Hindawi
Mobile Information Systems
Volume 2022, Article ID 5106001, 12 pages
https://doi.org/10.1155/2022/5106001

Research Article

Web Service Recommendation Based on Word Embedding and Node Embedding

Bo Jiang, Hang Li, Junchen Yang, Yanbin Qin, Liuhai Wang, and Weifeng Pan

School of Computer Science and Information Engineering, Zhejiang Gongshang University, Hangzhou 310018, China

Correspondence should be addressed to Bo Jiang; nancybjiang@zjgsu.edu.cn

Received 21 October 2021; Revised 4 May 2022; Accepted 25 May 2022; Published 13 July 2022

Academic Editor: Hammad Afzal

Thanks to the rapid development of service-oriented computing (SOC) technologies, the number of Web services, such as Web API, is increasing rapidly. However, this brings some difficulties for mashup (a kind of Web API composition) developers to choose appropriate Web Services to build their projects. Finding required Web APIs from a large number of candidates and recommending them to developers has become a vital issue in mashup development. The traditional collaborative filtering algorithm has the problems of cold start and sparse data. In order to solve the deficiency of the collaborative filtering algorithm, we propose an improved hybrid method that combines the two kinds of information to generate word embedding and node embedding, avoiding the cold start problem and data sparsity problem. Experiments on real-world data sets show that our proposed approach is better than five state-of-the-art approaches, which verifies the effectiveness of our approach.

1. Introduction

Web APIs are web applications that can provide some services. In recent years, with the rise of Web APIs, more and more developers tend to use Web APIs to develop new web applications [1]. The use of Web APIs enhances software reusability, reduces the development cost, and improves the software quality and development efficiency based on shortening the development cycle [2]. Therefore, the number of developers using Web APIs is increasing rapidly [3]. Correspondingly, a large number of Web APIs have emerged on the Internet in recent years. At present, the number of APIs recorded on ProgrammableWeb exceeds 18,000 [4] and continues to be growing rapidly. How to quickly recommend the one that meets user needs among a large number of Web APIs has become a difficult problem when developing Web APIs composition [5]. With the increasing complexity of software functions, the requirements for Web APIs are becoming higher and higher. The Web APIs used by each module need to be carefully screened.

At present, mashup (a kind of Web API composition) developers have limited tools to use. ProgrammableWeb provides a search service based on keyword matching [6]. Due to the limited number of keywords and little information available, the recommendation list provided by this service is always incomplete, and some high-quality APIs are often not recommended [7]. Some approaches based on collaborative filtering were proposed [6–10]. These approaches simply rely on the historical invocation information of mashups. Because the newly developed mashup does not have historical invocation information, these approaches have the problem of a cold start. What's more, a mashup only invokes a few APIs [11]. This is a problem for those collaborative filtering approaches because the effectiveness of collaborative filtering is built on finding similarities among a lot of users. When users are few, the effectiveness of collaborative filtering is poor. That is called the problem of sparse data.

Sparse data and cold start are two major problems in the field of service recommendation. We propose an approach called WMD-NV to try to solve these two problems. To address the cold start problem, some approaches extract the functional information of APIs or mashups (often by processing the description texts) and recommend APIs by comparing the similarity of functions. Due to the complexity of mashups and API functions, it is difficult to ensure the accuracy of the extracted information, which makes it difficult for these approaches to improve the accuracy. Besides, to address the problem of sparse data, some approaches based on matrix decomposition are proposed [12, 13]. These approaches mainly deal with the historical invocation information between mashups and APIs [14]. However, matrix decomposition is not designed for those sparse matrices. It is difficult to decompose a matrix with several nonzero numbers and hundreds of zeros [15]. The decomposed feature vectors are likely to be filled with zeros and thus have little value.

A hybrid algorithm based on multiple interactions is proposed in [16]. They use the interaction information of different domains to solve the problems of cold start and sparse data. At the same time, combined with the idea of collaborative filtering, they can efficiently recommend high-quality Web APIs to users in need. Inspired by their ideas, we get word embedding and node embedding of mashups and Web APIs by processing the text description information and interaction information of mashups and Web APIs. We combine these two different domains of embeddings and use the idea of collaborative filtering to filter similar mashups. We call these similar mashups neighbor mashups of target mashups. For the target mashup, we will recommend those Web APIs that are used more by neighbor mashups.

Our approach not only inherits the performance and advantages of the collaborative filtering algorithm but also solves the problems of data sparsity and the cold start of the collaborative filtering algorithm. First, we obtain the neighbor mashups of the target mashup according to word embedding and then obtain node embedding according to the interaction information between the mashup and the Web API. For node embedding, each embedding contains the interaction information between a mashup with all Web APIs. According to the neighbor mashups, we select the appropriate node embeddings and combine them by weight to get a composite node embedding. In this way, we obtain an alternative list of APIs, which is implicitly contained in the composite node embedding. Because we introduce weight when generating composite node embedding, the APIs used by the most similar mashups will get a higher score. In addition, the most popular APIs (which usually means that the API is of higher quality) will also receive higher scores. By using neighbor mashups, we avoid the problem of data sparsity. Since each mashup can easily obtain a text description (even if there is no one, it can take a few minutes to write one), our method does not have the problem of a cold start.

Our main contributions are as follows:

(i) We propose a Web Service recommendation approach, WMD-NV. Combining word embedding with node embedding, it solves the problems of sparse data and cold start well. It also makes good use of historical invocation information to recommend high-quality Web Services.

(ii) We experimented with our approach in a real-world data set containing more than 18,000 APIs and more than 7,000 mashups and the invocation history of mashups. Our replication package is publicly available online via https://github.com/lihang17/wmd-nv.

The result shows the validity of our approach compared to several existing approaches. In some cases, the performance of our approach is superior to the state of the art.

The rest of the paper is organized as follows. Section 2 presents the related work of Web Service recommendation. Section 3 defined the problem. In Section 4, we will describe the proposed approach in detail and the experiments we have done will be discussed in Section 5. Section 6 concludes this paper.

2. Related Work

In this section, we will discuss some existing Web Service recommendation approaches. So far, a lot of work has been proposed. Many approaches are based on the text information of mashups or APIs and recommend APIs by calculating the similarity of texts [17, 18]. These approaches are generally called content-based approaches. In addition, some approaches recommend APIs based on functions. These approaches hope to obtain the functional information of mashups and APIs in various ways, such as analyzing the description information [19] and then recommending the corresponding APIs to mashups with similar functions [20]. Other approaches combine several algorithms, which are called hybrid approaches. Most of these approaches use the idea of collaborative filtering. Collaborative filtering mainly depends on the user's historical invocation information when recommending, in which a variety of historical data may be used to judge the similarity between users or items. In addition, technologies in many fields are used to enhance the accuracy of recommendation results, such as NLP, deep learning, and knowledge graph.

