Abstract—In this paper, we explore service recommendation and selection in the reusable composition context. The goal is to aid developers finding the most appropriate services in their composition tasks. We specifically focus on mashups, a domain that increasingly targets people without sophisticated programming knowledge. We propose a probabilistic matrix factorization approach with implicit correlation regularization to solve this problem. In particular, we advocate that the co-invocation of services in mashups is driven by both explicit textual similarity and implicit correlation of services, and therefore develop a latent variable model to uncover the latent connections between services by analyzing their co-invocation patterns. We crawled a real dataset from ProgrammableWeb, and extensively evaluated the effectiveness of our proposed approach.

Keywords—Recommendation; matrix factorization; mashup; latent variable model

I. INTRODUCTION

Service-oriented computing promises the exposure and consumption of computing resources through the Internet in the form of platform-independent Web services. This leads to a large number of applications developed heavily based on Web services [1][2]. Among all these applications, mashup services are a type of lightweight Web applications that composes existing Web services to shorten development period and enhance scalability [3]. Nowadays, mashup services have become a popular form of applications and have gained support from multiple platforms, such as Google Mashup Editor1, IBM Mashup Center2, and Yahoo pipes3. Until Feb. 2015, there has already been over 12,839 Web APIs and over 6,168 Mashups on ProgrammableWeb4, and the number is still increasing.

As an example, Figure 1 shows the mashup scenario of a mobile application, WunderWalk5, which enables users to search for places of interest in urban settings. The application combines the Google Maps API6 and the Foursquare API7. In this application, the Google Maps API provides basic functionalities regarding the mapping, such as searching and marking a specific location, while the Foursquare API enables the basic functionalities regarding social activities, such as check-in services and sharing comments and pictures with friends. By invoking the two types of APIs, WunderWalk is able to provide new and powerful functionalities to users, such as viewing and reviewing marked locations and friends’ check-in records or sharing pictures associated with the locations.

Given the large number and diversity (e.g., QoS) of the available services on the Web, it has become more difficult than ever to develop a mashup service due to the unprecedentedly large scope of choices on selecting the services. Thus, it becomes a significant challenge as how to effectively recommend mashup developers with high-quality services in order to accelerate the mashup development. A desirable recommendation result should not only fit the users’ interest, but also be closely relevant to the other services in the mashup (some may have already been selected for the target mashup) to represent a high quality recommendation.

As a dominant approach to implementing the collaborative filtering methods, traditional matrix factorization based recommendation techniques use the historical interactions (e.g., API invocations in historical mashups) as input to make recommendations [4][5][6][7]. Traditional matrix factorization techniques focus on decomposing the interaction matrix (i.e., Mashup-API matrix in our paper) into two low rank matrix approximations, and then using these factorized matrices for recommendation. However, the matrix factorization based recommendation techniques rely on rich records of historical interactions to make accurate recommendations. For those newly released or rarely invoked APIs, the prediction accuracy of these methods is often quite limited.

Some recent works have demonstrated the effectiveness of integrating entity correlations into the matrix factoriza-
tion process. By incorporating users’ social relations [8] or location similarity dependency into service recommendation [9] as regularization terms, the recommendation quality can be greatly improved. The regularization terms introduced by social relations or location similarity can ensure that the distance of the latent feature vectors of two users or two locations is closer if they share some underlying similarity.

Inspired by above successful applications, we propose to incorporate the API service correlations as an extra regularization into the matrix factorization objective function. Instead of directly utilizing the explicit correlations (e.g., content-based similarity of API services), we design a latent variable model to infer implicit API service correlations, which reflect the latent similarities between APIs by studying API co-invocation patterns in historical mashup records. The motivation here is the generalized homophily in social science [10]; more similar two entities are, more interactions they may have (e.g., co-invoked by same mashups in this work). The co-invocations provide, for any pair of APIs, the number of mashups that comprise both APIs. The intuition behind this idea is that, similar API services are more possibly related to similar API services. To be more specific, API service $i$ and $j$ are more similar compared with API service $i$ and $k$ if the former pair are invoked by same mashups in a higher frequency than the latter pair. Our contributions are summarized as follows:

- We investigate the Web-based services (API) recommendation and selection in 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.
- We specially explore the co-invocation patterns between APIs and propose a latent variable model to uncover the implicit similarity by capturing the underlying causal dependency of API invocation process. The experimental results demonstrate that the implicit similarity can boost recommendation performance.
- We crawl the Web to collect real mashup datasets and conduct extensive experiments to validate the proposed approach. The experimental results show the effectiveness of our approach. We also publicly release our mashup dataset for future study.

