DeepWSC: Clustering Web Services via Integrating Service Composability into Deep Semantic Features

Guobing Zou, Zhen Qin, Qiang He, Pengwei Wang, Bofeng Zhang, and Yanglan Gan

Abstract—With an growing number of web services available on the Internet, an increasing burden is imposed on the use and management of service repository. Service clustering has been employed to facilitate a wide range of service-oriented tasks, such as service discovery, selection, composition and recommendation. Conventional approaches have been proposed to cluster web services by using explicit features, including syntactic features contained in service descriptions or semantic features extracted by probabilistic topic models. However, service implicit features are ignored and have yet to be properly explored and leveraged. To this end, we propose a novel heuristics-based framework DeepWSC for web service clustering. It integrates deep semantic features extracted from service descriptions by an improved recurrent convolutional neural network and service composability features obtained from service invocation relationships by a signed graph convolutional network, to jointly generate integrated implicit features for web service clustering. Extensive experiments are conducted on 8,459 real-world web services. The experiment results demonstrate that DeepWSC outperforms state-of-the-art approaches for web service clustering in terms of multiple evaluation metrics.

Index Terms—Web service, service clustering, deep neural network, service composability, mashup service

IEEE Proof

1 Introduction

With the advances of service-oriented architecture (SOA) in software integration and applications [1], web services are becoming popular and important building blocks for fast establishing next generation real-world applications. As the demand on service-oriented applications increases rapidly, more and more software vendors publish their applications as web services on the Internet. As of September 26, 2019, the world's largest online web service repository, ProgrammableWeb,[1] has registered more than 22,000 web services and counting. Web services significantly accelerate machine-to-machine interactions and promote the development of service-oriented software systems.

However, the explosion of web services has increased the burden of exploring and managing web services on online web service repositories like ProgrammableWeb [2]. For example, suppose a wants to find "Web services for retrieving keyword popularity by location and date". It is difficult to quickly and precisely find the desired web services among a huge number of available web services to respond to such a natural language query [3]. Therefore, how to accurately and efficiently find functionally similar or equivalent web services has become a fundamental and challenging research issue in field of service-oriented computing.

Clustering web services has been proved to be an effective way to facilitate a series of service-oriented tasks, e.g., service discovery [4], [5], service selection [6], service composition [1], [7] and service recommendation [8], [9], [10], by effectively finding desired web services. Taking the task of service composition as an example, when a mashup developer is finding the appropriate web services for matching the decomposed requirements, service clustering can help the mashup developer match the required functionalities with a limited number of service clusters instead of from a huge number of web services [1], [11]. In recent years, there are a lot of researchers focusing on improving the accuracy of service clustering [1], [3], [4], [7], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21].

The key to the performance of web service clustering is its accuracy and applicability. Traditional clustering approaches based on WSDL descriptions rely on service syntactic features, which did not consider the semantic information of service descriptions. Furthermore, ontology-based approaches apply high-quality ontologies with the combination of Information Retrieval [18] or with the assistance of domain-specific information as heuristics to enrich the semantic representation of few terms in WSDL web services. However, constructing high-quality ontologies is difficult and requires much human efforts [18], which restricts the applicability of ontology-based web service clustering. Along with the popularity of the API mode as the mainstream representation of

1. https://www.programmableweb.com

- *Guobing Zou, Zhen Qin, and Bofeng Zhang are with the School of Computer Engineering and Science, Shanghai University, Shanghai 200444, China. E-mail: {gbzou, zhenqin, bfzhang}@shu.edu.cn.*
- *Qiang He is with the Department of Computer Science and Software Engineering, Swinburne University of Technology, Melbourne 3122, Australia. E-mail: qhe@swin.edu.au.*
- *Pengwei Wang and Yanglan Gan are with the School of Computer Science and Technology, Donghua University, Shanghai 201620, China. E-mail: {wangpengwei, ylgan}@dhu.edu.cn.*

Manuscript received 5 Dec. 2019; revised 23 Aug. 2020; accepted 20 Sept. 2020. Date of publication 0 . 0000; date of current version 0 . 0000.
(Corresponding author: Yanglan Gan.)
Digital Object Identifier no. 10.1109/TSC.2020.3026188

1939-1374 © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

web services, researchers turned their focus from mining service syntactic features in WSDL descriptions to implementing probabilistic topic model, such as Latent Dirichlet Allocation (LDA) [22], to extract service semantic features from functionality description in natural language [14]. However, the major issue of topic model based approaches is that they mainly extract the explicit semantic features of service descriptions, whereas the implicit features of web services have been ignored to be explored for potentially enhancing the accuracy of web service clustering.

Recently, researchers are starting to attempt incorporating external heuristic information into traditional service clustering algorithms such as service invocation relationships [15] and user tagging [16]. However, this kind of approaches still cannot deeply extract implicit features for better clustering web services. This may severely impact the accuracy of web service clustering. As for the topic model based approaches integrated with service invocation relationships, the heuristics are primarily used to guide the training process of LDA model to mine explicit semantic features of web services [15], [16], where implicit features of web services are not thoroughly utilized. Therefore, there is an urgent need for an approach that can leverage implicit features to more accurately cluster web services. In our previous work [3] we proposed DeepWSC, a novel framework for web service clustering that makes a good use of implicit features of web services. However, domain knowledge in service computing as heuristics from the service invocation relationships is still not taken into account for more precisely extracting the implicit features of web services in our previous work. It is observed that service composability from mashup services can be applied as heuristics to better extract implicit features, which can be leveraged to facilitate web service clustering. For example, suppose a web service is invoked by another web service in a mashup service, they are tended to be partitioned into different clusters [16].

To this end, we propose an improved DeepWSC, where service composability features are integrated into deep neural network for web service clustering. DeepWSC first establishes the service composability network and generates the heuristics of service invocation relationships, which is then fed into an improved recurrent convolutional neural network to train a service feature extractor in an unsupervised manner. Finally, DeepWSC acquires the integrated implicit features of web services, which consists of deep semantic features and composability features of web services. To evaluate the effectiveness of DeepWSC in web service clustering, we conduct extensive experiments on 8,459 real-world web services from ProgrammableWeb. Benefiting from deep semantic features and composability features of web services, DeepWSC outperforms state-of-the-art web service clustering approaches on multiple evaluation metrics.

This work extends our previous conference paper [3] and effectively improves the DeepWSC's performance in web service clustering. The main differences between this paper and [3] are twofold. First, DeepWSC now employs the service composability features that are fed into the deep neural network as a whole, which are then combined to web services' deep semantic features by a combination strategy, to generate integrated implicit features of web services. Second, DeepWSC now employs BERT [23], a more advanced word embedding method, to embed web service descriptions. The main contributions of this paper are summarized as follows:

- We propose a novel heuristics-based framework DeepWSC for web service clustering. In DeepWSC, a deep neural network is trained as a service feature extractor which generates the integrated implicit features of web services for precisely clustering services.
- To extract the integrated implicit features of web services more effectively, we propose a strategy for combining web services' deep semantic features and service composability features, to generate service integrated implicit features.
- Extensive experiments are conducted on a large number of real-world web services crawled from ProgrammableWeb. The experimental results demonstrate that DeepWSC outperforms state-of-the-art approaches significantly in web service clustering.

The remainder of this paper is organized as follows. Section 2 formulates the research problem. Section 3 illustrates the overall framework of DeepWSC. Section 4 presents DeepWSC in detail. Experimental results and analyses are presented and discussed in Section 5. Section 6 reviews the related work. Finally, Section 7 concludes the paper and discusses the future work.

2 Problem Formulation

This section presents the formulations of our research to cluster web services.

Definition 1 (Web Service). *Web service refers to API service. It is denoted as a three-tuple* $s = \langle W^{(s)}, L^{(s)}, D^{(s)} \rangle$, *where* $W^{(s)} = \{w_1, w_2, \ldots\}$ *is a collection of words, constituting the functionality description of* s. $D^{(s)}$ *is the domain label corresponding to* s. $L^{(s)} = \{l_{ss'}, l_{ss''}, \ldots\}$ *is a set of undirected links, where each link indicates the composability relationship between web service* s *and another web service* s' *or* s''. *When* s *has no composability relationships with any other web services,* $L^{(s)}$ *holds an empty set.*

Definition 2 (Mashup Service). *A mashup service is a service composed of a set of existing web services, denoted by* $m = \{s_1, s_2 \ldots\}$, *where those singleton services are the components that make up* m. *Web services invoked by the same mashup service are considered to have composability relationships.*

Definition 3 (Web Service Repository). *All the* N *web services form a set* $\mathbb{S} = \{s_1, s_2, \ldots, s_N\}$. *All the* N' *mashup services constitute a set* $\mathbb{M} = \{m_1, m_2, \ldots, m_{N'}\}$. *A web service repository is the union of* $\mathbb{S}$ *and* $\mathbb{M}$, *i.e.,* $\mathbb{R} = \mathbb{S} \cup \mathbb{M}$.

Based on all the web services in $\mathbb{S}$, *we can construct a functionality description set, denoted as* $\mathbb{W} = \{W^{(s_1)}, W^{(s_2)}, \ldots, W^{(s_N)}\}$, *and a link set representing service composability relationships, denoted as* $\mathbb{L} = \{L^{(s_1)}, L^{(s_2)}, \ldots, L^{(s_N)}\}$.

