup development process. A recommended service should not only be suitable to the development intent of users but also be compatible with the other services selected for the mashup to ensure an overall effective and high-quality recommendation.

Although active research has been conducted on service recommendation, existing approaches still have the following issues that prevent them from being applicable to many real-world mashup applications:

- The QoS information of real-world APIs, especially the newly emerged ones, are generally incomplete and hard to obtain. Business APIs usually have limited allowable times of free invocations, which prohibit the continuous monitoring of their performance. This makes the QoS-based recommendation methods unreliable and even inapplicable in many cases. Besides, without knowing the functional relevance of APIs with a target mashup, concerns on the QoS is only supplementary.
- The real-world APIs commonly lack formal semantic information. There are hardly explicit input/output specifications or meaningful tags that can be used to compose the APIs effectively. Without such information, it is not only unfeasible to match different API in terms of their interfaces but also difficult to tell their intended purpose of usage.
- Even if the textual descriptions can be used to determine the relevance of APIs to a target mashup, the most relevant APIs are not guaranteed to work well with each other. In fact, if two APIs are very similar, they are more likely to be used as a substitute for each other rather than being selected to be used together in the same mashup task.

Although the recommendation methods based on service content analysis and collaborative filtering techniques can partially resolve the first two challenges, they commonly fail to handle the third challenge.

In this paper, we develop a novel approach to produce high-quality recommendation with better diversity. The approach leverages the advantages of not only the matrix factorization techniques but also the mutual relations of APIs by exploring the API usage history from previous mashups. As a dominant implementation of the collaborative filtering approach, the basic matrix factorization (MF) technique decomposes the Mashup-API interaction matrix into two low-rank matrix approximations and makes recommendations based on the resulting factorized matrices. Since the recommendation is fundamentally based on historical interactions of APIs (e.g., API invocations in historical mashups) [10] [11] [12] [13], its accuracy largely depends on the availability of rich mashup records. In this regard, we propose to incorporate the relations among APIs into matrix factorization to improve the robustness and accuracy of API recommendation. Several recent studies have demonstrated the effectiveness of incorporating additional information into the matrix factorization process. For example, improved recommendation quality is achieved by incorporating users' social relations or location similarity as regularization terms of the MF-based service recommendation in [14], [15]. The philosophy underlying both regularizing social relations or location similarity in the MF-based recommendation is that two entities (e.g., users or locations) should have a smaller distance with respect to their latent features.

Inspired by the success of social regularization in the social recommendation domain, we investigate the impact of incorporating API implicit correlations on the collaborative recommendation of APIs for mashups. In particular, we incorporate the correlations of *API services* as an extra regularization term in the traditional matrix factorization. Instead of defining the explicit correlations (e.g., the content-based similarity of API services) directly, we design a latent variable model to infer implicit API service correlations to induce diverse recommendation, where the latent similarities among APIs are reflected by the co-invocation patterns of APIs in historical mashup records. In particular, the co-invocations matrix provide, for any pair of APIs, the number of mashups that comprise both the APIs; and the regularization follows the generalized *homphily* in social science [16], i.e., the more interactions (i.e., co-invocations in previous mashups) two APIs have, the more similar their features would be in the objective function. Under such philosophy, the feature vectors of API services, i and j, should be more similar than the features of another pair of API services, i and k, if the former pair is co-invoked more frequently by the previous mashups than the latter pair.

In a nutshell, our contributions are summarized as follows:

- We investigate the problem of Web-based services (or API) recommendation for mashup under a regularized matrix factorization framework, where the API-mashup matrix is decomposed into two low-dimensional matrices, namely the *API latent subspace* and the *mashup latent subspace* for the effective recommendation.
- We explore the co-invocation patterns between APIs and propose a latent variable model to capture the implicit correlations of APIs. Such implicit information is used to make better API recommendation. Extensive experimental

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2803171, IEEE Transactions on Services Computing

results demonstrate the advantages taking this clue into account in boosting the recommendation performance.
- We crawl a collection of real mashup datasets from the Web and conduct extensive experiments to validate the proposed approach. The experimental results show the better performance of our approach. We also publicly release our mashup dataset to the community for future study.

The rest of this paper is organized as follows. Section 2 provides some background knowledge about API invocation in the real-world mashups and the matrix factorization method. Section 3 first gives an overview of our proposed approach, which describes the derivation of implicit pairwise API correlations, and then elaborates the model built to estimate the latent relations of APIs and presents the learning process of this model. Section 4 reports empirical studies on proposed approach and discusses the results. Section 5 gives a case study based on real-world APIs and mashups. Section 6 overviews related work and Section 7 concludes the paper.

2 PRELIMINARIES

In this section, we provide some motivation and background knowledge of our approach. We first investigate the characteristics of API invocations in real-world mashup applications to motivate the adoption of the matrix factorization based recommendation approach and then give a brief description to the latent class model and matrix factorization based recommendation approach.

2.1 Empirical Study

We investigate a dominant website, ProgrammableWeb[8], which has information about a large collection of APIs as well as Mashups built on these APIs.

To gain an insight of API invocations in real-world mashups, we crawled two types of entities from ProgrammableWeb—11,101 public Web-based APIs and 5,658 mashups. Each Mashup is described by a unique URL, a short description, a list of APIs invoked by it, a group of tags, and a series of categories to which it belongs. Some mashups also have comments and followers information. Similarly, each API also has information such as a short description, provider information, categories, endpoints, and authentication modes. Table 1 shows some basic statistics of the dataset.

TABLE 1
Statistics of ProgrammableWeb dataset

Data Type	Statistics
Number of API	11,101
Number of Mashup	5,658
Size of Service Corpus	25,256
Mashup-API (MA) Composition Matrix Density	1.8811×10^{-4}
API-API (AA) Mutual Matrix Density	1.6397×10^{-4}

We performed further statistical analysis of the dataset, which reveals the following characteristics of API invocations in real-world mashups:

- **Sparsity.** Although numerous services are available on the Web, most of them occur in mashup applications very sparsely. For example, Figure 2 (a) shows the mapping relation between mashups and APIs in form of a matrix, where each entry in the API-Mashup matrix indicates the occurrence of an API in a specific mashup. The results show an averagely very low rate of participation of an ordinary API in existing mashups and most APIs are never used in any mashups. This phenomenon is demonstrated by a low average density of the API-Mashup matrix of approximately 1.8×10^{-4}. In addition, Figure 2 (b) shows that a mashup usually only includes a very limited number of APIs. More specifically, 90% of the mashups include fewer than 5 APIs.
- **Imbalance.** The investigation shows usage imbalanced frequency of involvement of the API services in existing mashups. A small portion is used very frequently, while on the contrary, the majority of APIs are rarely used. Figure 2 (c) shows more detailed results, where the invocation frequency of every API is depicted. The results show that some popular APIs are used more than 2,000 times in total; while in contrast, many unpopular APIs are only included in no more than 10 mashups. Figure 2 (d) shows a closer insight of the API invocations. In this investigation, we have sorted all the API pairs in decedent order of their invocation frequency, with the x-axis showing the accumulative invocation counts of all the API pairs with respect to a growing number of API pairs considered from the top of the sorted list. The results show that the top 200 frequent API co-invocations already cover 99% of all API co-invocations by mashups.

Since the matrix factorization methods have been proved successful [10] [11] [12] in addressing both the imbalance and sparsity issues that characterize the aforementioned API invocation in existing mashups, we develop our approach to the API recommendation for mashups based on the matrix factorization techniques, to address the above challenges.

2.2 Latent Semantic Analysis

To uncover the hidden relations of APIs from previous Mashup invocations, we develop a latent variable model following the latent semantic analysis model scheme. In particular, we perform the probabilistic latent semantic analysis (PLSA) [17] to extract characteristics from API-Mashup interactions. Generally, PLSA considers a set of random latent variables such as topics that are associated with the observed variables based on the term co-occurrences in documents and then describes the generative process for each of the N documents in the collection. There are two key assumptions in applying this model: i) each document can be represented by an unsorted word collection, meaning the joint distribution probability of both words and documents are assumed to result from independently sampling; and ii) words and documents are conditionally independent given the latent topics.

2.3 Matrix Factorization

Matrix factorization is one of the most successful implementations of latent factor models and has been used widely in recent years owing to its exceptional scalability and predictive accuracy. Also, it offers the flexibility of modeling *dyadic* various real-life situations. For the above reasons, the collaborative filtering recommendation approach based on matrix factorization is the most widely

8. http://www.programmableweb.com/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2803171, IEEE Transactions on Services Computing

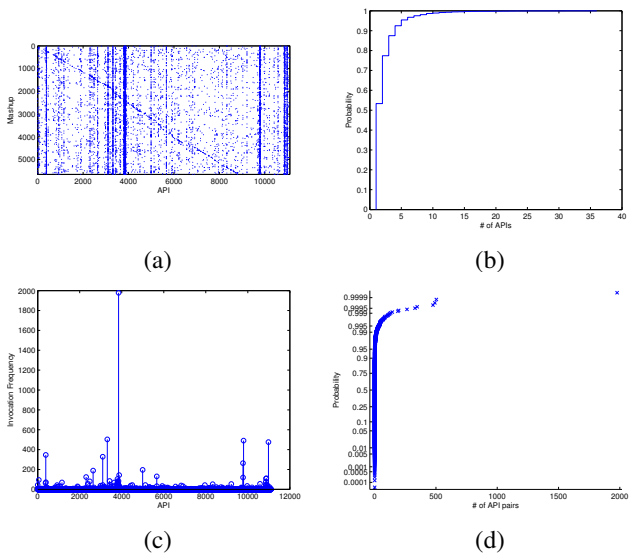


Fig. 2. (a) API-Mashup Matrix is highly sparse; (b) 90% mashup integrate less than 5 APIs; (c) The illustration of API invocation imbalance: infrequent APIS are only used less than 10 times, while frequent API over 2000 times; (d) The illustration of API invocation imbalance: top 200 most frequent APIs almost occupy 99% of all invocations between API and mashups

used recommendation model [10]. Given that historical data are often placed in a two-dimensional matrix, with one dimension representing users and the other dimension representing items of interest, matrix factorization characterizes both items and users by vectors of factors inferred from the interaction patterns between users and items implied by the item-user matrix. Generally, a higher correspondence between an item and user factors indicates a more competitive recommendation.