The rest of this paper is organized as follows. Section II introduces our proposed approach. Section III describes the derivation of implicit pairwise API correlations. Section IV reports the experimental settings and empirical study on proposed approach. Section V overviews related work and Section VI concludes the paper.

II. METHOD OVERVIEW

In this section, we first analyze data characteristics of the API invocation in mashup process. Then, we overview our proposed method.

A. Preliminary

To evaluate our proposed approach, we crawled 11,101 public Web-based API services and 5,658 mashups from ProgrammableWeb\(^8\) (on April 2, 2014). Table I shows some statistics of the dataset.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Statistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of API</td>
<td>11,101</td>
</tr>
<tr>
<td>Number of Mashup</td>
<td>5,658</td>
</tr>
<tr>
<td>Size of Service Corpus</td>
<td>25,256</td>
</tr>
<tr>
<td>Mashup-API (MA) Composition Matrix Density</td>
<td>$1.8811 \times 10^{-4}$</td>
</tr>
<tr>
<td>API-API (AA) Mutual Matrix Density</td>
<td>$1.6897 \times 10^{-4}$</td>
</tr>
</tbody>
</table>

The API invocations have the following characteristics:

- **Sparsity.** Although numerous services are available on the Web, their occurrence in mashup applications is sparsity. An observed entry in the API-Mashup matrix (Figure 2 (a)) indicates the low average frequency of an API to participate in a mashup. Most APIs are never used or not even discovered yet, and the density of API-Mashup matrix is only approximately $1.8 \times 10^{-4}$. Meanwhile, a mashup service usually includes a very limited number of APIs (Figure 2 (b), 90% include less than 5 APIs);

- **Imbalance.** Among the API services that are involved in mashups, their usage frequency is imbalanced. A small portion is used very frequently, while the others are actually not used often. The API-Mashup Matrix (API invocation frequency) dataset is also imbalanced (Figure 2 (c)), with the “frequent” API used over 2,000 times and the “infrequent” APIs less than 10 times. The top 200 most frequent API invocations cover 99% of all API invocations by mashups (Figure 2 (d)).

In order to handle imbalance and sparsity, we propose a matrix factorization based recommendation approach, which have been proved successful \cite{4}\cite{5}\cite{6} in addressing such two major challenges that also characterize the aforementioned API invocation in existing mashups.

B. The Proposed Method

We define the API recommendation problem as recommending a set of prospective APIs for a target mashup, given the invocation records of $n$ APIs in $k$ mashups. We denote invocation relation between APIs and mashups by a matrix $R \in \mathbb{R}^{n \times k}$, where each element $r_{ij}$ indicates whether or not an API $a_i$ is invoked by a mashup $m_j$ (true if $r_{ij} = 1$).

We explore the implicit functional correlations between services to make recommendations based on the matrix factorization approach. The basic idea is to take into account both API profiles and their co-invocation information in previous mashups to make recommendations. The involvement of co-invocation history can address the deficits of pure similarity based recommendation approaches, that sometimes some services simply supplement one another and would not be very similar in their profiles. In particular, we compute the explicit textual similarity of services to capture their domain

\(^8\)http://www.programmableweb.com/
\[ a_i \leftarrow a_i + \eta_i (\delta_{ij} m_j - \alpha \sum_{b \in N_i} z_{ib} (a_i - a_b)) \] (4)
\[ m_j \leftarrow m_j + \eta_j (\delta_{ij} a_i - \lambda_2 m_j) \] (5)

Based on above definitions, we can quantify the possibility of API being invoked by each targeting mashup by calculating the predict score using Eq.1.

### III. IMPLICIT CORRELATION Z DERIVATION

In this section, we first formulate our proposed latent variable model for inferring the implicit correlations of API services from API co-invocation interactions, along with API service textual similarity. Then, we describe the model inference process.

#### A. Model Specification

We believe some implicit ingredients drive the co-use of APIs in mashup services. This latent relationship cannot be directly observed or obtained from API’s descriptive profiles (e.g., category information and API description etc). We assume the latent correlation directly impacts the nature and frequency of API co-invocations in a mashup process. Intuitively, more co-invocations indicate stronger connections among pairwise APIs which may share stronger underlying similarities. In turn, the stronger the relationship, the higher likelihood that co-invocations will take place between the pair of APIs by the same mashup compositions. Thus, we model this latent correlations as a hidden factor of descriptive similarities indicated by API profiles.

