Category-Aware API Clustering and Distributed Recommendation for Automatic Mashup Creation

Bofei Xia, Yushun Fan, Wei Tan, Keman Huang, Jia Zhang, and Cheng Wu

Abstract—Mashup has emeraged as a promising way to allow developers to compose existed APIs (services) to create new or value-added services. With the rapid increasing number of services published on the Internet, service recommendation for automatic mashup creation gains a lot of momentum. Since mashup inherently requires services with different functions, the recommendation result should contain services from various categories. However, most existing recommendation approaches only rank all candidate services in a single list, which has two deficiencies. First, ranking services without considering to which categories they belong may lead to meaningless service ranking and affect the recommendation accuracy. Second, mashup developers are not always clear about which service categories they need and services in which categories cooperate better for mashup creation. Without explicitly recommending which service categories are relevant for mashup creation, it remains difficult for mashup developers to select proper services in a mixed ranking list, which lower the user friendliness of recommendation. To overcome these deficiencies, a novel category-aware service clustering and distributed recommending method is proposed for automatic mashup creation. First, *a Kmeans variant* (*vKmeans*) method based on topic model Latent Dirichlet Allocation is introduced for enhancing service categorization and providing a basis for recommendation. Second, on top of *vKmeans*, a *service category relevance ranking* (*SCRR*) model, which combines machine learning and collaborative filtering, is developed to decompose mashup requirements and explicitly predict relevant service categories. Finally, *a category-aware distributed service recommendation* (*CDSR*) model, which is based on a distributed machine learning framework, is developed for predicting service ranking order within each category. Experiments on a real-world dataset have proved that the proposed approach not only gains significant improvement at precision rate but also enhances the diversity of recommendation results.

Index Terms—Mashup, service clustering, service recommendation, probabilistic topic model, extreme learning machine

1 Introduction

Service-oriented computing (SOC) has led to a new generation for software engineering, changing the way of designing, developing, delivering and consuming software applications [1], [2]. As a result, service technology has been evolving rapidly and an increasing number of services have become available on the Internet [3], [4]. Meanwhile, since user requirements are comprehensive, the fulfillment of user's needs relies more on a set of composed services rather than a single service. Therefore, service composition plays a key role in SOC and how to facilitate the construction of service composition is critical to the wide adoption of service technology [1], [5], [6].

Recently, the mashup technology, which allows developers to compose existing services to create new or value-added services, has emerged as a promising service composition approach [7]. Generally speaking, a mashup process is usually operated at a web browser, by "dragging and dropping" APIs (programmable applications) from different sources. In recent years, a number of online mashup repositories have been established representative of ProgrammableWeb,[1] myExperiment,[2] and Biocatalogue.[3] In these repositories, a large number of published services offer interfaces or APIs for external invocation, and users can compose various services with different functionalities to create mashups for fulfilling comprehensive requirements. On ProgrammableWeb, for instance, mapping service *Google maps* and auction service *ebay* are recomposed to create a mashup service named *BidNearBy*, which searches local auctions in a map view.

When a user begins to develop a mashup, the first thing is to select proper existing services from the repository. However, rapid increasing number of services in the repository makes the selection difficult. For example, on ProgrammableWeb (till May 2014), 11,320 services have been published and 7,440 mashups have been created

- *B. Xia, Y. Fan, and C. Wu are with the Department of Automation, Tsinghua University, China. E-mail: 610306199@qq.com, {fanyus, wuc}@tsinghua.edu.cn.*
- *W. Tan is with the IBM Thomas J. WatsonResearch Center, Yorktown Heights, NY. E-mail: wtan@us.ibm.com.*
- *K. Huang is with the School of Computer Science and Technology, Tianjin University, China. E-mail: victoryhkm@gmail.com.*
- *J. Zhang is with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Moffett Field, CA. E-mail: jia.zhang@sv.cmu.edu.*

Manuscript received 2 Nov. 2014; accepted 2 Dec. 2014. Date of publication 9 Dec. 2014; date of current version 9 Oct. 2015.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSC.2014.2379251

1. www.programmableweb.com
2. www.myexperiment.org
3. www.biocatalogue.org

1939-1374 © 2014 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: SOUTHERN METHODIST UNIV. Downloaded on December 23,2020 at 05:10:47 UTC from IEEE Xplore. Restrictions apply.

with these services. Selecting suitable services is thus an intractable task even for experienced users. Therefore, service recommendation for automatic mashup creation has gained a lot of momentum. Some works [8], [9] analyze the requirement text and service description file then recommend services based on their semantic compatibility. Other approaches [10], [11], [12] predict service quality of service (QoS) and recommend services from the angle of QoS optimization. Furthermore, user interest and social relationship are considered by some works [13], [14], [15] when recommending services.

To summarize, most existing methods directly recommend services from the entire repository and rank all candidate services with different functionalities in a single list. Taking the aforementioned mashup on ProgrammableWeb for example: after searching services in the entire repository, the existing methods may recommend services like "*Google maps*, *Amazon Prodcut Advertisin*, *Shopping.com*, *Bing maps*, *Yahoo maps*, *eBay*..." However, this type of service recommendation has two main deficiencies:

1) Ranking services without considering to which categories they belong will lead to meanlingless ranking for mashup creation. For example, *Google maps* and *ebay* obviously belong to different service categories and are actually used for fulfilling different aspects of mashup. Therefore, the ranking order between these two services doesn't make much sense for mashup creation.
2) Mashup developers are not always clear about which service categories they need. Selecting proper services in a mixed ranking list is difficult for mashup developer. Furthermore, mashup developers may also have no idea about which service categories cooperate better for mashup creation.

Our investigation of the main causes of the deficiencies is summarized as follows. First, the service categorization in these repositories is weak which restricts considering category when recommending services. In most repositories, service categorization is currently realized through a manul process [16]. Furthermore, some existing service clustering algorithms only consider the functional similarity which restricts the clustering accuracy. Second, most service recommendation methods for mashup creation lack the component of explicitly predicting which service categories are more relevant given requirements.

In this paper, we propose a three-step approach to overcome the aforementioned restrictions and offer category-aware service clustering and recommending for automatic mashup creation. First, a service clustering method *Kmeans variant* (*vKmeans*) is developed for enhancing service categorization which provides a basis for recommendation. Second, for decomposing mashup requirements and explicitly predicting relevant categories, a service category relevance ranking model is proposed. Finally, a category-aware distributed service recommendation (CDSR) model is proposed for ranking services within each relevant category. In this way, mashup developer is informed with both "what service categories may cooperate better for fulfiling your requirement" and "what is the ranking order of services within each relevant category."

The main contributions of the paper are three-fold:

1) A service clustering algorithm *Kmeans variant* (*vKmeans*) is introduced for enhancing service categorization. Different from the traditional clustering method Kmeans, *vKmeans* considers different status of services regarding mashups. *vKmeans* first identifies core services of each category and then clusters other services by their functional similarity. This algorithm provides a basis for *category relevance ranking* and *category-aware service recommendation* in subsequent steps. Our experiments proved its suitability for category-aware service recommendation.
2) A *service category relevance ranking* (*SCRR*) model is proposed for decomposing mashup requirements and explicitly predicting relevant service categories. This model contains two phases: *Category Topic Matching* (*CTM*) and *Category Affinity Propagation* (*CAP*). *CTM* combines probablisitic topic model and machine learning technique, is designed for predicting the functional relevance probability of each category given mashup requirements. Based on Collaborative Filtering (CF), *CAP* takes the output of *CTM* as input and further considers the collaborative relationship among different categories in history information for predicting the final relevance probability of each category. Superior to existing recommendation methods, this model mainly addresses the problem when mashup developers are not clear about which service categories are needed for mashup creation.
3) A *Category-aware Distributed Service Recommendation* model is proposed for ranking services within each relevant category. On top of service clustering result of *vKmeans*, this model is designed with a distributed machine learning framework that predicts the service ranking order within different categories. Superior to existing recommendation methods, this model eliminates the meanlingless ranking among services in different categories and focuses on meaningful ranking order of services within each category.

Our experiements over a real-world dataset show that our method gains a 30 percent improvement on recommendation accuracy, compared to state-of-the-art approaches. Furthermore, our method achieves a 20 percent improvement for long tail recommendation, i.e., recommending not-so-popular services. This proves that our approach enhances the diversity of recommendation results.

The remainder of the paper is organized as follows. Section 2 shows a motivation use case; Section 3 illustrates the overall framework of our method; Section 4 presents the service clustering and recommending model and algorithms; Section 5 presents the experiments and discusses results on a real-world dataset; Section 6 discusses the related work and Section 7 draws a conclusion.

2 Motivation Use Case

In this section, we use a real case to illustrate how category-aware service clustering and distributed recommendation may enhance the process of automatic mashup creation.

The ProgrammableWeb is by far the larget online web open APIs (services) repository that contains more than

Authorized licensed use limited to: SOUTHERN METHODIST UNIV. Downloaded on December 23,2020 at 05:10:47 UTC from IEEE Xplore. Restrictions apply.