Content-based approaches mainly use text information of mashups or APIs to recommend APIs. The text information includes the description texts and keywords of mashups and APIs. The authors of [21, 22] proposed a keyword match approach, which calculates the similarity between APIs according to keywords for the recommendation. Due to less information that can be used and no semantic information in keywords, these approaches cannot meet the needs of users well. The authors of [23–25] add semantic information to their approach. They use explicit semantics to improve the accuracy of their approaches. These approaches use manual work to enrich API's description to improve search performance, which overcomes the limitations of the former approach to some extent. However, adding semantics manually brings some defects. For example, it is difficult to expand the scale of the data set, and the effect of the model is largely affected by human beings, and the effect is unstable. Therefore, some approaches based on the topic model have been proposed. For each piece of text information, after extracting the topic, the similarity between the texts is calculated. To a certain extent, these approaches work well. They can extract the similarity between Web APIs relatively accurately. However, the recommendation system should also consider the quality of

Web APIs. It is difficult to compare the quality of Web APIs with the topic model alone. In addition, the topic model requires a lot of data training, so it is not suitable for short texts such as text descriptions. Additionally, some interesting approaches are proposed. Risaralda et al. [26] proposed an approach based on Variational Autoencoders for syntactic Web Service discovery. It outperforms those based on traditional dimensionality reduction techniques. In [27, 28], Web Service is treated as an active user. In this way, they incorporated social networking into Web Service discovery and got interesting results.

Some early function-based approaches used keyword search to match the API profile. This approach is difficult to find the real same type of APIs. Therefore, many works want to add some semantic information manually to improve the effectiveness of the model, and many semantics-based recommendation algorithms have been proposed. The articles [29–31] leveraged domain ontology or dictionaries to enrich the semantics of descriptions of both APIs and user requests and adopted logic-based reasoning for semantic similarity calculation. However, as mentioned earlier, adding manual work will lead to the fact that the model cannot be applied to a large data set, and the effect is also affected by human beings, resulting in instability of the approach. In addition, some work uses machine learning and data mining technology [32] combines keywords and semantic approaches to form the preferences of each user and then recommends APIs according to these preferences. The article [33] targets intentional requests and domain service goals are extracted from the description information. The article [34] proposed an approach based on function and coinvocation. Because these approaches need to extract functional features from texts or other information, some problems arise, such as the low accuracy of extracted functions. These theories are difficult to tell whether the information extracted from the materials is complete [35], resulting in unstable model effectiveness.

In addition, some works have proposed hybrid approaches using multiple algorithms [36]. In [6], description texts and interaction information are used. They calculate the similarity between texts and between interaction information to obtain a candidate APIs list through the idea of collaborative filtering. Like other collaborative filtering approaches, it also has the problem of a cold start. Chen et al. proposed a hybrid approach based on the topic model and knowledge graph in [14]. This approach uses the topic model to calculate the similarity between mashups and then combines it with a knowledge graph to recommend APIs. As mentioned earlier, LDA [37], as a topic model, has considerable randomness [38] and does not perform well when dealing with short texts. Moreover, this approach only calculates the weighted sum of scores of the two models as the result. It is difficult to say whether this kind of combination method improves the effect of the whole approach [39]. The article [16] uses three types of interactions between mashups and APIs. First, it uses a deep neural network to learn the features in the description texts. And then, it also uses node2vec [40] to deal with the invocation matrix between mashups and APIs. Finally, the method of combining the two models is to use MLP, a deep learning model. Deep learning can automatically extract complex features from data. Our approach also uses MLP to get the final score. In fact, utilizing the information enclosed in the topological structure has been widely applied in the field of software engineering [2, 20, 41, 42].

3. Problem Statement

Before presenting our approach, we will elaborate on the definition of the problem in this paper.

When a developer wants to develop a new mashup, he can first describe the functions of the mashup accurately in text. So, based on the text, we can recommend some APIs commonly used in other mashups of the same type to the developer. The specific process is, according to this description text, we can find some similar mashups, which probably use the same APIs. And then, we can recommend some popular APIs that these mashups used.

More formally, we have a mashup collection A and an API collection B. For the interaction relationship between A and B, we use matrix Y to represent, where $|Y| = |A| \times |B|$. The interaction matrix Y stores the interaction information between mashups and APIs. For a mashup m and an API s, $Y_{m,s} = 1$ means mashup m invokes API s, or $Y_{m,s} = 0$ means mashup m does not invoke API s. In addition, we have all the description information of mashups composed of structured texts, and they are combined into a set. When giving a target mashup m, our goal is to accurately find k possible APIs which target mashup m may invoke for developers.

In particular, our model is mainly for those mashups under development. For each mashup under development, the model will give a list of APIs that the mashup may use next.

4. Proposed Approach: WMD-NV

In this section, we will first introduce the whole framework and then describe several components in detail. Finally, the approach will be presented completely.

4.1. Framework. The framework of our approach is shown in Figure 1. In WMD-NV, there are mainly two modules, one is WMD (Word Mover's Distance) [43, 44] module and the other is the node2vec module.

We use the WMD module to process the description texts, and the interaction matrix is used as the input of node2vec. WMD module learns the potential relationship between mashups in description texts. The output of the WMD module is the similarity between mashups. We will keep this similarity for the next step. After that, node2vec will deal with the information in the interaction matrix, which contains the features of mashups and APIs, as well as the potential relationship between mashups and APIs. Node2vec can learn these features and embed them into low-dimensional space. As a result, we will get the feature vectors of these mashups and APIs. According to the similarity between mashups, we will select 50 feature vectors of neighbor mashups that are similar to the target mashup m

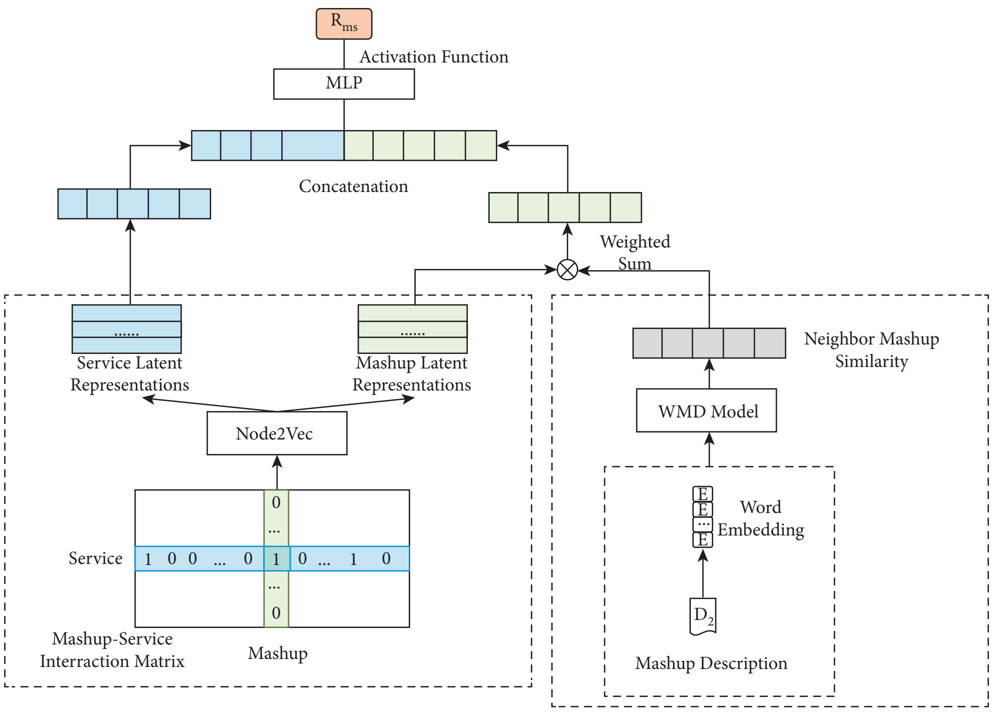
R_ms
Activation Function
MLP
Concatenation
Weighted Sum
Neighbor Mashup Similarity
WMD Model
Word Embedding
Service Latent Representations
Mashup Latent Representations
Node2Vec
Service
Mashup-Service Interraction Matrix
Mashup
D2
Mashup Description