Given two sets of entities (e.g., Mashups and APIs) $i \in \mathbf{I}$ and $j \in \mathbf{J}$, which interact with each other with a response $y_{ij} \in \mathbf{Y}$, the interaction matrix between the two kinds of entities is represented as:

$$\{(i,j) \rightarrow y_{ij} | i \in \mathbf{I} \; and \; j \in \mathbf{J}\} \tag{1}$$

where $\mathbf{Y} \in \mathbb{R}^{|\mathcal{I}| \times |\mathcal{J}|}$, only a part of which is known from previous observation. The basic idea of matrix factorization is to interpret $\mathbf{Y}$ into two low-rank matrices that share the same latent space to characterize the two entities, respectively. The recommendation can be then performed by inferring and ranking the missing values of entries based on the low-rank matrices derived from the values of observable entries.

3 Proposed Approach

In this section, we first briefly describe the API recommendation problem, and then introduce our proposed method in detail.

3.1 Problem Description

We define API recommendation as the problem of suggesting a set of promising APIs for a target mashup, given the invocation history of APIs in previous mashups. Specifically, in this study, we consider both the textual similarity and interactive patterns between APIs to make recommendations based on the matrix factorization approach, where textual similarity measures the word-based similarity of APIs' descriptive profiles and interactive patterns describe the latent complementary relations that are explored from the co-invocation records of APIs. In particular, the textual similarity is used to grasp the domain characteristics of APIs (e.g., Google Map API and Gas Station locations are likely to use some common words to indicate that they are both related to locations); while the involvement of co-invocation history can address the deficits of pure similarity based recommendation approaches, that it can only promote services sharing certain characteristics yet not able to suggest services complementing each other. Specifically, we investigate the co-invocation patterns of services to infer the implicit functional correlations between services and incorporate this correlation into the matrix factorization model as a regulation term to solve the recommendation problem. In this paper, we break down to the following two questions:

Problem 1 (Implicit Relation Derivation). Given a set of APIs and their invocation history in a set of Mashups, how to uncover their underlying connections?

Problem 2 (API Suggestion for Mashup). Given a target Mashup, how to find the most appropriate APIs to construct this mashup?

Suppose we have invocation records of n APIs in k mashups. We denote invocation relation between APIs and mashups by a matrix $\mathbf{R} \in \mathbb{R}^{n \times k}$, where each element r_{ij} indicates whether or not an API represented by $\mathbf{a}_i$ is invoked by a mashup represented by m_j (true if $r_{ij} = 1$). The primary idea of matrix factorization is to map the mashups (resp., APIs) into a shared lower dimensional space (the new dimensionality is $d \ll \min\{n, k\}$). Based on the factorization results of mashups $\mathbf{a}_i \in \mathbb{R}^d$ and the factorization results of APIs $\mathbf{m}_j \in \mathbb{R}^d$, the probability that $\mathbf{a}_i$ would be invoked by $\mathbf{m}_j$ is estimated by:

$$\hat{r}_{ij} = \mathbf{a}_i^T \mathbf{m}_j \tag{2}$$

Suppose the latent factors of mashups and APIs are both denoted as matrices, say $\mathbf{A} \in \mathbb{R}^{n \times d}$ and $\mathbf{M} \in \mathbb{R}^{k \times d}$, they can be learned from minimizing the ℓ_2 loss:

$$\min_{\mathbf{A},\mathbf{M}} \frac{1}{2} \sum_{i,j} I_{ij} (r_{ij} - \hat{r}_{ij})^2 + \frac{\lambda_{\mathbf{A}}}{2} ||\mathbf{A}||_F^2 + \frac{\lambda_{\mathbf{M}}}{2} ||\mathbf{M}||_F^2 \tag{3}$$

where I_{ij} equals 1 if API $\mathbf{a}_i$ is invoked by mashup $\mathbf{m}_j$, and 0 otherwise. $||\cdot||_F$ is the Frobenius norm of a matrix, $\lambda_{\mathbf{A}}$ and $\lambda_{\mathbf{M}}$ are the regularization parameters. For the simplicity of parameter tuning, we simply set $\lambda_{\mathbf{A}} = \lambda_{\mathbf{M}}$.

The following sections will first introduce the overall workflow of the proposed method, and show the formulation of the latent variable model used for inferring the implicit correlations of API services from historical API co-invocation interactions and the computation of API service textual similarity, followed by a description of the model inference process and collaborative filtering mashup recommendations.

3.2 Architectural Framework

Fig. 3 illustrates the framework of the proposed approach that leverages the implicit correlation between APIs for recommending APIs for mashups. In particular, we calculate the explicit textual similarity of APIs based on the textual specification of APIs (e.g., name, description) crawled from ProgrammableWeb and collect information about the co-invocations of APIs in previous mashups by scrawling existing mashup from ProgrammableWeb. The approach builds on a service recommendation process that involves two stages. First, it establishes the basic matrix factorization-based

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2803171, IEEE Transactions on Services Computing

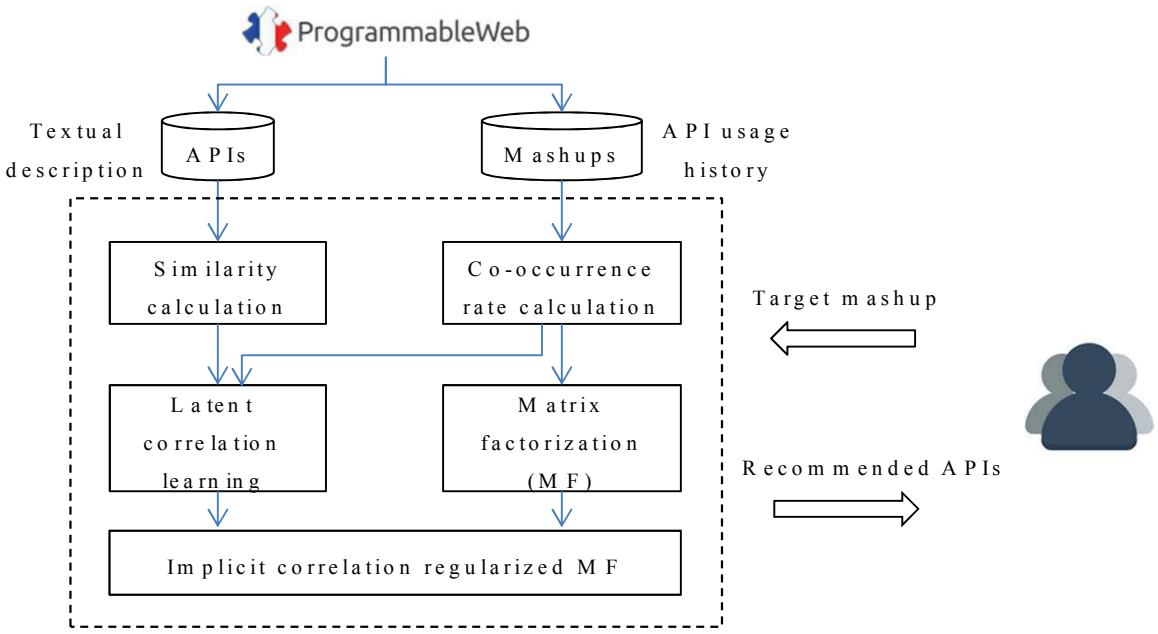


Fig. 3. The architectural framework of the proposed approach.

recommendation based on the mapping frequencies in existing mashups. Second, it calculates the latent implicit correlation between APIs by taking into account both the explicit similarity of APIs and the co-occurrence patterns of APIs in existing mashups. In particular, the implicit correlation is modeled and learned using a latent variable model. The implicit correlation information is then incorporated into the matrix factorization approach as a regularization factor to improve the recommendation accuracy. Once the approach is developed, given a target mashup, a ranked list of promising APIs will be returned by the approach to users as the recommendation results.

3.3 Implicit Relation Learning

The recommended APIs are expected to be both relevant to the target mashup as well as strongly correlated with each other: first, there are often a large number of available APIs in practical scenarios, which makes it important to identify and study only on the APIs that are relevant to the target domain of application; second, there are usually some implicit ingredients driving the co-use of APIs in mashup services, such as the compatibility or complementary relations among APIs. Such latent relationship cannot be directly observed or obtained from APIs' descriptive profiles (e.g., category information and API description). We model these latent correlations as a hidden factor of descriptive similarities indicated by API profiles. In particular, in our model, we compute the functional similarity of services based on the textual profile of services and regard the co-invocation history as observations (or results) for a generative method. In particular, we adopt the TF-IDF calculation to identify the term weights in each service profile to characterize the service, followed by comparing two services on their weightings to the terms, which are two vectors, to produce their textual similarity. Intuitively, more co-invocations indicate stronger connections among pairwise APIs. For this reason, we assume the latent correlation between APIs directly impacts the nature and frequency of API co-invocations in a mashup process, i.e., the stronger the correlation, the higher likelihood that co-invocations will take place between a pair of APIs in the same mashups.

Let $\mathbf{a}_i \in \mathcal{R}^m$ be the descriptive vectors of a API service i, y_{ij} be the counts of cases that API services i and j are co-invoked by the same mashups, and z_{ij} be the pairwise implicit correlations of API services i and j. We predict the implicit relations by adapting the statistical mixture model proposed in [17], [18] (as shown in Figure 4):

$$\begin{aligned} Pr(z_{ij}, y_{ij}|\mathbf{a}_i, \mathbf{a}_j) &= \underbrace{Pr(z_{ij}|\mathbf{a}_i, \mathbf{a}_j)}_{①} \underbrace{Pr(y_{ij}|z_{ij})}_{②} \\ &= Pr(z_{ij}|s_{ij}) Pr(y_{ij}|z_{ij}) \end{aligned} \tag{4}$$

where s_{ij} and z_{ij} are the explicit similarity (calculated from their descriptive vectors) and the latent relation of API services i and j, respectively. To obtain the latent relation z_{ij}, we need to specify two types of dependencies and solve Eq. 4: i) the dependency between the pairwise explicit similarity of API services s_{ij} and latent relations $Pr(z_{ij}|s_{ij})$; ii) the dependency between the co-invocation patterns of each pair of API services y_{ij} and latent relations $Pr(y_{ij}|z_{ij})$. We will describe the derivation process in detail in the following subsections.