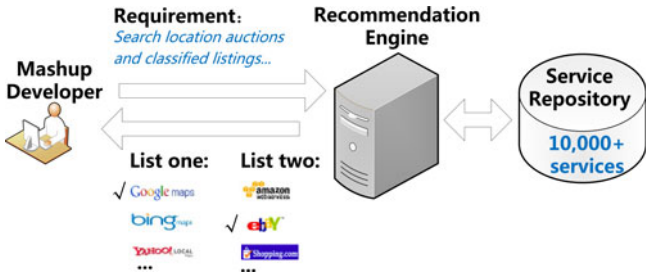


Fig. 1. Category-aware service recommendation use case for mashup creation. In this case, the mashup developer is not clear about which service categories are relevant. According to the mashup developer's requirement, the recommendation engine recommends two categories of services (*mapping* and *auction*) and the service ranking order within each category. Then developer can choose services from these two categories, respectively.

10,000 APIs with various functionalies. Such reusable and programmable APIs enable developers to compose some of them to creat mashups (compositions) for fulfilling comprehensive needs. For example, a developer plans to create a mashup application that can search bidding information near by. The requirement is described as "*Search local auctions and classified listings (craigslist) and show them on a map view*." Such a requirement is unlikely to be fulfilled by a single API. How to select proper APIs from the entire repository is intractable for a human developer. So there comes the desire of recommendation.

As shown in Fig. 1, the recommendation engine first understands the requirement text and infers that two kinds of APIs are needed for satifying the requirement: APIs with *maping and locating* function and APIs with *auction search* function. Then the engine searches for candidate APIs for the two kinds, respectively. Thereafter, the engine refers to the historical mashup-APIs invoking information and ranks candidate APIs within the two categories, respectively. Finally, the recommendation engine returns two candidate ranking lists to the developer: *Google maps*, *Bing maps*, and *Yahoo maps* for mapping service; *Amazon Prodcut Advertising*, *eBay*, and *Shopping.com* for auction searching service. The developer may finally select *Google maps* and *eBay* to build the mashup application. During this recommendation process for automatic mashup creation, three important issues can be identified as follows:

Service clustering. Mashup inherently requires services from different categories. How to cluster large numbers of services into different categories in an appropriate way for providing a basis for mashup creation?

Mashup requirements decomposing. Usually a developer is not clear about which categories are needed to fulfill the requirements. How to automatically decompose and map the mashup requirements to relevant service categories?

Category-aware distributed service ranking. In order to eliminate the meaningless service ranking among different categories and improve recommendation accuracy, how to design a recommendation framework for ranking candidate services within each relevant service category for the given mashup requirements?

3 Overview of Methodology Framework

To tackle the aforementioned challenges, we have designed a category-aware API clustering and distributed recommendation framework. The overview of the framework is shown in Fig. 2. The framework is composed of two parts: an offline service clustering part and an online service recommendation part.

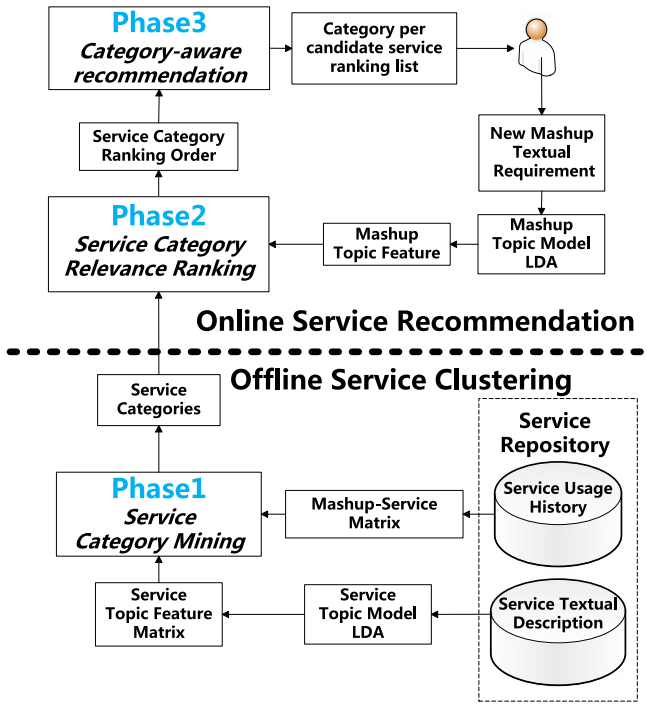


Fig. 2. Overview of methodology framework.

The offline part preprocesses the service and mashup historical information and clusters services into differnet service categories, which provides a basis for recommendation. The kernel phase of this part is a *Service Category Mining* method that takes two matrixes as input: service topic feature matrix (*captured from service textual description by topic model Latent Dirichlet Allocation (LDA)*) and the mashup-service matrix (*extracted from service usage history*). The service topic feature matrix is corresponding to the factor of service functionality and the mashup-service matrix is corresponding to the factor of service popularity. Taking both of the factors into account, the services in a repository are clustered in a more proper form for recommendation.

In the online part, after receiving a new mashup requirement, the textual requirement is first converted to a mashup topic feature vector that quantifies weights of various needs within the requirement. Then through the phase of *Service Category Relevance Ranking*, which takes the mashup topic feature and service category result as input, the result of service category relevance ranking order is obtained. This phase addresses the problem when mashup developers are not clear about the needed categories, and predicts the relevance probability of each service category. Afterwards, making use of the result of service category ranking order, the *Category-aware Recommendation* phase, which is based on a distributed recommendation framework, finally recommends services in the form of 'Category per candidate service ranking list' to the user. This phase eliminates the meanlingless service ranking among different categories and predicts the relevance probability of services with in a category. In this way, the user is informed 'which service category is

Authorized licensed use limited to: SOUTHERN METHODIST UNIV. Downloaded on December 23,2020 at 05:10:47 UTC from IEEE Xplore. Restrictions apply.

relevant to your composition requirement' and 'which services within a category may satisfy your requirement better'.

4 Model and Algorithms

In this section, we first introduce some preliminary knowledge of the probabilistic topic model LDA and the concept of service topic feature in Section 4.1. Then we propose a three-phase novel method for enhancing automatic mashup creation: the *Kmeans variant* (*vKmeans*) algorithm for service clustering is developed in Section 4.2; the *Service Category Relevance Ranking* model for category relevance ranking is proposed in Section 4.3; and the *Category-aware Distributed Service Recommendation* (*CRSR*) model for 'category per candidate service ranking list' is discussed in Section 4.4.

4.1 Preliminary Knowledge of Probabilistic Topic Model LDA and Topic Feature

Probabilisitc topic model *Latent Dirichlet Allocation* [17] is a known unsupervised statistical model for natural language processing. It is mainly used to discover the abstract 'topics' that occur in a collection of documents. Generally, a 'topic' here consists of a cluster of words that represent a certain aspect of content, and a document is typically mixed with multiple topics. Given a corpus, LDA model can not only automatically predict which words are relevant to a given topic, but also predict the weights of different topics (or topic distributions) in the document.

In the context of 'service' or 'mashup,' a topic represents a certain aspect of its functionality extracted from the description document; the topic distributions represent to which aspects of function that the service or mashup are relevant. For example, a topic that contains words of 'GIS' and 'location' indicates that this topic is highly correlated to 'mapping' functionality. If a service's or mashup's document has more weights on this topic, we can infer the service or mashup may be a mapping service or a mashup invoking mapping services.

The reason why we adopt topic features other than traditional keywords to represent the functionality of a service or mashup is that, topic feature is more robust and flexible [8], [18]. Assume that a user only inputs the keyword 'photo' for searching a service. Even though some services described with similar words (e.g., 'image,' 'album,' and 'picture') are relevant to the user's request, they may be neglected via simple keyword-based methods. When it comes to mashup, keyword-based methods also face the same problem. On the contrary, topic feature could remedy this flaw thus contributes to a better recommendation performance. The result of service topic feature extraction will be further discussed in the experiment part.

4.2 Service Category Mining

4.2.1 Motivation of Service Category Mining

Services in a repository gradually form different categories in which services have similar functionality. According to our previous empirical study [17] of service usage pattern, in a certain functionality aspect, there usually exists one core service that gains the most popularity. Some later emerging services are likely to imitate these core services and have similar functionality. However, some traditional clustering methods (e.g., Kmeans) that only cluster services according to their functionaly similarity are not suitable in the context of service clustering. To remedy this deficiency, we introduce a clustering algorithm based on Kmeans naming *Kmeans variant* (*vKmeans*) for service categorization. In this context, each service is represented by its topic feature vector as aforementioned. The similairity between two services is measured by the KL distance of their topic feature vectors. The main advantage of vKmeans is that, vKmeans first identifies the core services of the K service categories and assigns the values of these core services' topic feature vectors as the initial points of each category. In contrast, the tranditiaonl Kmeans only randomly selects K services' topic feature vectors as the initial points for each category. This improvement of service clustering is proved to enhance service recommendation in the experiment part.