Definition 4 (Web Service Clustering, WSC). *It is defined as a five-tuple,* $WSC = \langle \mathbb{S}, \mathbb{M}, \mathbb{W}, \mathbb{L}, K \rangle$, *where* $\mathbb{S}$ *is a collection of web services to be clustered,* $\mathbb{M}$ *is a set of mashup services,* $\mathbb{W}$ *is the set of functionality descriptions of web services,* $\mathbb{L}$ *is the set of composability links of web services, and* K *is the number of service clusters.*

IEEE Proof

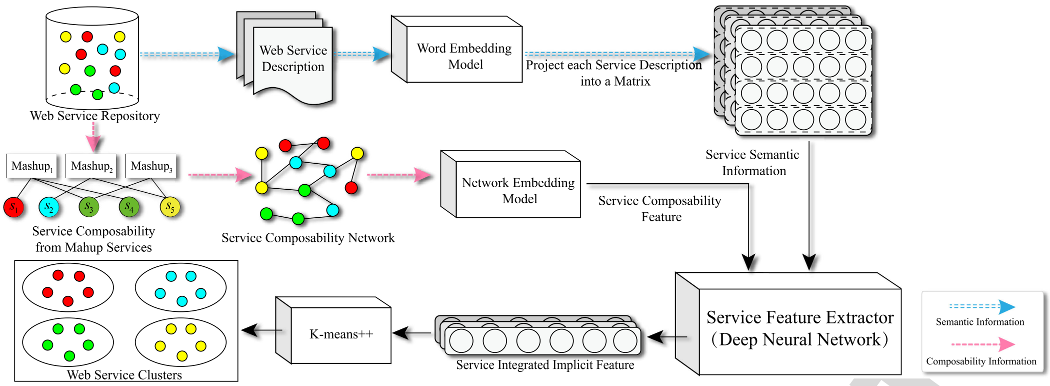


Fig. 1. Overall framework of DeepWSC for web service clustering.

The solution to a WSC problem is K clusters of services, denoted as $\mathbb{SC} = \{sc_1, sc_2, \ldots, sc_K\}$. Any two service clusters in $\mathbb{SC}$ do not have any web services in common, i.e., $sc_i \cap sc_j = \emptyset, \forall sc_i, sc_j \in \mathbb{SC}, i \neq j$, and each web service can only and must be included in one of the clusters.

3 Framework Overview

The overall framework of DeepWSC is illustrated in Fig. 1. The objective of DeepWSC is to partition a set of web services into several functionally-differentiated clusters according to their functionality descriptions and composability relationships. DeepWSC consists of four independent but correlative components, including service semantic representation, service composability mining, service implicit feature integration and service clustering:

- In the component of service semantic representation, web service descriptions are obtained from web service repository and a trained word embedding model is employed to transform each word into a dense vector. Thus, each of the descriptions is expressed as a matrix. These matrices as service semantic representations are fed into the deep neural network to train the service feature extractor.
- In the component of service composability mining, we first extract service invocation relationships from mashup services, and then build a service composability network. After that, network embedding is performed to embed service composability relationships into dense vectors. These vectors as the mined service composability features are fed into the deep neural network as heuristics to boost the training of the service feature extractor.
- In the component of implicit feature integration, the deep neural network is trained to generate integrated implicit features of web services by taking the matrix service semantic representations and the service composability features as inputs.
- In the component of service clustering, K-means++ [24], a widely-used clustering algorithm, is employed to cluster web services into a number of clusters.

Fig. 2. Mashup service example of price tracking and comparison.

4 Approach

The structure and training process of the service feature extractor powered by a deep neural network is illustrated in Fig. 5. It consists of four layers. (1) In the embedding layer, as shown in Fig. 5a, a service composability network is built and embedded to generate service composability features. (2) In the extraction layer, as shown in Fig. 5b, DeepWSC first extracts deep semantic features from service descriptions, and then integrates service composability features to generate service implicit features. (3) In the provision layer, as shown in Fig. 5c, a WE-LDA model [14] is trained to provide probabilistic topic distribution vectors as partially correct domain labels to help train the service feature extractor. (4) In the fitting layer, as shown in Fig. 5d, the service feature extractor is trained by updating its parameters.

4.1 Service Composability Feature Mining

To obtain the service composability features, we first establish a service composability network by using mashup services. Fig. 2 presents a mashup service from ProgrammableWeb, called PriceZombie Price Tracker.[2] It contains a mashup functionality description and a set of related web services with their functionality domain labels. Web services exhibit mutually composable relationships when they are invoked by the same mashup service, which are used to build a service composability network (SCN) defined as below.

Definition 5 (Service Composability Network, SCN). *Given a set of mashup services $\mathbb{M} = \{m_1, m_2, \ldots\}$ and their related web services $\mathbb{S} = \{s_1, s_2, \ldots\}$, an SCN is an undirected weighted graph, $G = (V, E, W)$, where V, E and W are the set of vertices, edges and weights, respectively. There is an edge $(s_i, s_j) \in E$ if s_i and s_j are invoked by a mashup service. A weight $w_e(s_i, s_j) \in W$ of the edge between two vertices s_i and*

2. https://www.programmableweb.com/mashup/pricezombie-price-tracker.

IEEE Proof

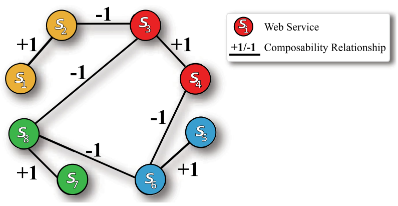


Fig. 3. Illustration of service composability network. Vertices with different colors represent web services with different functionalities.

s_j *is determined by their different or the same functionality as below*

$$w_e(s_i, s_j) = \begin{cases} 1, & s_i \text{ and } s_j \text{ share the same functionality} \\ -1, & s_i \text{ and } s_j \text{ are functionally different.} \end{cases} \tag{1}$$

From the above definition, the weight of each link in SCN represents the functional difference or similarity between two web services in terms of their corresponding domains. As shown in Fig. 4, a positive link ($w_e = +1$) means that two web services share the same or similar functionality, while a negative link ($w_e = -1$) means that the functionalities of the two web services differ from each other.

To integrate service composability features into deep neural network, vertices in the SCN need to be embedded into vector representations [25]. Learning vector vertex representations has been previously proven to be useful in many social network analysis tasks [26]. Our SCN contains links with a weight of $+1$ or -1, which is very similar to positive and negative links in signed social networks [25]. Motivated by the homogeneous structure of social network, we employ an effective signed network embedding technique, called signed graph convolutional networks (SGCN) [26], to embed the service composability network as a dense matrix, where each vertex is represented as a vector.

The foundation of SGCN is the balance theory in social network, which implies "the friend of my friend is my friend, and the foe of my friend is my foe". Based on this, paths in a signed network can be classified into balanced or unbalanced ones, where a balanced path consists of an even number of negative links, and an unbalanced one has an odd number of negative links. For a web service s_i, let $\mathfrak{N}^+_{s_i}$ and $\mathfrak{N}^-_{s_i}$ denote the set of services that are directly linked to s_i with a link weighted $+1$ and -1, respectively. $B_{s_i}(l)$ and $U_{s_i}(l)$ are the sets of services that reach s_i along a balanced and an unbalanced path of l hops, respectively. According to [26], $B_{s_i}(l)$ and $U_{s_i}(l)$ can be calculated as follows:

$$\begin{aligned} \text{For } l = 1,\ B_{s_i}(1) &= \left\{ s_j \middle| s_j \in \mathfrak{N}^+_{s_i} \right\} \\ U_{s_i}(1) &= \left\{ s_j \middle| s_j \in \mathfrak{N}^-_{s_i} \right\} \\ \text{For } l > 1,\ B_{s_i}(l+1) &= \left\{ s_j \middle| s_k \in B_{s_i}(l) \text{ and } s_j \in \mathfrak{N}^+_{s_k} \right\} \\ &\cup \left\{ s_j \middle| s_k \in U_{s_i}(l) \text{ and } s_j \in \mathfrak{N}^-_{s_k} \right\} \\ U_{s_i}(l+1) &= \left\{ s_j \middle| s_k \in U_{s_i}(l) \text{ and } s_j \in \mathfrak{N}^+_{s_k} \right\} \\ &\cup \left\{ s_j \middle| s_k \in B_{s_i}(l) \text{ and } s_j \in \mathfrak{N}^-_{s_k} \right\} \end{aligned} \tag{2}$$

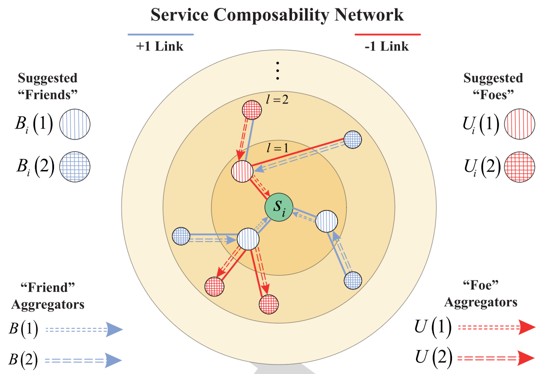


Fig. 4. Process illustration of SGCN to embedding service s_i in the service composability network based on [26].

As illustrated in Fig. 4, SGCN embeds s_i in the service composability network by continuously aggregating the information of its "friends" and "foes" layer by layer as the depth of the network gradually increases. The convolution and aggregation of s_i in layer 1 is defined as in (3) and (4),

$$h^{B(1)}_{s_i} = \sigma\left(\left[W^{B(1)} \sum_{s_j \in \mathfrak{N}^+_{s_i}} \frac{h^{(0)}_{s_j}}{\left|\mathfrak{N}^+_{s_i}\right|},\ h^{(0)}_{s_i} \right]\right) \tag{3}$$

$$h^{U(1)}_{s_i} = \sigma\left(\left[W^{U(1)} \sum_{s_k \in \mathfrak{N}^-_{s_i}} \frac{h^{(0)}_{s_k}}{\left|\mathfrak{N}^-_{s_i}\right|},\ h^{(0)}_{s_i} \right]\right) \tag{4}$$

where "[ ]" is the concatenating operation, $W^{B(1)}$ and $W^{U(1)}$ are the matrices to transform the representations of "friends" and "foes" of s_i, respectively, σ represents a non-linear activation function, and $h^{(0)}_{s_i}$ is the initial feature of s_i.