3.3.1 *Specifying* $Pr(z_{ij}|s_{ij})$

To measure the dependency between explicit similarity and implicit similarity of API services, We denote each API service file $\mathbf{a}_i \in \mathbb{R}^m$ using a TF/IDF weighting scheme and extract the terms and calculate the corresponding $tf \times idf$ scores to form its textual profile, which is a vector of $tf \times idf$ scores $\mathbf{a}_i = [w_1, w_2, ..., w_n]$). We then use cosine similarity to quantify the similarity given a pair of API services. The processing pipeline is described as follows:

- Keywords corpus construction. This step builds a keyword repository of services by segmenting the service descriptions into terms, removing those terms with little meaning such as 'a', 'the', and 'to', and eliminating the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2803171, IEEE Transactions on Services Computing

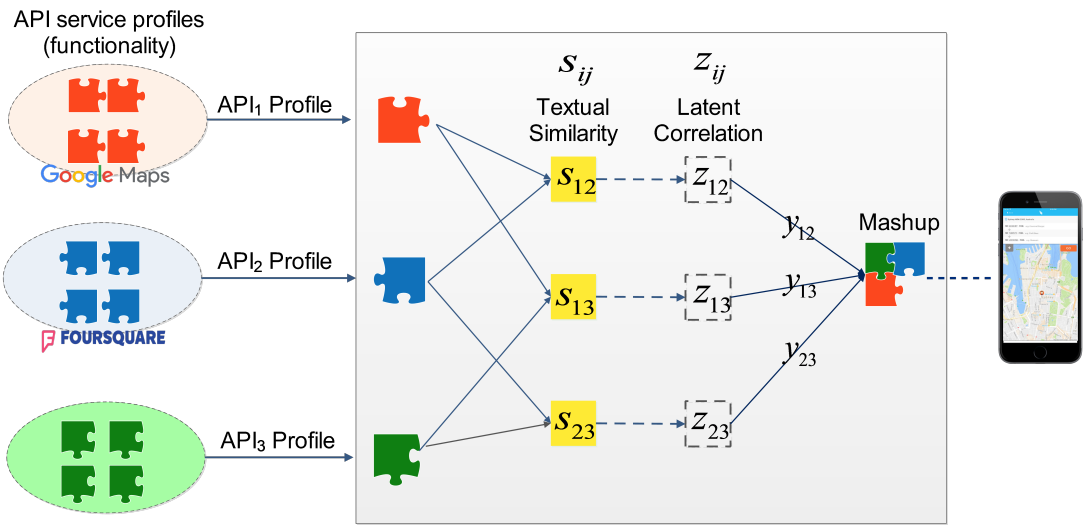


Fig. 4. The illustration of implicit correlation derivation model: s_{ij} denotes the explicit similarity between API services i and j, z_{ij} denotes their implicit relations between i and j, and the y_{ij} denotes the interaction between i and j, e.g., the frequency of service i and j engage in a same Mashup service.

differences among the words with prefixes (e.g., -in, -un, -dis, -non, et al) or suffixes (e.g., -s, -es, -ed, -er, or,-ing, -ion, et al). As an example, the words "automate(s)", "automatic", "automation" should all stem and lemmatize to "automate". The result is a service corpus containing generic descriptive keywords with the size $k \approx 14,000$, denoted by $c_i \in \mathcal{C}$.

- Term frequency calculation. We use tf (term frequency), the count of a term's occurrence in a given service description, to measure the importance of the term. The count is normalized to prevent bias towards longer documents (which tends to have a higher term count regardless of the actual importance of that term in the document). The calculation of tf is illustrated by:

$$tf(c_{ij}) = \frac{freq(c_{ij}, x_i)}{|x_i|} \tag{5}$$

where $tf(c_{ij})$ is the frequency of the occurrence of the j^{th} term c_j in the description of API service $x_i \in \mathcal{X}$.

- Inverse document frequency calculation. idf (inverse document frequency) measures the importance of a term in a set of API service descriptions, which can be calculated using:

$$idf(c_{ij}) = \begin{cases} \log \dfrac{|n|}{\sum_i \mathbb{I}(c_{ij} \in x_i)} & \text{if } \mathbb{I}(\cdot) \neq 0 \\ \log \dfrac{|n|}{1 + \sum_i \mathbb{I}(c_{ij} \in x_i)} & \text{if } \mathbb{I}(\cdot) = 0 \end{cases} \tag{6}$$

where $|n|$ is the number of services, $\mathbb{I}(\cdot)$ is the number of service descriptions in which term c_j appears.

Based on above defintions, we define $a_{ij} = tf \times idf^2$, where the idf value is assigned a higher weight to neutralize the bias brought by the tf measure in the widely existing short descriptions of most services [19]. To this point, each API service can be represented as a descriptive vector $\mathbf{a}_i$, and we can continue to compute the explicit similarity s_{ij} between API services i and j using the cosine similarity of their corresponding descriptive vectors as follows:

$$s_{ij} = \frac{\mathbf{a}_i \cdot \mathbf{a}_j}{||\mathbf{a}_i|| ||\mathbf{a}_j||} \tag{7}$$

Based on the above calculation, the first dependency in Eq. 4 can be specified:

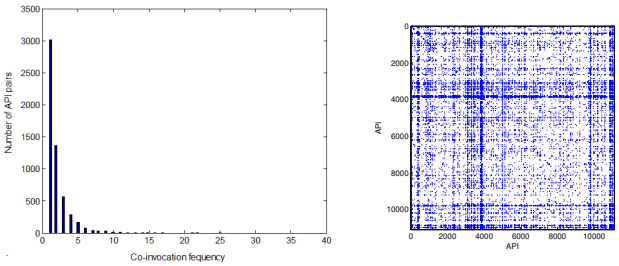
$$\begin{aligned} Pr(z_{ij}|\mathbf{a}_i, \mathbf{a}_j) &= Pr(z_{ij}|s_{ij}, \mathbf{w}) \\ &= \mathcal{N}(w_{ij}s_{ij}, \sigma^2) \\ &= \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z_{ij} - w_{ij}s_{ij})^2}{2\sigma^2}} \end{aligned} \tag{8}$$

Specially, in above calculation, we have assumed the relationship between z_{ij} and s_{ij} to follow a zero-mean Gaussian distribution $z_{ij} = w_{ij}s_{ij} + \epsilon$, $\epsilon \sim N(0, \sigma^2)$, where $w_{ij} \in \mathbf{w}$ denotes a weight vector to be estimated and associated with the similarity of each pair of APIs, σ^2 is the variance in Gaussian model.

3.3.2 *Specify* $Pr(y_{ij}|z_{ij})$

To infer the co-invocation of APIs, we introduce an additional layer, *the latent relations of APIs*. This is based on the insight that higher frequency of co-invocation does not necessarily imply higher textual similarity of APIs. For example, two functionally complementary APIs may have low textual similarity but be frequently used together in mashups. Clearly, such relations cannot be captured well by the textual features. To tackle this problem, we represent such *unobservable* relations as latent variables that measure the implicit similarity of APIs and specify those relations by studying the co-invocation behaviors of APIs in existing mashups. Figure 5 shows the co-invocations for pairwise API services, where Figure 5 (a) shows the distribution of co-invocation counts for all pairs of API services and Figure 5 (b) shows the specific co-invocation count for each pair of API services in mashups. As Poisson distributions can naturally model stochastic counting data,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2803171, IEEE Transactions on Services Computing



(a) (b)

Fig. 5. Illustrations of the co-invocation behaviors for API services

we use a *Poisson* distribution with a small parameter λ to model the times of co-invocations of each pair of API services, given the influence of their latent correlations z_{ij}:

$$Pr(y_{ij}|z_{ij}) = \frac{\lambda^{y_{ij}} e^{-\lambda}}{y_{ij}!} = \frac{(\theta z_{ij})^{y_{ij}} \cdot e^{-\theta z_{ij}}}{y_{ij}!} \tag{9}$$

where $\lambda = \theta_i z_{ij}$, $\theta_i \in \theta$ is the weight vector, and y_{ij} is the count that $\mathbf{a}_i$ and $\mathbf{a}_j$ are engaged in same mashups.