Figure 1: The framework of the proposed Web Service recommendation approach.

from all feature vectors. The weighted sum of the 50 neighbor mashups' feature vectors with the target mashup's feature vector, 51 features vectors in total, is calculated as the weighted representation of the target mashup m. The weights are calculated according to their similarity. And we will normalize the result to ensure that the result will not get bigger. The weight of the target mashup m is higher than that of other mashups, but its weight advantage is not prominent. Relatively speaking, the weight of 50 mashups is relatively average. Through this operation, the feature vectors of the 51 mashups can be superimposed on one vector, so that their relationships with different APIs are added to the final vector representation. This operation can be regarded as that we have obtained a series of candidate APIs, but these APIs are saved in the form of vectors. It is worth noting that these APIs have been invoked by mashups similar to the target mashup, and the target mashup will use these APIs with a high probability. In addition, we also obtained the feature vectors of all APIs through node2vec. Finally, we send the generated mashup representation and API feature vector into a 6-layer MLP to get the final score. In this way, we naturally integrate the text information of mashups and the invocation information in the interaction matrix, so that the approach can make more accurate judgments with more information.

Therefore, the final approach will have three components, which are composed of the WMD module, node2vec module, and MLP module. After the WMD module extracts the similarity information from the texts, node2vec extracts the invocation information. The results of the two modules are combined in the way of the weighted sum. Finally, MLP will process the weighted sum and give the score of each API.

4.2. Word Embedding. In [15], Mikolov et al. proposed word2vec. This new word embedding program can learn the features of words in texts and generate its vector representation through a neural network structure called skip-gram. This approach with a simple structure has amazing efficiency. It can learn a large number of corpora and obtain their word vectors in a relatively short time. The learning of word vectors is unsupervised, so you can choose to train the word vector yourself or use the word vector pretrained.

Suppose we have a word embedding matrix $X \in R^{d\times n}$, which stores n d-dimensional word vectors. For the i^{th} column, there is $x_i \in R^d$ means i^{th} word in d-dimensional space. We assume documents are represented as normalized bag-of-words (nBOW) vectors, $d_i \in R^n$. If word i appears c_i times in the paper, we denote

$$d_i = \frac{c_i}{\sum_{j=1}^{n} c_j}, \tag{1}$$

where d_i is an nBOW vector naturally very sparse as most words will not appear in any given document.

WMD is based on these pretrained word embedding sets, not only word2vec but also other word embedding approaches. For a given word embedding, we can perform many operations mathematically. For example, vec($Japan$) − vec($sushi$) + vec($Germany$) ≈ vec ($bratwurst$). Similarly, we can calculate the similarity between each word vector. This similarity is the Euclidean distance calculated directly from word2vec embedding space, and its formula is

$$c(i,tj) = \left\| x_i - x_j \right\|_2, \tag{2}$$

where $x_i \in R^d$ is the i^{th} column of D.

WMD compares the word embeddings in two documents and calculates the similarity between them. First, WMD allows some or all words in document $\mathbf{d}$ to be transferred to any word in another document $\mathbf{d}'$. We take $T \in R^{n\times n}$ to represent this process of transfer, where $T_{ij} \geq 0$ denotes how much of word i in $\mathbf{d}$ travels to word j in $\mathbf{d}'$. To transform $\mathbf{d}$ entirely into $\mathbf{d}'$ we ensure that the entire outgoing flow from word i equals d_i, i.e., $\sum_j T_{ij} = d_i$. Further, the amount of incoming flow to word j must match d'_j, i.e., $\sum_i T_{ij} = d'_j$. Finally, we can define the distance between the two documents as the minimum (weighted) cumulative cost required to move all words from d to d' as follows:

$$\min \sum_{i,j=1}^{n} T_{ij} c(i, j). \tag{3}$$

Then, we can get the distance (similarity) between $\mathbf{d}$ and $\mathbf{d}'$. Figure 2 visualizes the process of the comparison of two sentences D_1 and D_2 to the query sentence D_0.

For any given mashup description text, we send it into the trained WMD model, which will calculate the similarity $Sim_{m,n}$ between the target mashup m and another mashup n. We will select the k mashups with the closest distance. These mashups and the similarity between them will be used in the next module.

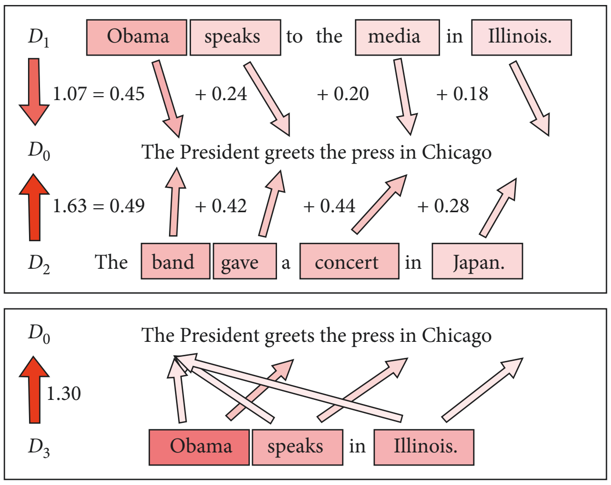

Figure 2: The components of the WMD metric.

Nowadays, researchers begin to apply random walks to the fields of information retrieval and image segmentation. And they have achieved some results. For example, Brin and Page created PageRank technology based on a random walk and founded Google.

4.3. Node Embedding.

4.3. Node Embedding. This module is based on an algorithm, node2vec [40], which can learn the continuous feature representation of nodes in the graph. The input of this module is an interaction matrix representing the interaction relationship between mashups and APIs (which can be regarded as an undirected graph).

4.3.1. Random Walk. When a drop of ink enters the water, only part of the water is dyed at first. However, soon the whole glass of water will be stained with the color of ink. This is because the molecules in the ink are moving irregularly, causing the molecules to slowly diffuse into the surrounding water molecules. Not only the molecules in ink but also water molecules, like any other molecules, will constantly impact in a random direction to a certain extent. This phenomenon is called Brownian motion in the field of physics. Inspired by this phenomenon, a random walk algorithm is created. And it describes the ideal mathematical state of Brownian motion.

The one-dimensional random walk algorithm holds that at a certain time t, the probability of the walker walking forward is p, and the probability of walking backward is $1-p$. For every moment, there is such a probability p; then for any moment n, its position L_n can be expressed as

$$L_n = x + X_1 + \cdots + X_n, \tag{4}$$

where x, X_1, ..., X_n are independent random variables, which meet

$$P(X_\mathrm{i} = 1) = p = 1 - P(X_i = -1). \tag{5}$$

Now, the random walk algorithm has been successfully applied to physics, economy, chemistry, and other fields.

4.3.2. Node2vec. Node2vec adopts the strategy of a random walk algorithm to generate multiple neighbor node subsets for each node in the graph and then combines the idea of the word2vec algorithm to transform these subsets into feature vectors. The subset is regarded as a document, and the node in it is regarded as a word. As a result, we get feature vectors of all nodes. Compared with the DNN approach, there is no need to mark the data manually, so node2vec has the characteristics of high efficiency and easy training, and the effect is quite good. The probability formula of the random sampling of node2vec is

$$P(c_i = x | c_{i-1} = v) = \begin{cases} \dfrac{\pi_{vx}}{Z}, & \text{if } (v, x) \in E, \\ 0, & \text{otherwise}, \end{cases} \tag{6}$$

where t is the previous sampling point, v is the current node, and x is the next sampling point.

