This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2681666, IEEE Transactions on Services Computing

DLTSR: A Deep Learning Framework for Recommendations of Long-tail Web Services

Bing Bai, Yushun Fan, Wei Tan, *Senior Member, IEEE,* Jia Zhang, *Senior Member, IEEE*

Abstract—With the growing popularity of web services, more and more developers are composing multiple services into mashups. Developers show an increasing interest in non-popular services (i.e., long-tail ones), however, there are very scarce studies trying to address the long-tail web service recommendation problem. The major challenges for recommending long-tail services accurately include severe sparsity of historical usage data and unsatisfactory quality of description content. In this paper, we propose to build a deep learning framework to address these challenges and perform accurate long-tail recommendations. To tackle the problem of unsatisfactory quality of description content, we use stacked denoising autoencoders (SDAE) to perform feature extraction. Additionally, we impose the usage records in hot services as a regularization of the encoding output of SDAE, to provide feedback to content extraction. To address the sparsity of historical usage data, we learn the patterns of developers' preference instead of modeling individual services. Our experimental results on a real-world dataset demonstrate that, with such joint autoencoder based feature representation and content-usage learning framework, the proposed algorithm outperforms the state-of-the-art baselines significantly.

Index Terms—Deep learning, mashup creation, service recommendation, long-tail

1 Introduction

With the growing popularity of Service-oriented computing (SOC) [1] and Internet of service (IoS) [2], more and more developers are benefiting from reusing web-based services (usually in the form of web APIs). Making software as a service allows users/developers to integrate software components from different providers, and ultimately make a value-added software composition (e.g., mashup [3]). Nowadays, various mashups have been published, and many services (such as *Google Maps* and *Twitter*) have become well-known and frequently used. As common demands have been met to a relatively large extent by these frequently used services, we find that mashup developers are beginning to investigate the potential of non-popular "long-tail" services.

Figure 1 shows some statistics about long-tail services (services with < 5 usage records, i.e., showing up in < 5 mashups) and hot ones (services with ≥ 5 usage records) in a representative web service ecosystem, i.e., ProgrammableWeb.com which is the largest online repository for web services [4], [5]. As we can see, about 70% of mashups published after 2014 involve at least one long-tail service, while in 2010, this number was only 40%. Moreover, long-tail services make up about 50% of the total service consumption by mashups after 2014, while in 2010, this number was only about 20%. This shows the fact that mashup developers are tending to consume more long-tail services now than before. Such growing interest calls for more effective long-tail web service recommendations.

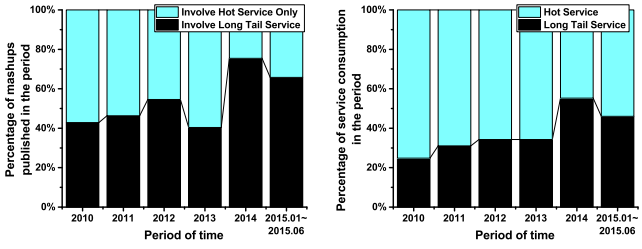


Fig. 1: Statistics about long-tail service and hot service consumption in ProgrammableWeb. In recent years, a growing percentage of mashups are published with long-tail web services involved, and the share of long-tail service consumption also rises.

From the perspective of individual web developers, long-tail web service recommendations are also helpful. For example, *Facebook* and *LinkedIn* are the popular services of "Social" category, however, if developers want to build mashups that can provide a "platform for readers to share information about what eBooks they are reading," *Readmill*, which is a long-tail service, would be a better choice. For developers with such demand, recommending *Readmill* are with more value-add compared with recommending well-known popular services such as *Facebook* and *LinkedIn* [6].

Therefore, we argue that accurate long-tail service recommendations can serve as a helpful complement to traditional service recommendations. However, traditional methods, especially collaborative filtering based ones, often fail to perform well on the long-tail side due to the sparsity problem [7], and tend to be biased towards popular, well-known services [8]. Moreover, there is very scarce work on long-tail service recommendations. Existing service rec-

- *B. Bai and Y. Fan are with the Department of Automation, Tsinghua University, Beijing, China. E-mail: bb13@mails.tsinghua.edu.cn, fanyus@tsinghua.edu.cn.*
- *W. Tan is with the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, NY. E-mail: wtan@us.ibm.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 July 14th, 2016; revised March 2nd, 2017.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2681666, IEEE Transactions on Services Computing

ommendation techniques are majorly either QoS-based or functionality-based. QoS-based algorithms aim at predicting the unknown non-functional properties (such as reliability, availability and response time) of a certain service, so they hardly give any help for finding unknown but interesting long-tail services [9], [10], [11], [12]. As for existing functionality-based algorithms, they do not focus on recommendations on the long-tail side [4], [5], [13], [14], [15].

The major challenges to perform high-performance long-tail web service recommendations include:

- Severe sparsity of historical usage data. The sparsity of historical usage data on the long-tail side strongly limits the applicability of traditional latent factor based collaborative filtering methods [7]. In addition, previous works [13], [16], [17] show that tightly coupling content extraction with the usage data can allow usage data to provide feedback, and guide the content extraction to achieve better performance. However, considering the severe sparsity of usage records in the long-tail side, tightly coupling the long-tail ratings will be of little help.
- Unsatisfactory quality of developer-provided service descriptions, including textual descriptions, tags, category information, protocols and so on. As the usage data is extremely sparse, we have to rely more on the description, however, different developers can use different terminologies, and may put insufficient, ambiguous, or even incorrect descriptive content.

To address the aforementioned challenges, we present a deep learning framework called DLTSR (Deep learning for Long-Tail web Service Recommendations). The overview of the proposed approach is illustrated in Figure 2.

To tackle the problem of unsatisfactory quality of descriptions, we use Stacked Denoising AutoEncoders (SDAE) [18] as the basic component in our framework, and use masking noise [18] to simulate the uncertainty of word (term) choosing by developers. Thanks to the deep denoising neural network, the model can capture a better representation of services and mashup queries. For further performance improvement, we impose the hot service usage data (which is relative rich) as a regularization on the encoding output of SDAE, thus enable a feedback from usage to content extraction. Therefore, our model can understand which service is functionally relevant to the mashup creation query better.

As the long-tail service usage records are too sparse to build high-quality latent factor models for every individual services [4], [13], we utilize long-tail service usage in a different manner, i.e., modeling the preference of mashup developers. We factorize the rating matrix into a linear combination of functional relevance and several pre-defined factors (i.e., side information, including update time and usage counts of long-tail services in this paper). In this way, we can prevent overfitting in such severe sparse condition, and besides, the performance of recommending cold-start services can also benefit. Although cold-start services do not come up with historical usage records, their side information still varies.

Through incorporating the above-mentioned ideas in a deep neural network, we can achieve a significantly better performance of long-tail web service recommendations. The main contributions of this paper are as follows.

- We propose DLTSR, a deep learning framework to address a gradually emergent challenge in web service economy, i.e., the long-tail web service recommendations.
- We use SDAE as a foundation. The transferred knowledge from usage in the hot service side, and the modeling of the developers' preference are incorporated tightly with the SDAE part to boost the performance of long-tail service recommendations.
- Experiments on real-world dataset from ProgrammableWeb show that DLTSR gains an improvement of 4.1% on Recall@5, 12.0% on Recall@50, and 7.1% on Recall@250, compared with the state-of-the-art baseline (modified) TCDR[1] [13].

The rest of the paper is organized as follows. Section 2 introduces the background of this paper and defines the problem, Section 3 gives an overview of our methodology, then Section 4 shows how the optimal parameters are learned from data and how to use the trained model to make recommendations, Section 5 shows the experimental results, Section 6 summarizes the related work, and Section 7 gives some conclusion remarks.

2 Problem Formulation

In this section, the background about service ecosystem is introduced, then the problem of long-tail service recommendations for mashup creation is formulated.

Definition 1 (Long-tail services and hot services). Long-tail services refer to services that are hardly used before. In this paper, a service that has been used less than 5 times is considered a long-tail service. On the contrary, hot services refer to the popular ones which have been used equal to or more than 5 times.