When $l > 1$, the aggregation is defined as in (5) and (6), where $W^{B(l)}$ and $W^{U(l)}$ are all the matrices to transform the representations of "friends" and "foes" of s_i for $l > 1$.

$$h^{B(l)}_{s_i} = \sigma\left(W^{B(l)} \left[\sum_{s_j \in \mathfrak{N}^+_{s_i}} \frac{h^{B(l-1)}_{s_j}}{\left|\mathfrak{N}^+_{s_i}\right|},\ \sum_{s_k \in \mathfrak{N}^-_{s_i}} \frac{h^{U(l-1)}_{s_k}}{\left|\mathfrak{N}^-_{s_i}\right|},\ h^{B(l-1)}_{s_i} \right]\right) \tag{5}$$

$$h^{U(l)}_{s_i} = \sigma\left(W^{U(l)} \left[\sum_{s_j \in \mathfrak{N}^+_{s_i}} \frac{h^{U(l-1)}_{s_j}}{\left|\mathfrak{N}^+_{s_i}\right|},\ \sum_{s_k \in \mathfrak{N}^-_{s_i}} \frac{h^{B(l-1)}_{s_k}}{\left|\mathfrak{N}^-_{s_i}\right|},\ h^{U(l-1)}_{s_i} \right]\right) \tag{6}$$

After multiple iterations of convolution and aggregation, the composability feature of s_i, denoted by h_c, is obtained by concatenating the two hidden representations, $h^{B(l_{\max})}_{s_i}$ and $h^{U(l_{\max})}_{s_i}$, where $l_{\max}$ is the number of layers of SGCN.

4.2 Service Integrated Implicit Feature Extraction

As illustrated in Fig. 5b, DeepWSC first employs a recurrent convolutional neural network (RCNN) [27] to extract service deep semantic features, and then combines the service composability features to generate service integrated implicit features for further clustering.

To embed service descriptions, each word w in a web service description is projected into a dense vector $e(w) \in \mathbb{R}^d$ by

IEEE Proof

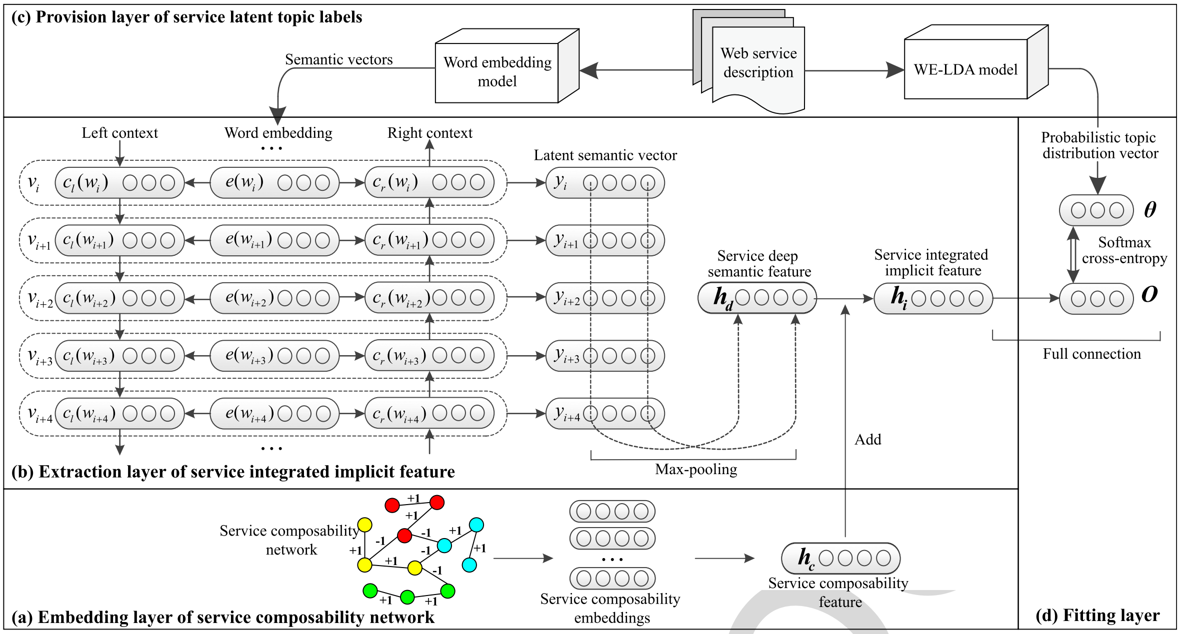


Fig. 5. Multiple layers and training process of the recurrent convolutional neural network for the extraction of service integrated implicit feature.

looking it up in a trained word embedding model, where d is the dimensionality of the embedded word vectors. Here, we use the state-of-the-art language representation model, BERT [23], as the word embedding model. Since the number of words varies in different web service descriptions, to ensure the performance of our model, we set L_{desc} as the uniform description length for all the web services. For those web services whose descriptions are shorter than L_{desc}, zero-padding is performed to pad them to L_{desc}; for those longer than L_{desc}, the extra words at the tail are removed.

After the original vector $e(w)$ of each word is obtained, DeepWSC further learns the enhanced representation of each word by combining its contextual information. Given a web service description $W^{(s)}$ as a word list, let w_i be the i-th word, $c_l(w_i)$ and $c_r(w_i)$ are denoted as the left and right contextual information of w_i, respectively. In [27], $c_l(w_i)$ and $c_r(w_i)$ are recursively calculated as follows:

$$c_l(w_i) = f\left(W^{(l)} c_l(w_{i-1}) + W^{(sl)} e(w_{i-1})\right) \tag{7}$$

$$c_r(w_i) = f\left(W^{(r)} c_r(w_{i+1}) + W^{(sr)} e(w_{i+1})\right) \tag{8}$$

where f is a non-linear activation function, $W^{(l)}$, $W^{(sl)}$, $W^{(r)}$ and $W^{(sr)}$ are matrices for linear transformation. Here the left context of w_i is obtained by scanning the word list from the left end to w_{i-1} with a recurrent structure. Similarly, the right context of w_i is obtained by scanning the word list from the right end to w_{i+1}. DeepWSC employs the Gated Recurrent Unit (GRU) [28], a widely-adopted recurrent structure, to scan the service description both forward and backward. As the dimensionality of GRU cells, the hyper parameter S_{cell} determines how much contextual information of $c_l(w_i)$ and $c_r(w_i)$ is included to enhance the original word vectors. It can be experimentally adjusted to a fixed setting to boost service clustering performance. By the concatenation of contextual information, the enhanced representation of word w_i in a service description consists of $c_l(w_i)$, $e(w_i)$ and $c_r(w_i)$ as in (9):

$$v_i = [c_l(w_i); e(w_i); c_r(w_i)]. \tag{9}$$

In service clustering tasks, the functionality features of web service descriptions should mainly rely on the embedded vectors of the words, rather than the contextual information that is primarily used as auxiliary. Specifically, the main portion of an enhanced word representation vector originates from the word itself instead of contextual information. As the recurrent structure is a biased model that assigns high weights to later inputs, we magnify the proportion of word w_i in the contextual information by optimizing the calculation process of $c_l(w_i)$ and $c_r(w_i)$ as in (10) (11), where $e(w_i)$ is appended to be scanned at the end of each recursive generation processes of them. After that, the enhanced word representation v_i is concatenated in the same way as in (9).

$${c_l}'(w_i) = f\left(W^{(l)} c_l(w_{i-1}) + W^{(sl)} e(w_i)\right) \tag{10}$$

$${c_r}'(w_i) = f\left(W^{(r)} c_r(w_{i+1}) + W^{(sr)} e(w_i)\right) \tag{11}$$

To generate the latent semantic vector y_i of word w_i based on the enhanced word vector v_i, we apply a linear transformation with a non-linear $\mathrm{leaky\ relu}$ activation function to v_i as in (12), where μ is a slope for negative inputs. Here, we use the $\mathrm{leaky\ relu}$ as the activation function instead of tanh in [27] because the convergence speed of $\tanh$ is much slower than that of relu series of the activation functions. Additionally, if we simply use the pure relu function, when a very large gradient flows through a neuron, it may no longer be activated by any data after updating the parameters.

IEEE Proof

$$y_i = \max\Big(W^{(y)} v_i + b^{(y)}, 0\Big) + \mu \times \min\Big(W^{(y)} v_i + b^{(y)}, 0\Big) \tag{12}$$

To further generate the deep semantic feature of a web service, we apply an element-wise max-pooling that is a sample-based discretization process to capture the most important service characteristics as in (13).

$$h_d = \max_{i=1}^{L_{desc}} y_i \tag{13}$$

where the j-th element of h_d is the maximum among all the jth elements of y_i. As a result, h_d is a vector with fixed-length and contains deep semantic feature of a web service.