For each node in the graph, first, node2vec will find r random walks for each node in the graph. For each walk starting from the starting node u, if the node t has just been sampled, in other words, passes through the edge (t, v), it now stays on node v. The random walk algorithm is to solve which position of the next node x will be. Node2vec algorithm defines a unique probability distribution to define the transfer probability of node transfer to different neighbor nodes:

$$\alpha_{pq}(t, x) = \begin{cases} \dfrac{1}{p}, & \text{if } d_{tx} = 0, \\ 1, & \text{if } d_{tx} = 1, \\ \dfrac{1}{q}, & \text{if } d_{tx} = 2. \end{cases} \tag{7}$$

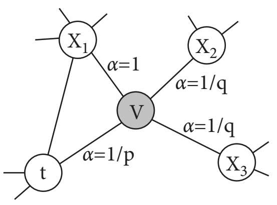

FIGURE 3: The strategy of sampling in node2vec.

For this probability distribution, if x is equal to t, that is, the next point to go is the previous node, then the sampling probability of this node is $1/p$. If t is connected to x, the probability of sampling x is 1. If t is not connected to x, the sampling probability of x is $1/q$. The two parameters p and q control how fast the walk explores and leaves the neighborhood of starting node u. Figure 3 shows this probability distribution.

Return parameter p: if $p > \max(q,1)$, the sampling will not return as much as possible, that is, the probability of sampling nodes that have been sampled is low, while the probability of sampling nodes that have not been sampled increases. If $p > \min(p,1)$, the sampling will tend to return to the previous node, so the walk will turn several nodes around the initial node u.

Access parameter q: if $q > 1$, the sampling will tend to run between nodes around the starting point, reflecting the BFS (Breath First Search) characteristics of the initial node. If $q < 1$, the swimming will tend to run away from the initial node, reflecting the DFS (Deep First Search) characteristics of the initial node u.

When $q = 1$ and $p = 1$, the walk mode is equivalent to the random walk in DeepWalk [45].

Generally, node2vec will generate r random walks for each mashup, all of which will be regarded as documents, and the nodes in each set of walks will be regarded as words. And then according to the idea of word2vec, SGD is used for training to obtain the feature vector V_i of the i^{th} mashup. At the same time, the model will generate the feature vector V_s of input Web API s.

In node2vec, we learn the mapping from nodes to low dimensional feature space, which maximizes the possibility of preserving the interaction information. As a result, we get the feature vector of each mashup and each API.

4.4. Aggregator. The function of the integration module is to give the final score according to the similarity obtained by the WMD module and the feature vectors obtained by the node2vec module. Our approach is to select the k neighbor mashups with the highest similarity and calculate the weighted sum of their feature vectors. The weight of each mashup is according to the similarity. In this way, we get a composite feature vector, which contains multiple interaction features of multiple mashups. The composite feature vector V' is as follows:

$$V_m' = \frac{\sum_{n=1}^{|A|} V_n Sim_{m,n}}{\sum_{n=1}^{|A|} Sim_{m,n}}. \quad (8)$$

Then, we send this feature vector V_m' into the MLP for training to get the final score $R_{m,s}$:

$$R_{m,s} = \text{MLP}\left(V_s \oplus V_m'\right), \quad (9)$$

where V_s is a feature vector of the input Web API. The vector is generated by node2vec.

We will select k Web APIs with the highest scores and recommend them to developers.

4.5. Offline Model Learning. Since the goal of our approach is to predict the score of a Web API over a mashup, each training sample as an input to WMD-NV consists of a mashup and a Web API. A positive sample (labeled as 1) is composed of a mashup and its component service, while a negative sample (labeled as 0) is a pair of a mashup and an irrelevant service without actual invocations.

The predicted score of WMD-NV should be approximately 1 for positive samples and 0 for negative samples. The likelihood function is defined as

$$P\left(Y^+, Y^- | \Theta\right) = \prod_{(m,s)\in Y^+} R_{m,s} \prod_{(m,s)\in Y^-} \left(1 - R_{m,s}\right), \quad (10)$$

where $R_{m,s}$ is the predicted rating of Web API s over mashup m. Θ is the parameter set Y^+ represents a set of positive samples, and Y^- denotes a set of negative ones.

Maximizing the likelihood probability of (17) is equivalent to minimizing the loss function described as follows:

$$J = -\sum_{(m,s)\in Y^+ \cup Y^-} \left(r_{m,s} \log R_{m,s} + \left(1 - r_{m,s}\right) \log\left(1 - \log R_{m,s}\right)\right), \quad (11)$$

where $r_{m,s}$ denotes the label (0 or 1) of a sample that consists of m and s.

Then, we use the well-known optimization algorithm Adaptive moment estimation (Adam) to minimize the loss function of our model. Adam is an extension of the stochastic gradient descent (SGD) and has been widely used in deep learning applications.

5. Experiments

In this section, we performed a series of experiments on a real data set. This data set was crawled down from ProgrammableWeb. We not only tested the effect of our approach on this data set but also compared our proposed approach with the most advanced approaches. The program of our approach is developed in Python and runs on a PC equipped with Intel Core CPU i5-2400 @3.1 GHz, 4GB DDR3 1333 MHz RAM, and Windows 10 OS.

5.1. Data Set. Since its establishment, ProgrammableWeb has been committed to recording the development of the API economy. At the same time, in the process of development, it has accumulated the most comprehensive API directory and information in the whole network. ProgrammableWeb is a leading source of news and information about API. The API directory it provides has always been the most important and extensive source of information when API-related statistics are included in various news, conferences, white papers, and various studies. All our data comes from ProgrammableWeb. The whole data set has 18,536 APIs and 7,876 mashups. After the certain screening, such as removing APIs without invocation information and mashups without text description, our data set finally left 7754 mashups and 1602 APIs, all of which have a description text. Another important data is the interaction matrix between mashups and APIs.

For the test set, we select all mashups that invoke three APIs as the test set. The test set contains 767 mashups and 179 APIs. In order to evaluate the effectiveness of the approach, we randomly eliminated one of the three APIs, and the remaining two APIs were used as the basis for the recommendation. Finally, if the deleted API appears in the top-N of the recommendation list, the recommendation is considered valid.

5.2. Evaluation Metrics. In the experiment, we used the widely used Precision@*N*, Recall@*N*, and F1@*N* to evaluate the effect of our approach and baselines. We will introduce these indicators, respectively, as follows.

(i) Precision@*N*: precision indicates the proportion of items related to the user in all items recommended to the user. Precision@*N* is a very popular indicator in the field of recommendation, indicating the precision of the top-*N* items recommended. It not only reflects the accuracy requirements of the recommendation system but also reflects the importance of rank. The calculation approach is as follows:

$$\text{Precision@}N = \frac{|\{\text{Real APIs}\} \cap \{\text{Recommend APIs}\}|}{N}. \tag{12}$$

(ii) Recall@*N*: it indicates the proportion of marked items listed in the top-N recommendation list. Ditto, Recall@*N* also takes into account the importance of rank. The calculation formula is as follows:

$$\text{Recall@}N = \frac{|\{\text{Real APIs}\} \cap \{\text{Recommend APIs}\}|}{\{\text{Real APIs}\}}. \tag{13}$$

(iii) F1@*N*: its calculation is based on accuracy and recall. For both indicators, F1 can be regarded as a weighted average, which is often regarded as an important indicator of the recommendation system. The calculation formula is as follows:

$$F1\text{@}N = \frac{2\text{Precision@}N \times \text{Recall@}N}{\text{Precision@}N + \text{Recall@}N}. \tag{14}$$

(iv) MAP@*N*: Mean Average Precision (MAP) is computed by considering the performance of precision at all positions of the top-N item list. The calculation formula is as follows:

$$\text{MAP@}N = \frac{1}{|M|} \sum_{\text{m} \in M} \frac{1}{N_m} \sum_{i=1}^{N} \left(\frac{N_i}{i} \times I(i) \right), \tag{15}$$

where $I(i)$ indicates whether service at the position i in the ranking list is an actual component service of m, N_m is the number of component services of m, and N_i denotes the number of actual component services of m that occurred in the top i services of the ranking list.

