Hindawi
Mathematical Problems in Engineering
Volume 2022, Article ID 3747033, 19 pages
https://doi.org/10.1155/2022/3747033

Research Article

Multigraph Convolutional Network Enhanced Neural Factorization Machine for Service Recommendation

Wei Gao and Jian Wu

School of Computer Science and Engineering, Zhejiang University, Hangzhou, Zhejiang 310027, China

Correspondence should be addressed to Jian Wu; wujian2000@zju.edu.cn

Received 21 December 2021; Accepted 16 February 2022; Published 1 April 2022

Academic Editor: Hua Ming

With an increasing number of web services on the Web, selecting appropriate services to meet the developer's needs for mashup development has become a difficult task. To tackle the problem, various service recommendation methods have been proposed. However, there are still challenges, including the sparsity and imbalance of features, as well as the cold-start of mashups and services. To tackle these challenges, in this paper, we propose a Multigraph Convolutional Network enhanced Neural Factorization Machine model (MGCN-NFM) for service recommendation. It first constructs three graphs, namely, the collaborative graph, the description graph, and the tag graph. Each graph represents a different type of relation between mashups and services. Next, graph convolution is performed on the three graphs to learn the feature embeddings of mashups, services, and tags. Each node iteratively aggregates the information from its higher-order neighbors through message passing in each graph. Finally, the feature embeddings as well as the description features learned by Doc2vec are modeled by the neural factorization machine model, which captures the nonlinear and higher-order feature interaction relations between them. We conduct extensive experiments on the ProgrammableWeb dataset, and demonstrate that our proposed method outperforms state-of-the-art factorization machine-based methods in service recommendation.

1. Introduction

Service-oriented computing (SOC) has become a significant paradigm for developing low-cost and reliable software applications in software engineering and cloud computing [1]. Web services are the basic building blocks of service-oriented computing, which encapsulate application functionalities and can be accessed through standard interfaces. Nowadays, an increasing number of web services, mainly in the form of RESTful Web APIs, have been published online. According to the recent statistics of ProgrammableWeb (https://www.programmableweb.com/), the largest online Web API registry, there are over 24,000 Web APIs available. However, the functionality of an individual service is limited and cannot satisfy the complex requirements of developers. As a result, it is common for developers to compose existing services and develop value-added services, also called mashups [2]. For example, a developer may create a new mashup that can display ratings and reviews of restaurants on a regional map by integrating Google Map Service and Yelp Service. However, creating a mashup can be difficult and time-consuming for an inexperienced developer due to the overwhelming number of services on the Internet. Therefore, it is vital to proactively recommend appropriate services that can satisfy the developer's complex requirements and reduce the burden of manual selection, especially in the era of the Internet of Things and Big Data [3].

A variety of methods have been proposed to recommend services for mashup development, and they can be divided into three classes: collaborative filtering-based methods, content-based methods, and hybrid methods [4]. Different types of methods focus on modeling different sources of information. Collaborative filtering-based methods make use of past composition history of mashups; Content-based methods make use of user requirements and service descriptions; Hybrid methods are a combination of the two

methods, and various kinds of additional information are incorporated, such as tags, categories, developers, QoS, etc.

In this paper, we adopt a factorization machine-based model, which is a type of hybrid method for service recommendation. Factorization Machine (FM), first proposed in [5] by Rendle, is a supervised machine learning algorithm for classification, regression, and ranking tasks. It achieves great success and is widely employed for product recommendation as it can model high-dimensional sparse features and their interactions in an efficient manner [6]. As a result, it attracts a lot of research attention for service recommendation. For instance, in [7], Cao et al. use FM with topic similarities, co-occurrence, and popularity as input features for service recommendation. In [8], Cao et al. propose an attentional FM model that discriminates the importance of different feature interactions with an attention mechanism. In [9], Kang et al. propose a hybrid FM model by integrating a deep neural network to capture the nonlinear and higher-order feature interactions.