TABLE 1
Notions in Latent Service Category Mining

Notions	Explanations
N_s	Total # of services
K_s	Total # of topics evolved in all services
S_i	Service i in repository
STF	$N_s \times K_s$matrix, $STF(i, j)$ represents the probability of topic j given S_i
M_i	Mashup i
N_m	Total # of mashup
MS	$N_m \times N_s$ matrix, $MS(i, j) = 1$ if S_j is invoked by M_i; otherwise $MS(i, j) = 0$
N_c	Preseted # of service category
C_c	Service Category c
N_{cc}	The # of services clustered in category c
S_c^*	The core service of category c

4.2.2 Materials Prepared for Category Mining

In vKmeans, both service's description plain text and service historical usage records are considered.

In recent years, plain texts (e.g., description text, tags, etc.) are more user friendly than traditional service description file (i.e. WSDL), and are often adopted by more repositories to describe the functionality of a service. We apply text mining approach to extract service topic features from these plain texts. We resort to the probabilistic topic model LDA to map the funtionality of a service to a fixed length vector. The value of every element in the vector ranges from 0 to 1, denoting the probability of a function may be involved in this service. The summation of all the elements of the service topic feature vector equals 1. Formally, we define *STF* matrix (illustrated in Table 1) to integrate the topic feature vectors of all services.

In addition, we quantitively measure the popularity of a service by the number of mashups that invoke it. The popularity of all services can be obtained from their historical usage by mashups. Similar to [19], [20], [21], the historical service usage information is integrated in the *MS* matrix (illustrated in Table 1). Afterwards, the summation of the ith column of *MS* matrix represents the popularity of service i (Algorithm 1, lines 1-3).

Authorized licensed use limited to: SOUTHERN METHODIST UNIV. Downloaded on December 23,2020 at 05:10:47 UTC from IEEE Xplore. Restrictions apply.

4.2.3 Two-Phase Category Mining Strategy

In this section, we introduce the vKmeans clustering method for clustering services into categories. The main idea of vKmeans is a two-phase clustering strategy. In the first phase, we refer to services' topic feature vectors and their popularity to assign per category a core service as the initial point for service clustering. In the second phase, other non-core services are clustered into different categories, similar to Kmeans. The similarities between services are measured by Kullback Leibler (KL) distance [22] of service topic feature vectors. We describe the pseudo code of vKmeans in Algorithm 1 and list all involved notions in Table 1.

Algorithm 1. Variant Kmeans

Input: *STF* matrix, *MS* matrix, N_c
Output: $S_{ci} : 1 \leq c \leq N_c, 1 \leq i \leq N_{ci}$
Phase 1.
1: **For** $i = 1$ to N_s
2: Calculate S_i's popularity by summing the *i*th column of *MS* matrix
3: **End For**
4: Rank all services by their popularities
5: $c = 1$
6: **For** $r = 1$ to N_s
7: **For** $j = 1$ to c
8: Calculate the KL distance between the *r*th popular service and S_j^*
9: **If** the KL distance $\geq$ Threshold
10: $c = c + 1$
11: **Break**
12: **End If**
13: **End For**
14: Assign the *r*th popular service as S_c^*
15: **If** $c > N_c$
16: **Break**
17: **End If**
18: **End For**
Phase 2.
19: **For** $i = 1$ to N_s
20: **If** S_i is not a core service
21: **For** $c = 1$ to N_c
22: Calculate the center point of service category c
23: Calculate KL distance between S_i and center point of service category c
24: **End For**
25: Assign S_i to the service category c with minima KL distance
26: $N_{cc} = N_{cc} + 1$
27: **End For**
28: Calculate the center point of each service categories
29: Repeat 19-28 until the center point of each categories converages.

Phase 1 of Algorithm 1 mainly mines the core services of each category. Lines 1-3 use the *MS* matrix to calculate the popularity of each service and line 4 ranks overall services considering their popularity. Lines 5-18 visit services by the order of popularity, and identify core services of each category regarding their topic feature vectors' KL distance. Topic features of all the services are caught in the *STF* matrix.

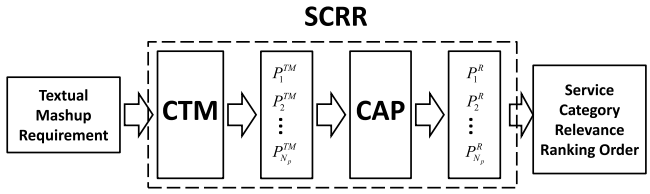


Fig. 3. Outline of service category relevance ranking model.

Phase 2 of Algorithm 1 clusters all the non-core services similari to traditional clustering process of Kmeans. Lines 21-24 calculate the KL distance between the unclustered services and the center point of each service category. Afterwards, in lines 25-26, we assign the unclustered service to the category with minimal KL distance. In lines 28-29, we iterate the Kmeans-like process until the value of center point of each service category converges.

In the context of service clustering, the major advantage of our vKmeans is that it uses both service topic feature similarity and service popularity when assigning the initial service of each category. In contrast, most traditional clustering methods randomly assign the intial point of each category, which neglects the status of different services thus may lead to unreasonable clustering results. On top of vKmeans, *service category relevance ranking* and *category-aware distributed service recommendation* become possible.

4.3 Service Category Relevance Ranking Model

4.3.1 Model Description

On top of service clustering results, given a mashup textual requirement, the intuitional task is to decompose the requirement and analyze which service categories may be relevant to the requirement. For this purpose, we have developed the *Service Category Relevance Ranking* model. In *SCRR*, two factors are considered to have influence on the category relevance ranking order.

- First, from the view of functional matching, which categories can probably satisfy the functional needs of the mashup requirement?
- Second, some service categories are frequently invoked together in historical mashups. How do the collaborative relationships among different service categories affect the relevance ranking? Which categories are more likely to be invoked together for fulfilling a new mashup requirement?

Following the aims above, as shown in Fig. 3, the *SCRR* model consists of two components: *category topic matching* and *category affinity propagation*. The input of C*TM* is the textual mashup requirement and the output is functional relevance probability of each service category regarding mashup creation. In *CAP*, besides these functional relevance probabilities, the historical collaborative relationships among categories are taken into account. *CAP* leverages the functional relevance factor and category collaborative relationship to obtain each service category's comprehensive relevance probabilities at the output side. Finally, we rank all service categories according to their relevance probabilities: a category with a higher relevance probability has more relevance for the given mashup requirement. In the following two sections, we give the details of *CTM* and

Authorized licensed use limited to: SOUTHERN METHODIST UNIV. Downloaded on December 23,2020 at 05:10:47 UTC from IEEE Xplore. Restrictions apply.

TABLE 2
Notions in Service Category Relevance Ranking

Notions	Explanations
P_i^{TM}	Topic matching probability of category i
P_i^R	Relevance probability of category i
T_i^m	Topic i on mashup side
K_m	Total # of topics evolved in all mashups
P_I	$N_m \times N_c$ matrix, $P_I(i, j) = 1$ if services in category j is invoked by mashup M_i; otherwise $P_I(i, j) = 0$
MTF	$N_m \times K_m$ matrix, $MTF(i, j)$ represents the probability of topic j given mashup M_i
N_h	# of hidden units of 'Topic-Cateogry' Matching Machine
I_c	$N_h \times K_m$ matrix, input weight matrix of 'Topic-Category' Matching Machine
B_c	$N_h \times 1$ vector, bias vector of 'Topic-Category' Matching Machine
O_c	$N_h \times N_c$ matrix, output weight matrix of 'Topic-Category' Matching Machine
AF	$N_c \times N_c$ symmetric matrix, if $i \neq j$, $AF(i, j)$ equals the co-occurrence time of category C_i and category C_j in the same mashup; if $i = j$, AF(i, j) equals 0.
λ	Ranges from 0 to 1, the leveraged parameter of functionality and category affinity.

CAP. To avoid ambiguity, we list the additional notions used in *SCRR* in Table 2.

4.3.2 Category Topic Matching

Upon receiving the textual mashup requirement, which service categories will better satisfy the functinonal needs of mashup requirement? *CTM* solves this problem by predicting the topic matching probabilities of each service category regarding the given mashup. We first explain the workflow of *CTM* in Fig. 4 and then show the parameter training process.

Firstly, the mashup requirement is converted to mashup topic feature vector by topic model LDA. Secondly, the mashup topic feature is inputted to the 'Topic-Category' Matching Machine. Afterwards, the functional relevance probabilities of each category are obtained at the output side. Obviously, the role of 'Topic-Category' Matching Machine is to imitate the mapping function from mashup requirement to the functional relevance of each service category. We adopt a machine learning approach to learn the mapping function and train the parameters using historical data.

We use extreme learning machine (ELM) [23], whose parameters are composed of an input weight matrix I_c, a bias vector B_c, and an output weight matrix O_c, to learn the implicit mapping functions. The training data is obtained as follows. Combining *MS* matrix (illustrated in Table 1) and the result of service clustering, which categories are involved in each historical mashup can be inferred. We formally define this historical information in $P_I$matrix (illustrated in Table 2) that is the training data at the input side. Similar to *STF* matrix, we use topic model LDA to integrate the topic feature vectors of all textual mashup requirements to *MTF* matrix (illustrated in Table 2), which is the training data at the output side. Afterwards, we apply the standard ELM training process in [23] and use *MTF* matrix and P_I matrix to train the model. The mapping relation is embedded in I_c, B_c, and O_c, respectively. The details of the training algorithm are illustrated in Algorithm 2.