(v) NDCG@*N*: normalized discounted cumulative gain (NDCG) is a standard measure of ranking quality, considering the graded relevance among positive and negative items within the top-*N* of the ranking list. The calculation formula is as follows:

$$\text{NDCG@}N = \frac{1}{|M|} \sum_{\text{m} \in M} \frac{1}{S_m} \sum_{i=1}^{N} \frac{2^{I(i)} - 1}{\log_2 (1 + i)}, \tag{16}$$

where S_m represents the ideal maximum DCG score that can be achieved for m.

5.3. Baselines. We compare our approach with some of the best existing approaches. Because our approach uses description texts and invocation information, we choose the same approaches based on this information, such as function-based recommendation and content-based recommendation. Several algorithms use the idea of collaborative filtering.

(i) Function-based recommendation (FBR): this approach expects to find information in the description texts and extract the functions of APIs. When recommending, it always recommends APIs with similar functions to users. In this approach, some tools are used to extract verbs and objects in the description texts, all of which are grouped into a functional set. Finally, through these sets, the API similarity is calculated, and the API is recommended on this basis.

(ii) API-based recommendation through LDA (ABR-LDA): this approach uses the LDA algorithm to model the description information of API. LDA algorithm will extract the topic probability distribution in the description information and calculate the similarity between APIs based on this topic information. Finally, the API set with the highest similarity with the target API is recommended to users.

(iii) Mashup-based recommendation through function (MBRF): this approach uses the same tool as approach 1 to extract the function information in the description texts of mashups, calculate the similarity between mashups based on this, and finally recommend the API used by similar mashups for the target mashup.

(iv) Mashup-based recommendation through VSM (MBR-VSM): this approach uses the VSM (Vector

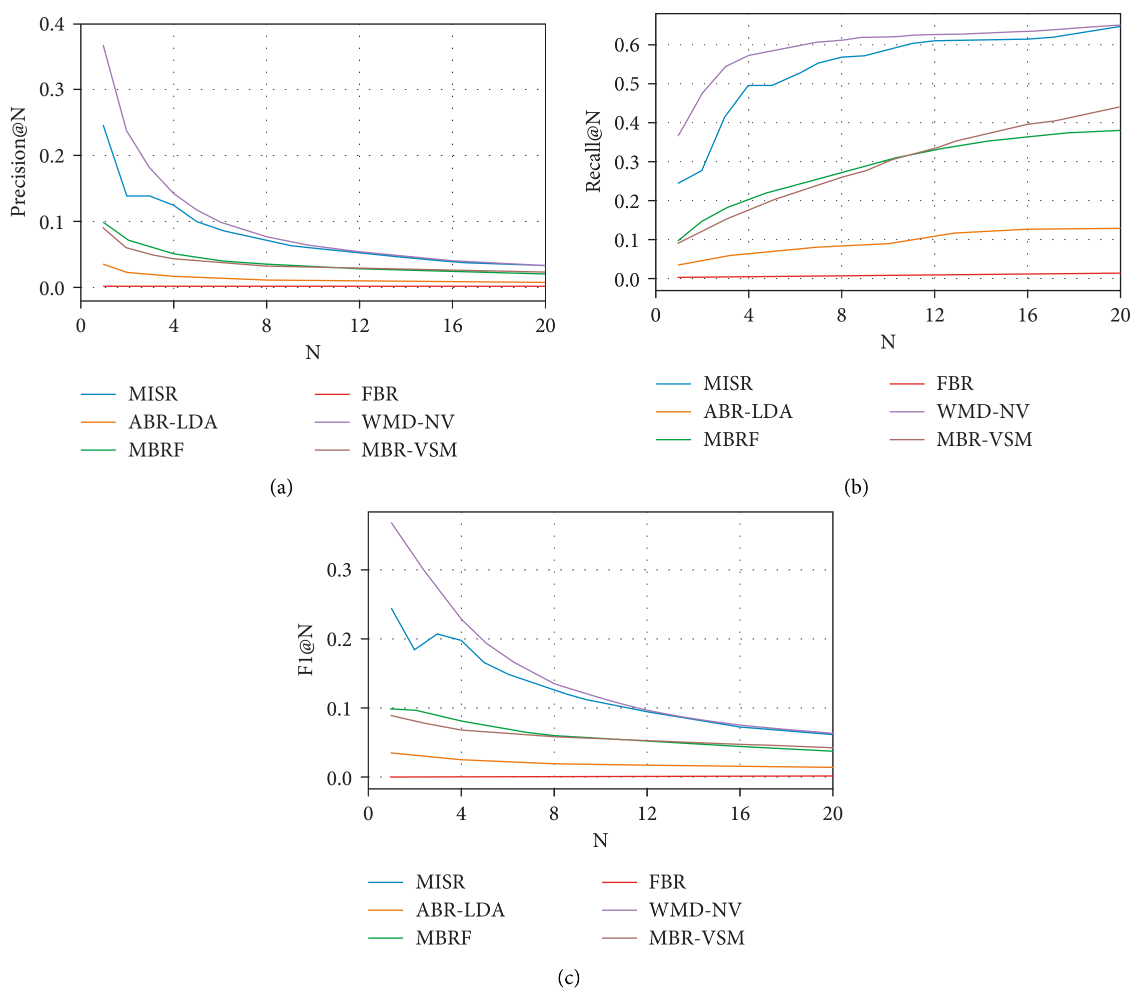

Figure 4: Comparison of top-N recommendations with different approaches on the data set.

Space Model) algorithm to reduce the dimension of mashup semantic description information to multidimensional vector. Using these vectors, the similarity between mashups is obtained. Finally, the APIs used by the k neighbor mashups with the highest similarity will be recommended to the target mashup.

(v) Multiplex interaction-oriented service recommendation (MISR): this approach merges three types of interactions between Web Services and mashups into a deep neural network. It uses powerful representation learning abilities provided by deep learning to extract hidden structures and features from various types of interactions between mashups and services [16].

5.4. Parameter Settings. For the word vector used in the WMD module, because we use the Google news word vector pretrained word embedding set published by Google, its word vector dimension is 300. In node2vec, the dimension of the node vector is set to 25, the number of walks per node is 16, the length of each walk is 80, parameter p is 1, and q is also 1, and most of the parameter settings follow [16]. After generating the node vector, we use a 6-layer MLP network to extract features from the mashup representation vector and API representation vector; by this way, we can combine two kinds of feature vectors. The numbers of units in six layers were set to 200, 100, 50, 25, 100, and 50, respectively. The activation function we use is ReLU. During training, the learning rate is 0.0003. These parameters are also totally used by reference papers, and only some optimization of the parameter is carried out when necessary.