Definition 2 (Service ecosystem). In this paper, we use the set $SE = \{M, \mathbf{X}_c, MT, S, \mathbf{Y}_c, ST, \mathbf{R}\}$ to model the service ecosystem. M is the set of I mashups, $S = LTS \bigcup HS$ is the set of J services, and LTS contains all the long-tail services while HS contains all the hot serivces. $\mathbf{X}_c$ is a I-by-W matrix that records the description content of mashups, where row i is the bag-of-words (terms) vector $\mathbf{X}_{c,i*}$ for mashup i based on a vocabulary of size W as in [16]. $\mathbf{Y}_c$ and $\mathbf{Y}_{c,j*}$ follows the similar definition on the service side. $MT = \{t_1^m, t_2^m, \ldots, t_I^m\}$ records the release time of mashups, and $ST = \{t_1^s, t_2^s, \ldots, t_J^s\}$ records the release time of services. $\mathbf{R}$ is the I-by-J binary rating matrix where $r_{ij} = 1$ when service j is invoked by mashup i, and $r_{ij} = 0$ when not.

As illustrated above, SE is general enough to model both services with WSDL descriptions and RESTful services [4]. With the definitions above, we can formally define the following problem:

Problem 1 (Long-tail web service recommendations for mashup creation). Given the information about the service ecosystem SE, for a new mashup query $k \in Q$

1. When evaluating long-tail recommendations, we make a small modification in vanilla TCDR to get a stronger baseline.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2681666, IEEE Transactions on Services Computing

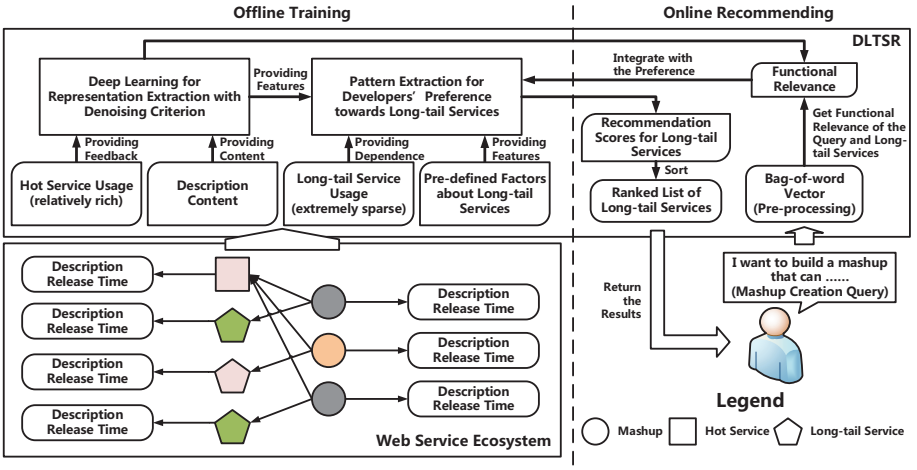


Fig. 2: The overview of the proposed approach. DLTSR takes various information into account to recommend long-tail web services for mashup creation queries.

described by a bag-of-word vector $\mathbf{Z}_{c,k*}$, the goal is to sort all the long-tail services in the set LTS and return a ranked list, in which a service with a higher rank is more likely to be adopted by query k.

3 Overview of Methodology

With the definitions above, we introduce the proposed model DLTSR. A brief review of SDAE is presented, then the deep learning architecture of DLTSR is introduced.

3.1 Stacked Denoising Autoencoders

We use Stacked Denoising Autoencoders (SDAE) as a component to build our deep learning framework. SDAE is a feedforward deep neural network with local denoising criterion for learning useful representations in an unsupervised manner [18]. Figure 3 shows the structure of an L-layer SDAE network. $\mathbf{X}_c$ is the clean input data and $\mathbf{X}_0$ is the corrupted version of $\mathbf{X}_c$ by random noise, which serves as the input of the neural network. By adding a bottleneck layer in the middle, i.e., $\mathbf{X}_{L/2}$, and trying to reconstruct the clean input $\mathbf{X}_c$ with corrupted input $\mathbf{X}_0$, the input is encoded as features which are robust to the noise. Besides, the noise used in corruption can be varied to simulate different scenes in practice. In this case, we use mask noise to simulate the uncertainty of word (term) choosing by developers.

3.2 DLTSR: Deep Learning Architecture for Long-tail Web Service Recommendations

With SDAE as a component, we build a specially designed deep learning architecture to perform high accuracy long-tail service recommendations. Figure 4 shows the architecture of our model (To prevent clutter, we don't distinguish the hot services and the long-tail ones in this figure). $\mathbf{X}_c$ is the matrix that records the clean description content of mashups, while $\mathbf{Y}_c$ is the matrix that records the clean description content of all the services. All these clean records are injected with random noise to get the corrupted version, i.e., $\mathbf{X}_0$ and $\mathbf{Y}_0$. The deep neural network tries to use

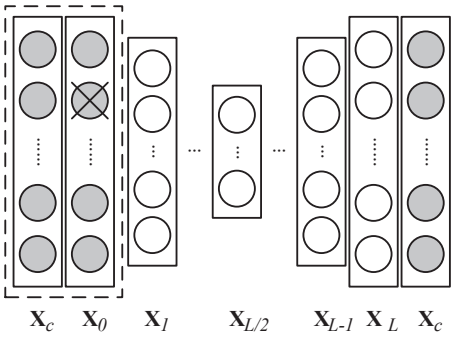


Fig. 3: The structure of an L-layer SDAE network. By using the corrupted input $\mathbf{X}_0$ to reconstruct the clean input $\mathbf{X}_c$ through a bottleneck layer $\mathbf{X}_{L/2}$, the input can be encoded as robust features.

the corrupted input to reconstruct the clean ones, thus the model can get rid of the uncertainty when developers choose words(terms), and learn effective features ($\mathbf{X}_{L/2}$ and $\mathbf{Y}_{L/2}$). Moreover, by incorporating the features of mashups and services[2], and pre-defined factors which are helpful for the long-tail recommendations, we can make a prediction of how the long-tail services are likely to be involved by the mashups.

Corresponding to the architecture described above, we can formally define the generative process of DLTSR as:

1) For each layer l of the SDAE network,
   a) For each column n of the weight matrix $\mathbf{W}_l$, draw
   $$\mathbf{W}_{l,*n} \sim \mathcal{N}(\mathbf{0}, \lambda_w^{-1}\mathbf{I}_{K_l}).$$
   b) Draw the bias vector $\boldsymbol{b}_l \sim \mathcal{N}(\mathbf{0}, \lambda_w^{-1}\mathbf{I}_{K_l})$.
   c) For each row i of $\mathbf{X}_l$, draw
   $$\mathbf{X}_{l,i*} \sim \mathcal{N}\big(\sigma(\mathbf{X}_{l-1,i*}\mathbf{W}_l + \boldsymbol{b}_l), \lambda_s^{-1}\mathbf{I}_{K_l}\big).$$

2. When predicting the usages, we allow a small offset to the features, i.e., $\boldsymbol{u}$ and $\boldsymbol{v}$ can be slightly different with $\mathbf{X}_{L/2}$ and $\mathbf{Y}_{L/2}$, as sometimes the descriptions cannot reflect how the developers use the services in practice.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2681666, IEEE Transactions on Services Computing

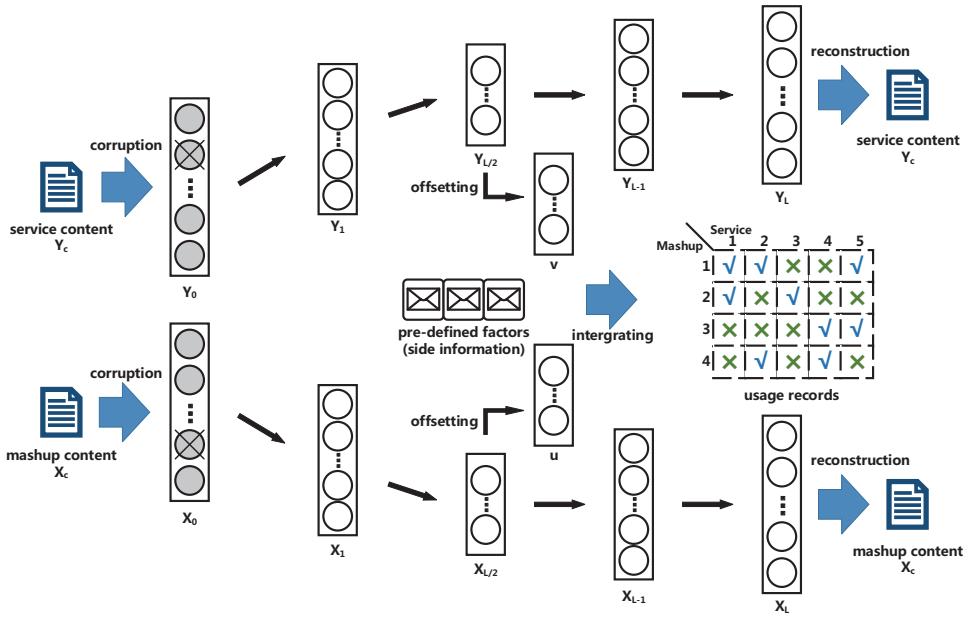


Fig. 4: The deep learning architecture for long-tail web service recommendations. To prevent clutter, we don't distinguish the hot services and the long-tail ones in this figure.