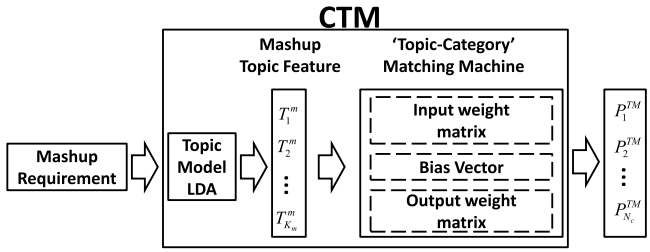


Fig. 4. Working process of category topic matching.

Algorithm 2. Parameter training of PTM

Input: *MTF* matrix, $P_I$matrix
Output: I_c, B_c, O_c
1: Initialize each element of matrix I_c by random value between [–1, 1]
2: Initialize each element of vector B_c by random value between [0, 1]
3: Extend vector B_c to a $N_h \times N_m$ matrix B_{Extend} by column
4: tempOutput $= I_c \cdot MTF^T + B_{Extend}$
5: Output $= e^{-tempOutput^2}$
6: $H^+ =$ Moore-penrose inverse matrix of Output
7: $O_c = (H^+)^T \bullet P_I$
8: Return I_c, B_c, O_c

This parameter training process consists of three main steps. In the first step, lines 1-2 randomly assign I_c and B_c. In the second step, lines 3-5 calculate the hidden layer output matrix *Output*. Finally, lines 6-8 calculate the Moore-Penrose inverse matrix of hidden layer output matrix *Output* and obtain O_c. Specificaly, we resort to the SVD approach for calculating Moore-Pensore inverse matrix.

4.3.3 Category Affinity Propagation

As aforementioned, some service categories are invoked together frequently in historical mashups. The co-occurrence times of two service categories in historical mashups indicates their affinity regarding mashup creation. Therefore, *CAP* learns the affinity among categories from historical data and quatifies the strength of collaboration among different service categories. Furthermore, *CAP* leverages the functional relevance probability of each category (obtained by *CTM*) and the affinity among service categories, then predicts the comprehensive relevance probability of each category according to the given mashup requirement.

In order to quantify the affinity among categories, *CAP* measures the affinity strength between two categories by the times of their co-occurrence in the same mashup in historical records. We formalize the affinity strength among all categories in *AF* matrix (illustrated in Table 2).

Afterwards, we introduce the following formula for calculating the comprehensive category relevance probability:

$$P_i^R = \lambda P_i^{TM} + (1 - \lambda) \sum_{k=1}^{N_c} \frac{AF(i, k) \cdot P_k^{TM}}{\sum_{l=1}^{N_c} AF(k, l)}.$$

Authorized licensed use limited to: SOUTHERN METHODIST UNIV. Downloaded on December 23,2020 at 05:10:47 UTC from IEEE Xplore. Restrictions apply.

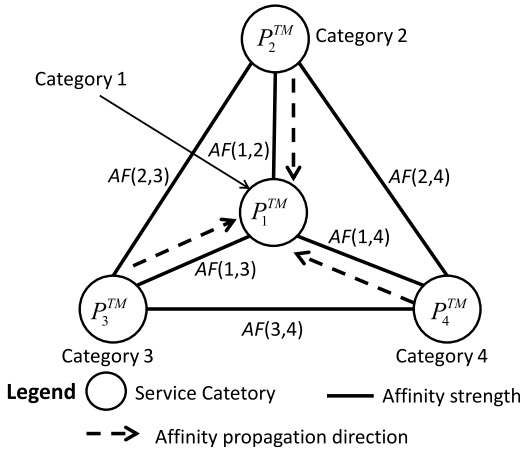


Fig. 5. Graphical representation of category affinity propagation. *AF*(*i*, *j*) represents the co-occurrence times of service category *i* and category *j* in historical mashups. Notice *AF* matrix is symmetric, thus *AF*(*i*, *j*) = *AF*(*j*, *i*). This figure shows how the *CAP* model leverages the functional relevance probability and affinity propagation to calculate the comprehensive relevance probability of category 1.

P_i^R denotes the comprehensive relevance probability of category i and consists of two parts: the λP_i^{TM} part represents category i's functional relevance component that can be obtained from *CTM* process; the other part represents the affinity propagation component from other categories. Here, we introduce a parameter λ to adjust the strength of the two parts. The calculating process by the graphical representation is illustrated in Fig. 5.

As shown in Fig. 5, let us assume four categories: $P_1^{TM}, \ldots, P_4^{TM}$ are their functional relevance probabilities given mashup requirement; *AF*(1, 2), …, *AF*(3, 4) are the affinity strength among the four categories. Assume that all the other three service categories have influence on category 1. *CAP* thus calculates the comprehensive relevance probability of category 1 as follows:

$$P_1^R = \lambda P_1^{TM} + (1-\lambda)\left(\frac{AF(1,2)\cdot P_2^{TM}}{AF(2,1)+AF(2,3)+AF(2,4)} + \frac{AF(1,3)\cdot P_3^{TM}}{AF(3,1)+AF(3,2)+AF(2,4)} + \frac{AF(1,4)\cdot P_4^{TM}}{AF(4,1)+AF(4,2)+AF(4,3)}\right).$$

The comprehensive relevance probability P_1^R of category 1 consists of two parts. For the functional relevance part, P_1^{TM} presents the functional relevance probability of category 1 itself regarding mashup requirement, and λ is the strength coefficient of P_1^{TM}. Obviously, high funtional relevance probability increases the comprehensive relevance probability. For the affinity propagation part, we take the affinity propagation from category 2 to category 1 as an example: *AF*(1, 2)/(*AF*(2, 1) + *AF*(2, 3) + *AF*(2, 4)) presents the normalization affinity from category 2 to category 1, then multiply with P_2^{TM} to obtain the relevance influence from category 2 to category 1. Obviously, if category 2 and category 1 has a high co-occurrence ratio in mashups, then the functional relevance probability of category 2 has more influence on category 1's comprehensive relevance probability. Following this way, the affinity propagation from category 3 and 4 to category 1 can be obtained.

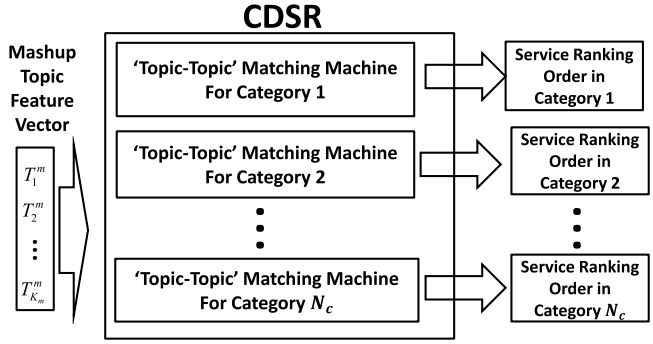


Fig. 6. Outline of category-aware distributed service recommendation model. In *CDSR*, each category is distributed with a specific machine learning component called '*Topic-Topic' Matching Machine*. Receiving the new mashup topic feature vector, the 'Topic-Topic' Matching Machine outputs the service ranking order within each category.

All the comprehensive relevance probabilities of other three categories can be obtained in the same way. Therefore, we can rank the categories according to their comprehensive relevance probabilities. The novelty of *CAP* is that it considers both the influence of functional relevance and category affinity propagation when calculating comprehensive relevance probability of a category introduces. Furthermore, a parameter λ is introduced for leveraging the strength of the two parts. In the experiment part, we will further discuss the impact of λ and how to tune it.

4.4 Category-Aware Distributed Service Recommendation Model

4.4.1 Model Description

Through the *SCRR* model, we have obtained the comprehensive relevance ranking order of different services categories. Afterwards, we aim to go a step further and tell the user "what's the ranking order of candidate services within each category." For this purpose, we have designed a *Category-aware Distributed Service Recommendation* model. The outline of *CRSR* model is shown in Fig. 6.

In *CDSR*, each service category is distributed with a 'Topic-Topic' Matching Machine. The input of each category's 'Topic-Topic' Matching Machine is the mashup topic feature vector; the output is the service ranking order within each category. In this process, the role of each category's 'Topic-Topic' Matching Machine aims to imitate the category-specific mapping function between mashup topic feature and service ranking order. Similar to 'Topic-Category' Matching Machine, the 'Topic-Topic' Matching Machine of category c is also an ELM-based unit with the three-layer structure: input weight matrix I_c^T, bias vector B_c^T, and output weight matrix O_c^T. We list the nontions involved in *CDSR* in Table 3, and show the parameter training process of *CRSR* in the next section.

4.4.2 Model Training

The model training process is composed of two stages. In the first stage, using the result of service category mining and historical service usage data, we obtain the training

Authorized licensed use limited to: SOUTHERN METHODIST UNIV. Downloaded on December 23,2020 at 05:10:47 UTC from IEEE Xplore. Restrictions apply.