To integrate service deep semantic features and composability features, we first add a linear transformation with a leaky relu activation function to h_c as in (14). It transforms h_c to h'_c that shares the same dimensionality with h_d.

$$h'_c = \max\Big(W^{(c)} h_c + b^{(c)}, 0\Big) + \mu \times \min\Big(W^{(c)} h_c + b^{(c)}, 0\Big) \tag{14}$$

Then, the integrated implicit feature h_i of a web service is obtained with the fixed length by adding h'_c to h_d. The reason why we do not integrate these two kinds of features in a concatenating way is that not all the services have composability information. Specifically, for services that do not appear in any of the mashup services, their service composability features cannot be mined, resulting in h_c being a zero vector. In such case, when the integration is performed in the concatenating way, the distance between two services without composability information decreases compared with others, because parts of the elements of their service integrated implicit features are exactly equivalent. Conversely, integrating service deep semantic feature and service composability feature in a vector-adding way can address this issue.

4.3 Service Latent Topic Distribution Generation

In service clustering, there is no service domain label as ground truth. The service feature extractor cannot be trained in a supervised manner. Inspired by [29], we utilize an augmented probabilistic topic model, WE-LDA [14], to assign each web service a probabilistic topic distribution vector θ with K elements. These vectors act as a partially correct domain labels that help train the service feature extractor.

We train a WE-LDA with Skip-Gram algorithm on all the service descriptions [14]. All the terms in the data set constitute the term set V. They are represented as word vectors by a trained Word2vec model [30]. Then, K-means++ algorithm is employed to cluster these terms into K clusters, denoted as $\Omega = \{\omega_1, \omega_2, \ldots, \omega_K\}$. Ω is used to semi-supervise the sampling process for the words in the service descriptions as in (15), where α, β and λ are prior parameters, $n_{t,\neg i}^{(w_i)}$ indicates the number of times that word w_i is observed with topic t, $n_{t,\neg i}^{(\cdot)}$ denotes the number of times words in V are assigned to topic t, $v_{t,\neg i}^{(s)}$ denotes the number of times words are assigned to topic t in the functionality description of s, and $v_{\cdot,\neg i}^{(s)}$ is the number of all the words in the functionality description of s.

$$\begin{aligned} p(z_i = t|\Omega, \lambda) \propto\ & \frac{n_{t,\neg i}^{(w_i)} + \beta}{n_{t,\neg i}^{(\cdot)}} \times \frac{v_{t,\neg i}^{(s)} + \alpha}{v_{\cdot,\neg i}^{(s)} + K\alpha} \\ & \times \prod\nolimits_{w_{j,\neg i}\in\omega_i} \exp\left(\frac{\lambda}{|\omega_i|} \times z_t^{(w_{j,\neg i})}\right) \end{aligned} \tag{15}$$

After the above sampling process, the service latent topics can be obtained as in (16), where $p(z_i = t)$ represents the topic probability of topic t.

$$p(z_i = t) \propto \frac{v_t^{(s)} + \alpha}{v_{\cdot}^{(s)} + K\alpha} \tag{16}$$

Finally, the trained WE-LDA model generates a probabilistic topic distribution vector θ for each web service, which are used as service labels to train the service feature extractor.

4.4 Model Training and Service Clustering

Fig. 5d illustrates how we train the service feature extractor in DeepWSC by updating its parameters. First, a fully-connected layer is added to transform the service integrated implicit feature h_i to O as in (17). O is an unnormalized probabilistic topic distribution of a web service which has the same dimensionality with θ derived from WE-LDA model.

$$O = W^{(O)} h_i + b^{(O)} \tag{17}$$

To convert vector O into a normalized topic probabilistic distribution $P = \{p_1, p_2, \ldots, p_K\}$ to align with θ, we apply softmax function to O as in (18), where each p_i represents the probability of a web service that belongs to the ith topic.

$$p_i = \frac{\exp(O_i)}{\sum_{j=1}^{K} \exp(O_j)} \tag{18}$$

Consequently, the objective loss function of the model training of all the N web services is defined as in (19), where Θ denotes all the parameters to be trained as in (20). The loss function J calculates the summation of the cross entropy between P and θ of all the web services.

$$J = -\sum_{i=1}^{N} (P\log\theta + (1-P)\log(1-\theta)|s_i, \Theta) \tag{19}$$

$$\begin{aligned} \Theta = \{ & W^{(l)}, W^{(sl)}, W^{(r)}, W^{(sr)}, W^{(y)}, b^{(y)}, \\ & W^{(c)}, b^{(c)}, W^{(O)}, b^{(O)}\} \end{aligned} \tag{20}$$

The aim of the training process is to minimize J for all the web services. We use the Adam [31] optimizer to update all the parameters that need to be trained in DeepWSC's RCNN for service implicit feature extraction as in (21)

$$\Theta - \eta\frac{\partial J}{\partial \Theta} \rightarrow \Theta \tag{21}$$

where η is an initial learning rate. When the model converges, the fitting layer and the WE-LDA model are removed, the remaining components are used to generate integrated implicit feature h_i for web service clustering.

The generated integrated implicit features of all the web services share the same dimensionality, thus, the widely-

IEEE Proof

TABLE 1
Distribution of the Number of Web Services in Each Category

Category	# of services	Category	# of services
Tools	887	Telephony	342
Financial	757	Security	312
Messaging	591	Reference	304
eCommerce	553	Email	299
Payments	553	Search	290
Social	510	Travel	294
Enterprise	509	Video	281
Mapping	429	Education	277
Government	371	Advertising	274
Science	357	Transportation	269

TABLE 2
Statistics of the Service Data Set

Item Name	Value	Item Name	Value
Number of Services	8,459	Length (min)	16
Number of Categories	20	Length (average)	69.3
Number of Terms	25,479	Number of Vertices	513
Length (max)	406	Number of Links	2,421

used K-means++ algorithm can be applied to partition web services into several functionally differentiated clusters.

5 Experiments

5.1 Experimental Setup and Data Set

All the experiments were conducted on our workstation equipped with an NVIDIA GTX 1080TI GPU, an Intel(R) Xeon(R) Gold 6130 @2.60 GHz CPU and 192 GB RAM.

To validate the performance of DeepWSC, we crawled web services from ProgrammableWeb until July 1, 2018. This data set contains 17,923 real-world web services with domain labels and 6,392 mashup services, and is available on GitHub.[3] These web services correspond to 384 categories by their domain labels. However, the number of services in each category is uneven, i.e., the category *Tools* contains 887 web services while *Solar* only contains two. To prevent DeepWSC from being impacted by extremely small clusters, we conducted experiments on the top 20 categories with the most web services. Note that service domain labels are only used in the clustering evaluation, instead of the training process of DeepWSC. The experimental data set contains 8,459 web services. The numbers of web services in each category are listed in Table 1, and more statistical information about the data set is shown in Table 2.

As for the service composability network, we select 513 web services that are used by the 6,392 mashup services as its vertices. Then, we add weighted links among the vertices according to their composability relationships, where the number of links in the network is 2,421. Then, an SGCN model is trained on this service composability network with the implementation published on GitHub.[4]

As for the word embedding model, BERT, we use an open source implementation on GitHub.[5] The pre-trained BERT model we used in our experiments can be accessed online.[6]

5.2 Evaluation Metrics

We evaluate the performance of DeepWSC by four widely-used evaluation metrics: Purity, Normalized Mutual Information (NMI), Recall and F_1-measure. Let $\mathbb{C}^* = \{c_1^*, c_2^*, \ldots, c_K^*\}$ be the set of K original categories, $\mathbb{SC} = \{sc_1, sc_2, \ldots, sc_K\}$ be the set of K partitioned service clusters, and n be the number of web services to be clustered in the data set. Purity calculates the proportion of correctly clustered services to the total number of services and is calculated as in (22).

$$\mathrm{Purity}(\mathbb{SC}, \mathbb{C}^*) = \frac{1}{n}\sum_{i=1}^{K} \max_j \left| sc_i \cap c_j^* \right| \tag{22}$$

NMI evaluates clustering based on information theory. It is calculated as in (23), (24) and (25), where P indicates the probability that the service appears in the corresponding set.

$$\mathrm{NMI}(\mathbb{SC}, \mathbb{C}^*) = \frac{\mathrm{I}(\mathbb{SC}; \mathbb{C}^*)}{(\mathrm{H}(\mathbb{SC}) + \mathrm{H}(\mathbb{C}^*))/2} \tag{23}$$

$$\mathrm{I}(\mathbb{SC}; \mathbb{C}^*) = \sum_{i=1}^{K}\sum_{j=1}^{K} \mathrm{P}\left(sc_i \cap c_j^*\right) \log \frac{\mathrm{P}\left(sc_i \cap c_j^*\right)}{\mathrm{P}(sc_i) \cap \mathrm{P}\left(c_j^*\right)} \tag{24}$$

$$\mathrm{H}(\mathbb{SC}) = -\sum_{i=1}^{K} \mathrm{P}(sc_i) \log \mathrm{P}(sc_i) \tag{25}$$

Recall and F_1-measure regard the clustering as a series of decisions. Recall is calculated as in (26), where TP and FN denote the numbers of decisions that two similar services are assigned to the same and different clusters, respectively.

$$Recall = \frac{TP}{TP + FN} \tag{26}$$

F_1-measure combines Recall and Precision. It is calculated as in (28), where Precision is defined as in (27).

$$Precision = \frac{TP}{TP + FP} \tag{27}$$

$$F_1\text{-}measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{28}$$

All the four evaluation metrics are real numbers ranged in $[0, 1]$. Higher values indicate the higher clustering accuracy.

5.3 Competing Methods

Our main approach is DeepWSC implemented by an RCNN integrated with the service composability relationships and an WE-LDA model, called DeepWSC (RCNN, WE-LDA, Heuristics). To demonstrate the clustering performance, we compare it with nine competing methods. In the following, we refer to LDA and WE-LDA model as "LDAs". The comparing methods are detailedly described as below.