Although previous FM-based models have achieved great results for service recommendation, there are still challenges that can limit the performance of the existing methods:

(i) Feature sparsity and imbalance problems: the performance of the factorization machine-based models heavily relies on the co-occurrence of different features. However, the features used for service recommendation are generally sparse: (1) although there are numerous services on the web, each mashup only composes a few services, and most services are composed by mashups only a few times, leading to a sparse mashup-service composition record. Moreover, only a small portion of services are composed frequently, while the majority of services are rarely composed, resulting in the imbalance problem. It is difficult to learn latent factors for mashups and rarely composed services with limited data; (2) even if contextual information such as categories, tags, providers, etc. is incorporated, the features themselves may suffer from the sparsity and imbalance problem. For instance, a majority of tags are used in a few services, while a few popular tags are frequently used by a majority of services. In this case, the model suffers from learning informative latent factors for infrequently used tags, and the benefit of incorporating tag information into the model is reduced.

(ii) Cold-start problem: when a new mashup is first included in the development cycle, it does not contain any component services. We refer to them as "cold-start mashups". As FM can only learn features that appear in the training data, the latent factors for the cold-start mashup cannot be learned due to a lack of composition data. Similarly, we refer to services that have never been composed by any mashup as "cold-start services", and it is impossible to learn the latent factors of cold-start services. As a result, when a cold-start mashup or service is encountered, FM can only make predictions by setting the latent factors of the mashup or service to random values, and relying on the latent factors of contextual information such as description information. Therefore, it is unlikely to lead to good recommendation results for cold-start mashups.

To address the aforementioned issues, we aim to learn informative latent factors for sparse and cold-start mashups and services in FM. To this end, we enhance FM with Multigraph Convolutional Network (MGCN), which exploits various relations between mashups and services. We dub our model the Multigraph Convolutional Network enhanced Neural Factorization Machine, or MGCN-NFM for short. In particular, we construct three graphs, namely, collaborative graph, description graph, and tag graph. Each graph reflects different types of relations, and different relations can complement each other to enrich the information in a mashup or service. Next, we use graph convolution on each graph to further enrich the relational features. The intuition is that by exploiting information from higher-order neighbors, the representations of sparse nodes can be enhanced. To reduce the computational complexity, we use a neighbor sampling technique to maintain a small representative set of neighbors for each node in the graph convolution process. After graph convolution, each mashup or service has three enhanced feature representations that possess different characteristics. These feature embeddings, as well as tag feature embeddings learned from the tag graph and the description features learned from Doc2vec, are used as the input of FM to capture the fine-grained feature interactions. In this paper, we adopt the neural factorization machine model (NFM) proposed in [10]. It enhances vanilla FM by modeling higher-order and nonlinear feature interactions with a bilinear interaction pooling layer and several deep neural network layers. Different from [10], the embedding layer of NFM is obtained based on the MGCN instead of an embedding lookup table. Our model effectively reduces the feature sparsity, imbalance, and cold-start problems due to the multigraph construction and convolution process. For sparse features, their embeddings have more chances of being updated in training as they are connected to more multihop nodes with different types of relations. For cold-start mashups or services, since they have (multihop) connections to other nodes in the description graph and tag graph, their feature embedding can be learned when updating the embeddings of their neighbors.

The contributions of this work can be summarized as follows:

(i) We introduce a novel framework, MGCN-NFM, for service recommendation by enhancing the neural factorization model with a multigraph convolutional network that learns the latent factors for various features.

(ii) We alleviate the feature sparsity, imbalance, and cold-start problems by constructing three graphs reflecting different types of relations between mashup and service, and using multigraph convolution to enrich the sparse relations.

(iii) We conduct extensive evaluations of our proposed model on the ProgrammableWeb dataset, and the results show that our model achieves better performance than state-of-the-art FM methods for service recommendation.

The rest of this article is organized as follows: Section 2 presents the related work of service recommendation. Section 3 introduces the details of our proposed approach. Section 4 reports the experimental results and analysis. Section 5 concludes this paper with future work.