TABLE 3
Notions in Category-Aware Distributed Service Recommendation

Notions	Explanations
S_c^T	$K_s \times 1$ vector, the predicted service topic feature for category *c*
N_c^T	# of hidden units of 'Topic-Topic' Matching Machine of category *c*
I_c^T	$N_c^T \times K_m$ matrix, input weight matrix of 'Topic-Topic' Matching Machine of category *c*
B_c^T	$N_c^T \times 1$ vector, bias vector of 'Topic-Topic' Matching Machine of category *c*
O_c^T	$N_c^T \times K_s$ matrix, output weight matrix of 'Topic-Topic' Matching Machine of category *c*
N_m^c	# of mashups involved in category *c*
S_{TF}^c	$N_m^c \times K_s$ matrix, service topic feature matrix for training 'Topic-Topic' Matching Machine of category *c*
C_{TF}^c	$N_m^c \times K_c$ matrix, mashup topic feature matrix for training 'Topic-Topic' Matching Machine of category *c*

data for 'Topic-Topic' Matching Machine of each category. In the second stage, we follow the standard ELM parameter training process in [23] and train the parameters of each category. The pseudo code of CDSR model training is shown in Algorithm 3.

Algorithm 3. Parameter training of CDSR

Input: $S_{ci} : 1 \le c \le N_c, 1 \le i \le N_{ce}$, *STF* matrix *MTF* matrix, P_I matrix
Output: $\{I_c^T, B_c^T, O_c^T, 1 \le c \le N_c\}$
Stage 1
1: **For** $c = 1$ to N_c
2: $N_m^c = \text{sum}(P_I(:, c))$
3: $S_{TF}^c = \text{zeros}(N_m^c, K_s)$
4: $C_{TF}^c = \text{zeros}(N_m^c, K_m)$
5: **End For**
6: **For** $i = 1$ to N_m
7: **For** $c = 1$ to N_c
8: **If** $P_I(i, c)$ equals 1
9: Assign the *i*th row of *MTF* matrix to the current beginning zero row of C_{TF}^c matrix
10: Randomly choose a service of category *c* used in mashup M_i, the index of the service is *j*
11: Assign the *j*th row of STF matrix to the current beginning zero row of S_{TF}^c matrix
12: **End If**
13: **End For**
14: **End For**
Stage 2
15: **For** $c = 1$ to N_c
16: Use $\{S_{TF}^c, C_{TF}^c\}$ train $\{I_c^T, B_c^T, O_c^T\}$
17: **End For**

Lines 1-5 initialize the matrix of training data. Line 2 counts the number of mashups involved in each category, thus confirms the dimension of training data matrix. Lines 3-4 intialize training data matrix as zero matrix. K_s presents the dimension of service topic feature vector; K_m presents the diemension of mashup topic feature vector. Lines 6-14 visit all the mashups, and obtain the training data for each category. Line 9 gathers the input-side training data, C_{TF}^c, for each category from *MTF* matrix. Lines 10-11 gather the output-side training data, S_{TF}^c, for each category from *STF* matrix. Lines 15-17 learn the parameters of distributed 'Topic-Topic' Matching Machines similar to Algorithm 2.

4.4.3 Category-Aware Distributed Service Recommendation

Combining the result of service category relevance ranking and the trained *CRSR* model, the category-aware distributed service recommendation result is obtained by Algorithm 4.

Algorithm 4. Category-aware distributed service recommendation

Input: $\{I_c^T, B_c^T, O_c^T, 1 \le c \le N_c\}$, mashup topic feature vector
Output: Category per candidate service ranking order
1: **For** $c = 1$ to N_c
2: Input the mashup topic feature vector to 'Topic-Topic' Matching Machine of category *c*
3: Obtain the predicted service topic feature vector S_c^T at the output side
4: Calculate the KL distance between S_c^T and all the services in category *c*
5: Rank all the services in category *c* according to their KL distance with S_c^T
6: **End For**
7: **Return** *each category's candidate service ranking list* together with the *category relevance ranking order*

Lines 1-6 follow the working process *CDSR* model and obtain the candidate service ranking list of different categories. Line 7 recommends service combining the service ranking order within a category and the category relevance ranking order. For example, if there are five categories, *CDSR* places the first service in each category at top five corresponding to the relevance order of the category; then *CDSR* places the second service in each category at 6 to 10 also corresponding to the relevance order of their categories, and so on.

5 Experiments

We have conducted a series of experiments on ProgrammableWeb dataset, to compare the prediction accuracy of our proposed *CDSR* approach with other state-of-the-art service recommendation methods.

5.1 Data Set Description

The metadata of services and mashups are crawled from ProgrammableWeb.com, in the range from June 2005 to June 2013. In this data set, the textual information of each service consists of its *description file*, *service tags* and *summary*; the textual description of each mashup consists of its *requirement text*, *mashup tags*, *summary* and the list of services invoked by it. Furthermore, every service in the repository is organized into a certain category by manual work.

After removing meaningless or vacant mashups and services, we obtained a collection of 6,813 mashups and

Authorized licensed use limited to: SOUTHERN METHODIST UNIV. Downloaded on December 23,2020 at 05:10:47 UTC from IEEE Xplore. Restrictions apply.

TABLE 4
Data Set on ProgrammalbeWeb.com

Statistics	Values
Total # of services	7,186
Total # of compositions	6,813
Total # of service categories of PW	62
The # of services used in at least one composition	1,155
Total # of terms in composition textual corpus	205,494
Total # of terms in services textual corpus	350,101

7,186 services. The details of the data set is illustrated in Table 4.

5.2 Evaluation Metrics

In this paper, we choose the following metrics from information retrieval to evaluate the recommendation performance.

5.2.1 NDCG@N

Normalized Discounted Cumulative Gain @ top N services in ranking list is defined as follows:

$$NDCG@N = \frac{1}{S_N}\sum_{j=1}^{N}\frac{(2^{r(j)}-1)}{\log_2(1+j)}.$$

Where $r(j)$ is the relevant score (0 or 1) of the jth recommended service on the ranking list; and S_N represents the ideal maximum score that the cumulative component can reach.

5.2.2 MAP@N

Mean Average Precision @ top N services in ranking list is defined as follows:

$$MAP@N = \frac{\sum_{r=1}^{N}\left(\frac{N_r}{r}\bullet I(r)\right)}{N_{used}}.$$

Where N_r denotes the number of actual used services in the top r services of the ranking list; I(r) indicates whether the service at ranking position r is actually used; and N_{used} represents the total number of actual used services in composition.

5.3 Baseline Methods

We chose the following six types of methods for comparison: the details of how we used the baseline methods in our experiments are shown as below:

1) *TF-IDF*: For TF-IDF method, we first obtain the Term frequency Document Matrix (TDM) from the corpus. Afterwards, TDM is transferred to Tf-idf Weighted-Document Matrix (TWDM). Next, services are recommended according to their tf-idf weight similarity with composition requirement, measured by KL divergence defined as follows:

$$D_{KL}(query||service) = \sum_{i=1}^{N} TWDM(i,q)\log\frac{TWDM(i,q)}{TWDM(i,s)}.$$

Where N denotes the total number of term; *TWDM* (*i*,*q*) denotes the tf-idf weight of *i*th term given the *query*; and *TWDM*(*i*,s) denotes the tf-idf weight of the *i*th term given a *service*.

2) *PopK*: For PopK method, we refer to [24]. In this context, the popularity of a service is measured by the usage frequency by compositions from historical information. Only the top K services in relative service categories are recommended according to the query.
3) *SCTM*: *Single Category Topic Matching* (*SCTM*) is a special case of CDSR. SCTM considers the entire services as one category, which means that it only recommends all candidate services in a single ranking list.
4) *Km-SCTM*: For Kmeans-SCTM method, we use the traditional Kmeans algorithm to cluster services into different categories according to their topic feature vector's KL divergence, then resort to SCTM to recommend services within each category.
5) *OTM*: *Original Topic Matching* (*OTM*) recommends services based on the original service categories provided by ProgrammableWeb. Such categories are divided by manual work.
6) *DTM*: *Domain Topic Matching* (*DTM*) is a subset of CDSR. The major difference of CDSR over DTM is that CDSR considers the affinity propogation among service categories when predicting the relevant categories according to mashup requirement.

5.4 Experiment Results

In this section, we will discuss our experimental results. We will first compare the recommendation performance on the overall dataset using different methods. Then, we will compare different methods on long-tail recommendation, which reflects the diversity of recommendation. After that, we will discuss the impact of category amount. Finally, we will discuss the impact of parameterλ.

5.4.1 Overall Recommendtion Comparison

We have conducted a 10-fold cross validation on the overall dataset to evaluate different methods: TF-IDF, PopK, SCTM, Km-SCTM (fix category amount at 10), OTM, DTM (fix category amount at 10), CDSR (fix category amount at 10 and fix λ at 0.5). After parameters trained, we use the mashup textual requirement as input of recommendation methods and return a ranking list of candidate services. A service in the list is considered a positive recommendation only if it is actually used in the mashup; otherwise, it is considered a negative recommendation. With varying N, the length of recommendation list, we report the NDCG and MAP of the seven methods in Fig. 7.