3.3.3 Model Learning

Given a training dataset $\mathcal{D}$ (denoted by n API pairs), the explicit similarity of services s_{ij} (Section 3.3.1) and the co-invocation of services y_{ij} (Section 3.3.2), the likelihood function of our latent variable model can be specified below:

$$\begin{aligned} & Pr(\mathcal{D}|\mathbf{w},\theta)Pr(\mathbf{w})Pr(\theta) \\ &= \prod_{(i,j)\in\mathcal{D}} \Big(Pr(z_{ij}, y_{ij}|s_{ij}, \mathbf{w}, \theta) \Big) Pr(\mathbf{w})Pr(\theta) \\ &= \prod_{(i,j)\in\mathcal{D}} \Big(Pr(z_{ij}|s_{ij}, \mathbf{w}) Pr(y_{ij}|z_{ij}, \theta) \Big) Pr(\mathbf{w})Pr(\theta) \end{aligned} \tag{10}$$

Specially, to avoid overfitting, we put a ℓ_2 regularization on the both parameters $\mathbf{w}$ and θ, i.e., $Pr(w) \propto e^{-\frac{\lambda_w}{2}\mathbf{w}^T\mathbf{w}}$ and $Pr(\theta) \propto e^{-\frac{\lambda_\theta}{2}\theta^T\theta}$. We now have

$$\begin{aligned} & \prod_{(i,j)\in\mathcal{D}} \Big(Pr(z_{ij}|s_{ij}, \mathbf{w}) Pr(y_{ij}|z_{ij}, \theta) \Big) Pr(\mathbf{w})Pr(\theta) \\ & \propto \prod_{(i,j)\in\mathcal{D}} \Big(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z_{ij} - w_{ij}s_{ij})^2}{2\sigma^2}} \cdot (\theta z_{ij})^{y_{ij}} \cdot e^{-\theta z_{ij}} \Big) \\ & \quad \cdot e^{-\frac{\lambda_w}{2}\mathbf{w}^T w} \cdot e^{-\frac{\lambda_\theta}{2}\theta^T\theta} \end{aligned} \tag{11}$$

Besides, we take the logarithm of Equation 10 and adopt a stochastic gradient-based method to optimize the model parameters ($\mathbf{w}$ and θ) and the latent variables z_{ij}, with the goal of maximizing this logarithm function.

$$\begin{aligned} \mathcal{L}(z_{ij\in\mathcal{D}}, \mathbf{w}, \theta) = \sum_{(i,j)\in\mathcal{D}} \Big(& -\frac{1}{2\sigma^2}(z_{ij} - w_{ij}s_{ij})^2 \\ & + y_{ij}\ln(\theta z_{ij}) - \theta z_{ij} \Big) - \frac{\lambda_w}{2}\mathbf{w}^T\mathbf{w} - \frac{\lambda_\theta}{2}\theta^T\theta \end{aligned} \tag{12}$$

In particular, both the two types of parameters (model parameters $\mathbf{w}$ and θ, and the latent variable z_{ij}) are learned using a generic coordinate ascent method, which conducts optimization by updating one type of parameters while fixing the other type during this step. The whole process runs iteratively until reaching certain convergence threshold.

$$\begin{aligned} \frac{\partial\mathcal{L}}{\partial z_{ij}} &= \frac{1}{\sigma}\sum_{(ij)\in\mathcal{D}} \Big((w_{ij}s_{ij} - z_{ij}) + \frac{y_{ij}}{z_{ij}} \Big) \\ \frac{\partial\mathcal{L}}{\partial w_{ij}} &= \frac{1}{\sigma}\sum_{(i,j)\in\mathcal{D}} \Big((w_{ij}s_{ij} - z_{ij})s_{ij} \Big) - \lambda_w w_{ij} \\ \frac{\partial\mathcal{L}}{\partial \theta} &= \sum_{(i,j)\in\mathcal{D}} \Big(\frac{1}{\theta} y_{ij} - z_{ij} \Big) - \lambda_\theta \theta \end{aligned} \tag{13}$$

More specifically, $\mathbf{w}$ can be calculated as:

$$\mathbf{w}^{new} = (\lambda_w \mathbf{I} + \mathbf{S}^T\mathbf{S})^{-1}\mathbf{S}^T\mathbf{Z} \tag{14}$$

where $\mathbf{S} = [\mathbf{s}_{i_1,j_1}, ..., \mathbf{s}_{i_n,j_n}]^T$ and $\mathbf{Z}^T = [\mathbf{z}_{i_1,j_1}, ..., \mathbf{z}_{i_n,j_n}]^T$ Both the parameter θ and the latent variable z_{ij} can be solved by iteratively updating their values until convergence using the Newton method (Eq. 15 and Eq. 16, respectively).

$$\begin{aligned} z_{ij}^{new} &= z_{ij}^{old} - \frac{\partial\mathcal{L}}{\partial z_{ij}} \Big/ \frac{\partial^2\mathcal{L}}{\partial (z_{ij})^2} \\ \text{where } & \frac{\partial^2\mathcal{L}}{\partial (z_{ij})^2} = \frac{1}{\sigma^2}\sum_{(i,j)\in\mathcal{D}} \Big(1 + \frac{y_{ij}}{z_{ij}^2} \Big) \end{aligned} \tag{15}$$

$$\begin{aligned} \theta_{ij}^{new} &= \theta_{ij}^{old} - \frac{\partial\mathcal{L}}{\partial \theta_{ij}} \Big/ \frac{\partial^2\mathcal{L}}{\partial (\theta_{ij})^2} \\ \text{where } & \frac{\partial^2\mathcal{L}}{\partial (\theta_{ij})^2} = -\sum_{(i,j)\in\mathcal{D}} (y_{ij}\frac{1}{\theta^2}) \end{aligned} \tag{16}$$

Specially, the log likelihood function in Equation 12 is globally concave as the Hessian matrix $\nabla^2\mathcal{L}(\theta)$ is negative semidefinite. Therefore, Eq. 16 satisfies $-\sum_{(i,j)\in\mathcal{D}}(y_{ij}\frac{1}{\theta^2}) \leq 0$. After obtaining the model parameters, given any API service i, we can extract its co-invocation patterns with another API service j, say y_{ij}, by calculating its descriptive vector $\mathbf{a}_i$ and its textual similarity s_{ij} with other training APIs.

3.4 Mashup Recommendation with Integrated with Implicit Correlations

One important advantage of matrix factorization is that it allows incorporation of additional information. Therefore, after inferring the relations between APIs, we can easily integrate them into a general matrix factorization framework. Generally, a recommender system can leverage either explicit or implicit feedback to infer user preferences over items. While the implicit feedback indirectly reflects opinions by observing user's previous behaviors, it usually denotes the presence or absence of an event and is thus typically represented by a densely filled matrix. Specifically for the API recommendation problem for mashups, mashups are regarded as users and APIs correspond to items. Despite of the consideration of both explicit and implicit clues (in terms of textual information and co-invocations between APIs) in our approach, the inference of implicit correlations conducted by the previous step enables us to interpret all those clues as implicit

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2803171, IEEE Transactions on Services Computing

correlations of APIs and to conduct recommendation solely based on such implicit correlations learned from the known invocation history.

In particular, to leverage the implicit relations between APIs for the API recommendation, we use those relations to regularize the latent spaces of APIs and Mashups when incorporate them into the general matrix factorization framework (Equation 3).

$$L = \min_{\mathbf{A},\mathbf{M}} \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{k} I_{ij}(r_{ij} - \mathbf{a}_i^T\mathbf{m}_j)^2 + \frac{\lambda_{\mathbf{A}}}{2}||\mathbf{A}||^2 + \frac{\lambda_{\mathbf{M}}}{2}||\mathbf{M}||^2 + \overbrace{\frac{\alpha}{2}\sum_{i=1}^{n}\sum_{b=1}^{n}\mathbf{Z}_{ib}||\mathbf{a}_i - \mathbf{a}_b||^2}^{1} \tag{17}$$

where the last term (part 1) of Eq.17 integrates the link information of APIs, $\mathbf{Z}$ indicates the pairwise latent relations of APIs (as described in Section 3.3), and the regularization terms $||\mathbf{A}||^2$ and $||\mathbf{M}||^2$ control the complexity of model. The intuition of incorporating the regularization term $\mathbf{Z}_{ib}||\mathbf{a}_i - \mathbf{a}_b||^2$ is to make the latent representations of the implicitly connected APIs as close as possible.

Eq. 17 can be easily solved by coordinating optimization methods, which alternately fix one variable ($\mathbf{A}$ or $\mathbf{M}$) and optimize the other by using gradient updating rules to progressively find a local minimum [20] The updating rules for the two variables are shown in Eq.18 and Eq.19, respectively.

$$\mathbf{a}_i \leftarrow \mathbf{a}_i + \eta_1(\delta_{ij}\mathbf{m}_j - \alpha\sum_{b\in\mathcal{N}_i} z_{ib}(\mathbf{a}_i - \mathbf{a}_b)) \tag{18}$$

$$\mathbf{m}_j \leftarrow \mathbf{m}_j + \eta_2(\delta_{ij}\mathbf{a}_i - \lambda_2\mathbf{m}_j) \tag{19}$$

Based on the results of above calculation, we can predict the possibility of an API being invoked by a target mashup by calculating the score using Eq. 2.

4 Experiments

In this section, we introduce the experimental settings and analysis of our experimental studies to evaluate the proposed approach. The experiments focus on the following aspects: i) comparing the performance of our proposed approach and the representative baseline methods, ii) studying the impact of the proposed implicit correlations on the recommendation results, and iii) evaluating the impact of different parameter settings (dimensionality and regularization) on the performance of our approach.

4.1 Experimental Settings

We use the dataset crawled from the ProgrammableWeb (introduced in Section 2.1) for the experiments. In the following, we briefly describe the validation strategy and performance metrics used in this work.

4.1.1 Validation Strategy

We construct the training set and the testing set as follows: firstly, we randomly select 20% API-Mashup pairs as the testing set and randomly split the rest data into 8 parts, each containing 10% API-Mashup pairs. These parts are added incrementally to the training set to represent different sparsity levels. Table 2 shows in detail the statistics of the testing set and the training sets of different sparsity levels, where 10% means the training set contains 10% API-Mashup pairs, 20% means it contains 20% API-Mashup pairs, and so on.

TABLE 2
Training Data with Different Sparsity Levels

Data	Density
10%	1.8604×10^{-4}
20%	1.8671×10^{-4}
30%	1.8712×10^{-4}
40%	1.8751×10^{-4}
50%	1.8774×10^{-4}
TestingData	1.8774×10^{-4}

4.1.2 Performance Metrics

We adopt four commonly used metrics, namely Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE), to measure the recommendation performance. For the both metrics, smaller values indicate better performance.

$$MAE = \frac{1}{N}\sum_{ij}|r_{ij} - \hat{r}_{ij}| \tag{20}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{ij}(r_{ij} - \hat{r}_{ij})^2} \tag{21}$$

To evaluate the quality of the proposed method in recommending appropriate APIs for a given mashup, we also use two other metrics, $precision@x$ and $recall@x$, to evaluate our proposed method on the testing data. Specifically, $precision@x$ measures how many previously marked off APIs are selected to the mashups among the total number of recommended APIs, and $recall@x$ measures how many previously marked off APIs are selected to the mashups among the total number of marked off APIs. The two metric are computed as follows:

$$precision@x = \frac{\sum_{\mathbf{a}_i\in\mathbf{A}}|Res(a_i)\cap Rec(\mathbf{a}_i)|}{\sum_{\mathbf{a}_i\in\mathbf{A}}|Res(\mathbf{a}_i)|} \tag{22}$$

$$recall@x = \frac{\sum_{\mathbf{a}_i\in\mathbf{A}}|Res(\mathbf{a}_i)\cap Rec(\mathbf{a}_i)|}{\sum_{\mathbf{a}_i\in\mathbf{A}}|Rec(\mathbf{a}_i)|} \tag{23}$$

where $Res(\mathbf{a}_i)$ denotes the set of corresponding invoked APIs in the testing dataset for given mashup $\mathbf{m}_i$, and $Rec(u_i)$ denotes the set of suggested APIs by proposed method for given mashup $\mathbf{m}$. By studying the mashup services in the experimental dataset, we find most of them have one to five APIs. For this reason, we set x= 1, 2, 3, 4, 5.