3. https://github.com/zhenqincn/ProgrammableWebDataSet
4. https://github.com/benedekrozemberczki/SGCN
5. https://github.com/hanxiao/bert-as-service
6. https://storage.googleapis.com/bert_models/2018_10_18/uncased_L-12_H-768_A-12.zip

IEEE Proof

TABLE 3
Performance Comparisons of Web Service Clustering Among Competing Methods

Methods	Purity	NMI	Recall	F_1-measure
TF-IDF	0.4673	0.3930	0.3427	0.1996
LDA+K	0.5200	0.4262	0.3199	0.3383
LDA	0.5285	0.4341	0.3321	0.3503
WE-LDA+K	0.5372	0.4363	0.3282	0.3466
WE-LDA	0.5420	0.4403	0.3370	0.3543
DeepWSC (Text-CNN, LDA)	0.5400	0.4625	0.3484	0.3662
DeepWSC (Text-CNN, WE-LDA)	0.5553	0.4668	0.3572	0.3733
DeepWSC (RCNN, LDA)	0.5492	0.4704	0.3614	0.3784
DeepWSC (RCNN, WE-LDA)	0.5708	0.4856	0.3821	0.3969
DeepWSC (RCNN, WE-LDA, Heuristics)	**0.6379**	**0.5273**	**0.4186**	**0.4356**
Gains on WE-LDA	17.69%	19.76%	24.21%	22.95%
Gains on DeepWSC (without Heuristics)	11.76%	8.59%	9.55%	9.75%

- TF-IDF [12]: This method obtains service features based on the term frequency and inverse document frequency. Based on the syntactic service features, the K-means++ algorithm is adopted to cluster web services.
- LDA [13]: It generates a probabilistic topic distribution vector for each web service, which is assigned to the latent topic with the maximum probability. Web services assigned to the same latent topic are partitioned in the same cluster.
- LDA+K [4]: Unlike pure LDA, this method calculates the similarity between two web services based on the probabilistic topic distribution vectors from LDA and then employs the K-means++ algorithm to cluster web services.
- WE-LDA [14]: It uses the WE-LDA model to assign web services to latent topics that have the highest values in their probabilistic topic distributions. Web services assigned to the same latent topic are clustered together.
- WE-LDA+K [14]: This method first calculates the similarity between two web services based on the probabilistic topic distribution vector using WE-LDA and then employs the K-means++ algorithm to cluster web services.
- DeepWSC (Text-CNN, LDA): It is our first self-developed method from [3]. Its service feature extractor is implemented with a Text-CNN [32] trained based on LDA.
- DeepWSC (Text-CNN, WE-LDA): It is our second self-developed method from [3]. Its service feature extractor is implemented with a Text-CNN trained under the guidance of a WE-LDA model.
- DeepWSC (RCNN, LDA): It is our third self-developed method from [3]. Its service feature extractor is implemented with an RCNN trained based on an LDA model.
- DeepWSC (RCNN, WE-LDA): It is our self-developed method from [3] with the best performance. Its service feature extractor is implemented with an RCNN trained under the guidance of a WE-LDA model.

5.4 Experiment Results

5.4.1 Overall Clustering Comparison

For the TF-IDF method, we perform the K-means++ algorithm for five times based on the term frequency and inverse document frequency features and calculate its average performance. For the four LDAs-based methods, we conduct a group of experiments under different prior-parameter settings and select the LDAs with the best performance. The best LDAs are obtained when both α and β are 0.1 and λ is 3.0 in the WE-LDA model. For the four self-developed methods from [3], we conduct the experiments for five times at each of the different hyperparameter settings and then select the trained models with the best average performance for feature extraction and web services clustering. For our two new methods, we conduct a series of experiments with different hyperparameter combinations to find the best hyperparameter setting. Then, we performed model training and service clustering for five times under the best hyperparameter setting that yields the average performance. Table 3 compares the performance in web services clustering among the eleven competing methods.

It can be observed from Table 3 that DeepWSC (RCNN, WE-LDA, Heuristics) outperforms the five existing traditional methods, including TF-IDF and the LDAs-based ones. Taking the best two traditional methods, WE-LDA and WE-LDA+K, as an example, DeepWSC (RCNN, WE-LDA, Heuristics) achieves an average advantage of 21.15 percent over WE-LDA and 23.21 percent over WE-LDA+K across all the evaluation metrics. It proves the superior clustering performance of our new DeepWSC.

To further validate the performance of our new DeepWSC that combines service composability features and utilizes a new word embedding method, we compare DeepWSC (RCNN, WE-LDA, Heuristics) with the version without the heuristics. The results show that the former is on average 9.91 percent better than the latter. This indicates that the enhancement on DeepWSC presented in this paper can effectively improve the clustering performance.

To evaluate the effectiveness of service deep semantic features, we remove the use of the service composability features from DeepWSC and obtain the self-developed approach named DeepWSC (RCNN, WE-LDA), where the service implicit feature is only composed of service deep semantic feature obtained by the RCNN model. It is observed from Table 3 that DeepWSC (RCNN, WE-LDA) outperforms the existing LDAs-based methods. Specifically, it achieves an average advantage of 10.25 percent over WE-LDA and 12.12 percent over WE-LDA+K across all the

IEEE Proof

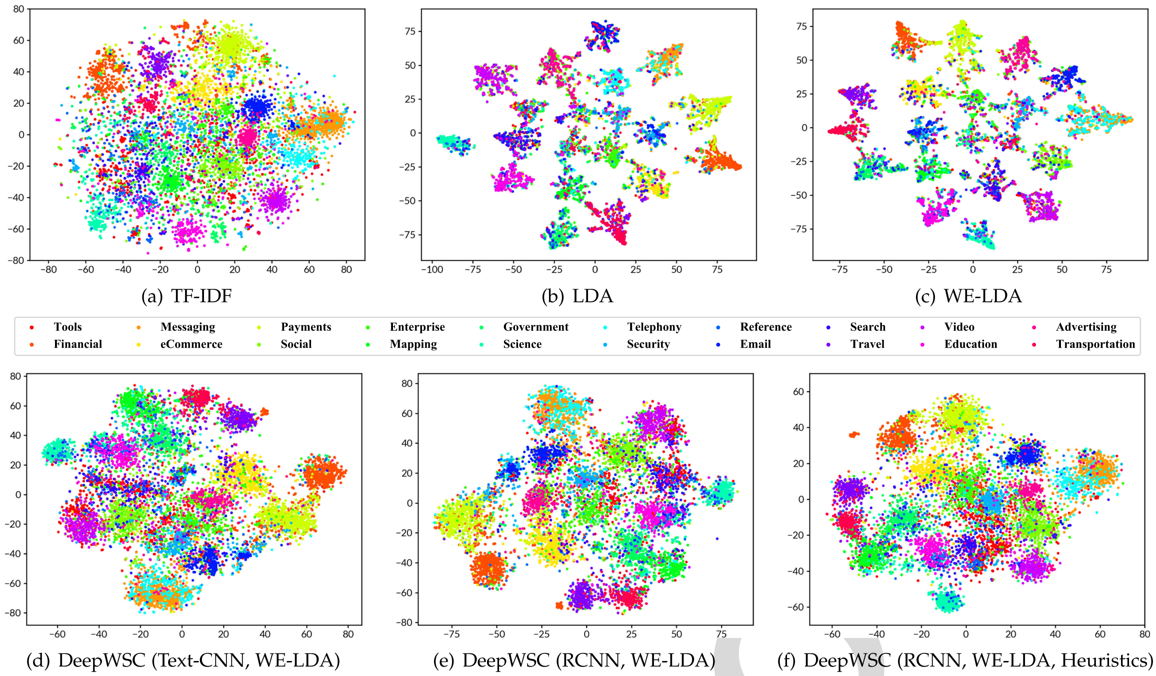


Fig. 6. 2-dimensional service clustering visualization where web service features are extracted by competing approaches. Each point represents a web service, and the points sharing the same color represent that these services belong to the same domain.

evaluation metrics. Moreover, by replacing DeepWSC (RCNN, WE-LDA) with a Text-CNN [32], DeepWSC (Text-CNN, WE-LDA) is compared with WE-LDA. The results show that it has an average advantage of 4.75 percent over WE-LDA and 6.72 percent over WE-LDA+K across all the evaluation metrics. Therefore, we can demonstrate that the use of deep semantic features of web services helps DeepWSC outperforms LDAs-based methods that only leverages explicit semantic features.

From the above, we conclude that DeepWSC, taking into account deep semantic feature and composability feature of web services, outperforms all the comparing methods for clustering web services in multiple evaluation metrics.

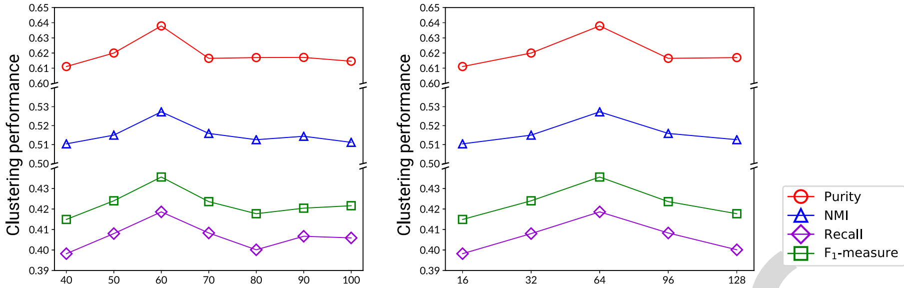
5.4.2 *Analysis of Scatter Diagrams*

To illustrate the features of web services for further clustering by different approaches, we visualize 2-dimensional embeddings of service features in Fig. 6 by t-SNE [33]. For our DeepWSC based methods shown in Figs. 6d, 6e, and 6f, the results exhibit clear color boundaries between different categories of web services. Especially, in Fig. 6f, the colors of the points within each cluster are almost the same, which indicates that most of the web services in each cluster share the same domain label. For LDAs-based methods as illustrated in Figs. 6b and 6c, many web service points in different colors are included in the same clusters, leading to a low service clustering accuracy.