d) For each row j of $\mathbf{Y}_l$, draw

$$\mathbf{Y}_{l,j*} \sim \mathcal{N}\big(\sigma(\mathbf{Y}_{l-1,j*}\mathbf{W}_l + \boldsymbol{b}_l), \lambda_s^{-1}\mathbf{I}_{K_l}\big).$$

2) For each mashup i,

a) Draw a clean input

$$\mathbf{X}_{c,i*} \sim \mathcal{N}(\mathbf{X}_{L,i*}, \lambda_n^{-1}\mathbf{I}_{K_L}).$$

b) Draw a latent vector $\boldsymbol{\varepsilon}_i \sim \mathcal{N}(\mathbf{0}, \lambda_v^{-1}\mathbf{I}_{K_{L/2}})$ and set the latent vector to be $\boldsymbol{u}_i = \mathbf{X}_{L/2,i*}^T + \boldsymbol{\varepsilon}_i$.

3) For each service j,

a) Draw a clean input

$$\mathbf{Y}_{c,j*} \sim \mathcal{N}(\mathbf{Y}_{L,j*}, \lambda_n^{-1}\mathbf{I}_{K_L}).$$

b) Draw a latent vector $\boldsymbol{\delta}_j \sim \mathcal{N}(\mathbf{0}, \lambda_v^{-1}\mathbf{I}_{K_{L/2}})$ and set the latent vector to be $\boldsymbol{v}_j = \mathbf{X}_{L/2,j*}^T + \boldsymbol{\delta}_j$.

4) For each mashup-service pair (i, j), draw a rating

$$r_{ij} \sim \mathcal{N}(\hat{r}_{ij}, c_{ij}^{-1}),$$

where $\hat{r}_{ij}$ is the estimated rating for mashup-service pair (i, j), we define that

$$\hat{r}_{ij} = \begin{cases} \alpha \cdot \boldsymbol{u}_i^T \boldsymbol{v}_j + \gamma_j, & j \in HS \\ \beta_1 \cdot \boldsymbol{u}_i^T \boldsymbol{v}_j + \beta_2 \cdot d_{ij} & \\ \quad +\beta_3 \cdot h_{ij} + \beta_4, & j \in LTS \end{cases} \tag{1}$$

d_{ij} is the time period since service j was released (if the service has never been used before mashup i) or used last time (if the service has been used before), so $d_{ij} = t_i^m - \max(\{t_j^s, t_{i'}^m | r_{i'j} = 1, t_{i'}^m < t_i^m\})$. h_{ij} records how many times that service j has been used before mashup i, so $h_{ij} = \text{card}(\{i' | r_{i'j} = 1, t_{i'}^m < t_i^m\})$. And c_{ij} is the confidence factor for r_{ij} similar with [16], [17], we define that

$$c_{ij} = \begin{cases} c_H g(r_{ij}), & j \in HS \\ c_{LT} g(r_{ij}), & j \in LTS \end{cases}, \tag{2}$$

and

$$g(r_{ij}) = \begin{cases} a, & r_{ij} = 1 \\ 1, & r_{ij} = 0 \end{cases} \tag{3}$$

As illustrated in (1), we design a different mechanism for the generation of long-tail ratings. By setting the estimated rating as a weighted linear sum of functional relevance (i.e., $\boldsymbol{u}_i^T \boldsymbol{v}_j$) and pre-defined factors (i.e., d_{ij} and h_{ij}), user preference over these factors can be learned. By setting a small c_{LT} in (2), we can avoid the overfitting problem in the severe sparsity condition of long-tail service side.

Using what pre-defined factors in (1) can be different in different service ecosystems, and the set of pre-defined factors can be easily enriched to achieve a better recommendation accuracy. For our study in ProgrammableWeb service ecosystem, we use d_{ij} and h_{ij} defined above. The intuition is: a service is less likely worth a try if it has been published for a long time without being used, and given other factors fixed, developers should be more prudent to use a service without any usage record before.

The probabilistic graphical model of the generative process is shown in Figure 5. For concision, the SDAE portions (which are in the dashed boxes) are simplified, $\mathbf{W}^+$ is used to represent the parameters of the SDAE part, i.e., $\{\mathbf{W}_l, \boldsymbol{b}_l, l = 1, 2, \cdots, L\}$, and we use a single rectangle to represent both hot services and long-tail services.

4 Parameter Learning and Prediction

In this section, how to obtain optimal parameters of DLTSR is presented, and how to use the trained model to make recommendations is illustrated.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2681666, IEEE Transactions on Services Computing

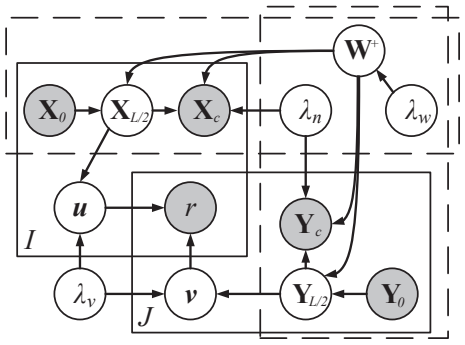


Fig. 5: The probabilistic graphical model of DLTSR. For concision, the SDAE parts are simplified in this graphical model and both hot services and long-tail services are represented by a single rectangle.

4.1 Parameter Learning for the Model

Based on the generative process above, maximizing the posterior probability is equivalent to maximizing the joint log-likelihood of $\boldsymbol{u}_{1:I}$, $\boldsymbol{v}_{1:J}$, $\mathbf{X}_c$, $\mathbf{X}_0$, $\mathbf{X}_{1:L}$, $\mathbf{Y}_c$, $\mathbf{Y}_0$, $\mathbf{Y}_{1:L}$, $\mathbf{W}_{1:L}$, $\boldsymbol{b}_{1:L}$, α, $\{\gamma_j, j \in HS\}$ and $\beta_{1:4}$ given λ_v, λ_w, λ_s, λ_n and $\{c_{ij}\}$:

$$\begin{aligned}\mathscr{L} =& -\frac{\lambda_n}{2}\sum_i \|\mathbf{X}_{c,i*} - \mathbf{X}_{L,i*}\|_2^2 - \frac{\lambda_n}{2}\sum_j \|\mathbf{Y}_{c,j*} - \mathbf{Y}_{L,j*}\|_2^2 \\ &- \frac{\lambda_v}{2}\sum_i \|\boldsymbol{u}_i - \mathbf{X}_{L/2,i*}^T\|_2^2 - \frac{\lambda_v}{2}\sum_j \|\boldsymbol{v}_j - \mathbf{Y}_{L/2,j*}^T\|_2^2 \\ &- \frac{\lambda_s}{2}\sum_l\sum_i \|\sigma(\mathbf{X}_{l-1,i*}\mathbf{W}_l + \boldsymbol{b}_l) - \mathbf{X}_{l,i*}\|_2^2 \\ &- \frac{\lambda_s}{2}\sum_l\sum_j \|\sigma(\mathbf{Y}_{l-1,j*}\mathbf{W}_l + \boldsymbol{b}_l) - \mathbf{Y}_{l,j*}\|_2^2 \\ &- \sum_{i,j}\frac{c_{ij}}{2}(r_{ij} - \hat{r}_{ij})^2 - \frac{\lambda_w}{2}\sum_l(\|\mathbf{W}_l\|_F^2 + \|\boldsymbol{b}_l\|_2^2).\end{aligned}$$

For computational efficiency, we let λ_s go to infinity [16], [19]. The likelihood becomes:

$$\begin{aligned}\mathscr{L} =& -\frac{\lambda_n}{2}\sum_i \|\mathbf{X}_{c,i*} - f_r(\mathbf{X}_{0,i*}, \mathbf{W}^+)\|_2^2 \\ &- \frac{\lambda_n}{2}\sum_j \|\mathbf{Y}_{c,j*} - f_r(\mathbf{Y}_{0,j*}, \mathbf{W}^+)\|_2^2 \\ &- \frac{\lambda_v}{2}\sum_i \|\boldsymbol{u}_i - f_e(\mathbf{X}_{0,i*}, \mathbf{W}^+)\|_2^2 \\ &- \frac{\lambda_v}{2}\sum_j \|\boldsymbol{v}_j - f_e(\mathbf{Y}_{0,j*}, \mathbf{W}^+)\|_2^2 \\ &- \sum_{i,j}\frac{c_{ij}}{2}(r_{ij} - \hat{r}_{ij})^2 - \frac{\lambda_w}{2}\sum_l(\|\mathbf{W}_l\|_F^2 + \|\boldsymbol{b}_l\|_2^2),\end{aligned} \tag{4}$$

where $f_e(\cdot, \mathbf{W}^+)$ computes the encoding results, i.e., the output of the bottleneck layer. And $f_r(\cdot, \mathbf{W}^+)$ computes the reconstructing results, i.e., the output of the last layer. The definitions of $\hat{r}_{ij}$ and c_{ij} are illustrated in (1) and (2) respectively.