4.2 Results

We present the experimental results on the performance of our proposed model in the following aspects:

- How the proposed model performs when compared to the state-of-the-art methods over the experimental dataset (Section 4.2.1).
- Whether implicit correlations derived from API co-invocations can improve the recommendation performance (Section 4.2.2).
- What is the impact of key parameters of the proposed model (Sections 4.2.4 and 4.2.3).
- What are the insights on applications and possible extensions of the proposed model (Section 4.3).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2803171, IEEE Transactions on Services Computing

4.2.1 Comparison with Other Approaches

In this section, we compare our proposed approach with the following approaches under different sparsity levels (shown in Table 2. The textual similarity of the baseline approaches is calculated by using two similarity functions with same API descriptive vectors (description in Section III.A). The baseline methods are briefly depicted as follows:

- API-based Neighborhood (**AN**). This method uses Pearson Correlation to calculate the similarity between APIs and makes recommendation solely based on such similarity [21].
- Mashup-based Neighborhood (**MN**). This method also uses Pearson Correlation to compute the similarity between mashups but predicts invocations based on similar mashups.
- Non-negative Matrix Factorization (**NMF [22], [23]**). The method applies non-negative matrix factorization on API-Mashup matrix to predict the missing invocations. The invocation matrix between API and mashup $\mathbf{R}$ can be decomposed into two lower dimension matrices

$$\mathbf{R}_{ij} = \mathbf{A}_{ik}\mathbf{M}_{kj} \tag{24}$$

The model can be solved by the following optimization process

$$\min_{\mathbf{A},\mathbf{M}} \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{k}(r_{ij} - \mathbf{a}_i^T\mathbf{m}_j)^2 \tag{25}$$

- Regularized NMF (**RNMF**). This method imposes two regularizations of $\mathbf{a}$ and $\mathbf{m}$ to avoid overfitting, which is formulated as:

$$\min_{\mathbf{A},\mathbf{M}} \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{k}(r_{ij} - \mathbf{a}_i^T\mathbf{m}_j)^2 + \frac{\lambda_{\mathbf{A}}}{2}||\mathbf{a}||_F^2 + \frac{\lambda_{\mathbf{M}}}{2}||\mathbf{m}||_F^2 \tag{26}$$

- Probabilistic Matrix Factorization (**PMF**). This is one of the most famous MF models in collaborative filtering [12]. It assumes a Gaussian distribution on the residual noise of the observed data and places Gaussian priors on the latent matrices U and V. The objective function of PMF for the frequency data is defined as follows:

$$\min_{\mathbf{A},\mathbf{M}} \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{k}(g(r_{ij}) - g(\mathbf{a}_i^T\mathbf{m}_j)^2) + \frac{\lambda_{\mathbf{A}}}{2}||\mathbf{a}||_F^2 + \frac{\lambda_{\mathbf{M}}}{2}||\mathbf{m}||_F^2 \tag{27}$$

where $g(\cdot) = 1/(1+\exp(-x))$ is the logistic function.

- Content-based Recommendation (**CBR**). This method purely uses semantic information of APIs and mashups, such as the extracted textual similarity of APIs and tentative Mashups from their profile, and does not consider any API invocation history in making recommendations.
- Ours-bin. Instead of using Poisson distribution to model the co-occurrence of APIs, another option of our approach is to adopt a sigmoid function, in which $y_{ij} = 1$ if two APIs appear in a same mashup, and 0 otherwise.

$$Pr(y_{ij} = 1|z_{ij}) = \frac{1}{1 + e^{\theta_{ij}z_{ij}}} \tag{28}$$

TABLE 3
MAE and RMSE Performance Comparison

		10%	20%	30%	40%	50%
AN	**MAE**	0.2259	0.2135	0.2005	0.2002	0.1994
	RMSE	0.3646	0.3572	0.3469	0.3325	0.3253
MN	**MAE**	0.2284	0.2140	0.2067	0.2021	0.2003
	RMSE	0.3705	0.3525	0.3510	0.3497	0.3438
NMF	**MAE**	0.2413	0.2377	0.2215	0.2142	0.2123
	RMSE	0.3892	0.3811	0.3672	0.3511	0.3487
RNMF	**MAE**	0.1524	0.1447	0.1414	0.1392	0.1277
	RMSE	0.3002	0.2784	0.2646	0.2546	0.2452
PMF	**MAE**	0.1189	0.1142	0.1138	0.1119	0.1085
	RMSE	0.2867	0.2634	0.2554	0.2433	0.2325
CBR	**MAE**	0.2432	0.2355	0.2283	0.2280	0.2223
	RMSE	0.3626	0.3611	0.3487	0.3442	0.3329
Ours-bin	**MAE**	**0.1157**	0.1105	0.1082	0.1043	0.1017
	RMSE	**0.2702**	0.2548	0.2394	0.2209	0.2018
Ours	**MAE**	0.1164	**0.1101**	**0.1062**	**0.1033**	**0.1014**
	RMSE	0.2714	**0.2552**	**0.2389**	**0.2205**	**0.2014**

For our approach, we set the dimensionality $d = 15$. For simplicity, we set the same value for the two regularization parameters $\lambda_{\mathbf{A}} = \lambda_{\mathbf{M}} = 0.01$ in the experiments. The impact of different parameter settings is studied in details in the following subsections.

It is worth noting that there are some approaches that combine matrix factorization models with content-based methods for better performance. For this reason, they require rich historical information of mashups. Some newly emerging ones also employ techniques like topic models or clustering techniques to further boost the performance. Although certain improvement can be achieved, as they claim, the improvement is generally incremental.

Table 3 shows that matrix factorization based methods generally achieve better performance than both AN and MN. Specially, our proposed method obtains better accuracy than other matrix factorization methods consistently. The reason lies in that our method integrates the inherent relations among APIs with the historical relations between APIs and mashups. The performance of our approach verifies the advantages of using the implicit relation information in combination with the sparse API-Mashup matrix for improving the recommendation accuracy. Compared to the traditional method of modeling item co-occurrences as binary relations using a sigmoid function, the Poisson distribution produces better results.

For the above reason, we set our default modeling function as Poisson in the following experiments. We also fix the training set as 40% percentage of the whole dataset to run the comparison. Fig. 6 shows the Precision and Recall of different methods, where we observe our proposed method consistently outperforms other methods.

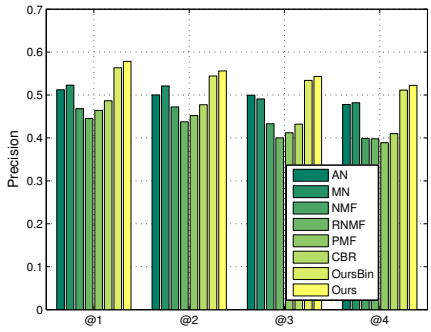
4.2.2 Impact of Implicit Correlations

To evaluate the impact of implicit relations of APIs, we compare them with two explicit similarity measures:

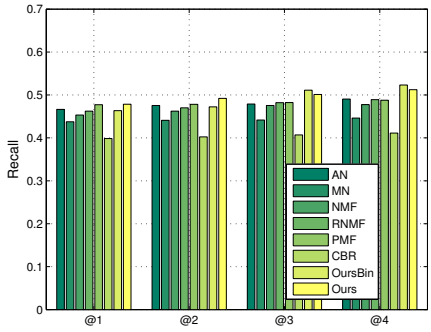
- Cosine similarity. The cosine similarity computes the relations between APIs $\mathbf{a}_i$ and $\mathbf{a}_j$, both denoted as TF/IDF vectors, as follows:

$$sim(\mathbf{a}_i, \mathbf{a}_j) = \frac{<\mathbf{a}_i, \mathbf{a}_j>}{||\mathbf{a}_i||||\mathbf{a}_j||} \tag{29}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2803171, IEEE Transactions on Services Computing



(a)



(b)

Fig. 6. Overall performance comparison (a) Precision x (b) Recall x

- Jaccard similarity. The Jaccard similarity computes the relations between any pair of APIs as:

$$sim(\mathbf{a}_i, \mathbf{a}_j) = \frac{|B_i \cap B_j|}{|B_i \cup B_j|} \tag{30}$$

where both B_i and B_j are binary vectors denoting which mashup sets that $\mathbf{a}_i$ and $\mathbf{a}_j$ participate in, respectively.

Fig. 7 shows that the matrix factorization approach incorporating implicit API correlations outperforms the same approach that incorporates either of the other two similarity-based methods. The reason lies in that implicit relations are derived from both the profiles and co-invocation records of APIs. Such implicit information cannot be easily identified solely from content-based correlations as the implicit connections between APIs can diversify the recommendation results of generic methods. The Jaccard similarity-based factorization achieves better performance than the pure content-based method, as it partially captures the intersections of different APIs in mashup activities.