Let \( a_i \in \mathbb{R}^m \) be the descriptive vectors of API service, \( y_{ij} \) be the counts of API services \( i \) and \( j \) being invoked by the same mashups, and \( z_{ij} \) be the pairwise implicit correlations of API services \( i \) and \( j \). Specifically, we predict the implicit relations by adapting the statistical mixture model proposed in [11], [12] (as shown in Figure 3):

\[ Pr(z_{ij}, y_{ij}|a_i, a_j) = Pr(z_{ij}|a_i, a_j) Pr(y_{ij}|z_{ij}) \]

\[ = Pr(z_{ij}|s_{ij}) Pr(y_{ij}|z_{ij}) \] (6)

Where \( s_{ij} \) indicates the explicit similarity of API services \( i \) and \( j \) calculated from their descriptive vectors. To obtain the latent relation \( z_{ij} \), we need to solve Eq. 6 by specifying two types of dependencies: i) the dependency between the pairwise explicit similarity of API services \( s_{ij} \) and latent relations,
denotes $c \in C$ and using (7) Term frequency calculation. Keywords corpus construction. This step aims to build $\text{tf}_{z \in X}$ and $\text{idf}_{c \in C}$, and i) 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 details in the following subsections.

1) Specifying $Pr(z_{ij}|s_{ij})$: To measure the dependency between explicit similarity and implicit similarity between pairwise API services, we first use cosine similarity to quantify each pair of API services by evaluating their profile in the text level. We denote each API service file $a_i \in \mathbb{R}^m$ using a TF/IDF weighting scheme. The process pipeline is briefly described as follows:

- Keywords corpus construction. This step aims to build a keyword repository of services, it is mainly divided into two stages. We first segment the service descriptions into terms and remove those terms with little meaning, such as ‘a’, ‘the’, and ‘to’, then eliminate the differences among the words with prefixes (e.g., -in, -un, -dis, -non, et al) or suffixes (e.g., -es, -ed, -ing, -ion, et al). As an example, the words “automate(s)”, “automatic”, “automation” should all map to “automate”. In the end, we have a service corpus containing generic descriptive keywords with the size $k \approx 14,000$, denoted by $c_i \in C$.

- Term frequency calculation. $tf$ (term frequency) is the count of a term’s occurrence in a given service description to measure the importance of the term. This count is usually 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 is obtained by:

$$tf(c_{ij}) = \frac{freq(c_{ij}, x_i)}{|x_i|}$$  \hspace{1cm} (7)

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

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

$$idf(c_{ij}) = \begin{cases} \log \frac{|n|}{\sum_i I(c_{ij} \in x_i)} & \text{if } I(\cdot) \neq 0 \\ \log \frac{|n|}{1 + \sum_i I(c_{ij} \in x_i)} & \text{if } I(\cdot) = 0 \end{cases}$$  \hspace{1cm} (8)

where $|n|$ is the number of services, $I(\cdot)$ denotes the number of service descriptions where term $c_j$ appears.

It should be noted that because most services have short description [13], we assign a higher weight to the $idf$ value (i.e., $a_{ij} = tf \times idf^2$) to normalize the inherent bias of the $tf$ measure in short documents.

Each API service can be represented as a descriptive vector $a_i$. Thus, we can compute the explicit similarity $s_{ij}$ between API services $i$ and $j$ using cosine similarity of their corresponding descriptive vectors as follows:

$$s_{ij} = \frac{a_i \cdot a_j}{||a_i|| ||a_j||}$$  \hspace{1cm} (9)

up to this point, the first dependency in Eq. 6 can be specified:

$$Pr(z_{ij}|a_i, a_j) = Pr(z_{ij}|s_{ij}, w) = N(w_{ij}s_{ij}, \sigma^2)$$

$$= \frac{1}{\sigma \sqrt{2\pi}} \exp \left( -\frac{(z_{ij} - w_{ij}s_{ij})^2}{2\sigma^2} \right)$$  \hspace{1cm} (10)

We assume the relationship between $z_{ij}$ and $s_{ij}$ follows a zero-mean Gaussian distribution $z_{ij} = w_{ij}s_{ij} + \epsilon$, $\epsilon \sim N(0, \sigma^2)$. $w_{ij} \in \mathbb{W}$, which denotes a weight vector that is to be estimated and associated with the similarity of each pair of APIs. $\sigma^2$ is the variance in Gaussian model.