2. Related Work

Recommending web services to developers according to their mashup requirements is a hot topic in service computing. Most works use the dataset from ProgrammableWeb, and they exploit different kinds of auxiliary information, including functional descriptions, tags, categories, providers, architectural styles, etc., as the mashup-service composition record is extremely sparse. Based on the modeling of the mashup-service composition record, existing research can be roughly divided into three categories: neighbor-based collaborative filtering (CF) methods [11–15], latent factor-based CF methods [6, 16–21] and deep learning-based methods [8, 9, 22–27].

Neighbor-based CF methods recommend services based on the identification of similar users (user-based CF) or similar items (item-based CF). It is widely used in the scenario of QoS-aware service recommendation, where the Quality of Service (QoS) information of services is available [28, 29]. Absent from the QoS information, existing works in service recommendation for mashups calculate the matching degree between mashup and service from different information sources with different methods and aggregate them together. As there are multiple types of objects (mashup, service, content, tag, etc.) and rich relations among those objects, they usually build a heterogeneous graph to capture them. In [11], Cao et al. recommend services by building a social network based on social relationships among mashups, services, and their tags. In [12], Gao et al. propose to use a generalized manifold ranking algorithm on the graph with relations among mashups and services. In [13], Liang et al. measure the meta-path-based similarity between mashups under different semantic meanings based on a heterogeneous information network (HIN). Based on the HIN, in [14], Xie et al. propose a mashup group preference-based service recommendation, where mashup group preference is utilized to capture the rich interactions among mashups. In [15], Wang et al. design a knowledge graph to encode the mashup-service relations and exploit random walks with restart to assess the similarities.

Latent factor-based CF methods aim to learn the latent factors of mashups and services to discover potential features. The two most widely adopted models are the matrix factorization-based model (MF) and the factorization machine-based model (FM). MF models decompose the mashup-service composition matrix to derive the mashup feature vectors and service feature vectors (embeddings). The key is to build relevant features that are helpful for improving the recommendation performance. In [16], Xu et al. propose a social-aware service recommendation model, where a coupled matrix factorization model is used to predict the multidimensional relations among users, mashups, and services. In [17], Yao et al. propose a probabilistic MF with implicit correlation regularization, where they develop a latent variable model to uncover the coinvocation patterns of services driven by explicit textual relations and implicit similar or complement relations. In [18], Gao et al. compute different similarity scores between services through heterogeneous functional aspects, and MF is used to learn the embeddings of mashups and services for each functional aspect. The FM model is a generalization of the linear regression model and the MF model. It is more powerful than the MF model because it can entail contextual features and model second-order feature interactions of various sparse features. In [6], Cao et al. propose a QoS-aware service recommendation based on relational topic model (RTM) and FM, where RTM is first used to mine the latent topics derived from the relations among mashups, services, and their links, and then FM is used to predict their link relations. In [19], Li et al. integrate tag, topic, co-occurrence, and popularity factors in the FM for service recommendation, where they exploit the enriched tags and topics of mashups and services derived by RTM and use the invocation times and category information of services to derive their popularity. In [20], Cao et al. extend the description of services using Word2vec and derive latent topics by the hierarchical dirichlet process (HDP). FM is then applied to train these latent topics for service recommendation. In [21], Xie et al. propose to use FM on the features of mashups and services learned from different kinds of metapaths of HIN.

With the rapid development of deep learning in the past few years, it has become popular to adopt neural networks for service recommendation. Compared with traditional methods, deep learning-based methods can perform feature engineering automatically and provide a more powerful representation capability. In [22], Zhang et al. cluster the descriptions of services using Doc2Vec and use the DeepFM model [30] to mine the higher-order composition relations. In [23], Chen et al. also adopt the DeepFM model by taking the features learned from word embedding and the Dirichlet mixture model (DMM) as input. In [8], Cao et al. propose an attention FM [31] by employing an attention mechanism that learns the importance of each input feature interaction via a neural network model. In [9], Kang et al. further combine the advantages of the DeepFM model and the attention FM model, and propose the NAFM model that can capture the nonlinear feature interactions with different degrees of importance. In [24], Xiong et al. adopt three kinds of similarity feature extractors on textual descriptions by a variety of pretrained word embeddings and integrate the

mashup-service composition record and their textual descriptions by using a multilayer perceptron.