From a perspective of optimization, the first and second terms in (4) are the reconstruction error, and λ_n controls to what extent the reconstruction error affects the optimal model. The fifth term is the collaborative filtering error, and controls to what extent the collaborative filtering error affects the optimal model (the definition of c_{ij} follows (2), c_H and c_{LT} affect c_{ij} for hot services and long-tail services separately). The third and fourth terms tightly couple the collaborative filtering part and SDAE part together, and λ_v controls the freedom of each side. And the last term serves as a $L2$-norm regularization for $\mathbf{W}^+$ to prevent overfitting.

The training procedure of DLTSR is similar with [16]. First we pre-train the SDAE part layer-by-layer to initialize network parameters, then we alternate the update of $\boldsymbol{u}_{1:I}$, $\boldsymbol{v}_{1:J}$, α, $\{\gamma_j, j \in HS\}$, $\beta_{1:4}$ and $\mathbf{W}^+$. In other words, after layer-wise pre-training, we repeat the following steps:

1) $\boldsymbol{u}_{1:I}, \boldsymbol{v}_{1:J}, \alpha, \{\gamma_j, j \in HS\}, \beta_{1:4}$ are optimized while holding $\mathbf{W}^+$ fixed;
2) $\mathbf{W}^+$ is optimized using back-propagation learning algorithm [20] while $\boldsymbol{u}_{1:I}, \boldsymbol{v}_{1:J}, \alpha, \{\gamma_j, j \in HS\}$ and $\beta_{1:4}$ are fixed.

By repeating the steps above, we can find a (local) optimum for $\mathscr{L}$. When $\mathbf{W}^+$ is being optimized, commonly used techniques, like momentum term [21], stochastic gradient descent (SGD), tied weights [18] and dropout learning [22] can also be used to fasten the convergence and get a more effective deep neural network.

4.2 Prediction

When a mashup query arrives, trained DLTSR can be adopted to make recommendations. Let $\mathbf{Z}_{c,k*}$ be the bag-of-word(term) vector of query $k \in Q$, the predicted rating is as follows:

$$\begin{aligned}r_{kj}^* =& \beta_1\big(f_e(\mathbf{Z}_{c,k*}, \mathbf{W}^+)\big)^T\boldsymbol{v}_j \\ &+ \beta_2 d_{kj} + \beta_3 h_{kj} + \beta_4, j \in LTS.\end{aligned}$$

Note that when making predictions, we use the clean input $\mathbf{Z}_{c,k*}$.

If there are any new services that are introduced into the service repository just now, its offset $\boldsymbol{\delta}_j^*$ will be $\mathbf{0}$ (i.e., $\boldsymbol{v}_j = f_e(\mathbf{Y}_{c,j*}, \mathbf{W}^+)$), so the model proposed in this paper can settle service-side cold-start.

5 Experiments

In this section, information about the dataset is presented first, followed by evaluation metrics and baseline methods, and evaluation results are introduced in the end[3].

5.1 Experimental Dataset

As ProgrammableWeb is the largest online repository of public web services and their mashups [4], [5] and in recent years long-tail web services are playing a more and more important role in ProgrammableWeb service ecosystem, we chose the data from ProgrammableWeb.com as the experimental dataset to validate our approach.

We crawled the data of ProgrammableWeb.com from June 2005 to June 2015. The description content consisted of

3. Code and data are available at www.simflow.net/Team/baibing/DLTSR.rar

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2681666, IEEE Transactions on Services Computing

TABLE 1: Information about the Dataset

Total # of services	12384
Total # of hot services by June 2015	329
Total # of long-tail services by June 2015	12055
Total # of mashups	6228
Vocabulary Size	7500
Average # of services in the mashups	2.09
Average # of word (term) tokens in the descriptions of mashups and services	22.42

textual descriptions, tags and category information. During the evaluation, we used the descriptions of mashups in the test set as queries, and the services involved as ground truth for recommendations. After processing the descriptions by stemming and stop words removing, the top 7500 discriminative words according to tf-idf values were chosen to form the vocabulary [23], then we performed min-max normalization independently for the descriptions as in [16]. The details about the dataset are illustrated in Table 1.

5.2 Evaluation Scheme

In large service repositories like ProgrammableWeb, there are many long-tail services that can deliver similar functionalities. Therefore, a non-rated long-tail service may be due to that the developers are not aware of its existence, and not necessarily means that the service is not useful. So, similar with [16], [17], we used Recall@N to evaluate the accuracy of each method.

In the experiments, the performance of our proposed method and baselines was evaluated from July 2013 to June 2015. We used the same approach of generating training and testing datasets as [5], [13], each test contained three months, and the time granularity was set to one month. After 8 rounds of experiments, the final metric was the weighted average number of the Recall@N results with the numbers of mashups in the testing sets as weights.

To evaluate the recommendation diversity of all the baselines and the proposed algorithm, similar with [7], we used Diversity@N defined as follows:

$$\text{Diversity@}N = \frac{|\cup_{k\in Q} R_{k,N}|}{|LTS|}$$

where Q is the set of queries, LTS is the set of long-tail services, and $R_{k,N}$ is the set of top-N recommended services for query k. $|\cdot|$ denotes the number of items in a set.

5.3 Baselines and Hyperparameter Settings

The baseline methods are listed as follows:

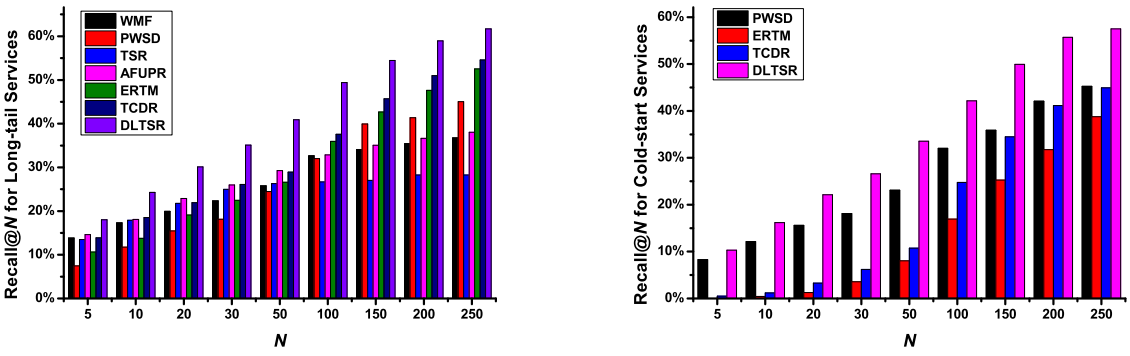
- **WMF**. Weighted Matrix Factorization (WMF) [24] is a collaborative filtering model that introduces different trust to ratings. By given a larger trust to one ratings, the problem of sparsity can be mitigated. To overcome the mashup-side cold-start problem [13] in the problem of service recommendations for mashup creation, the latent factors of mashups are fixed as the topic proportions of the descriptions by LDA. The form of weights is set as $w_{ij} = g(r_{ij})$. The definition of $g(r_{ij})$ is illustrated in (3). This method cannot make cold-start service recommendations.
- **PWSD**. Probabilistic Web Service Discovery (PWSD) method is proposed in [14]. It uses a probabilistic model to characterize the latent topics [25] between services and queries, and long-tail services are ranked based on the topic relevance. This method can make cold-start service recommendations.
- **TSR**. Time-aware Service Recommendations (TSR) is proposed in [5]. TSR promotes the performance of service recommendations by predicting service activity in the near future. It can give a reference of how well traditional service recommendation algorithms work in the long-tail side. This method cannot make cold-start service recommendations.
- **AFUPR**. This method [26] aggregates functionality, use history and popularity of web APIs for mashup recommendations. It first selects candidate APIs based on topic distributions and usage, then re-ranks the candidate APIs based on the popularity. This method cannot make cold-start service recommendations.
- **ERTM**. The Enhanced Relational Topic Model [27] incorporates the popularity of APIs to vanilla relational topic models [28] for web API recommendations. This method can make cold-start service recommendations.
- **(Modified) TCDR**. Time-aware Collaborative Domain Regression (TCDR) [13] is a service recommendation model that couples LDA and WMF tightly to recommend web services and can make cold-start service recommendations. Additionally, time information is also taken into consideration. To cope with the sparsity of long-tail service recommendations for a better accuracy, we make a modification on the confidence factors in [13] as:

$$c_{ij} = \lambda_\eta \frac{e^{-\lambda_t(t_{\text{current}}-t_i)} g(r_{ij})}{\frac{1}{I\cdot J}\sum_{i,j} e^{-\lambda_t(t_{\text{current}}-t_i)} g(r_{ij})}$$

In the experiments, TCDR showed superior performance to other baselines. Compared with the proposed DLTSR, TCDR can also make use of content information, rating information and temporal information, except that it is a shallow latent-factor based model. So we chose TCDR as the primary baseline.

The hyperparameters were set as follows. All the size of latent factors (i.e., topic number) was set to 40 as in [5], [13]. When adopting LDA, we set $\boldsymbol{\alpha} = \mathbf{1.25}$ and $\boldsymbol{\beta} = \mathbf{0.01}$ according to the empirical formula [29], dropped the first 8000 Gibbs sweeps, and then used the average results of the next 2000 Gibbs sweeps. Then we found the optimal hyperparameters for WMF, TSR, AFUPR, ERTM and TCDR with grid searching. For WMF, best performance was achieved with a set to 100, and the $\ell 2$ regularization parameter set to 0.001. For TSR, best performance was achieved with K set to 250. For AFUPR, we set the hidden dimension size of collaborative filtering to 600. For ERTM, we set λ to 0.4. For TCDR, best performance was achieved with $a = 100$, $\lambda_v = 10$, $\lambda_\eta = 0.1$, and $\lambda_t = 0.06$. For DLTSR, we directly set $a = 100$, $\lambda_n = 1$ and then performed grid searching on the other hyperparameters, and found that best

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2681666, IEEE Transactions on Services Computing



(a) Recall@N comparison

(b) Recall@N comparison for cold-start ones

Fig. 6: Performance comparison of the baselines and the proposed method. (a) Recall@N results of all the long-tail services (including the cold-start ones). (b) Recall@N results of cold-start services only. WMF and TSR cannot make cold-start recommendations.

performance was achieved with $\lambda_v = 10$, $\lambda_w = 1e-4$ and $c_H = 0.1$. PWSD do not have hyperparameters out of LDA.

For other detail settings about the deep denoising neural network, we used the masking noise with a noise level of 0.3, a dropout rate of 0.1 to achieve adaptive regularization, a momentum term of 0.99 and tied weights. In terms of network architecture, the number of hidden units K_l was set to 200 as in [16] when $l \neq L/2$ and $0 < l < L$. For example, the architecture of the DLTSR model with a 4-layer SDAE was '7500-200-40-200-7500'.

5.4 Evaluation Results

The evaluation results are presented in this section. The performance of our method and the baseline methods is presented, then the impact of some hyperparameters are studied and case studies are illustrated.

5.4.1 Accuracy Comparison

Recommendation accuracy is the first important property of a long-tail web service recommender system. Figure 6 illustrates the Recall@N of different algorithms on different sizes of recommendation list. Additionally, Recall@N results for cold-start services only (i.e., the out-matrix prediction task in [17]) are also reported[4].

Figure 6a demonstrates the accuracy of long-tail service recommendations, WMF only takes rating information into consideration and overlooks the content of services, while PWSD only takes content into consideration and overlooks the ratings, thus the results of WMF and PWSD show a significantly different trend. When N is smaller than 100, WMF outperforms PWSD, for rating information contains the relative popularity of long-tail services. However, when N is larger than 100, the performance gap between PWSD and WMF grows rapidly because WMF cannot recommend cold-start services, which have no ratings. This shows that traditional collaborative filtering methods only taking rating information into consideration, are not suitable for long-tail service recommendations. The trend of performance of TSR and AFUPR is quite similar with WMF, for they cannot recommend cold-start services either. Among them, AFUPR performs the best. On the other side, ERTM introduces popularity into the topic model, thus outperforms PWSD, which only consider content information. TCDR outperforms WMF and PWSD significantly all the time, since TCDR uses content information, rating information, and temporal information together to make long-tail recommendations. As for DLTSR, it outperfomes all the baselines and gains an improvement of around 4.1% on Recall@5, 12.0% on Recall@50, and 7.1% on Recall@250 compared with TCDR.

Figure 6b demonstrates the accuracy of cold-start service recommendations, and WMF, TSR, as well as AFUPR, cannot recommend cold-start services. An interesting phenomenon is that TCDR and ERTM get poor results in terms of Recall@N when N is small, indicating that both TCDR and ERTM still tend to put ever-used long-tail services in the front of recommendation lists. PWSD does not consider popularity factors, thus it's not inclined to ever-used services, and can provide a superior performance compared with ERTM and TCDR when recommending cold-start services. Thanks to the deep architecture and modeling of developers' preference, DLTSR gets a better performance when making cold-start recommendations. Compared with the primary baseline TCDR, DLTSR enjoys a larger improvement of 9.8% on Recall@5, 22.8% on Recall@50, and 12.6% on Recall@250 for cold-start recommendations.

5.4.2 Diversity Comparison

For a recommender system, overall recommendation diversity is another key property that we are concerned about. However, increasing the diversity of a recommender system often leads to decreasing of accuracy [30]—in extreme cases, if we recommend randomly, the diversity would be ideal but there would be no accuracy at all.

Figure 7 illustrates the recommendation diversity of baselines and the proposed method. As WMF, TSR and AFUPR can only recommend ever-used long-tail services, their results are very poor, and hardly become better when N is enlarged. On the contrary, the other three methods can provide significantly better recommendation diversity.

4. When evaluating the performance for cold-start services, we used the recommendation list that includes not only cold-start ones but also ever-used long-tail ones.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2681666, IEEE Transactions on Services Computing

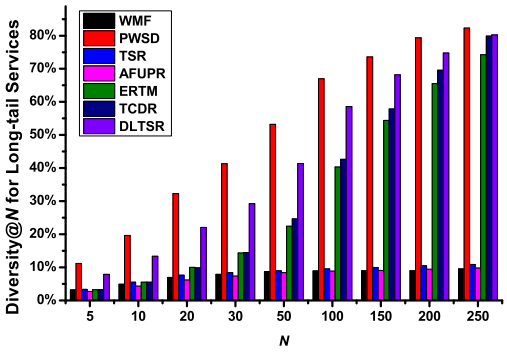


Fig. 7: Diversity comparison of the baselines and the proposed method. DLTSR can deliver the 2nd best performance in terms of recommendation diversity.