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 in an explicit way. For example, two functionally complementary APIs may have low textual similarity but be frequently used together in mashups. Clearly, such similarity cannot be captured well by the textual features.

In order to tackle this problem, we represent such un-observable similarity as latent variables, and specify them by studying the co-invocation behaviors of APIs in mashup activities. Figure 4 shows the co-invocation behaviors for pairwise API services. Figure 4 (a) shows the distribution of co-invocation counts for all pairs of API services, and
Figure 4 (b) shows the specific co-invocations of each pair of API services in mashups. As the natural stochastic model for the counting data is the Poisson distribution, we use Poisson distribution with a small $\lambda$ to model how many times of co-invocations for each pair of API services given the influence of their latent correlations $z_{ij}$.

$$Pr(y_{ij}|z_{ij}) = \frac{\lambda y_{ij} e^{-\lambda}}{y_{ij}!} = \frac{(\theta z_{ij})^{y_{ij}} e^{-\theta z_{ij}}}{y_{ij}!}$$  \hspace{1cm} (11)$$

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

B. Model Learning

Given three types of inputs, i.e., a training dataset $D$ (denoted by $n$ API pairs), the explicit similarity of services $s_{ij}$ (Section III-A1), and the co-invocation of services $y_{ij}$ (Section III-A2), the likelihood function of this model can be specified as below:

$$Pr(D|w, \theta) Pr(w) Pr(\theta)$$

$$= \prod_{(i,j) \in D} \left( Pr(z_{ij}|s_{ij}, w, \theta) Pr(w) Pr(\theta) \right)$$

$$= \prod_{(i,j) \in D} \left( \frac{1}{\sigma} \left( \frac{1}{\sqrt{2\pi}} e^{-\frac{(z_{ij} - w_{ij}s_{ij})^2}{2\sigma^2}} \left( \theta z_{ij} \right)^{y_{ij}} e^{-\theta z_{ij}} \right) \right)$$

$$\propto \prod_{(i,j) \in D} \left( \frac{\lambda_w}{\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}(z_{ij} - w_{ij}s_{ij})^2} \cdot e^{-\theta i} \cdot e^{-\frac{\theta_j}{2}} \cdot \frac{\lambda_\theta}{2} \cdot \theta \cdot \theta \right)$$

(12)

To avoid overfitting, we put $\ell_2$ regularization on the parameters $w$ and $\theta$ as $Pr(w) \propto e^{-\frac{\lambda_w}{2} w^T w}$ and $Pr(\theta) \propto e^{-\frac{\lambda_\theta}{2} \theta^T \theta}$.

Then we take the logarithm of Equation 12 and adopt the stochastic gradient-based method to maximize the logarithm function by optimizing the model parameters $w$ and $\theta$ and the latent variables $z_{ij}$.

$$\mathcal{L}(z_{ij} \in D, w, \theta) = \sum_{(i,j) \in D} \left( -\frac{1}{2\sigma^2}(z_{ij} - w_{ij}s_{ij})^2 \right. + y_{ij} \ln(\theta z_{ij}) - \theta z_{ij} \left. \right) - \frac{\lambda_w}{2} w^T w - \frac{\lambda_\theta}{2} \theta^T \theta$$

(13)

The model parameters $w$, $\theta$, and $z_{ij}$ can be learned by following a generic coordinate ascent method, in which we update one parameter while fixing the other two parameters. The whole process runs iteratively until reaching a convergence threshold.

$$\frac{\partial \mathcal{L}}{\partial z_{ij}} = \frac{1}{\sigma} \sum_{(i,j) \in D} \left( (w_{ij}s_{ij} - z_{ij}) + \frac{y_{ij}}{z_{ij}} \right)$$

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{1}{\sigma} \sum_{(i,j) \in D} \left( (w_{ij}s_{ij} - z_{ij})s_{ij} \right) - \lambda_w w_{ij}$$

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_{(i,j) \in D} \left( \frac{1}{\sigma} y_{ij} - z_{ij} \right) - \lambda_\theta \theta$$

(14)

after obtaining the model parameters, for any given testing sample of an API service $i$, we first calculate its descriptive vector $a_i$, and its textual similarity $s_{ij}$ with other training APIs, then we can extract its co-invocation patterns with other training APIs $y_{ij}$.

$$z_{ij}^{\text{new}} = z_{ij}^{\text{old}} - \frac{\partial \mathcal{L}}{\partial z_{ij}} / \left( \frac{\partial^2 \mathcal{L}}{\partial (z_{ij})^2} \right)$$

(15)

Up to this point, combined with learned model parameters, we can estimate the latent correlations between the testing API service $i$ and the training API services $z_{ij}$ using Eq. 15, where $\frac{\partial^2 \mathcal{L}}{\partial (z_{ij})^2} = \frac{1}{\sigma} \sum_{(i,j) \in D} (1 + \frac{y_{ij}}{z_{ij}})$.