Graph Neural Network (GNN) [32] has recently emerged as a popular deep learning model for extracting features from graph-structured data, and research has begun to focus on the use of various GNN models for service recommendation. In [25], Zhang et al. propose a Semantic Variational Graph Auto-Encoder model, where they construct the service graph using the composition relations in the mashup, and use a variational graph autoencoder model as a link prediction task for service recommendation. In [26], Lian and Tang propose a service recommendation method that exploits the higher-order connectivity between mashups and services based on the neural graph collaborative filtering technique. In [27], He et al. propose a service link prediction method based on a heterogeneous graph attention network, where they select five types of neighbors associated with service links, and two levels of attention are applied to learn the importance of different nodes and their associations. Our method is different from theirs in that the constructed graphs incorporate different kinds of relations, making them more comprehensive, and the learned features go through the FM model that can better exploit their hidden interactions.

3. MGCN-NFM Approach

In this section, we will describe our proposed multigraph convolutional network enhanced neural factorization machine model (MGCN-NFM) in detail. First, in Section 3.1, we describe the problem of service recommendation for the mashup. Next, we present the overall framework in Section 3.2. The main components of the model, namely, graph construction, multigraph convolutional network, and neural factorization machine, are described in Sections 3.3, 3.4, and 3.5, respectively. Finally, the training process and model complexity are discussed in Section 3.6.

3.1. Problem Definition. We denote $\mathcal{M} = \{m_1, m_2, \ldots, m_M\}$ as a set of M mashups, $\mathcal{S} = \{s_1, s_2, \ldots, s_N\}$ as a set of N services. $Y \in \{0,1\}^{M \times N}$ denotes the mashup-service invocation history, where $Y_{ij} = 1$ means mashup m_i has composed service s_j in the past, otherwise $Y_{ij} = 0$. Each mashup and each service are associated with its title, description, and tags. In particular, titles and descriptions consist of a sequence of words. For each mashup m, we concatenate its title and description and denote as $d_m = \{w_1, w_2, \ldots, w_{|d_m|}\}$, where w_i is a word and $|d_m|$ is the length of the description. $\mathcal{D}_{\mathcal{M}} = \{d_{m_1}, d_{m_2}, \ldots, d_{m_M}\}$ denotes the descriptions of all mashups in $\mathcal{M}$. For each service s, d_s can be defined in a similarly manner, and $\mathcal{D}_{\mathcal{S}} = \{d_{s_1}, d_{s_2}, \ldots, d_{s_N}\}$ denotes the descriptions of all services in $\mathcal{S}$. We denote $\mathcal{T} = \{t_1, t_2, \ldots, t_T\}$ as a set of T tags. $MT \in \{0,1\}^{M \times T}$ denotes the tagging assignment of mashups, where $MT_{ij} = 1$ means tag t_j is tagged by mashup m_i, and otherwise $MT_{ij} = 0$. Similarly, $ST \in \{0,1\}^{N \times T}$ denotes the tagging assignment of services. The problem of service recommendation can be defined as follows: given $\mathcal{M}, \mathcal{S}, \mathcal{T}, Y, \mathcal{D}_M, \mathcal{D}_S, MT$, and ST that are recorded in the service repository, and a newly developed mashup with requirement description, specified tags, and possibly composed services, the goal is to recommend a list of appropriate services that are likely to be composed by the new mashup.