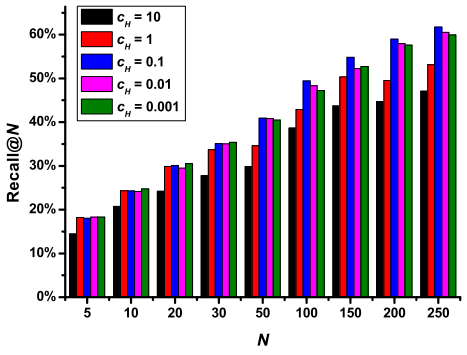


Fig. 8: Recall@N of DLTSR with different c_H

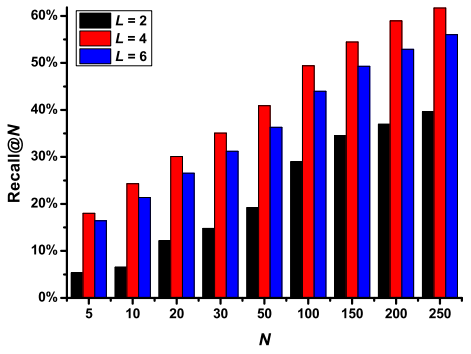


Fig. 9: Recall@N of DLTSR with different L

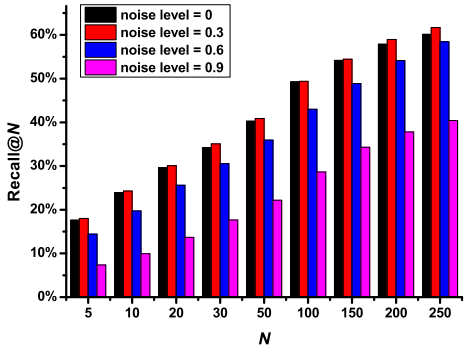


Fig. 10: Recall@N of DLTSR with different noise level

Among the methods, PWSD gets the best results, followed by DLTSR. TCDR tends to put ever-used long-tail services in the front, thus the diversity performance is worse than PWSD and DLTSR especially when N is small. A similar phenomenon can be found for ERTM. The performance gap of DLTSR and TCDR is 4.6% on Diversity@5, 16.7% on Diversity@50 and 0.4% on Diversity@250.

In conclusion, DLTSR outperforms the primary baseline TCDR in terms of both accuracy and diversity. And at a small price of diversify, DLTSR can beat PWSD significantly in terms of accuracy.

5.4.3 Impact of the Confidence Factor c_H

c_H is one of the key hyperparameters in our algorithm, which refers to how much we trust the transferred knowledge from hot service side. In terms of optimization, a larger c_H will force the algorithm to pay more attention to the loss of collaborative filtering error in the hot service side, while a smaller c_H will make the algorithm pay more attention to other terms (majorly the reconstruction error). In other words, a too large c_H may result in overlooking the content information, while a too small c_H may result in overlooking the transferred information from the hot service side.

Figure 8 shows the Recall@N of DLTSR with different c_H. As we can see, the performance degrades significantly as c_H grows, for overlooking the content information will easily result in overfitting. However, if c_H is too small, little transferred information from the hot service side is considered, and consequently, the performance will also suffer. The performance gap between different hyperparameters gets more significant when N is large.

5.4.4 Impact of the Number of Layers L

Different from traditional shallow models, the deep architecture of deep learning brings great potential for learning effective representations from complex raw data like images and documents. However, without sufficient training data and careful hyperparameter selection, performance will still face the restriction of overfitting. Figure 9 shows the performance of DLTSR with $L = 2, 4, 6$.

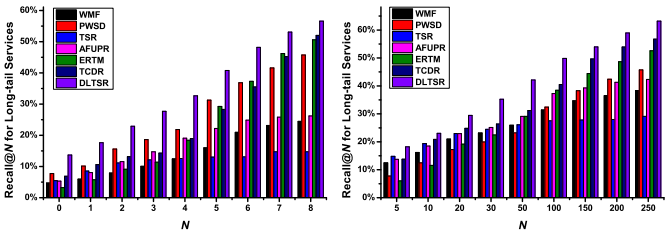
As we can see, when $L = 2$, the model is too shallow to learn effective representations from the bag-of-word vectors, so the performance is rather poor. However, when $L = 6$, performance also drops a bit, indicating that overfitting occurs and more training data or more effective regularization is needed. In conclusion, a 4-layer network is deep enough for this problem.

5.4.5 Impact of the noise level

As the basic component in our framework, Stacked Denoising Autoencoders (SDAE) use a local denoising criterion for learning useful and robust representations [18]. We also study the impact of the noise level in DLTSR. Figure 10 shows the results.

As we can see, injecting a proper level of noise can bring a performance improvement. However, if the noise is too strong, hardly can the SDAE reconstruct clean input from such corrupted version.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2681666, IEEE Transactions on Services Computing



(a) Long-tail threshold = 3 (b) Long-tail threshold = 7

Fig. 11: Performance comparison on different long-tail threshold.

5.4.6 Impact of the long-tail threshold

In our study, we choose 5 as the threshold for long-tail services. This threshold is chosen based on the following results: given 5 as the threshold, (1) from the very beginning of ProgrammableWeb.com to June 2015, the long-tail services takes about 20% of the overall total service consumption by mashups; (2) if we concentrate on the data after 2014, we can found that long-tail services takes about half of the total service consumption.

We also investigate what the performance would be like if we change the threshold. Results are illustrated in Figure 11. We can find that the performance of PWSD hardly changes with the threshold, for PWSD only considers content information while ignores historical usage information. The trend of DLTSR and other baselines are similar, i.e., increasing the threshold will lead to better Recall@N results, for the more usage data we have, the easier for recommendation algorithms to work well. Compared with the baselines other than PWSD, DLTSR is less sensitive to the change of long-tail threshold. Generally, a smaller threshold will enlarge the performance gap. This shows the effectiveness of our specially designed framework for long-tail recommendations, when the services become more "long-tailed", DLTSR suffers less from performance loss.

5.4.7 Case Studies

To gain a better insight into DLTSR and the baseline methods, we take three cases and study the effectiveness. Example mashup queries are *Deal Tracker*, *Egypt Forecast* and *MapMyRun*. Detailed results can be found in Table 2.

Case 1. *Deal Tracker* is a mashup offering discount information, and the query is of average length. It uses two cold-start services, i.e., *I Need a Bargain* and *Woot*. As a result, methods that can only recommend ever-used services, i.e., WMF TSR and AFUPR, cannot give any recommendation. By comparing the recommendation results of TCDR and PWSD, we find that TCDR outperforms PWSD a bit, indicating that tightly coupling content and usage [13] is effective. ERTM also outperforms PWSD in this case. As for DLTSR, since it can better capture the representation from the description and understand the preference of developers, the recommendation result is significantly better than both TCDR and PWSD. For example, the rank position of *Woot* is 22nd for DLTSR, significantly better than the results of TCDR and PWSD, which is 124th and 137th respectively as shown in row 6 of table 2.

To find out why TCDR, ERTM and PWSD perform significantly worse when recommending *Woot* than DLTSR, we dig into the data and find that the description of Woot is *"Woot is a daily deals site, offering curated collections of themed deals called Woot Plus Events, and a daily deal called Today's Woot. The Woot API provides developers with automated access to daily deals, events, videos, and more. Results are JSON formatted."* The words "themed", "videos", "collections" and "events" mislead TCDR and PWSD to think that Woot is a video-related API, while DLTSR can better understand the key point of *Woot* thanks to the deep denoising architecture and transferred regularization. This may partially explain why DLTSR performs significantly better when recommending cold-start services, for it can better understand the descriptions of services.

Case 2. *Egypt Forecast* is a mashup providing weather conditions of Egyptian cities, and the query is relatively short. It uses three hot services, i.e., *Google Maps*, *Google Chart* and *Panoramio*, and two long-tail services, including *World Weather Online* and *OpenWeatherMap*. According to the data from ProgrammableWeb.com, *World Weather Online* has already been used by 3 mashups before *Egypt Forecast*; as a result, WMF, TSR, ERTM and TCDR do a great job in this case, and put it to the 4th, 2nd, 5th and 2nd position in the recommendation list. As for DLTSR, it put *World Weather Online* to the 3rd position, which performs equally well. As for the cold-start service *OpenWeatherMap*, results given by methods except PWSD are quite similar with the results in the case *Deal Tracker*, i.e., DLTSR outperforms TCDR significantly, while WMF, TSR and AFUPR cannot give any recommendation as shown in row 11 of table 2. The reason why PWSD does quite a good job may be that there is a common word "map" in both the query and the description of *OpenWeatherMap*, indicating that PWSD is more sensitive to the certain word in the query.

Case 3. *MapMyRun* allows users to track their fitness activities using the GPS, and the query is relatively informative. It uses two long-tail services, i.e., *Fitbit* and *MapMyFitness*, which both have been used before *MapMyRun* was released. In this case, DLTSR, TCDR and TSR deliver similar performance, while ERTM gives a slightly worse result. For *MapMyFitness*, which has been used only once before *MapMyRun* was released, WMF performs worse compared with the other baselines.