4.2.3 Impact of Dimensionality

In this section, we study the impact of dimensionality, i.e., the number of latent features used to characterize APIs and mashups, by varying its value from 5 to 50 with a step size of 5. Fig. 8 shows that our approach achieves the best performance when the dimensionality value reaches 20 to 30 (depending on the API number), indicting the most appropriate latent factors for the specific API-Mashup history. Besides, we observe both MAE and RMSE keep decreasing with the increase of latent factor number from 5 to 35. This observation is consistent with the intuition that a bigger number of latent factors can extract more informative structures. However, when the dimensionality exceeds a certain number (e.g., 30 with API number being 4), the performance begins to drop. The reason is that an excessively larger dimensionality causes the over-fitting problem. As a result, either too small (e.g., 5) or too large a dimensionality (over 35) would degrade the recommendation performance.

4.2.4 Impact of Regularization

To determine the best regularization level, we study the sensitivity of regularization λ by tuning this parameter within the range of $\{10^{-4}, 10^{-3}, ..., 1\}$, with the step size of 10^{-1}. Fig. 9 shows that both MAE and RMSE keep dropping as λ increases until $\lambda = 0.01$, when both begin to increase. That means the performance of our proposed approac performs best when $\lambda = 0.01$. This is why we take the value as the default setting in the comparison with other methods.

4.3 Discussion

In this section, we present our insights on model analysis and discuss some possible extensions, followed by pointing out several other potential service applications of our implicit relation model.

4.3.1 Possible Model Extension

The traditional matrix-factorization based service recommendation models are limited to pursuing some feature representations by decomposing the dyadic matrix (e.g., Mashup-API matrix in this work). In comparison, our model additionally leverages the co-invocation information among APIs (represented as an API-API matrix) by discerning the complex non-linearity via a latent variable model. The model redecodes the API co-invocation matrix as a complementary to linearity extracted from the basic Mashup-API matrix. This makes it easy to be extended to broader scenarios by following these directions:

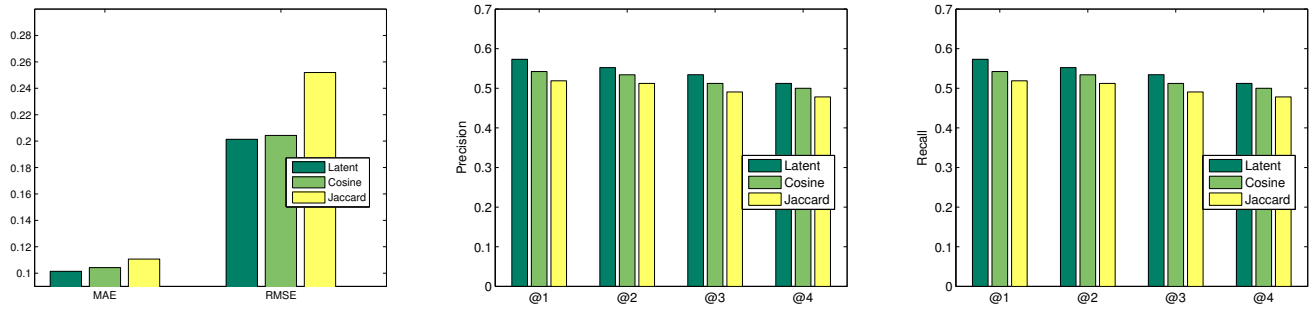
- A possible extension for integrating richer information is to further fuse the co-mashup matrix from the perspective of Mashups. In this way, we can look at if each pair of Mashups use the same API (e.g., a Mashup-Mashup matrix) and boost the recommendation performance by providing additional information representations through a re-decoding process.
- Another direction is to turn to employ deep learning techniques, as such techniques are more advantageous in learning complicated and non-linear representation via stacking multiple layers of information processing modules in hierarchical architectures [24] [25].
- To fully utilize the multi-dimensional information such as the topics, invocation relationships, and temporal information of APIs, we may use the tensor-based matrix factorization to enhance the recommendation performance.

4.3.2 Application Scenarios

Being a generic approach, our proposed model can potentially applied to various application scenarios other than the API recommendation problem. Several potential application domains are:

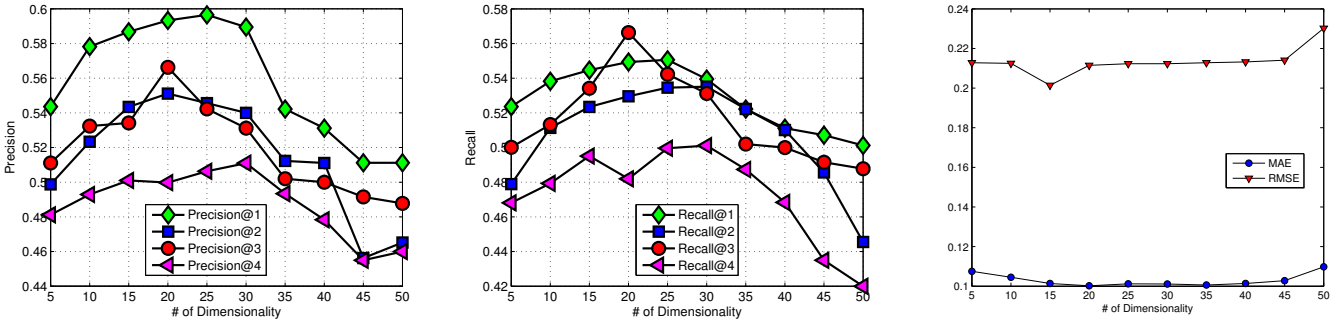
- *Service discovery.* Our proposed method can contribute to service discovery [26], [27] in fundamental problems like service ranking and selection. There has been a long-standing challenge in semantic web service discovery, i.e., given a set of semantically equivalent web services, how to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2803171, IEEE Transactions on Services Computing



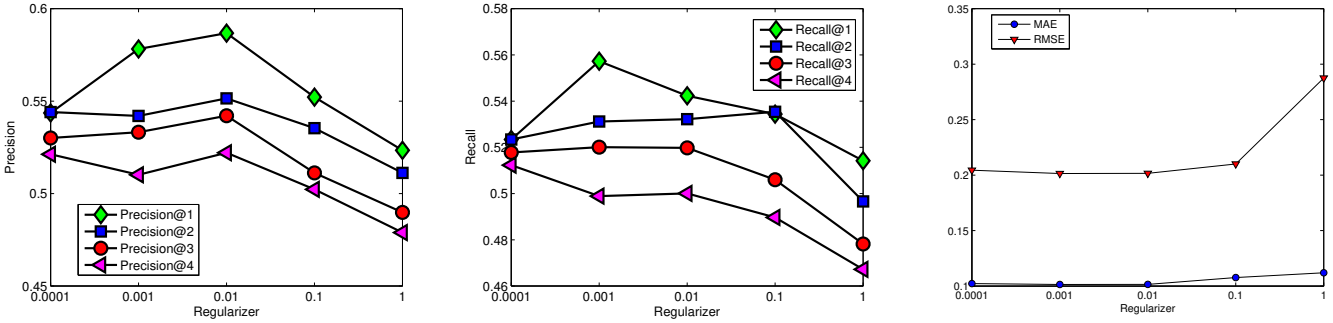
(a) (b) (c)

Fig. 7. Impact of implicit correlations comparison (a) MAE and RMSE (b) Precision x (c) Recall x



(a) (b) (c)

Fig. 8. Impact of dimensionality (a) Precision x (b) Recall x (c) RME and RMSE



(a) (b) (c)

Fig. 9. Impact of regularizer (a) Precision x (b) Recall x (c) RME and RMSE

discern which one is the best or most desirable? Most previous studies on this problem use the semantic similarity or collaborative filtering techniques to produce a personalized rank list of the candidate services [28], [29]. In a similar way, our proposed approach can be easily adapted by replacing mashups with users for API discovery. That means, we only need to replace $\mathbf{m} = \{\mathbf{m}_1, ..., \mathbf{m}_j\}$ with a set of users $\mathbf{u} = \{u_1, ..., u_m\}$, and then API discovery can be conducted by our approach follow Eq. 17.

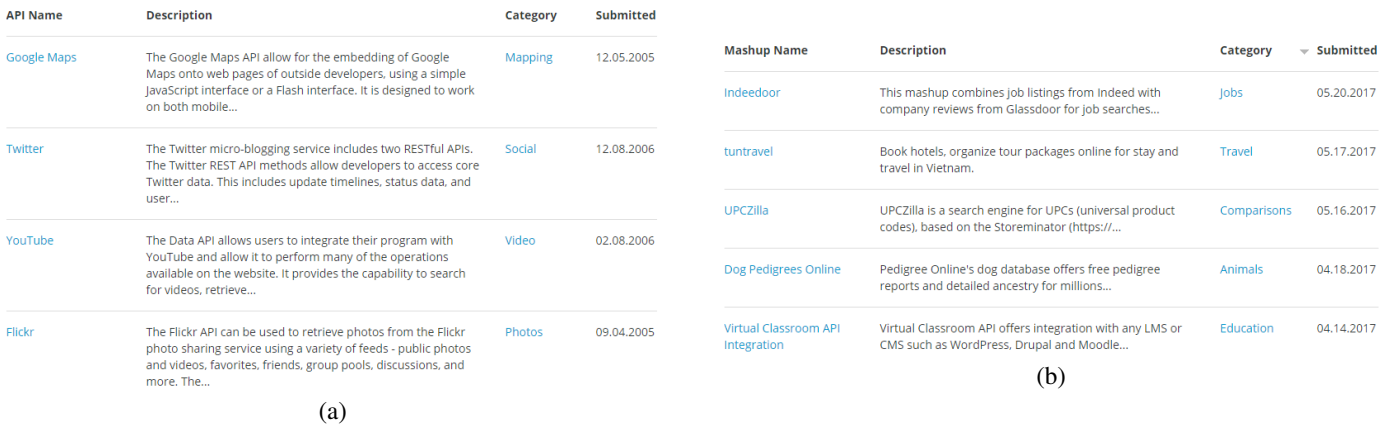
- *Service clustering.* Our methods can also contribute to service clustering [30], [31]. With the estimated hidden discriminative strengths for pairwise services derived from our proposed method, along with semantic similarity, we can easily define a criterion (e.g., a distance function) to build a service graph within a certain distance (e.g., k-nearest services). Based on such graphs, we may develop a more discernible service clustering mechanism other than the tradition mechanism that uses only semantic proximity.