We observe that the service features embedded in Figs. 6b and 6c show much clearer clusters but less color boundaries compared with those in Figs. 6d, 6e, and 6f. It can be explained that the 2-D embeddings are obtained by reducing the service features generated by corresponding approaches. For LDAs, the dimensionality of a service feature is K, where each element represents the relevance between a web service to its corresponding latent topic. Therefore, most service features obtained by LDAs have an element whose value is much higher than the others, resulting in the 2-D embeddings to form more obvious clusters. As for DeepWSC, the generated integrated implicit features of web services are very high-dimensional, of which any single element has no specific meaning. Accordingly, which elements in the service features have relatively larger values is irregular, leading to a relatively lower marginal distance between service clusters of the 2-D embeddings. However, service features from DeepWSC contain more deep semantic features and composability relationships as heuristics than those from LDAs. By applying K-means++, these high-dimensional service features can be better clustered according to their functionalities compared with those generated by LDAs.

More importantly, from the 2-D embeddings, we observe some interesting and meaningful phenomena out of domain labels. As shown in Figs. 6c, 6d, and 6e, the 2-D embeddings of web services affiliated to *Message* and *Telephony* are mixed together, however, they are partitioned into two distinct clusters in Figs. 6a and 6b. The primary reason of this phenomenon is the functionality descriptions of web services within the two categories share high similarities in semantics. For example, web service *Twilio SMS*[7] and *TelAPI*[8] have the similar semantic meanings from the perspective of human understanding. As we know, TF-IDF and LDA are based on Bag-of-Words model, where description terms with similar semantics are regarded as completely unrelated ones. Since functionality descriptions of service within *Message* and *Telephony* contain specific terms related to their corresponding domains, the service features obtained by TF-IDF and LDA can generate two distinct clusters. Furthermore, WE-LDA as

7. https://www.programmableweb.com/api/twilio-sms
8. https://www.programmableweb.com/api/telapi



(a) L_{desc} (b) S_{batch}

Fig. 7. Clustering performance of DeepWSC versus description length L_{desc} and batch size S_{batch}.

the word embeddings augmented LDA model leverages the augmented semantics from word vectors to boost its service clustering performance, while DeepWSC (Text-CNN, WE-LDA) and DeepWSC (RCNN, WE-LDA) depend on the deep semantic features to cluster web services. Both the augmented semantics and deep semantic features enable the corresponding approaches to handle functionality descriptions closer to human understanding, leading to small distance between the extracted service features of the two categories in Figs. 6c, 6d, and 6e. It may lead to better clustering results in terms of human understanding, instead of evaluated by domain labels.

In DeepWSC (RCNN, WE-LDA, Heuristics), service composability features are introduced as heuristics and integrated into deep semantic features. As show in Fig. 6f, web services within *Message* and *Telephony* are partitioned as two adjacently mixed but mutually independent clusters. Since they belong to different domains, the spatial distance between their composability features is relatively far, which is beneficial to better differentiate the two categories of web services with semantically similar functional descriptions. That is, the 2-D embeddings of web services with similar semantics of functional descriptions are gathered together, while the service composability features as heuristics can help divide them into independent clusters. It improves clustering accuracy in terms of domain labels, as shown in Table 3. In such case, when the granularity of service clustering is enlarged in application scenarios, web services within the two categories could be merged together as one cluster. As a result, we can obtain diverse and reasonable clusters with other granularities different from domain labels.

Additionally, we also found that there is a category *Tools* where some services are incorrectly clustered. Specifically, the 2-D embeddings of many services within this category are scattered throughout the plane space of Fig. 6. The underlying reason is that the category of *Tools* is so abstract that it may contain a variety of web services across multiple different functionalities. This phenomenon may potentially decrease the clustering accuracy in terms of both human understanding and domain labels.

5.5 The Performance Impact of Hyperparameters

In the experiments, two groups of hyperparameters impact the clustering performance of DeepWSC. (1) The quality of the service feature extractor is mainly related to four hyperparameters, including the service description length L_{desc}, training batch size S_{batch}, the size of GRU cells S_{cell}, and the dimensionality of integrated implicit features S_{h_i}. (2) Additionally, α as a hyperparameters in WE-LDA model reflects the relationships among latent topics that impacts the quality of the service feature extractor. We test the two groups of hyperparameters and analyze how they impact the clustering performance of DeepWSC. Fig. 7 presents the clustering performance of DeepWSC with different values of L_{desc} and S_{batch} by four evaluation metrics.

Service descriptions are different from common short texts. Usually, it first describes the service functionality and then how to invoke the service. It is observed that clustering task mainly focuses on service functionality descriptions rather than invocation descriptions. Moreover, a lot of information in the invocation descriptions is repetitive. It hinders the extraction of service integrated implicit features, lowering the clustering performance. In the experiments, we uniformly prune the service descriptions to L_{desc} words in order to remove the invocation information in the latter part of a service description and improve the quality of service clustering. Fig. 7a illustrates the changes of DeepWSC's clustering performance under different settings of L_{desc}. DeepWSC achieves the best clustering accuracy when L_{desc} is set as 60. However, if we abandon too many words, e.g., setting L_{desc} as 40, DeepWSC's clustering performance decreases due to the loss of functionality description.

Note that uniformly pruning the tail of all the services potentially causes the useful parts of descriptions to be lost. Theoretically, the best way to eliminate the influence of the redundant descriptions is to individually identify the useless part of each service description and prune it accordingly. However, due to the lack of existing precise technique to recognize the redundant part of descriptions individually, we use an uniform but effective scheme to prune the descriptions to a suitable length in order to reduce the influence of the redundant parts. It improves the performance of DeepWSC over the entire web service repository.

A smaller S_{batch} allows DeepWSC to more precisely learn the integrated implicit features. However, it may trigger overfitting of the trained model by amplifying the discriminations among integrated implicit features. Conversely, a large S_{batch} allows DeepWSC to once handle more services when calculating the loss function, which may cause it underfitting. Therefore, how to balance the value of S_{batch} is a tradeoff. The performance achieved by DeepWSC with different S_{batch} is shown in Fig. 7b. It is observed that when S_{batch} is set as 64, DeepWSC achieves the best clustering performance. As it becomes larger or smaller, the performance decreases.

IEEE Proof

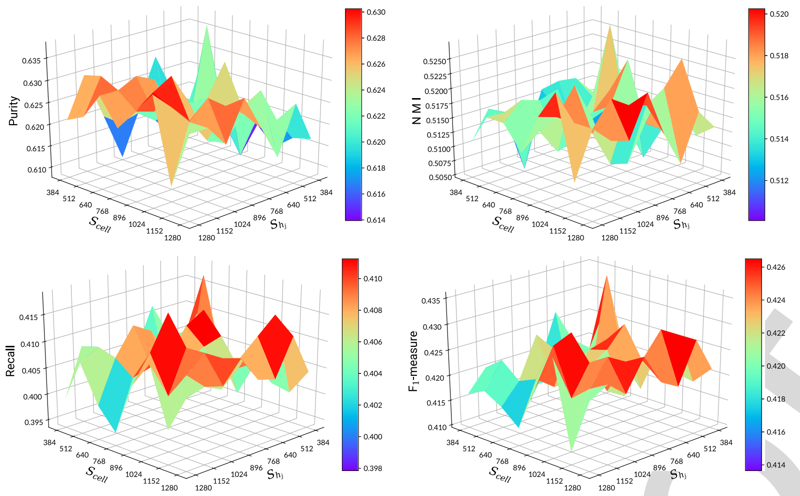


Fig. 8. Clustering performance of DeepWSC with different combinations of feature size S_{h_i} and cell size S_{cell}.

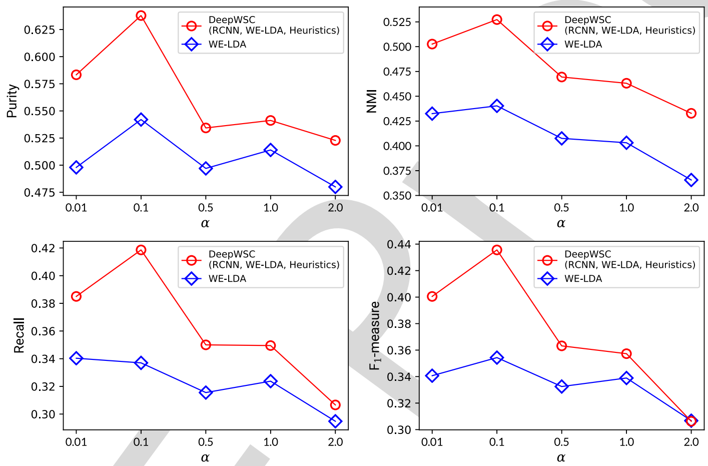


Fig. 9. Clustering performance between DeepWSC and WE-LDA with different hyperparameter α.