As illustrated in Fig. 7, CDSR shows a superior recommendation performance over other methods. Let us take MAP@20 for example: CDSR (71.03 percent) is higher than DTM (67.63 percent), OTM (64.52 percent), Km-SCTM (55.5 percent), PopK (46.72 percent), SCTM (38.28 percent), and TF-IDF (20.1 percent). The interpretation of this result is as follows. Since TF-IDF is only a functional keywords-based method, it shows a relative poor performance. PopK performs relatively well because the service usage pattern on this dataset presents a strong power-law distribution, which means that the popular

Authorized licensed use limited to: SOUTHERN METHODIST UNIV. Downloaded on December 23,2020 at 05:10:47 UTC from IEEE Xplore. Restrictions apply.

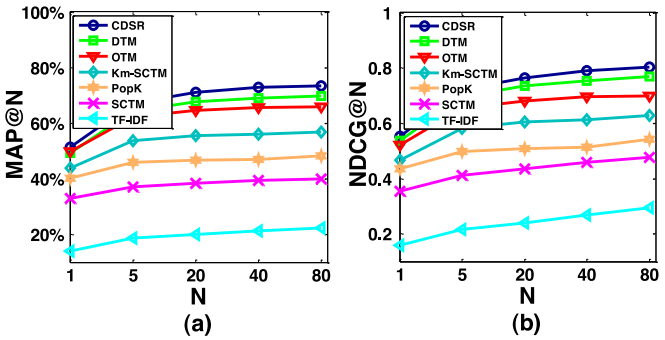


Fig. 7. MAP and NDCG on overall service recommendation.

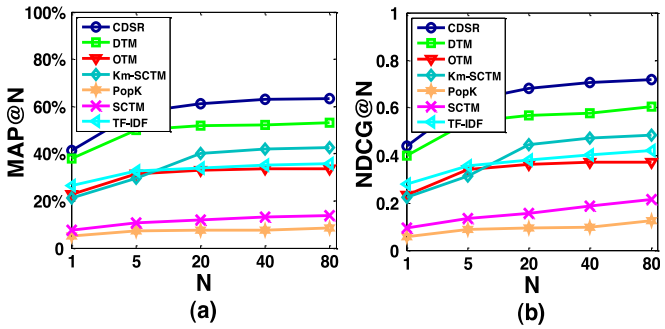


Fig. 8. MAP and NDCG on longtail service recommendation.

service may gain more preference from developers [17]. SCTM considers both functionality and service popularity; however, it recommends services in a single list without considering to which categories they belong. Thus it leads to meanlingless ranking and restricts its performance. Both Km-SCTM and OTM cluster services first and then recommend services within each category. But the service clustering part of these two methods only considers service functional similarity. Our previous method DTM remedies this flaw and takes the service popularity into account when clustering services. However, DTM overlooks the factor of collaborative history among different service categories.

Compared with all above baseline methods, three advancements contribute to our CDSR's superiority. First, the service clustering strategy of CDSR is better than traditional functionality-based algorithm and manual work. Second, CDSR considers the collaborative influence among service categories, thus promotes the category relevance ranking. Third, CDSR recommends services in different categories according to their category-specific service usage patterns, thus eliminates the meanlingless ranking among services from different categories.

5.4.2 Recommendation Diversity Comparison

In recent years, long tail recommendation gains a lot of momentum because it reflects the diversity of the recommendation results [25]. In the context of service recommendation for mashup, long tail services refer to those sevices not popular and only used in few mashups. According to our previous empirical work [17], the service usage pattern on ProgrammableWeb.com shows a significant power-law distribution: nearly 5 percent popular services ever occurred in 78 percent mashups and the other 95 percent services are in long tail. Therefore, the potential value of 95 percent services in long tail should not be ignored. We have evaluated the performance of different methods on long tail recommendation (Fig. 8).

Comparing Figs. 7 and 8, we can see that most methods are subject to the power law property and show different performances on long tail recommendation. PopK sharply decreases in the two metrics, because it only recommends the most popular services in each category but ignores the services in longtail. SCTM recommends services from the entire repository and suffers significantly from power law distribution. On the contrary, TF-IDF becomes better in longtail recommendation as it only cares about functionality and keyword, which helps it get rid of the influence of power law distribution. Km-SCTM and OTM also descrease a lot when it comes to long tail recommendation. DTM reforms the services clustering strategy and performs better. Our proposed CDSR method further considers the collaborative influence among different service categories, which in turn further remedies the influence of power law and improves the precision in longtail recommendation.

5.4.3 Impact of Service Category Amount

Our proposed CDSR method provides a mechanism to adjust the amount of service category amount. Larger service category amount means a finer-grained service clustering strategy and smaller service category amount means a coarser-grained service clustering strategy. We have designed expeirments to study the impact of service category amount.

Fig. 9 shows CDSR's result of MAP and NDCG with different service category amounts (fix λ at 0.5). In some range, CDSR shows a general tendency of higher prediction precision with more service category amout. Let us take Fig. 9c for example. When there are only two categories, CDSR shows a relative low precision rate at both overall (e.g., MAP@20 is 38.26 percent) and longtail (e.g., MAP@20 is 29.65 percent) recommendation. When the service category amount goes to ten, the precision rate increases sharply at overall (e.g., MAP@20 is 71.03 percent) and longtail (e.g., MAP@20 is 61.26 percent) recommendation. However, if the category amount continues to grow, for example to reach 20, the precision rate of overall performance decreases in some degree (e.g., MAP@20 is 66.07 percent) while longtail performance almost remains at the same level (e.g., MAP@20 is 61.93 percent). The reason may be too large preseted category amount may weaken too much the influence of power law distribution and the status of those popular services. These observations imply that we should choose appropriate service category amount for superior recommendation performance.

5.4.4 Impact of λ

As aforementioned in the model part, CDSR considers the relevance of service category with mashup requirement is the result of two factors: the functionality relevance factor and the categories collaboration factor. The parameter λ is the coefficient to control the strength of the two factors. The larger λ is, the functionality relevance factor has more strength and the categories collaboration factor has less strength, and vise versa. In this section, we discuss the impact of λ on CDSR's recommendation performance.

Authorized licensed use limited to: SOUTHERN METHODIST UNIV. Downloaded on December 23,2020 at 05:10:47 UTC from IEEE Xplore. Restrictions apply.

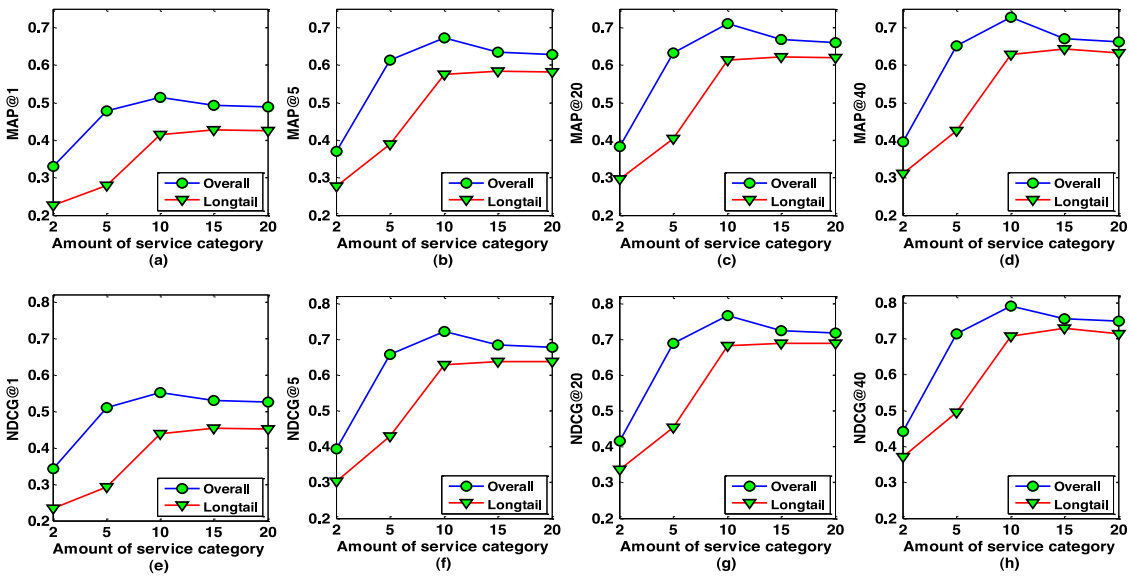


Fig. 9. Impact of category amount on CDSR. Large service category amount means a finer-grained service clustering strategy while small amount means a coarser-grained strategy. In our experiment, to fix the service category amount at 10 may be a wise choice.

Fig. 10 demonstrates CDSR's overall and longtail performance with a range of λ (fix service category amount at 10). When λ is 0, both of overall and longtail show low precision rates, which means taking no consideration of functionality relevance factor is unreasonable. As we gradually increase the value ofλ, the functionality relevance factor and categorires collaboration factor are leveraged: when λ is 0.5, the overall precision rate nearly reaches its peak point; when λ is 0.75, the longtail precision rate reaches nearly its peak point. If we preset λ at 1, both of overall and longtail precision rate decrease, which confirms that categories collaboration factor should be taken into account. Furthermore, the value of λ can be flexibly adjusted to satisfy different requirements: either high overall prediction accuracy rate or high longtail prediction accuracy rate.