5 Case Study

This section gives snippets of the APIs and Mashups listed in the ProgrammableWeb website to provide an intuitive impression of real-world scenarios (see Fig. 10). We further illustrate the real-world cases of service recommendation for two target mashups (Authorize.Net and Yahoo Travel) to help readers better understand the different recommendation results obtained by different approaches.

In particular, Table 4 shows the top 3 recommendation results obtained by the API-based Neighborhood (AN), Mashup-based Neighborhood (MN), Non-negative Matrix Factorization

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2803171, IEEE Transactions on Services Computing



API Name	Description	Category	Submitted
Google Maps	The Google Maps API allow for the embedding of Google Maps onto web pages of outside developers, using a simple JavaScript interface or a Flash interface. It is designed to work on both mobile...	Mapping	12.05.2005
Twitter	The Twitter micro-blogging service includes two RESTful APIs. The Twitter REST API methods allow developers to access core Twitter data. This includes update timelines, status data, and user...	Social	12.08.2006
YouTube	The Data API allows users to integrate their program with YouTube and allow it to perform many of the operations available on the website. It provides the capability to search for videos, retrieve...	Video	02.08.2006
Flickr	The Flickr API can be used to retrieve photos from the Flickr photo sharing service using a variety of feeds - public photos and videos, favorites, friends, group pools, discussions, and more. The...	Photos	09.04.2005

(a)

Mashup Name	Description	Category	Submitted
Indeedoor	This mashup combines job listings from Indeed with company reviews from Glassdoor for job searches...	Jobs	05.20.2017
tuntravel	Book hotels, organize tour packages online for stay and travel in Vietnam.	Travel	05.17.2017
UPCZilla	UPCZilla is a search engine for UPCs (universal product codes), based on the Storeminator (https://...	Comparisons	05.16.2017
Dog Pedigrees Online	Pedigree Online's dog database offers free pedigree reports and detailed ancestry for millions...	Animals	04.18.2017
Virtual Classroom API Integration	Virtual Classroom API offers integration with any LMS or CMS such as WordPress, Drupal and Moodle...	Education	04.14.2017

(b)

Fig. 10. Examples of Recommended APIs and Mashups on ProgrammableWeb produced by our method. (a) APIs (b) Mashups

(NMF), Regularized NMF (RNMF), Probabilistic Matrix Factorization (PMF), Content-based Recommendation (CBR) approaches, against the top 3 results generated by our proposed approach. A comparison with the ground truth shows that our method outperforms all the other methods by achieving two hits for the first mashup and one hit for the second mashup.

6 Related Work

In this section, we introduce three categories of the related work, followed by a brief summary and discussion.

6.1 Collaborative Service Recommendation Methods

Service recommendation has been an active area of research for years. Traditional service recommendation approaches focus on the quality of mashup service to achieve high-quality service recommendation [6] [32]. Those methods require explicit specification of users' requirements to recommend the appropriate services. In contrast, collaborative filtering (CF) models [33] [34] can capture to some extent users' implicit requirements. Thus, most recent service recommendation approaches are based on CF models. CF is a popular collaborative filtering-based recommendation algorithm, which makes automatic predictions (filter) about the interests of a user by collecting the preference or taste information from many users. Such approaches typically compute similarity of users or services, predict missing QoS values based on the QoS records of similar users or services, and recommend the best services to users. For example, Hu et al. [35] consider users' personalized factors in the service QoS and address the limitations of existing QoS prediction methods by proposing a novel personalized QoS prediction approach. This approach incorporates both the temporal dynamics of QoS attributes and the personalized factors of users and combines collaborative filtering with improved time series forecasting (which uses Kalman filtering) to compensate for the shortcomings of ARIMA models. The approach improves the user-experienced QoS of the recommended service through more accurate QoS predictions.

6.2 Matrix Factorization-based Service Recommendation

Matrix factorization techniques [11] [12] [36] have gained popularity in recent years as a dominant class of CF methods, due to the high accuracy and scalability [10]. This class of techniques focuses on fitting the user-item rating matrix using low-rank approximations and use the obtained matrices to make further predictions. As an example, Yu et al. [37] develop a trace norm regularized matrix factorization algorithm for recommending services with the best QoS to users. This work incorporates the low-rank structure and the clustered representation of real-world QoS data into a unified objective function to estimate users' QoS experience.

To pursue higher accuracy, recent research commonly combines different types of additional information into Matrix Factorization. For example, Zheng et al. [13] propose a neighborhood-integrated Matrix Factorization approach for collaborative and personalized web service QoS value prediction. Chen et al. [38] take location in term of IP addresses of services into account to make more accurate recommendations. The approach combines *user interest value* based on the content similarity between user history records and Mashup services and *QoS predictive value* of Mashup services by collaborative filtering. Ma et al. [14] fuse MF with geographical and social influence for personalized point of interest (POI) recommendation in location-based social networks. Jamali et al. [39] incorporate trust propagation into the matrix factorization model for the recommendation in social networks. Specially, Liu et al. [40] develop two extensions of the matrix factorization models, the data weighting approach and the time-aware modeling approach, for incorporating the social network structure in the context-aware recommendation. More recently, Yao et al. [19] propose a service recommendation approach based on both content similarity and collaborative filtering.

6.3 Matrix Factorization Recommendation for Mashups

Service recommendation for mashups is very similar to the recommendation to the generic service composition. The only differences are the mashup applications are usually data-centric and have no explicit quality requirements or workflows to specify the target mashup. Some typical research on this topic include: Cao et al. [41] use the content similarity between services to recommend services for mashups using the generic MF approach. Xu et al. [42] propose a coupled matrix model to describe the multi-dimensional social relationships among users, mashups, and services. They then design a factorization algorithm to predict unobserved relationships in the model to support more accurate service recommendations. The above recommendation approaches commonly have the following deficiencies: 1) they only provide a single service ranking list without considering the categories

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2803171, IEEE Transactions on Services Computing

TABLE 4
Service Suggestion Results Using Different Approaches

Target Mashup	Method	Top 3 Recommendation API		
Authorize.Net	AN	Yahoo Shopping	Converter	UPS
	MN	AlsaMarketing	ICanLocalize	eBay Shopping
	NMF	AllTrust	Amazon Flexible Payments Service	ecommstats
	RNMF	Yahoo Shopping	Amazon Flexible Payments Service	EgoPay
	RMF	**eDigiCash Shopping Cart**	eBay Shopping	eHealth Technology
	CBR	MasterCard Payments	GeoCash	Yahoo Shopping
	OURS	**KeepMore**	**eDigiCash Shopping Cart**	Macy's Shopping Bag Services
	Groudtruth	**KeepMore**	**Southwind Fan and Light**	**eDigiCash Shopping Cart**
Yahoo Travel	AN	Yahoo Live	Yahoo Local Search	Microsoft SensorMap DataHub
	MN	WhereIsNow	Microsoft SensorMap DataHub	YellowBot Location
	NMF	Yahoo Search	Feest.je	Walker Tracker
	RNMF	Microsoft Bing	VisiStat	Geographic Location
	RMF	MyTravelToolbox	Yahoo Search	Guidebox
	CBR	OctopusTravel	World Travel and Tours	MyTravelToolbox
	OURS	Travel Booking Engine	**Skyscanner Hotels**	WebHotelier
	Groundtruth	**Edamam Diet Recommendations**	**Skyscanner Car Hire**	**Skyscanner Hotels**

to which the services belong, and 2) they neglect the situation when mashup developers are not clear about which categories they need to fulfill the requirement. To address the meaningless ranking of services caused by the above issues, Xia et al. [43] propose a three-step approach, including service clustering for each category, relevant categories identification, and category-aware service recommendation, to enhance the recommendation for mashup creation. This work shows the importance of fully leverage the meta-information in service profiles to achieving effective recommendations.

Recently, Rahman et al. [44] suggest a matrix factorization method based on integrated content and network-based service clustering. This method ensures that the recommendation can be made within a comparable short list of related services, with the latent relationship taken into account.

Considering the impact of service domain evolution, mashup-side cold-start, and the information evaporation problems overlooked by existing work, in a most recent work [45], Bai et al. extend the collaborative topic regression model and develop a generative process to take into account both content and historical usage information for future service recommendation.

6.4 Alternative approaches of recommendation for mashups

Apart from matrix factorization related methods, there are alternative approaches to the recommendation for mashups. Usually, meta-information in service profiles is critical for such methods. Some typical work of this kind includes the following: Cao et al. [46] propose to recommend services as a package for mashup development. In the package, they construct a multi-level relational network model, considering the latent relationship of topics, tags, and services. Instead of outputting a list of similar services, their recommendations give a package of compatible services to further ease mashup development. Wan et al. [47] present a probabilistic model to capture semantic information and components of mashups. The model works with a heterogeneous information network and the regularized framework ensures the consistency of the outputs of the model and the network.

6.5 Summary

The generic recommendation techniques for service recommendation usually considers external knowledge such as social information to improve the recommendation effect. Though several efforts have been contributed on modifying the Matrix Factorization-based service recommendation model for mashup composition, few of them consider the impact of service invocation history to the probability of future invocations.

Inspired by the above approaches and in view of their shortcomings, we propose a novel recommendation approach that integrates an API correlation regularization to the matrix factorization approach. Note that, Lo et al. [48] also combine service similarity and Matrix Factorization in their missing value prediction. Besides serving a different purpose, we infer implicit correlations among APIs rather than directly using the explicit API similarity for making recommendations. The implicit relations are more informative than simple explicit relations because they take both the explicit descriptive information and the influence of historical invocation relations between mashups and APIs into account.

7 Conclusion

This paper presents a mashup service recommendation approach by integrating the implicit API correlations regularization into the matrix factorization model. The intuition is that both the content features of APIs and the historical invocation relations between APIs and mashups are critical in determining the future invocation of APIs by a target mashup. We define the model components and propose corresponding methods for inferring the proposed model. Experimental results over a large real-world service dataset show that our approach outperforms the state-of-the-art collaborative filtering algorithms in term of the recommendation accuracy. This work can be considered as a preliminary step towards systematically exploring automatic service selection and recommendation methods for building optimal mashups by reusing existing online Web-based services.