5.5. Performance Comparison. Figure 4 shows the performance of all baselines and our approaches to the three indicators. In all cases, our WMD-NV is superior to the several state-of-the-art approaches in the three indicators. From top 1 to top 3, our approach performs particularly well, which proves that our approach always puts the APIs required by users in front. This is very important for a recommendation algorithm.

It can be seen from Figure 4 that the performance of the FBR algorithm is very poor. Part of the reason is that the objects and verbs extracted from short texts such as

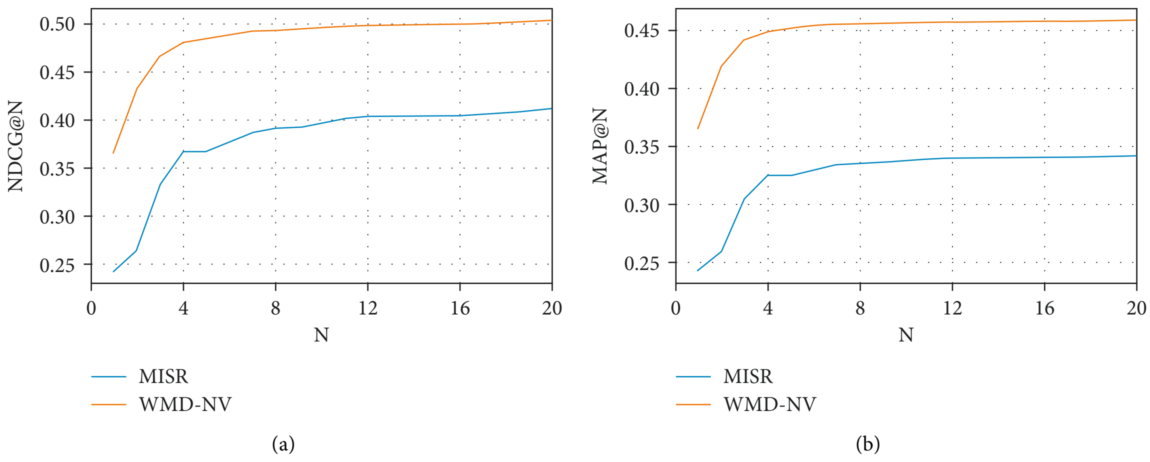

Figure 5: Comparison of our approach with MISR in NDCG@N and MAP@N.

description information are difficult to describe the functions of the API clearly, and the vector representations of its functions are likely to be inaccurate. For similarity is extracted from inaccurate function information, the result is very unreliable naturally. ABR-LDA algorithm uses the LDA algorithm based on the bag-of-words model, which only focuses on the similarity between texts. Therefore, although the semantic information is lost, it performs better than FBR because its judgment on the topics is relatively accurate. However, they are all algorithms looking for similar APIs. Due to the particularity of mashups, two APIs with almost the same functions are often not required, so the performance of this approach is also limited.

Relatively speaking, the approach based on collaborative filtering performs better. Both MBRF and MBR-VSM are recommended based on the similarity of mashups because the similarity between APIs is not a good recommendation basis. Similar APIs are often competitive. For example, when developing a map-related mashup, if you have used the map API provided by Google, you basically do not need to use the map API provided by Baidu. In addition, there are a large number of APIs, but the APIs often used the account for only about 10% of the total. Therefore, similar APIs in a sense do not necessarily mean that they are equally easy to use. It is not difficult to imagine that these recommended algorithms based on API similarity are easy to find an API with similar functions but poor quality according to a popular high-quality API, which is often not what developers want. Therefore, the effects of these two approaches are relatively good. However, like all collaborative filtering approaches, both approaches have the problem of a cold start.

After that, MISR performs best among these approaches because it uses multiple interaction information, so the information of the approach is more sufficient and comprehensive when recommended. MISR not only uses the description information of API but also uses the description information of mashup and then selects the neighbor mashups based on the similarity of mashup's description texts. At the same time, the interaction matrix of mashups and APIs is also used. What's more, the interaction information in another sense, the overall popularity of the APIs, is extracted based on the interaction matrix. This information makes its recommendation quality higher, and its approach of extracting text information retains semantic information, which is more advanced.

However, too much information is easy to bring more interference to the approach. In WMD-NV, we did not use many types of interaction information but achieved better results. The NDCG@2, MAP@2, Precision@2, Recall@2, and F1@2 values of the WMD-NV were higher than those of the MISR; these indexes improved by 16.97%, 16.00%, 9.86%, 19.71%, and 13.15% respectively. It is worth noting that although there is no obvious advance between our approach and MISR on Precision@*N* and Recall@*N* after the top 10, the effect of our approach is much higher than MISR on NDCG@*N* and MAP@*N* as both indicators consider ranking. Figure 5 shows the comparison of our approach with MISR in NDCG@*N* and MAP@*N*. This means that our approach always puts users' required APIs at the forefront. As mentioned earlier, this is very important for recommendation approaches, because users are likely to give up using the recommendation after trying the first few APIs.

We also conducted experiments on the number of neighbors of the mashup; we can see the results in Figure 6. The results show that the number of neighbors has little effect on the performance of WMD-NV. It shows that our approach has certain stability on the change of parameters, and experiments in other parameters also show this conclusion. In total, the approach performs the best result when the number of neighbors is 50.

5.6. Threats to Validity. Some potential factors may threaten the validity of our work, and we discuss them in the following.

Internal threats to validity: internal threats mainly focus on whether the conclusion of the experiment itself has a causal relationship with the experimental process. In our experiment, the main internal threats to validity lie in evaluation criteria and data sets.

At present, there is no suitable evaluation data set, and there is a lack of a recognized evaluation approach to evaluate the effect of the APIs recommendation system. In this