As illustrated in all three cases, DLTSR performances significantly better than TCDR and PWSD when recommending long-tail services with no rating before, i.e., cold-start services. For the ever-used but still long-tail ones, it performs at least equally well compared with TCDR, TSR and WMF, since DLTSR can capture better service representations from the description content than LDA-based methods, and can incorporate the representations with developers' preference.

5.5 Complexity Analysis and Implementation

For the parameter learning stage, we can get that the computational complexity of calculating the gradient of $\mathbf{U}$ is $O(IJK_{L/2} + IWK_1)$, where I is the number of mashups, J the number of services, W the size of the vocabulary, and $K_{L/2}$ is the dimensionality of the learned

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2681666, IEEE Transactions on Services Computing

TABLE 2: Example mashup queries and the rank positions of the long-tail services given by each method.

Mashup	*Deal Tracker*								
Query	*Deal Tracker* finds you the biggest discounts and best deals from around the Internet. Never pay full price again and get to all the best deals before they sell out! *Deal Tracker* will help you save over 50% on many products!								
Hot Services	None								
Long-tail Services		DLTSR	TCDR	ERTM	AFUPR	TSR	PWSD	WMF	Remark
	I Need a Bargain	6	60	82	–	–	91	–	A cold-start service.
	Woot	22	124	93	–	–	137	–	A cold-start service.
Mashup	*Egypt Forecast*								
Query	*Egypt Forecast* provides weather conditions of over 70 Egyptian cities using *Google Maps* and different weather APIs.								
Hot Services	*Google Maps*, *Google Chart*, and *Panoramio*								
Long-tail Services		DLTSR	TCDR	ERTM	AFUPR	TSR	PWSD	WMF	Remark
	World Weather Online	3	2	5	3	2	146	4	Had been used 4 times.
	OpenWeatherMap	8	32	47	–	–	2	–	A cold-start service.
Mashup	*MapMyRun*								
Query	*MapMyRun* is an application that allows users to track their fitness activities using the built-in GPS of their mobile phones. Users of this application can save and upload their workout data to their *MapMyRun* account, where they can see the information in more details. Additionally, if the city is on the map of the application, runners will be able to search for, and compare different running routes.								
Hot Services	None								
Long-tail Services		DLTSR	TCDR	ERTM	AFUPR	TSR	PWSD	WMF	Remark
	Fitbit	4	1	13	2	3	236	1	Had been used 3 times.
	MapMyFitness	9	11	26	9	10	11	29	Had been used once.

representation while K_1 the dimensionality of the output in the first layer. Similar results can be drawn that the computational complexity of calculating the gradient of $\mathbf{V}$ is $O(IJK_{l/2} + JWK_1)$. For the update of all the weights and biases in SDAE part, the complexity is $O(IWK_1 + JWK_1)$ since the computation is dominated by the first layer. For the update of the rest parameters, the complexity is $O(IJK_{L/2})$. Thus for a complete epoch, the total time complexity is $O(IJK_{L/2} + IWK_1 + JWK_1)$. For the prediction stage, the computational complexity of calculating predicted ratings and ranking them in the descend order is $O\big(WK_1 + JK_{L/2} + J\log(J)\big)$.

We conduct the experiments on a server with 2 Intel Xeon E5-2650-V2 CPUs, 128GB RAM and NVIDIA Tesla K20 GPU using single precision. We use a MATLAB implementation with GPU/C++ acceleration. For the dataset we test, each epoch of model training takes about 20 seconds with a batch size of 128, and about 16 seconds with a batch size of 512. When training the model, we first run 400 epochs with a batch size of 128 and then 200 epochs with a batch size of 512. All the model training can be finished offline. The time cost of making online predictions can be ignored.

In conclusion, the model proposed in this paper is scalable enough for datasets like ProgrammableWeb.com, which is the largest online repository of public web services and their mashups [4], [5]. What's more, an even faster implementation can be achieved by using specialized deep learning software tools like Theano[5] and Tensorflow[6], and we can also reduce the time complexity by techniques like subsampling.

5. http://www.deeplearning.net/software/theano/
6. http://www.tensorflow.org/

6 Related Work

As our work touches on a couple of different areas, in this section, we describe several representative related works and differentiate them with our approach.

6.1 Web Service Recommendations

Existing personalized service recommendation approaches can be divided into two categories, i.e., QoS-based services recommendations and functionality-based services recommendations. The method proposed in this paper belongs to functionality-based approaches.

6.1.1 QoS-based web service recommendations

For this kind of algorithms, researchers assume that the mashup developers are aware of functionally which categories of services should be involved, and focus on the non-functional properties of services such as reliability, availability and response time. Among the works, [11] introduced the non-negative tensor factorization to perform temporal-aware missing QoS prediction with the triadic relations of user-service-time model. [12] fused the neighborhood-based and model-based collaborative filtering approaches to achieve a higher prediction accuracy. [9] proposed a QoS prediction by considering temporal information and employing the random walk algorithm to overcome the sparsity. [10] unified the modeling of multi-dimensional QoS data via tensor and applied tensor decomposition to predict missing QoS value.

However, limited by the objective, QoS-based service recommendation algorithms cannot help developers find an unknown but interesting service.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2681666, IEEE Transactions on Services Computing

6.1.2 Functionality-based web service recommendations

For functionality-based service recommendation, researchers focus on finding the services that meet the functional requirements the best. Mashup developers do not need to be an expert of judging which service categories should be involved. Some of the works recommended services according to potential interests of developers [4], others recommended services according to mashup queries [5], [13], [14], [15]. [4] exploited the idea of collaborative topic regression [17] and recommend services to meet the potential interests. [14] proposed a method where LDA is used to model latent topics from WSDL documents, and then recommend services for mashup queries based on the topic relevance. [5] promoted the performance of service recommendations by predicting service activity in the near future. [13] tightly coupled matrix factorization and LDA, and take information evaporation into consideration. [15] emphasized the necessity of recommending services by categories, and come up with a service clustering method and recommend both service clusters and individual services.

However, to the best of our knowledge, none of the existing web service recommendation approaches have stressed the problem of long-tail web service recommendations, as long-tail web service becomes a more and more important issue for web service ecosystems.

6.2 Long-tail Recommendations

Thanks to the Internet technologies, it has been a lot easier for consumers to find and consume long-tail products and services, and how to recommend long-tail items effectively and efficiently becomes a popular topic for both academia and industry. As long-tail recommendations shows significantly different properties compared with popular item recommendations, various approaches have been proposed based on the different scenes. [7] proposed a graph-based algorithm for the long-tail recommendations by using random walk similarity and hitting time. [31] focused on combining usage and content to recommend music in the long-tail. [32] came up with a solution of clustering long-tail items and recommend items based on the ratings in the clusters. [33] used LDA to discover the trending categories and to describe a user's interest, and then recommend shops.

Existing long-tail recommendation approaches are usually based on what the users have consumed and then recommend related long-tail items, which cannot settle the long-tail web service recommendations due to the mashup-side cold start problem [13]. In this paper, we come up with an approach by transferred knowledge and modeling the preference of mashup developers. Experiments show that our approach is effective.

6.3 Deep Learning for Natural Language Processing

For mashup developers, recommending services based on the queries is a promising and effective solution. This requires a better understanding of both the queries and description documents of services in the form of nature language. Deep learning shows a great potential for natural language processing recently. [34] proposed a Dynamic Convolutional Neural Network to model sentences of varying length. [35] used a Deep Boltzmann Machine to learning better representations of documents than LDA. [36] presented how to get representations of words and phrases with the Skip-gram model. [37] proposed to use Word Mover's Distance (WMD) to measure the distance between two text documents, so provided a method to transfer word embeddings to document similarity. [16] came up with a hierarchical Bayesian model named with collaborative deep learning (CDL), which tightly couples deep representation learning for the content information and collaborative filtering to make recommendations.

In this paper, we build a deep learning architecture to incorporating various information to recommend long-tail services for mashup queries. For the content modeling side, we use SDAE as the basic component, and tightly couple the SDAE parts with historical usage information like [16].

7 Conclusion and Future Work

As long-tail services are playing an increasingly important role in web API economy, how to recommend long-tail web services effectively is becoming a key issue. However, very scarce work has focused this problem, and traditional web service recommendation methods perform poorly on the long-tail side.