IV. EXPERIMENTS

In this section, we first introduce the experimental settings and then present the analysis of the experimental results. The experiments focus on the following aspects: i) comparing the performance of our proposed method and the representative baseline methods, ii) studying the impact of proposed implicit correlations, and iii) evaluating the impact of different parameter settings in terms of dimensionality and regularization.

A. Experimental Settings

1) Validation: The training set and the testing set are constructed as follows: firstly, we randomly select 20% API-Mashup pairs as the testing set; then we 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 (as shown in Figure 5).

Table II shows statistics of the testing set and the training sets with different sparsity levels, where 10% means the training set contains 10% API-mashup pairs, and 20% means it contains 20% API-mashup pairs, and so on.

<table>
<thead>
<tr>
<th>Table II</th>
<th>Training Data with Different Sparsity Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Density</td>
</tr>
<tr>
<td>10%</td>
<td>$1.8604 \times 10^{-4}$</td>
</tr>
<tr>
<td>20%</td>
<td>$1.8671 \times 10^{-4}$</td>
</tr>
<tr>
<td>30%</td>
<td>$1.8712 \times 10^{-4}$</td>
</tr>
<tr>
<td>40%</td>
<td>$1.8751 \times 10^{-4}$</td>
</tr>
<tr>
<td>50%</td>
<td>$1.8774 \times 10^{-4}$</td>
</tr>
</tbody>
</table>
2) Metrics: We adopt two commonly used metrics in collaborative filtering, Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE), to measure the recommendation performance. For the both metrics, smaller values indicate better performance.

\[
\text{MAE} = \frac{1}{N} \sum_{ij} |r_{ij} - \hat{r}_{ij}|
\]

\[
\text{RMSE} = \sqrt{\frac{1}{N} \sum_{ij} (r_{ij} - \hat{r}_{ij})^2}
\]

B. Comparison with Other Approaches

In this section, we investigate and compare our proposed approach with the following approaches over different sparsity levels (shown in Table II). The baseline methods are briefly depicted as follows:

- API-based Neighborhood (AN). This method uses Pearson Correlation to calculate similarities between APIs, and makes recommendation based on similar APIs [14].
- Mashup-based Neighborhood (MN). This method uses Pearson Correlation to compute the similarities between mashups and predicts invocations based on similar mashups.
- Non-negative Matrix Factorization (NMF). It applies non-negative matrix factorization on API-Mashup matrix for predicting the missing invocations. The invocation matrix between API and mashup \( R \) can be decomposed into two lower dimension matrices

\[
R_{ij} = A_{iA}M_{kj}
\]

The model can be solved by the following optimization process

\[
\min_{A,M} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{k} (r_{ij} - A_i^T M_j)^2
\]

- Regularized NMF (RNMF). It imposes two regularizations of \( a \) and \( m \) to avoid overfitting, and is formulated as:

\[
\min_{A,M} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{k} (r_{ij} - A_i^T M_j)^2 + \frac{\lambda_A}{2} ||a||_F^2 + \frac{\lambda_M}{2} ||m||_F^2
\]

- Probabilistic Matrix Factorization (PMF). It is one of the most famous MF models in collaborative filtering [6]. It assumes Gaussian distribution on the residual noise of 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:

\[
\min_{A,M} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{k} (g(r_{ij}) - g(A_i^T M_j))^2 + \frac{\lambda_A}{2} ||a||_F^2 + \frac{\lambda_M}{2} ||m||_F^2
\]

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

For our approach, we set the dimensionality \( d = 15 \). For simplicity, we set the same value for the two regularization parameters \( \lambda_A = \lambda_M = 0.01 \) in the experiments. It should be noted that the impact of different parameter settings are studied in details in the rest of the section.

Table III 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 effectiveness of bringing the implicit relation information in the highly sparse API-Mashup matrix for improving the recommendation accuracy.