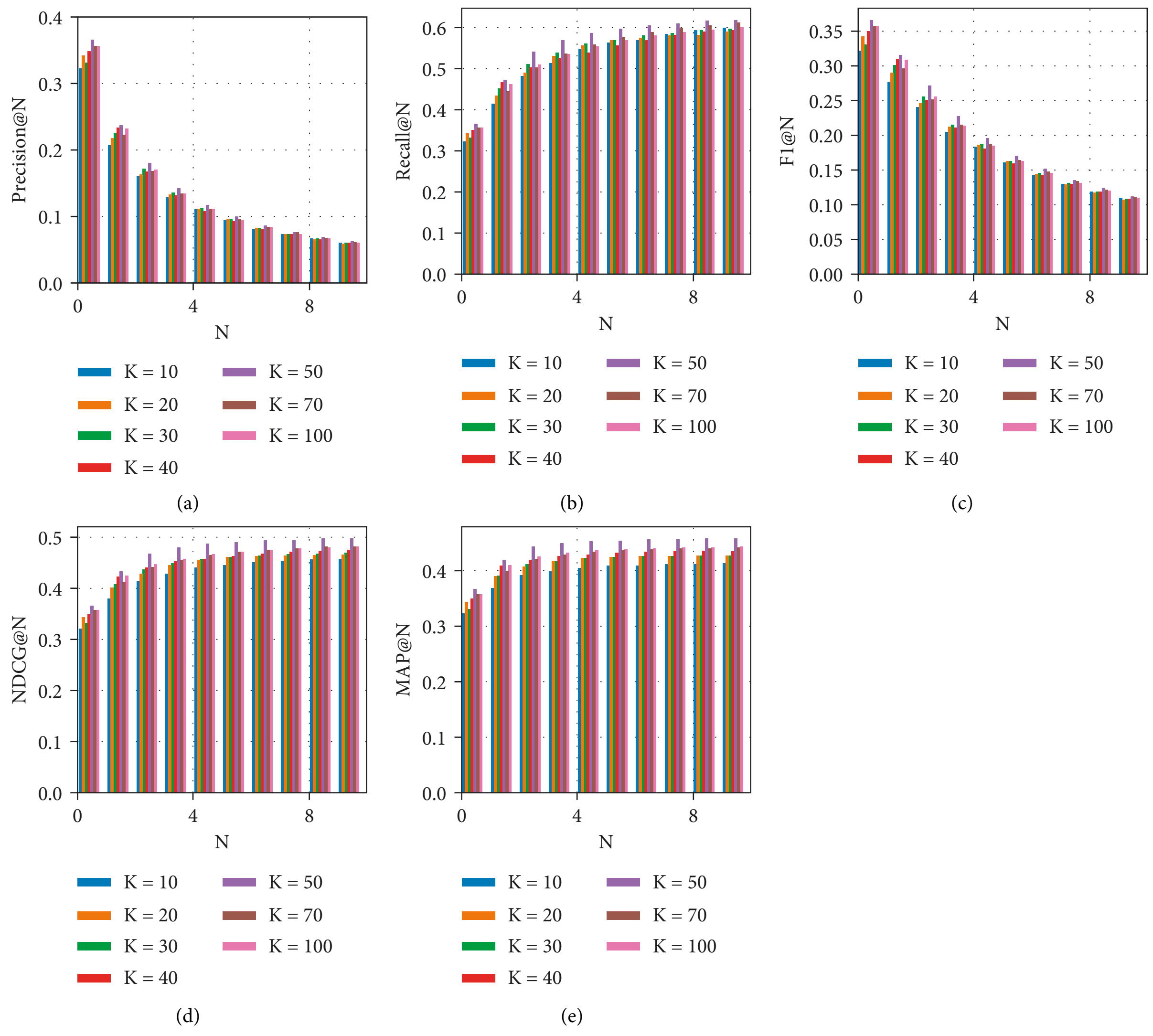

Figure 6: The performance of our approach in different numbers of neighbors.

experiment, we include the mashup that invokes three APIs into the test set and marks one API randomly as the evaluation material. If the approach recommends the marked API with the help of the remaining two APIs, the recommendation result is considered effective. This approach is not used in other papers. For example, in MISR, they include all mashups that call more than two APIs into the test set, randomly select one API, and mark the remaining APIs. Except for MISR, the source code of most of the baseline approaches is not publicly available, so we can only implement these approaches ourselves (which is also one of the internal threats) and test them with the same standards. Our test approach can well test the recommendation ability of the approaches to deal with interaction information, but it is unfavorable to those approaches optimized for cold start.

External threats to validity: external threats focus on the value of the model itself. Reflected in the recommended algorithm, it examines whether the effectiveness of the algorithm can be extended to real-world data set.

In our experiment, we use the data crawled down from the ProgrammableWeb as the real-world data set. For our approach, it is a great challenge to consider all possible real scenarios. Because the input of our approach is the description texts of the mashup, users must edit the texts themselves before using our approach. The problem is that everyone's writing style and language level are different, and the input text information will have many differences. The two texts expressing the same meaning may look very different. These different words will have a great impact on the results. Nevertheless, the description information set on the ProgrammableWeb is already the most comprehensive data set that can be found. ProgrammableWeb stores the world's largest Web API directory, as well as the most mashup information. More than 1000 companies and individuals are using the data on the ProgrammableWeb. It is the first source of API information collected by the news, white papers, and various related papers. Therefore, we have reason to believe that our data set is reliable enough.

6. Discussion

In this paper, we propose a hybrid Web Service recommendation approach called WMD-NV, which integrates mashup description text information and mashup-API interaction matrix. The goal of the approach is to recommend APIs that mashup developers may use. We model the mashup and API information, overlay the vector representations of the mashups into one feature vector, and maximize the list of APIs that may be used by the target mashup. Among the three indicators, our approach performs well and is better than several existing state of the art. Moreover, among the recommended top 3 indicators, it is far better than other approaches, which proves that our approach can not only recommend the APIs required by developers but also rank them at the forefront. These are due to the strategy of superimposing vectors, which makes the APIs used many times (usually means better) gain more weight, so they can get a higher ranking.

The future work may be improved in the following aspects. For example, API version iteration will lead to great changes in API performance indicators and usage approaches, which is not considered in our approach. In addition, for the similarity between mashups, we are only limited to the similarity of description texts. There are other indicators that can be considered, such as tag and function.

Data Availability

The data presented in this study are available on request from the corresponding author.

Conflicts of Interest

The authors declare no conflicts of interest.

Authors' Contributions

Conceptualization was done by B. J. and W. P.; methodology was developed by H. L. and W. P.; validation was performed by H. L.; original draft was written by H. L.; reviewing and editing were done by J. Y., Y. Q., and L. W.; project administration was done by B. J.; funding acquisition was done by B. J. All authors have read and agreed to the published version of the manuscript

Acknowledgments

This work was supported in part by the Natural Science Foundation of Zhejiang Province under Grant LY21F020002, in part by the Key Research and Development Program Project of Zhejiang Province under Grant 2019C01004 and Grant 2019C03123, and in part by the Commonweal Project of Science and Technology Department of Zhejiang Province under Grant LGF19F020007.

References

[1] T. Wei, Y. Fan, A. Ghoneim, M. A. Hossain, and S. Dustdar, "From the service-oriented architecture to the Web API economy," *IEEE Internet Comput*, vol. 20, pp. 64–68, 2016.

[2] W. Pan, H. Ming, C. K. Chang, Z. Yang, and D.-K. Kim, "ElementRank: ranking java software classes and packages using a multilayer complex network-based approach," *IEEE Transactions on Software Engineering*, vol. 47, no. 10, pp. 2272–2295, 2021.

[3] W. Pan, X. Xu, H. Ming, and C. K. Chang, "Clustering mashups by integrating structural and semantic similarities using fuzzy AHP," *International Journal of Web Services Research*, vol. 18, no. 4, pp. 34–57, 2021.

[4] B. Jiang, J. Yang, Y. Qin, T. Wang, M. Wang, and P. Weifeng, "A service recommendation algorithm based on knowledge graph and collaborative filtering," *IEEE Access*, pp. 50880–50892, 2021.

[5] X. Du, T. Wang, L. Wang et al., "CoreBug: improving effort-aware bug prediction in software systems using generalized k-core decomposition in class dependency networks," *Axioms*, vol. 11, no. 5, p. 205, 2022.

[6] T. Espinha, A. Zaidman, and H. G. Gross, "Web API growing pains: stories from client developers and their code," in *Proceedings of the Software Maintenance, Reengineering & Reverse Engineering (CSMR-WCRE)*, pp. 84–93, Antwerp, Belgium, February 2014.

[7] C. Li, R. Zhang, J. Huai, and H. Sun, "A novel approach for API recommendation in mashup development," in *Proceedings of the IEEE International Conference on Web Services*, vol. 1, pp. 730–734, Anchorage AK, USA, July 2014.

[8] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Transactions on Information Systems*, vol. 22, pp. 5–53, 2004.

[9] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative fifiltering," in *Proceedings of the 26th International Conference on World Wide Web 2017*, pp. 173–182, New Yark, USA, April 2017.