6 Related Work

In this paper, we have presented an integrated method for automatic mashup creation, which combines service clustering and service recommendation. Therefore, we review related work from two aspects: service clustering and service recommendation.

6.1 Service Clustering

With the rapid increasing number of services in a repository, proper service clustering has become critical to facilitate service searching process.

Functional similarity is the basis of service clustering. Khalid et al. [26] analyze WSDL documents and cluster them into groups based on functional similarity. Using WSDL documents as well, Cristina et al. [27] propose an

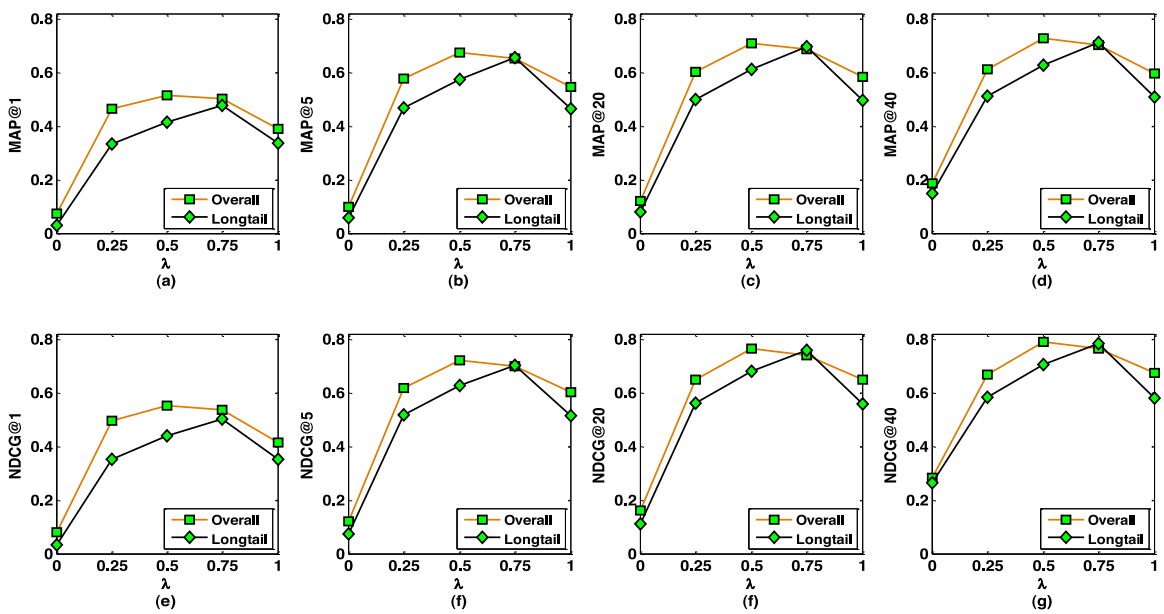


Fig. 10. Impact of λon CDSR. λ leverages the strength of functional factor and category collaborative factor when predicting the relevance service category regarding mashup creation. The overall and longtail peak points correspond to different value of λ. The CDSR model can be adapted to different recommendation purpose by adjusting parameter λ.

Authorized licensed use limited to: SOUTHERN METHODIST UNIV. Downloaded on December 23,2020 at 05:10:47 UTC from IEEE Xplore. Restrictions apply.

ant-based service clustering method that groups services based on semantic similarity. Chen et al. [28] believe utilizing WSDL documents alone for service clustering limits its accuracy. Thus, they develop an approach called *WTCluster*, in which tags are taken into consideration in addition to WSDL documents.

Non-functional factors are also considered by many researchers for enhancing service clustering process. Zhou et al. [29] consider dynamic situations when a data providing (DP) service emerages or disappears, supported by a dynamic fuzzy *C*-means algorithm. Zhang et al. [30] considers the hierarchical structure within each service category. In their two-phase service clustering method, the traditional K-means method is first used to cluster services into different groups, followed by an algorithm to construct the hirerarch within each service category. Skoutas et al. [31] develop a thematically service clustering method, where services are categorized into groups based on customizable query parameters.

Our service clustering method proposed in this paper takes into account both functional and non-functional factors. Different from exsiting works, we learn from our previous empirical work and summarize the formation process of a service category. After a core service gains popularity, follower services imitate the core service and join the service repository. Inspired by the typical service category formation process, our proposed clustering method first identifies the core services of each category, then clusters other non-core services.

6.2 Service Recommendation

In recent years, a number of recommendation methods have been developed for service compositions.

Semantic matching approaches recommend services based on semantic relevance between services and queries. Meng et al. [9] develop a keyword-aware service recommendation method named KASR. Key words are used to indicate users' preferences, and a user-based collaborative filtering algorithm is adopted to generate appropriate recommendations. Unlike this exact keyword-based matching method, Li et al. [8] use probabilistic topic model LDA to extract functional attributes of services from WSDL documents. Their recommendation mthod is based on the topic-level semantic matching probability.

QoS prediction approaches recommend services from the angle of QoS optimization. Tang et al. [32] consider locations of both users and services when predicting QoS values of Web services and recommend service candidates. Ahmed et al. [10] propose a Hidden Markov Models (HMM) method for QoS metrification, which measures and predicts the behavior of web services in terms of response time. Zheng et al. [12] use matrix factorization technique to develop a collaborative QoS prediction approach for web services, taking advantages of the past web service usage experiences of services users.

Social relationship-based approaches use the collaborative relation to enhance the service recommendation. Cao et al. [13] extract users' interests from their mashup service usage history, and build a social network based on social relationships information to support service recommendation for mashup. Xu et al. [14] present a social-aware service recommendation approach, where multi-dimensional social relationships among potential users, topics, mashups, and services are described by a coupled matrix model.

Regardless of clustering methods, most existing recommendation methods directly recommend from all services and offer a single service candidates ranking list. However, mashup inherently demand services from different categories and users are not always clear about which categories can satisfy their mashup query. Therefore, different from existing methods, our recommendation method in this paper offers "per category service candidate ranking list" recommendation.

7 Conclusions

With the rapid increasing number of published services on the Internet, how to effectively and efficiently recommend proper services for automatic mashup creation has become an imporatant issue. Two main deficiencies affects the accuracy of existing recommending approaches: first, most existing recommendation approaches only provide a single service ranking list without considering which categories the services belong to, which leads to meanlingless ranking for mashup creation; second, this type of recommendation neglect the situation when mashup developers are not clear about which categories they need to fulfil the requirement. Overcoming these two deficiencies and improving the service recommendation for automatic mashup creation motivate our work.

In this paper, a three-step approach is presented to enhance the recommendation for mashup creation. We first cluster services according to the formation process of different categories in repository, which provides basis for recommending. Then, a service category relevance ranking model is proposed to decompose mashup requirement and explicitly predict the relevant service categories. Finally, a distributed machine learning method is introduced for category-aware service recommendation. Our experiments on real-world datasets have shown that our method gains a 30 percent improvement on accuracy rate and a 20 percent improvement for long-tail recommendation.

From a dynamic view, new services keep on joining the repository; published services may be no longer available; service's popularity and usage patterns for mashup creation are both evolving overtime. In the future, we will extend our method and further consider these dynamic factors when recommending service for mashup creation.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China (No. 61033005, No. 61174169) and the Independent Research Fund of Tsinghua University (No. 20111080998) and the Specialized Research Fund for the Doctoral Program of Higher Education (No. 20120002110034). Yushun Fan is the corresponding author.

References

[1] L.-J. Zhang, J. Zhang, and H. Cai, *Services Computing*. New York, NY, USA: Springer, 2007.

[2] W. Yi and M. B. Blake, "Service-oriented computing and cloud computing: Challenges and opportunities," *IEEE Internet Comput.*, vol. 14, no. 6, pp. 72–75, Nov./Dec. 2010.

Authorized licensed use limited to: SOUTHERN METHODIST UNIV. Downloaded on December 23,2020 at 05:10:47 UTC from IEEE Xplore. Restrictions apply.