C. Impact of Implicit Correlations

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

- Cosine similarity. We use cosine similarity to compute the relations between APIs \( a_i \) and \( a_j \), both denoted as TF-IDF vectors:

\[
sim(a_i, a_j) = \frac{a_i \cdot a_j}{||a_i|| ||a_j||}
\]

- Jaccard similarity. The Jaccard similarity of each pair of APIs engaged in the mashup activities are calculated as:

\[
sim(a_i, a_j) = \frac{|B_i \cap B_j|}{|B_i \cup B_j|}
\]

where both \( B_i \) and \( B_j \) are binary vectors denoting the mashup sets that \( a_i \) and \( a_j \) participate in, respectively.
The parameter dimensionality determines the number of latent features used to characterize APIs and mashups. In this section, we study the impact of parameter dimensionality by varying the dimensionality value from 5 to 50 with a step value of 5. Figure 6 (b) shows that our approach achieves the best performance when the dimensionality value is 15, which may indicate the most appropriate latent factors for the API-Mashup histories. From the results, we can observe with the increase of latent factor number from 5 to 35, the MAE and RMSE keep decreasing. This observations are consistent with the intuitions that bigger number of latent factors can extract more informative structures. However, when the dimensionality exceeds 35, the performance begins to drop. The reason is that a larger dimensionality causes the over-fitting problem. As a result, either too small (e.g., 5) or too large a dimensionality (e.g., 40, 50 etc) would degrade the recommendation performance.

E. Impact of Regularization

To decide the most appropriate regularization level, we study the sensitivity of regularization $\lambda$ by tuning this parameter within the range of $\{10^{-4}, 10^{-3}, ..., 1\}$ with the step of $10^{-1}$. Figure 6 (c) shows that both MAE and RMSE keep dropping as $\lambda$ increases until $\lambda = 0.01$, and then begin to increase. That means the performance of our proposed approach achieves the best at $\lambda = 0.01$, which is also the reason that we take the value as the default setting in our comparative experiments.

V. RELATED WORK

Service recommendation have been an active area of research for years. Traditional service recommendation approaches locate quality of mashup service to realize high-quality service recommendation [3][15]. Such methods require explicit specification of users’ requirements to recommend the appropriate services. On the other hand, collaborative filtering (CF) models [16][17] can reflect to some extent users’ implicit requirements. Thus, most recent service recommendation approaches are based on CF models. CF is a popular recommendation algorithm, which makes automatic prediction (filter) about the interests of a user by collecting preferences or taste information from many users (collaborating). These approaches 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.

Among different CF methods, matrix factorization techniques [5][6] have gained popularity as a dominant class of methodology within collaborative filtering recommenders [4], due to the high accuracy and scalability. These methods focus on fitting the user-item rating matrix using low-rank approximations, and use it to make further predictions. Specially, Yu et al. [18] 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 single unified objective function to estimate users’ QoS experience.

In pursuit of higher accuracy, recent research commonly combines additional information into Matrix Factorization. Zheng et al. [7] propose a neighborhood-integrated Matrix Factorization approach for collaborative and personalized web service QoS value prediction. Chen et al. [19] take location in term of IP addresses of services into account to make more accurate recommendation. 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. [8] fuse MF with geographical and social influence for personalized point of interest (POI) recommendation in location-based social networks. Jamali et al. [20] incorporate trust propagation
into the matrix factorization model for recommendation in social networks. More recently, Cao et al. [21] and Yao et al. [13] respectively propose a service recommendation approach based on both content similarity and collaborative filtering. Liu et al. [22] develop two extensions of the matrix factorization models, data weighting approach and time-aware modeling approach, for incorporating the social network structure in context-aware recommendation. Xu et al. [23] propose a coupled matrix model to describe the multi-dimensional social relationships among users, mashups, and services, and design a factorization algorithm to predict unobserved relationships in the model to support more accurate service recommendations. Though tremendous efforts on modifying the Matrix Factorization model, few of them consider the impact of service invocation history to the probability of future invocations.

Inspired by above approaches and in sight of their shortcomings, we propose to extend a recommendation approach by integrating the API correlations regularization to the matrix factorization objective function. Lo et al. [24] also combine implicit correlations and Matrix Factorization in their missing value prediction. Besides serving different purposes, 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 by taking the influence of historical invocation relations between mashups and APIs into account.

VI. 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 essential in determining the future invocation of APIs in a target mashup. We define the model components and propose corresponding methods for inferring the model. The 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 accuracy. This work can be considered our preliminary step of systematically exploring automatic service selection and recommendation in mashup composition by reusing online Web-based services.

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

REFERENCES