When S_{cell} and S_{h_i} are overly large, redundant information may be included in the enhanced representations of words and integrated implicit features of web services. Conversely, if they are too small, DeepWSC may not be able to acquire rich information when extracting integrated implicit features. To find the best setting of S_{cell} and S_{h_i}, we train DeepWSC by tuning the combination of them under a wide range of search space. Fig. 8 shows the clustering performance with different combinations of S_{cell} and S_{h_i}, indicating that DeepWSC achieves the best performance when S_{cell} and S_{h_i} are set as 512 and 640, respectively.

Since the deep neural network in DeepWSC is trained based on LDAs, the hyperparameter α in WE-LDA impacts the clustering accuracy. Fig. 9 compares the clustering performance between DeepWSC and WE-LDA with different values of α on four evaluation metrics. It shows that DeepWSC outperforms WE-LDA under different settings of α across all the evaluation metrics, and they both achieve the best performance when α is set to 0.1.

5.6 Time Overhead

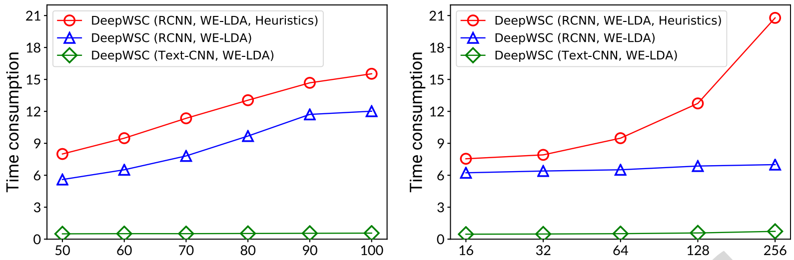
To evaluate the applicability of DeepWSC, we present the additional time overhead incurred by training and applying the service feature extractor. Since it is closely related to L_{desc} and S_{batch}, we change them to test DeepWSC's corresponding time overhead by fixing the other hyperparameters to their optimal values.

As shown in Fig. 10a, the additional time overhead of DeepWSC (RCNN, WE-LDA, Heuristics) is at its top with approximately 15 minutes when L_{desc} is 90, and at its bottom wtih around 8 minutes when L_{desc} is 50. In the case with the highest service clustering accuracy (L_{desc} is set to 60), the additional time overhead is about 9 minutes, which is a short time compared with the consumption for training a WE-LDA model. In regard to DeepWSC (RCNN, WE-LDA), the additional time overhead is slightly lower. Furthermore, when we use Text-CNN as the service feature extractor, the additional time overhead is lower than a minute. Similarly, the additional time overhead of DeepWSC increases along with the growth of S_{batch}, which is shown in Fig. 10b.

5.7 Discussion

5.7.1 *Clustering Accuracy Discussion*

Intuitively, the clustering accuracy of DeepWSC can be further improved by more sophisticated techniques. Currently, it is mainly restricted by the following three reasons.



(a) L_{desc} (b) S_{batch}

Fig. 10. Additional time overhead for training and applying the service feature extractor in DeepWSC along with the changes of L_{desc} and S_{batch}.

First, there are some traditional web service clustering approaches which present a relatively high clustering accuracy [17], [18], [19], [20], [21], however, they mainly cluster web services described in standard and strictly structured language such as WSDL. In recent years, web services described in natural language are becoming mainstream, e.g., ProgrammableWeb as the world's largest online web service repository, manages all the API and mashup web services that are described in natural language. Since natural language has more expressiveness than WSDL, it also brings greater flexibility and complexity. Existing widely-used advanced approaches of natural language oriented applications have demonstrated the common disadvantage of achieving relatively low accuracy [27], [29], [32]. Thus, web service clustering approaches on ProgrammableWeb are intrinsically difficult to receive a very high absolute value of clustering accuracy compared with those by WSDL descriptions.

Second, the number of web services and their corresponding domains can both affect the clustering accuracy. As observed in [18], [19], the performance of service clustering declines quickly as the number of web services increases. Although these approaches [17], [18], [19], [20], [21] can achieve a relatively high clustering accuracy, they are experimentally validated on a small scale data set, including no more than 500 web services distributed in less than 5 categories. On the contrary, the experimental results of our research are obtained on a large number of real-world web services from ProgrammableWeb, leading to the clustering accuracy with possibility of being further improved.

Finally, as discussed in Section 5.4.2, the composability of service invocations as heuristics can help distinguish those semantically similar web services that are distributed among different domains, which can effectively promote the service clustering performance when domain labels are taken as evaluation criteria. However, as shown in Table 2, there are only 513 web services invoked by mashup services, indicating that most web services have not been invoked by any mashup service, resulting in the limited heuristic information mined from service composability relationships. This phenomenon is also mentioned in [1], which may potentially affect the improve of service clustering accuracy of DeepWSC.

5.7.2 *Threats to Validity*

In the experiments, the performance of DeepWSC can be affected by the tuning and optimization of hyperparameters. As show in Section 5.5, L_{desc}, S_{batch}, S_{cell}, S_{h_i} and α can affect the accuracy of service clustering of DeepWSC. However, there is no effective theoretical method to guide the settings of these hyperparameters. The process of finding a set of optimal hyperparameters depends on the comparison of multiple rounds of experiments. Thus, when faced with different data sets, it is better to reassign suitable hyperparameter values to ensure the effectiveness of the approach.

6 Related Work

Functional-based service clustering initially focused on the similarity of service functionality descriptions by WSDL. Some researchers applied text mining techniques to WSDL descriptions to cluster web services. Elgazzar *et al.* proposed [5] an approach that used five key features extracted from WSDL descriptions to cluster web services. In order to improve the clustering accuracy, subsequent researchers proposed enhanced approaches that aim at further mining semantic information in WSDL descriptions based on topic models [12], [13]. However, due to the limited number of terms in WSDL descriptions, it is difficult for these traditional approaches to precisely obtain service features.

Furthermore, some researchers exploited domain ontologies to cluster web services. Xie *et al.* [17] proposed an ontology-based semantic clustering approach which measured the service similarity in two aspects, including function similarity and process similarity. Kumara *et al.* [18] proposed a hybrid approach which calculated service similarity by generating an ontology via hidden semantic patterns, or using an information retrieval-based way equipped with term-similarity measuring techniques. Additionally, they proposed an approach to identify cluster centers by using similarity values and TF-IDF values of service names, handling the issue that clustering accuracy can be affected by unsuitable cluster centers. Considering domain specific terms can describe more semantic information than general ones, Rupasingha *et al.* [19] proposed an ontology-based web service clustering approach that takes domain specificity into account. It is observed that these approaches are based on WSDL or other strictly structured semantic oriented description languages, which are more conductive to perform concept extraction from constructed ontology and useful to mine semantic features for clustering web services. However, it is hard for domain ontologies to cover all functionality descriptions of web services across multiple application domains. What's more, when facing a large number of services from different service providers or developers, it is a challenging and labor-intensive task to manually designate domain terms from ontologies for service descriptions.

IEEE Proof

In recent years, there are also context-based approaches for web service clustering. Zhang *et al.* [20] proposed a context-based approach that jointly inherent WSDL service description and service usage context. The topology modeled from service collaboration relationships is used to make up for the limitations of keyword retrieval in WSDL web services. However, it is difficult to obtain the topology of service collaboration relationships from online web service repository. Kumara *et al.* [21] extracted domain context and generated feature vectors through WSDL web services, which is fed to train SVMs whose outputs are converted to the posterior probabilities for calculating term similarity. The calculated term similarity is used to help fine-tuning the wrongly clustered web services. Unlike the ontology-based and context-based approaches, DeepWSC aims to cluster web services described in unstructured natural language, which has been becoming the mainstream way for building service-oriented software systems.

As API-based web services described by natural language become increasingly popular and mainstream, many researchers have investigated new approaches to cluster web services described in natural language. Shi *et al.* [14] proposed a word embedding augmented LDA model for web service clustering, which extracts service semantic feature from service functionality description and leads to superior service clustering accuracy. He *et al.* [15] extended probabilistic model for more accurate service clustering by incorporating mutual invocation relationships with service characterization. Cao *et al.* [16] boosted clustering performance by taking into account service invocation relationships and the tags, where the semantic features are extracted by a Doc2vec model.

With the advances of deep learning, many approaches based on deep neural network outperform traditional ones in natural language processing tasks [23], [27], [29], [32]. Some researchers have also utilized deep learning techniques to boost the accuracy of web service classification. Yang *et al.* [34] proposed a deep neural network called ServeNet, which applied CNN to obtain local relations and LSTM to retain global long-term dependencies. By the combination of deep neural networks, it can automatically extract high-level features without manual feature engineering, which achieves the state-of-the-art performance on web service classification task. The main differences between ServeNet with our approach DeepWSC are twofold. On one hand, ServeNet is trained in a supervised manner for web service classification, while DeepWSC is designed as an unsupervised manner for web service clustering. On the other hand, ServeNet mainly combines deep neural networks to extract deep semantic features from functionality descriptions, while it has not taken into account any domain heuristics. However, DeepWSC not only leverages deep neural network to extract deep semantic features of web services, but also incorporates composability features of web services. These two kind of features are jointly synthesized as integrated implicit features for more precisely clustering web services.

7 Conclusion and Future Works

In this paper, we propose a novel framework for web service clustering that integrates deep neural network with service composability relationship, called DeepWSC. It first generates deep semantic features and service composability features that can be leveraged by the deep neural network. Then, we train a service feature extractor to extract integrated implicit feature of each web service. Finally, the task of clustering web services is performed by a widely-used K-means++ clustering algorithm. The results demonstrate that DeepWSC outperforms the state-of-the-art approaches for web service clustering in multiple evaluation metrics.

In the future, we plan to further explore advanced clustering algorithms to improve the clustering accuracy and obtain diverse granularities of service clusters.