3.2. Overall Framework. Figure 1 shows the overall framework of our proposed MGCN-NFM model. It mainly consists of three modules: graph construction, multigraph convolutional network, and neural factorization machine. In the graph construction module, we construct three graphs capturing the three relations between mashups and services, namely, collaborative graph, description graph, and tag graph. In constructing the description graph, the descriptions of mashups and services are transformed to the low-dimensional description embedding by the Doc2vec technique. In the multigraph convolutional network module, we adopt the simplified graph convolution network for each graph to propagate higher-order information between mashups and services and output the aggregated feature representations of each node. In the neural factorization machine module, the feature representations of mashup, service, and tags learned in each graph, as well as the description embedding of mashup and service, are used as inputs to the neural factorization machine model, which can model nonlinear and higher-order feature interactions. Finally, the model outputs a score representing the probability of a mashup composing a service. For model training, the predicted score is compared with the actual composition record, and the error is back-propagated to update the model parameters. For service recommendation, the scores of the mashup and all candidate services are computed and ranked, and the top-scoring services are recommended.

3.3. Graph Construction. To fully exploit different kinds of information in mashups and services, we construct three graphs reflecting diverse relations between mashups and services: the collaborative graph, the description graph, and the tag graph. Figure 2 illustrates an example of the three constructed graphs.

3.3.1. Collaborative Graph. The collaborative graph reflects the historical composition relation between mashup and service. However, each mashup generally only composes a small number of services, making the composition relation extremely sparse. Inspired by the idea of collaborative filtering that mashups composed of the same set of services tend to be similar, we expand the composition relation to alleviate the sparsity problem by exploiting the collaborative signal. The invocation matrix Y can be regarded as a mashup-service bipartite graph $\mathcal{G}^{MS} = (\mathcal{M} \bigcup \mathcal{S}, \mathcal{E}^{MS})$, where there is an edge between mashup m and service s, i.e. $(m, s) \in \mathcal{E}^{MS}$ if $Y_{ms} = 1$. We also consider the collaborative relations inside the mashups and services. We build a mashup-mashup collaborative graph $\mathcal{G}^{MM} = (\mathcal{M}, \mathcal{E}^{MM})$ and a service-service collaborative graph $\mathcal{G}^{SS} = (\mathcal{S}, \mathcal{E}^{SS})$ to complement the graph $\mathcal{G}^{MS}$. The mashup-mashup