[10] R. Xiong, J. Wang, N. Zhang, and Y. Ma, "Deep hybrid collaborative filtering for web service recommendation," *Expert Systems with Applications*, vol. 110, pp. 191–205, 2018.

[11] W. Pan, H. Ming, D.-K. Kim, and Z. Yang, "PRIDE: prioritizing documentation effort based on a PageRank-like algorithm and simple filtering rules," *IEEE Transactions on Software Engineering*, 2022.

[12] M. M. Rahman, X. Liu, and B. Cao, "Web API recommendation for mashup development using matrix factorization on integrated content and network-based service clustering," in *Proceedings of the IEEE International Conference on Services Computing*, pp. 225–232, Helsinki, Finland, Aug 2017.

[13] L. Yao, X. Wang, Q. Z. Sheng, B. Boualem, and C. Huang, "Mashup recommendation by regularizing matrix factorization with API co-invocations," *IEEE Transactions on Services Computing*, vol. 14, no. 2, pp. 502–515, 2021.

[14] J. Chen, B. Li, J. Wang, Y. Zhao, and Y. Xiong, "Knowledge graph enhanced third-party library recommendation for mobile application development," *IEEE Access*, vol. 14, pp. 502–515, 2020.

[15] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proceedings of the ICLR*, Scottsdale Arizona, USA, January 2013.

[16] Y. Ma, X. Geng, and J. Wang, "A deep neural network with Multiplex interactions for cold-start service recommendation," *IEEE Transactions on Engineering Management*, vol. 68, pp. 105–119, 2019.

[17] W. Pan and C. Chai, "Structure-Aware mashup service clustering for cloud-based Internet of things using genetic

algorithm based clustering algorithm," *Future Generation Computer Systems*, vol. 87, pp. 267–277, 2018.

[18] A. Zhu, W. Chen, J. Zhang, X. Zong, W. Zhao, and Y. Xie, "Investor immunization to Ponzi scheme diffusion in social networks and financial risk analysis," *International Journal of Modern Physics B*, vol. 33, no. 11, Article ID 1950104, 2019.

[19] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston MA, USA, June 2015.

[20] H. Li, T. Wang, W. Pan, M. Wang, and J. Wang, "Mining Key classes in java projects by examining a very small number of classes: a complex network-based approach," *IEEE Access*, vol. 9, pp. 28076–28088, 2021.

[21] H. Qiang, Z. Rui, X. Zhang et al., "Efficient keyword search for building service-based systems based on dynamic programming," in *Proceedings of the International Conference on Service-Oriented Computing*, Malaga, Spain, October 2017.

[22] Q. He, R. Zhou, X. Zhang, Y. Wang, D. Ye, and F. Chen, "Keyword search for building service-based systems," *IEEE Transactions on Software Engineering*, vol. 43, pp. 658–674, 2017.

[23] M. Aznag, M. Quafafou, and Z. Jarir, "Leveraging formal concept analysis with topic correlation for service clustering and discovery," in *Proceedings of the IEEE International Conference on Web Services*, vol. 1, pp. 153–160, Anchorage AK, USA, July 2014.

[24] M. Al-Hassan, H. Lu, and J. Lu, "A semantic enhanced hybrid recommendation approach: a case study of e-Government tourism service recommendation system," *Decision Support Systems*, vol. 72, pp. 97–109, 2015.

[25] L. Yu, J. Zhou, J. Zhang, F. Wei, and J. Wang, "Time-aware semantic web service recommendation," in *Proceedings of the IEEE International Conference on Web Services*, New York, USA, June 2015.

[26] L. Ignacio, M. Cristian, Z. Alejandro, A. M. Tim, and G. Tor-Morten, "Discovering web services in social web service repositories using deep variational autoencoders," *Information Processing & Management*, vol. 57, no. 4, 2020.

[27] A. Hafsi, G. Youssef, B. N. Cheyma, and L. B. R. SemLinkWS, "Collaboration social network of web services to aid service discovery," *Procedia Computer Science*, vol. 192, 2021.

[28] H. Amal, Y. Gamha, and L. Romdhane, *Social Web Service Discovery Model*, pp. 105–112, 2019.

[29] A. V. Paliwal, B. Shafiq, J. Vaidya, H. Xiong, and N. Adam, "Semantics-based automated service discovery," in *Proceedings of the IEEE Transactions on Services Computing*, Washington, USA, January 2012.

[30] P. Rodriguez-Mier, C. Pedrinaci, M. Lama, and M. Mucientes, "An integrated semantic web service discovery and composition framework," in *Proceedings of the IEEE Transactions on Services Computing*, Rio de Janeiro, Brazil, August 2016.

[31] R. Dumitru, K. Jacek, V. Tomas, and D. John, "WSMO-Lite and hRESTS: lightweight semantic annotations for Web services and RESTful APIs," *Web Semantics Science Services and Agents on the World Wide Web*, vol. 31, pp. 39–58, 2014.

[32] M. Shunmei, D. Wanchun, Z. Xuyun, and C. K. A. S. R. Jinjun, "A keyword-aware service recommendation approach on MapReduce for big data applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, pp. 3221–3231, 2014.

[33] Z. Neng, W. Jian, and M. Yutao, "Mining domain knowledge on service goals from textsual service descriptions," *IEEE Transactions on Services Computing*, vol. 13, pp. 488–502, 2017.

[34] Y. Lina, W. Xianzhi, Z. S. Quan, R. Wenjie, and Z. Wei, "Service recommendation for mashup composition with implicit correlation regularization," in *Proceedings of the IEEE International Conference on Web Services*, New York, USA, July 2015.

[35] S. Richard, C. Danqi, M. Christopher, and N. Andrew, "Reasoning with neural tensor networks for knowledge base completion," in *Proceedings of the 26th International Conference on Neural Information Processing Systems*, pp. 926–934, New York, United States, December 2013.

[36] A. Jain, X. Liu, and Y. Qi, *Aggregating Functionality, Use History, and Popularity of APIs to Recommend Mashup Creation*, Heidelberg Springer, Berlin, 2015.

[37] B. David, N. Andrew, and J. Michael, "Latent dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

[38] P. Samanta and X. Liu, "Recommending services for new mashups through service factors and top-K neighbors," in *Proceedings of the IEEE International Conference on Web Services*, Honolulu HI, USA, June 2017.

[39] W. Pan, L. Bing, L. Jing, M. Yutao, and H. Bo, "Analyzing the structure of Java software systems by weighted k-core decomposition," *Future Generation Computer Systems*, vol. 83, pp. 431–444, 2018.

[40] A. Grover and J. Leskovec, "node2vec: scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco California, USA, August 2016.

[41] W. Pan, S. Beibei, L. Kangshun, and Z. Kejun, "Identifying Key classes in object-oriented software using generalized k-core decomposition," *Future Generation Computer Systems*, vol. 81, pp. 188–202, 2018.

[42] W. Pan, H. Ming, Z. Yang, and T. Wang, "Comments on "using k-core decomposition on class dependency networks to improve bug prediction model's practical performance"," *IEEE Transactions on Software Engineering*, 2022.

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

[44] G. Huang, C. Quo, M. J. Kusner, Yu Sun, K. Q. Weinberger, and F. Sha, "Supervised word mover's distance," in *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS'16)*, pp. 4869–4877, Red Hook, NY, USA, December 2016.

[45] P. Bryan, A. R. Rami, and S. S. DeepWalk, "Online learning of social representations," *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data miningAugust*, pp. 701–710, 2014.