Acknowledgments

This work was supported in part by the National Key Research and Development Program of China (No. 2017YFC0907505), Shanghai Natural Science Foundation (No. 18ZR1414400, 17ZR1400200), National Natural Science Foundation of China (No. 61772128, 61602109), and Shanghai Sailing Program (No. 16YF1400300).

References

[1] 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 Trans. Services Comput.*, vol. 8, no. 5, pp. 674–687, Sep./Oct. 2015.
[2] F. Chen, S. Yuan, and B. Mu, "User-QoS-based web service clustering for QoS prediction," in *Proc. IEEE Int. Conf. Web Services*, 2015, pp. 583–590.
[3] G. Zou, Z. Qin, Q. He, P. Wang, B. Zhang, and Y. Gan, "DeepWSC: A novel framework with deep neural network for web service clustering," in *Proc. IEEE Int. Conf. Web Services*, 2019, pp. 434–436.
[4] B. Cao *et al.*, "Mashup service clustering based on an integration of service content and network via exploiting a two-level topic model," in *Proc. IEEE Int. Conf. Web Services*, 2016, pp. 212–219.
[5] 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*, 2010, pp. 147–154.
[6] Y. Xia, P. Chen, L. Bao, M. Wang, and J. Yang, "A QoS-aware web service selection algorithm based on clustering," in *Proc. IEEE Int. Conf. Web Services*, 2011, pp. 428–435.
[7] B. Cao, X. Liu, M. M. Rahman, B. Li, J. Liu, and M. Tang, "Integrated content and network-based service clustering and web apis recommendation for mashup development," *IEEE Trans. Services Comput.*, vol. 13, no. 1, pp. 99–113, Jan./Feb. 2020.
[8] 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, Third Quarter 2010.
[9] M. Shi, Y. Tang, and J. Liu, "Functional and contextual attention-based LSTM for service recommendation in mashup creation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 5, pp. 1077–1090, May 2019.
[10] M. Shi, J. Liu, D. Zhou, M. Tang, F. Xie, and T. Zhang, "A probabilistic topic model for mashup tag recommendation," in *Proc. IEEE Int. Conf. Web Services*, 2016, pp. 444–451.
[11] 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, Fourth Quarter 2013.
[12] L. Chen *et al.*, "WTCluster: Utilizing tags for web services clustering," in *Proc. Int. Conf. Service-Oriented Comput.*, 2011, pp. 204–218.
[13] L. Chen, Y. Wang, Q. Yu, Z. Zheng, and J. Wu, "WT-LDA: User tagging augmented LDA for web service clustering," in *Proc. Int. Conf. Service-Oriented Comput.*, 2013, pp. 162–176.
[14] M. Shi, J. Liu, D. Zhou, M. Tang, and B. Cao, "WE-LDA: A word embeddings augmented LDA model for web services clustering," in *Proc. IEEE Int. Conf. Web Services*, 2017, pp. 9–16.
[15] D. He *et al.*, "A probabilistic model for service clustering-jointly using service invocation and service characteristics," in *Proc. IEEE Int. Conf. Web Services*, 2018, pp. 302–305.

IEEE Proof

[16] Y. Cao, J. Liu, M. Shi, B. Cao, X. Zhang, and Y. Wang, "Relationship network augmented web services clustering," in *Proc. IEEE Int. Conf. Web Services*, 2019, pp. 247–254.

[17] L. Xie, F. Chen, and J. Kou, "Ontology-based Semantic Web services clustering," in *Proc. IEEE Int. Conf. Ind. Eng. Eng. Manage.*, 2011, pp. 2075–2079.

[18] B. T. G. S. Kumara, I. Paik, and W. Chen, "Web-service clustering with a hybrid of ontology learning and information-retrieval-based term similarity," in *Proc. IEEE Int. Conf. Web Services*, 2013, pp. 340–347.

[19] R. A. H. M. Rupasingha, I. Paik, and B. T. G. S. Kumara, "Improving web service clustering through a novel ontology generation method by domain specificity," in *Proc. IEEE Int. Conf. Web Services*, 2017, pp. 744–751.

[20] R. Zhang, K. Zettsu, T. Nakanishi, Y. Kidawara, and Y. Kiyoki, "Context-based web service clustering," in *Proc. Int. Conf. Semantics Knowl. Grid*, 2009, pp. 192–199.

[21] B. T. G. S. Kumara, I. Paik, H. Ohashi, W. Chen, and K. R. C. Koswatte, "Context aware post-filtering for web service clustering," in *Proc. IEEE Int. Conf. Services Comput.*, 2014, pp. 440–447.

[22] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, 2003.

[23] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Annu. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Human Lang. Technol.*, 2019, pp. 4171–4186.

[24] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proc. Annu. ACM-SIAM Symp. Discrete Algorithms*, 2007, pp. 1027–1035.

[25] S. Wang, J. Tang, C. Aggarwal, Y. Chang, and H. Liu, "Signed network embedding in social media," in *Proc. SIAM Int. Conf. Data Mining*, 2017, pp. 327–335.

[26] T. Derr, Y. Ma, and J. Tang, "Signed graph convolutional networks," in *Proc. IEEE Int. Conf. Data Mining*, 2018, pp. 929–934.

[27] S. Lai, L. Xu, K. Liu, and J. Zhao, "Recurrent convolutional neural networks for text classification," in *Proc. AAAI Conf. Artif. Intell.*, 2015, pp. 2267–2273.

[28] K. Cho *et al.*, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2014, pp. 1724–1734.

[29] J. Xu *et al.*, "Short text clustering via convolutional neural networks." in *Proc. Annu. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Human Lang. Technol.*, 2015, pp. 62–69.

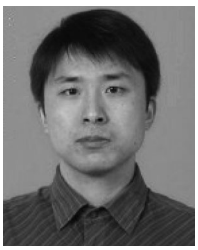
[30] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Advances Neural Inf. Process. Syst.*, 2013, pp. 3111–3119.

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Representations*, 2015, pp. 1–15.

[32] Y. Kim, "Convolutional neural networks for sentence classification," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2014, pp. 1746–1751.

[33] L. V. D. Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2605, 2008.

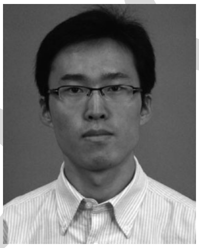
[34] Y. Yang, P. Liu, L. Ding, B. Shen, and W. Wang, "ServeNet: A deep neural network for web service classification," 2019, *arXiv:1806.05437*. [Online]. Available: https://arxiv.org/abs/1806.05437



Guobing Zou received the PhD degree in computer science from Tongji University, Shanghai, China, 2012. He is currently an associate professor and dean of the Department of Computer Science and Technology, Shanghai University, China. He has worked as a visiting scholar with the Department of Computer Science and Engineering, Washington University in St. Louis from 2009 to 2011, USA. His current research interests mainly focus on services computing, data mining, intelligent algorithms and recommender systems. He has published around 70 papers on premier international journals and conferences, including the *IEEE Transactions on Services Computing*, IEEE International Conference on Web Services, International Conference on Service-Oriented Computing, IEEE International Conference on Services Computing, etc.

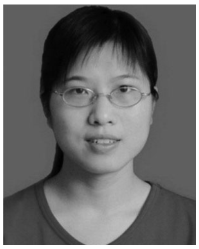
Zhen Qin received the bachelor's degree in computer science and technology from Shanghai University, 2018. He is currently working toward the master's degree with the School of Computer Engineering and Science, Shanghai University, China. His research interests include web service clustering, deep learning, and intelligent algorithms. He has published a paper on the 26th IEEE International Conference on Web Services (ICWS).

Qiang He received the 1st PhD degree from the Swinburne University of Technology (SUT), Australia, in 2009, and the 2nd PhD degree in computer science and engineering from the Huazhong University of Science and Technology (HUST), China, in 2010. He is currently working as a senior lecturer with the Department of Computer Science and Software Engineering, Swinburne University of Technology, Australia. His research interests include software engineering, cloud computing, services computing, big data analytics, and green computing. He received the Best Paper Awards from ICWS 2017, ICSOC 2018, and SCC 2018. For more details please visit: https://sites.google.com/site/heqiang/.



Pengwei Wang received the BS and MS degrees in computer science from the Shandong University of Science and Technology, Qingdao, China, in 2005 and 2008, respectively, and the PhD degree in computer science from Tongji University, Shanghai, China, in 2013. He is currently an associate professor with the School of Computer Science and Technology, Donghua University, Shanghai, China. He experienced a postdoctoral research fellow with the Department of Computer Science, University of Pisa, Italy. His research interests include services computing, cloud computing, and Petri nets. He has published more than 30 papers on premier international journals and conferences.

Bofeng Zhang received the PhD degree from the Northwestern Polytechnic University (NPU), in 1997, China. He is currently a full professor with the School of Computer Engineering and Science, Shanghai University. He experienced a postdoctoral research with Zhejiang University from 1997 to 1999, China. He worked as a visiting professor with the University of Aizu from 2006 to 2007, Japan. His research interests include personalized service recommendation, intelligent human-computer interaction, and data mining. He has published more than 150 papers on international journals and conferences.



Yanglan Gan received the PhD degree in computer science from Tongji University, Shanghai, China, 2012. She is currently an associate professor with the school of Computer Science and Technology, Donghua University, Shanghai, China. Her research interests include bioinformatics, web services, and data mining. She has published more than 30 papers on premier international journals and conferences, including *Bioinformatics*, *BMC Bioinformatics*, the *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, IEEE International Conference on Web Services, IEEE International Conference on Service-Oriented Computing, Neurocomputing, and Knowledge-Based Systems.

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