As part of our future work, we will continue investigating more promising features of mashup applications to further improve the proposed approach. We will particularly focus on exploring the structural information between service providers and users by analyzing their *following* relations, e.g., the APIs may have different groups of users as followers. We will also perform further verification of the proposed methods in more practical mashup applications.

References

[1] A. Bouguettaya et al., "A Service Computing Manifesto: The Next 10 Years," *Communications of the ACM*, vol. 60, no. 4, pp. 64–72, 2017.

[2] L. J. Zhang, J. Zhang, and H. Cai, *Services computing*. Springer, 2007.

[3] Y. Wei and M. B. Blake, "Service-oriented computing and cloud computing: challenges and opportunities," *IEEE Internet Computing*, no. 6, pp. 72–75, 2010.

[4] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu, "Web services composition: A decades overview," *Information Sciences*, vol. 280, pp. 218–238, 2014.

[5] Q. Yu, X. Liu, A. Bouguettaya, and B. Medjahed, "Deploying and managing web services: issues, solutions, and directions," *The VLDB Journal*, vol. 17, no. 3, pp. 537–572, 2008.

[6] C. Cappiello, F. Daniel, M. Matera, and C. Pautasso, "Information quality in mashups," *Internet Computing, IEEE*, vol. 14, no. 4, pp. 14–22, 2010.

[7] X. Liu, Y. Hui, W. Sun, and H. Liang, "Towards service composition based on mashup," in *Services, 2007 IEEE Congress on*. IEEE, 2007, pp. 332–339.

[8] E. Al-Masri and Q. H. Mahmoud, "Investigating web services on the world wide web," in *Proceedings of the 17th international conference on World Wide Web*. ACM, 2008, pp. 795–804.

[9] M. Maleshkova, C. Pedrinaci, and J. Domingue, "Investigating web apis on the world wide web," in *Web Services (ECOWS), 2010 IEEE 8th European Conference on*. IEEE, 2010, pp. 107–114.

[10] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[11] J. D. Rennie and N. Srebro, "Fast maximum margin matrix factorization for collaborative prediction," in *Proceedings of the 22nd international conference on Machine learning*. ACM, 2005, pp. 713–719.

[12] A. Mnih and R. Salakhutdinov, "Probabilistic matrix factorization," in *Advances in neural information processing systems*, 2007, pp. 1257–1264.

[13] Z. Zheng, H. Ma, M. R. Lyu, and I. King, "Collaborative web service qos prediction via neighborhood integrated matrix factorization," *Services Computing, IEEE Transactions on*, vol. 6, no. 3, pp. 289–299, 2013.

[14] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King, "Recommender systems with social regularization," in *Proceedings of the fourth ACM international conference on Web search and data mining*. ACM, 2011, pp. 287–296.

[15] X. Chen, Z. Zheng, Q. Yu, and M. Lyu, "Web service recommendation via exploiting location and qos information," 2013.

[16] M. Granovetter, "The strength of weak ties: A network theory revisited," *Sociological theory*, vol. 1, no. 1, pp. 201–233, 1983.

[17] T. Hofmann, "Probabilistic latent semantic indexing," in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 1999, pp. 50–57.

[18] R. Xiang, J. Neville, and M. Rogati, "Modeling relationship strength in online social networks," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 981–990.

[19] L. Yao, Q. Z. Sheng, A. Segev, and J. Yu, "Recommending web services via combining collaborative filtering with content-based features," in *Web Services (ICWS), 2013 IEEE 20th International Conference on*. IEEE, 2013, pp. 42–49.

[20] R. M. Neal and G. E. Hinton, "A view of the em algorithm that justifies incremental, sparse, and other variants," in *Learning in graphical models*. Springer, 1998, pp. 355–368.

[21] Z. Zheng, H. Ma, M. R. Lyu, and I. King, "Wsrec: A collaborative filtering based web service recommender system," in *Web Services, 2009. ICWS 2009. IEEE International Conference on*. IEEE, 2009, pp. 437–444.

[22] S. Sra and I. S. Dhillon, "Generalized nonnegative matrix approximations with bregman divergences," in *Advances in neural information processing systems*, 2006, pp. 283–290.

[23] R. Tandon and S. Sra, "Sparse nonnegative matrix approximation: new formulations and algorithms," *Max Planck Institute for Biological Cybernetics*, 2010.

[24] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[25] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[26] C. Li, R. Zhang, J. Huai, X. Guo, and H. Sun, "A probabilistic approach for web service discovery," in *Services Computing (SCC), 2013 IEEE International Conference on*. IEEE, 2013, pp. 49–56.

[27] G. Cassar, P. Barnaghi, and K. Moessner, "Probabilistic matchmaking methods for automated service discovery," *Services Computing, IEEE Transactions on*, vol. 7, no. 4, pp. 654–666, 2014.

[28] W. Rong, K. Liu, and L. Liang, "Personalized web service ranking via user group combining association rule," in *Web Services, 2009. ICWS 2009. IEEE International Conference on*. IEEE, 2009, pp. 445–452.

[29] Q. Zhang, C. Ding, and C.-H. Chi, "Collaborative filtering based service ranking using invocation histories," in *Web Services (ICWS), 2011 IEEE International Conference on*. IEEE, 2011, pp. 195–202.

[30] L. Chen, L. Hu, Z. Zheng, J. Wu, J. Yin, Y. Li, and S. Deng, "Wtcluster: Utilizing tags for web services clustering," in *Service-Oriented Computing*. Springer, 2011, pp. 204–218.

[31] D. Skoutas, D. Sacharidis, A. Simitsis, and T. Sellis, "Ranking and clustering web services using multicriteria dominance relationships," *Services Computing, IEEE Transactions on*, vol. 3, no. 3, pp. 163–177, 2010.

[32] X. Wang, Z. Wang, and X. Xu, "Semi-empirical service composition: A clustering based approach," in *Web Services (ICWS), 2011 IEEE International Conference on*. IEEE, 2011, pp. 219–226.

[33] Z. Zheng, H. Ma, M. R. Lyu, and I. King, "Qos-aware web service recommendation by collaborative filtering," *Services Computing, IEEE Transactions on*, vol. 4, no. 2, pp. 140–152, 2011.

[34] L. Liu, N. Mehandjiev, and D.-L. Xu, "Multi-criteria service recommendation based on user criteria preferences," in *Proceedings of the fifth ACM conference on Recommender systems*, 2011, pp. 77–84.

[35] Y. Hu, Q. Peng, X. Hu, and R. Yang, "Web service recommendation based on time series forecasting and collaborative filtering," in *Web Services (ICWS), 2015 IEEE International Conference on*. IEEE, 2015, pp. 233–240.

[36] L. Yao, Q. Z. Sheng, A. H. Ngu, J. Yu, and A. Segev, "Unified collaborative and content-based web service recommendation," *IEEE Transactions on Services Computing*, vol. 8, no. 3, pp. 453–466, 2015.

[37] Q. Yu, Z. Zheng, and H. Wang, "Trace norm regularized matrix factorization for service recommendation," in *2013 IEEE 20th International Conference on Web Services (ICWS)*, 2013, pp. 34–41.

[38] X. Chen, Z. Zheng, X. Liu, Z. Huang, and H. Sun, "Personalized qos-aware web service recommendation and visualization," *Services Computing, IEEE Transactions on*, vol. 6, no. 1, pp. 35–47, 2013.

[39] M. Jamali and M. Ester, "A matrix factorization technique with trust propagation for recommendation in social networks," in *Proceedings of the fourth ACM conference on Recommender systems*, 2010, pp. 135–142.

[40] N. N. Liu, B. Cao, M. Zhao, and Q. Yang, "Adapting neighborhood and matrix factorization models for context aware recommendation," in *Proceedings of the Workshop on Context-Aware Movie Recommendation*, 2010, pp. 7–13.

[41] C. Buqing, M. Tang, and X. Huang, "Cscf: A mashup service recommendation approach based on content similarity and collaborative filtering." *International Journal of Grid & Distributed Computing*, vol. 7, no. 2, 2014.

[42] W. Xu, J. Cao, L. Hu, J. Wang, and M. Li, "A social-aware service recommendation approach for mashup creation," in *IEEE 20th International Conference on Web Services (ICWS)*, 2013, pp. 107–114.

[43] B. Xia, Y. Fan, W. Tan, K. Huang, J. Zhang, and C. Wu, "Category-aware api clustering and distributed recommendationfor automatic mashup creation."

[44] M. M. Rahman, X. Liu, and B. Cao, "Web api recommendation for mashup development using matrix factorization on integrated content and network-based service clustering," in *Services Computing (SCC), 2017 IEEE International Conference on*. IEEE, 2017, pp. 225–232.

[45] B. Bai, Y. Fan, K. Huang, W. Tan, B. Xia, and S. Chen, "Service recommendation for mashup creation based on time-aware collaborative domain regression," in *Web Services (ICWS), 2015 IEEE International Conference on*. IEEE, 2015, pp. 209–216.

[46] J. Cao, Y. Lu, and N. Zhu, *Service Package Recommendation for Mashup Development Based on a Multi-level Relational Network*. Cham: Springer International Publishing, 2016, pp. 666–674. [Online]. Available: https://doi.org/10.1007/978-3-319-46295-0_46

[47] Y. Wan, L. Chen, Q. Yu, T. Liang, and J. Wu, *Incorporating Heterogeneous Information for Mashup Discovery with Consistent Regularization*. Cham: Springer International Publishing, 2016, pp. 436–448. [Online]. Available: https://doi.org/10.1007/978-3-319-31753-3_35

[48] W. Lo, J. Yin, S. Deng, Y. Li, and Z. Wu, "An extended matrix factorization approach for qos prediction in service selection," in *IEEE Ninth International Conference on Services Computing (SCC)*, 2012, pp. 162–169.