[3] M. Maleshkova, C. Pedrinaci, and J. Domingue, "Investigating web APIs on the world wide web," in *Proc.IEEE 8th Eur. Conf. Web Services*, Ayia Napa, Cyprus, 2010, pp. 107–114.
[4] E. Al-Masri and Q. H. Mahmoud, "Investigating web services on the world wide web," in *Proc. 17th Int. Conf. World Wide Web*, Beijing, China, 2008, pp. 795–804.
[5] L. Chen, J. Wu, H. Jian, H. Deng, and Z. Wu, "Instant recommendation for web services composition," *IEEE Trans. Services Comput.*, vol. 7, no. 4, pp. 586– 598, Oct.-Dec. 2014.
[6] P. Leitner, W. Hummer, and S. Dustdar, "Cost-based optimization of service compositions,"*IEEE Trans. Services Comput.*, 2013, vol. 6, no. 2, pp. 239–251, Apr.–Jun. 2013.
[7] L. Xuanzhe, H. Yi, S. Wei, and L. Haiqi, "Towards service composition based on mashup," in *Proc. IEEE World Congress Services*, Salt Lake City, UT, USA, 2007, pp. 332–339.
[8] C. Li, R. Zhang, J. Huai, X. Guo, and H. Sun, "A probabilistic approach for web service discovery," in *Proc. IEEE Int. Conf. Services Comput.*, Santa Clara, CA, USA, 2013, pp. 49–56.
[9] S. Meng, W. Dou, X. Zhang, and J. Chen, "KASR: A keyword-aware service recommendation method on mapreduce for big data application," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 12, pp. 3221–3231, Dec. 2014.
[10] W. Ahmed, Y. Wu, and W. Zheng, "Response time based optimal web service selection," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 2, pp. 551–561, Feb. 2015.
[11] Q. Yu, Z. Zeng, and H. Wang, "Trace norm regularized matrix factorization for service recommendation," in *Proc. IEEE 20th Int. Conf. Web Services*, Santa Clara, CA, USA , 2013, pp. 34–41.
[12] Z. Zheng, H. Ma, M. R. Lyu, and I. King, "Collaborative web service qos prediction via neighborhood integrated matrix factorization," *IEEE Trans. Services Comput.*, vol. 6, no. 3, pp. 289–299, Jul.–Sep. 2013.
[13] B. Cao, J. Liu, M. Tang, Z. Zheng, and G. Wang, "Mashup service recommendation based on user interest and social network," in *Proc. IEEE 20th Int. Conf. Web Services*, Santa Clara, CA, USA, 2013, pp. 99–106.
[14] W. Xu, J. Cao, L. Hu, J. Wang, and M. Li, "A social-aware service recommendation approach for mashup creation," in *Proc. IEEE 20th Int. Conf. Web Services*, Santa Clara, CA, USA, 2013, pp. 107–114.
[15] Y. Zhou, L. Liu, C.-S. Perng, A. Sailer, I. Silva-Lepe, and Z. Su, "Ranking services by service network structure and service attributes," in *Proc. IEEE 20th Int. Conf. Web Services*, Santa Clara, CA, USA, pp. 26–33.
[16] J. Zhang, J. Wang, P. C. K. Hung, Z. Li, J. Liu, and K. He, "Leveraging incrementally enriched domain knowledge to enhance service categorization," *Int. J. Web Services Res.*, vol. 9, no. 3, pp. 43–66, 2012.
[17] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learning Res.*, vol. 3, pp. 993–1022, 2003.
[18] G. Cassar, P. Barnaghi, and K. Moessner, "Probabilistic matchmaking methods for automated service discovery," *IEEE Trans. Services Comput.*, vol. 7, no. 4, pp. 654–666, Oct.-Dec. 2014.
[19] K. Huang, Y. Fan, and W. Tan, "An empirical study of programmable web: A network analysis on a service-mashup system," in *Proc. IEEE 19th Int. Conf. Web Services*, 2012, pp. 552–559.
[20] W. Tan, J. Zhang, and I. Foster, "Network analysis of scientific workflows: A gateway to reuse," *IEEE Comput.*, vol. 43, no. 9, pp. 54–61, Sep. 2010.
[21] J. Zhang, W. Tan, J. Alexander, I. Foster, and R. Madduri, "Recommend-as-you-go: A novel approach supporting services-oriented scientific workflow reuse," in *Proc. IEEE Int. Conf. Services Comput.*, Washington DC, USA, , Jul. 4–9, 2011, pp. 48–55.
[22] M. N. Do and M. Vetterli, "Wavelet-based texture retrieval using generalized gaussian density and kullback-leibler distance," *IEEE Trans. Image Process.*, vol. 11, no. 2, pp. 146–158, Feb. 2002.
[23] G. Huang, Q. Zhu, and C. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, pp. 489–501, 2006.
[24] K. Huang, Y. Fan, and W. Tan," "Recommendation in an evolving service ecosystem based on network prediction," *IEEE Trans. Autom. Sci. Eng.*, 2014, vol. 11, no. 3, pp. 906–920, Jul. 2014.
[25] H. Yin, B. Cui, J. Li, J. Yao, and C. Chen, "Challenging the long tail recommendation," in *Proc. 38th Int. Conf. Very Large Data Bases*, Istanbul, Turkey, Aug. 27–31, 2012, pp. 896–907.
[26] K. Elgazzar, A. E. Hassan, and P. Martin, "Clustering WSDL documents to bootstrap the discovery of web services," in *Proc. IEEE Int. Conf. Web Services*, Miami, FL, USA, 2010, pp. 147–154.
[27] M. Essaaidi, M. Malgeri, C. Badica, C. B. Pop, V. R. Chifu, I. Salomie, M. Dinsoreanu, T. David, and V. Acretoaie, *Semantic Web Service Clustering for Efficient Discovery Using an Ant-Based Method*, in *Intelligent Distributed Computing IV*, M. Es-Saaidi, M. Malgeri, and C. Badica, Eds. Berlin, Heidelberg: Springer, 2010, pp. 23–33.
[28] G. Kappel, Z. Maamar, H. R. Motahari-Nezhad, L. Chen, L. Hu, Z. Zheng, J. Wu, J. Yin, Y. Li, and S. Deng, "WTCluster: utilizing tags for web services clustering," in *Proc. Int. Conf. Service-Oriented Comput.*, 2011, pp. 204–218.
[29] Z. Zhou, M. Sellami, W. Gaaloul, M. Barhamgi, and B. Defude, "Data providing services clustering and management for facilitating service discovery and replacement," *IEEE Trans. Autom. Sci. Eng.*, vol. 10, no. 4, pp. 1131–1146, Oct. 2013.
[30] L.-J. Zhang, S. Cheng, C. K. Chang, and Q. Zhou, "A pattern-recognition-based algorithm and case study for clustering and selecting business services," *IEEE Trans. Syst., Man Cybernetics, Part A: Syst. Humans*, vol. 42, no. 1, pp. 102–114, Jan. 2012.
[31] D. Skoutas, D. Sacharidis, A. Simitsis, and T. Sellis, "Ranking and clustering web services using multicriteria dominance relationships," *IEEE Trans. Services Comput.*, vol. 3, no. 3, pp. 163–177, Jul.–Sep. 2010.
[32] M. Tang, Y. Jiang, J. Liu, and X. Liu, "Location-aware collaborative filtering for qos-based service recommendation," in *Proc. IEEE 19th Int. Conf. Web Services*, 2012, Honolulu, HI, USA, pp. 202–209.

Bofei Xia is currently working toward the PhD degree in the Department of Automation, System Integegration Institute Tsinghua University, Beijing, China. His research interests include services computing, service recommendation and big data.

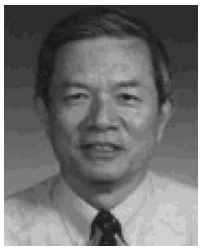
Yushun Fan received the PhD degree in control theory and application from Tsinghua University, China, in 1990. He is currently a professor with the Department of Automation. His research interests include enterprise modeling methods and optimization analysis, business process reengineering, workflow management, system integration, object-oriented technologies and flexible software systems, Petri nets modeling and analysis, and workshop management and control.

Wei Tan received the BS and PhD degrees from the Department of Automation, Tsinghua University, China, in 2002 and 2008, respectively. He is currently a research staff member with the IBM T. J. Watson Research Center, NY. His research interests include NoSQL, big data, cloud computing, service-oriented architecture, business and scientific workflows, and Petri nets.

Keman Huang received the BS degree in automation and another BS degree in economics from Tsinghua University, China, in 2009. He received the PhD degree in control theory and application in 2014 from Tsinghua University, China. He is currently an assistant professor with the School of Computer Science and Technology, Tianjin University, China. His research interests include services computing, web service composition, social network analysis, data mining, and service recommendation. He is a member of the ACM and IEEE.

Authorized licensed use limited to: SOUTHERN METHODIST UNIV. Downloaded on December 23,2020 at 05:10:47 UTC from IEEE Xplore. Restrictions apply.

Jia Zhang received the BS and MS degrees in computer science from Nanjing University, China. She received the PhD degree in computer science from the University of Illinois at Chicago. She is currently an associate professor at the Department of Electrical and Computer Engineering, Carnegie Mellon University. Her recent research interests center on service-oriented computing, with a focus on collaborative scientific workflows, Internet of Things, cloud computing, and big data management. She has coauthored one textbook titled *Services Computing* and has published more than 130 refereed journal papers, book chapters, and conference papers. She is currently an associate editor of the *IEEE Transactions on Services Computing (TSC)* and of *International Journal of Web Services Research (JWSR)*, and an editor-in-chief of *International Journal of Services Computing (IJSC)*.



Cheng Wu received the BS and MS degrees in electrical engineering from Tsinghua University, Beijing, China. He is currently a fellow of Chinese Academy of Engineering. His research interests include complex system modeling and optimization, and modeling and scheduling in supply chains.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.

Authorized licensed use limited to: SOUTHERN METHODIST UNIV. Downloaded on December 23,2020 at 05:10:47 UTC from IEEE Xplore. Restrictions apply.