In this paper, we propose a deep learning framework which is specifically designed for this problem. To tackle the problem of unsatisfactory quality of description given by service developers and mashup queries, we use the deep learning model SDAE as the basic component to learn robust and effective representations. Moreover, the knowledge from usages in the hot service side is transferred and imposed as a regularization on the output of SDAE to guide the learning of representations. To make the best use of the few long-tail historical ratings, a special mechanism is designed to model developers' preference. Experiments demonstrate that our method gains a significant improvement compared with the state-of-the-art baseline methods.

In the future, we plan to incorporate more information, such as QoS, user profiles and social connection between services into DLTSR to promote the accuracy of recommendations. We also plan to investigate more sophisticated deep learning models such as convolutional neural networks or recurrent neural networks, and activations such as ReLU or PReLU to further boost the performance.

Acknowledgments

This research has been partially supported by the National Natural Science Foundation of China (No.61174169). Yushun Fan is the corresponding author.

References

[1] Y. Wei and M. B. Blake, "Service-oriented computing and cloud computing: challenges and opportunities," *IEEE Internet Computing*, vol. 14, no. 6, p. 72, 2010.

[2] C. Schroth and T. Janner, "Web 2.0 and soa: Converging concepts enabling the internet of services," *IT professional*, vol. 9, no. 3, pp. 36–41, 2007.

[3] 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 *Services Computing (SCC), 2011 IEEE International Conference on*. IEEE, 2011, pp. 48–55.

[4] X. Liu and I. Fulia, "Incorporating user, topic, and service related latent factors into web service recommendation," in *Web Services (ICWS), 2015 IEEE International Conference on*. IEEE, 2015, pp. 185–192.

[5] Y. Zhong, Y. Fan, K. Huang, W. Tan, and J. Zhang, "Time-aware service recommendation for mashup creation," *IEEE Transactions on Services Computing*, vol. 8, no. 3, pp. 356–368, 2015.

[6] S. M. McNee, J. Riedl, and J. A. Konstan, "Being accurate is not enough: how accuracy metrics have hurt recommender systems," in *CHI'06 extended abstracts on Human factors in computing systems*. ACM, 2006, pp. 1097–1101.

[7] H. Yin, B. Cui, J. Li, J. Yao, and C. Chen, "Challenging the long tail recommendation," *Proceedings of the VLDB Endowment*, vol. 5, no. 9, pp. 896–907, 2012.

[8] X. Zhao, Z. Niu, and W. Chen, "Opinion-based collaborative filtering to solve popularity bias in recommender systems," in *International Conference on Database and Expert Systems Applications*. Springer, 2013, pp. 426–433.

[9] Y. Hu, Q. Peng, and X. Hu, "A time-aware and data sparsity tolerant approach for web service recommendation," in *Web Services (ICWS), 2014 IEEE International Conference on*. IEEE, 2014, pp. 33–40.

[10] Y. Ma, S. Wang, F. Yang, and R. N. Chang, "Predicting qos values via multi-dimensional qos data for web service recommendations," in *Web Services (ICWS), 2015 IEEE International Conference on*. IEEE, 2015, pp. 249–256.

[11] W. Zhang, H. Sun, X. Liu, and X. Guo, "Temporal qos-aware web service recommendation via non-negative tensor factorization," in *Proceedings of the 23rd international conference on World wide web*. ACM, 2014, pp. 585–596.

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

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

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

[15] B. Xia, Y. Fan, W. Tan, K. Huang, J. Zhang, and C. Wu, "Category-aware api clustering and distributed recommendation for automatic mashup creation," *IEEE Transactions on Services Computing*, vol. 8, no. 5, pp. 674–687, 2015.

[16] H. Wang, N. Wang, and D.-Y. Yeung, "Collaborative deep learning for recommender systems," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1235–1244.

[17] C. Wang and D. M. Blei, "Collaborative topic modeling for recommending scientific articles," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2011, pp. 448–456.

[18] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, no. Dec, pp. 3371–3408, 2010.

[19] Y. Bengio, L. Yao, G. Alain, and P. Vincent, "Generalized denoising auto-encoders as generative models," in *Advances in Neural Information Processing Systems*, 2013, pp. 899–907.

[20] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[21] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural networks*, vol. 12, no. 1, pp. 145–151, 1999.

[22] S. Wager, S. Wang, and P. S. Liang, "Dropout training as adaptive regularization," in *Advances in neural information processing systems*, 2013, pp. 351–359.

[23] D. Blei and J. Lafferty, "Text mining: Theory and applications, chapter topic models," 2009.

[24] K.-Y. Chen, H.-M. Wang, B. Chen, and H.-H. Chen, "Weighted matrix factorization for spoken document retrieval," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 8530–8534.

[25] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[26] A. Jain, X. Liu, and Q. Yu, "Aggregating functionality, use history, and popularity of apis to recommend mashup creation," in *International Conference on Service-Oriented Computing*. Springer, 2015, pp. 188–202.

[27] C. Li, R. Zhang, J. Huai, and H. Sun, "A novel approach for api recommendation in mashup development," in *IEEE International Conference on Web Services*, 2014, pp. 289–296.

[28] J. Chang and D. M. Blei, "Hierarchical relational models for document networks," *The Annals of Applied Statistics*, pp. 124–150, 2010.

[29] T. L. Griffiths and M. Steyvers, "Finding scientific topics," *Proceedings of the National academy of Sciences*, vol. 101, no. suppl 1, pp. 5228–5235, 2004.

[30] V. Ghanghas, C. Rana, and S. Dhingra, "Diversity in recommender system," *International Journal of Engineering Trends & Technology*, vol. 4, no. 6, 2013.

[31] M. A. Domingues, F. Gouyon, A. M. Jorge, J. P. Leal, J. Vinagre, L. Lemos, and M. Sordo, "Combining usage and content in an online recommendation system for music in the long tail," *International Journal of Multimedia Information Retrieval*, vol. 2, no. 1, pp. 3–13, 2013.

[32] Y.-J. Park and A. Tuzhilin, "The long tail of recommender systems and how to leverage it," in *Proceedings of the 2008 ACM conference on Recommender systems*. ACM, 2008, pp. 11–18.

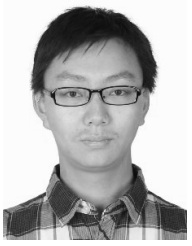
[33] D. J. Hu, R. Hall, and J. Attenberg, "Style in the long tail: Discovering unique interests with latent variable models in large scale social e-commerce," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 1640–1649.

[34] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," *arXiv preprint arXiv:1404.2188*, 2014.

[35] N. Srivastava, R. R. Salakhutdinov, and G. E. Hinton, "Modeling documents with deep boltzmann machines," *arXiv preprint arXiv:1309.6865*, 2013.

[36] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[37] M. J. Kusner, Y. Sun, N. I. Kolkin, and K. Q. Weinberger, "From word embeddings to document distances," in *Proceedings of the 32nd International Conference on Machine Learning (ICML 2015)*, 2015, pp. 957–966.



Bing Bai received the BS degree in control theory and application in 2013 from Tsinghua University, China. He is currently working toward the PhD degree at the Department of Automation, Tsinghua University. 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, Director of the System Integration Institute, and Director of the Networking Manufacturing Laboratory, Tsinghua University. From September 1993 to 1995, he was a visiting scientist, supported by Alexander von Humboldt Stiftung, with the Fraunhofer Institute for Production System and Design Technology (FHG/IPK), Germany. He has authored 10 books in enterprise modeling, work-flow technology, intelligent agent, object-oriented complex system analysis, and computer integrated manufacturing. He has published more than 300 research papers in journals and conferences. 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. From 2008 to 2010, he was a researcher at the Computation Institute, University of Chicago and Argonne National Laboratory. At that time, he was the technical lead of the caBIG workflow system. His research interests include NoSQL, big data, cloud computing, service-oriented architecture, business and scientific workflows, and petri nets. He has published more than 50 journal and conference papers, and a monograph Business and Scientific Workflows: A Web Service-Oriented Approach (272 pages, Wiley-IEEE Press). He received the Best Paper Award from the IEEE International Conference on Services Computing (2011), the Pacesetter Award from the Argonne National Laboratory (2010), and caBIG Teamwork Award from the National Institute of Health (2008). He is an associate editor of the IEEE Transactions on Automation, Science and Engineering. He was in the program committees of many conferences and has co-chaired several workshops. He is a member of the ACM and a senior member of the IEEE.

Jia Zhang received the MS and BS degrees in computer science from Nanjing University, China and 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 co-authored 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 editor-in-chief of International Journal of Services Computing (IJSC). She is a senior member of the IEEE.