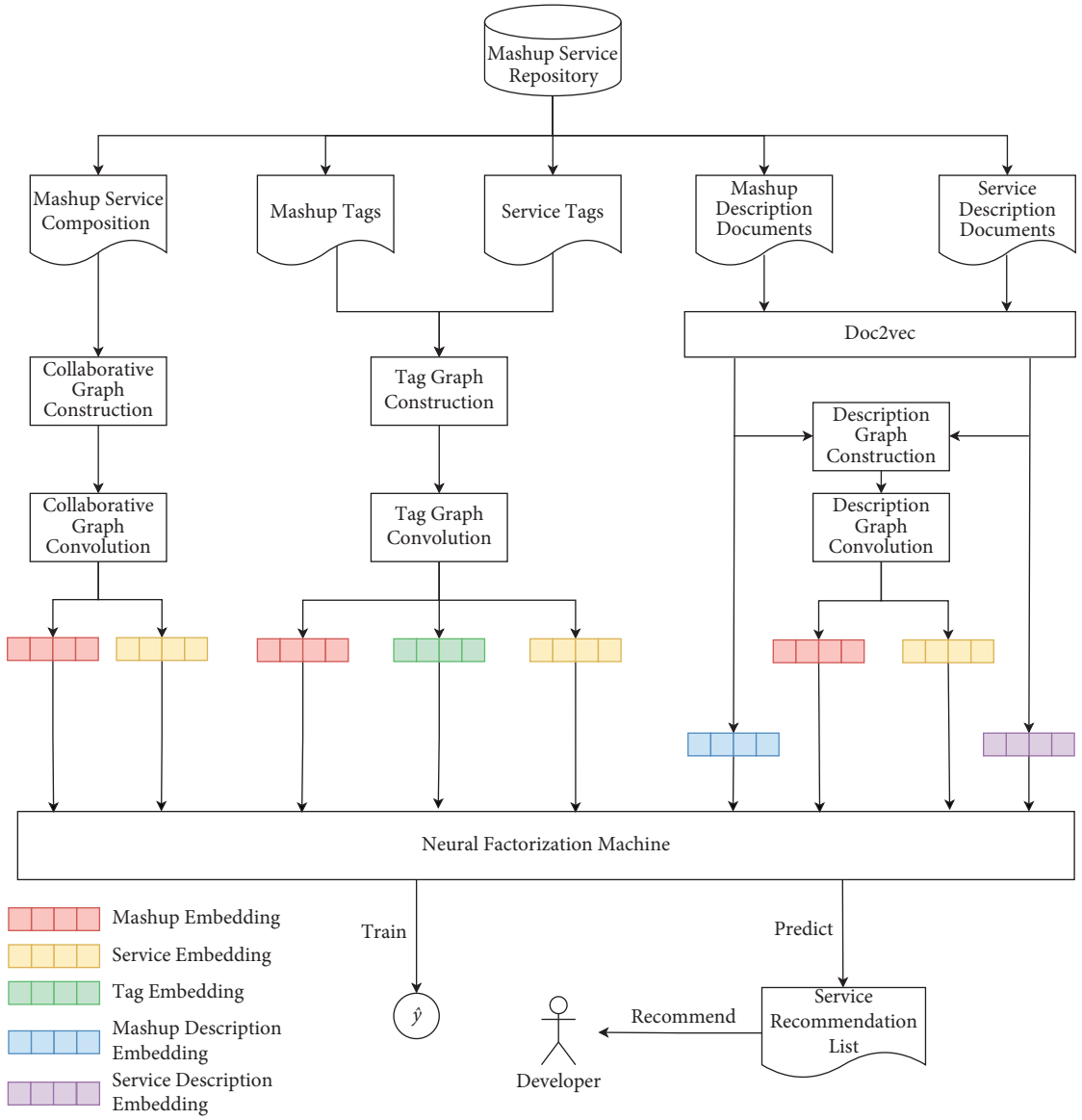


Figure 1: Overall framework of MGCN-NFM model.

collaborative graph is constructed based on the similarity of mashups measured by the number of commonly composed services. Specifically, if mashups i and j have co-composed services, then there is an edge $(i, j) \in \mathcal{E}^{MM}$ between them. The similarities between them are calculated as the number of co-composed services normalized by the number of composed services for each mashup, i.e., $sim(i, j) = Y_i \cdot Y_j / Y_i Y_j$, where Y_i and Y_j denote the i-th and j-th rows of the matrix Y. The normalization keeps the similarity in the range of [0,1]. In a similar manner, the service-service collaborative graph can be constructed based on the similarity of services measured by the number of co-appearing mashups. If services i and j have both occurred in the same mashup, they are connected by an edge $(i, j) \in \mathcal{E}^{SS}$. The similarity between them is defined as $sim(i, j) = Y_{:,i} \cdot Y_{:,j} / Y_{:,i} Y_{:,j}$, where $Y_{:,i}$ and $Y_{:,j}$ denote the i-th and j-th columns of the matrix Y. With the constructed mashup-mashup collaborative graph $\mathcal{G}^{MM}$ and service-service collaborative graph $\mathcal{G}^{SS}$, one mashup can directly utilize the information of the neighboring mashups, thereby alleviating the sparsity issue with the mashup-service invocation record. We merge the three graphs to get an overall collaborative graph $\mathcal{G}^{C} = (\mathcal{M} \bigcup \mathcal{S}, \mathcal{E}^{MS} \bigcup \mathcal{E}^{MM} \bigcup \mathcal{E}^{SS})$.

3.3.2. Description Graph. The description graph represents the similarities between mashups and services in terms of their textual descriptions. We employ the state-of-the-art Doc2vec technique [33] to measure the description similarity. Doc2vec maps the textual descriptions of mashup d_m and service d_s into latent semantic embeddings $\mathbf{d}_m$ and $\mathbf{d}_s$. Doc2vec follows the idea of word2vec [34] for learning word representations. Specifically, each document and each word is mapped to a unique vector. The model concatenates the document vector with a sequence of word vectors from the

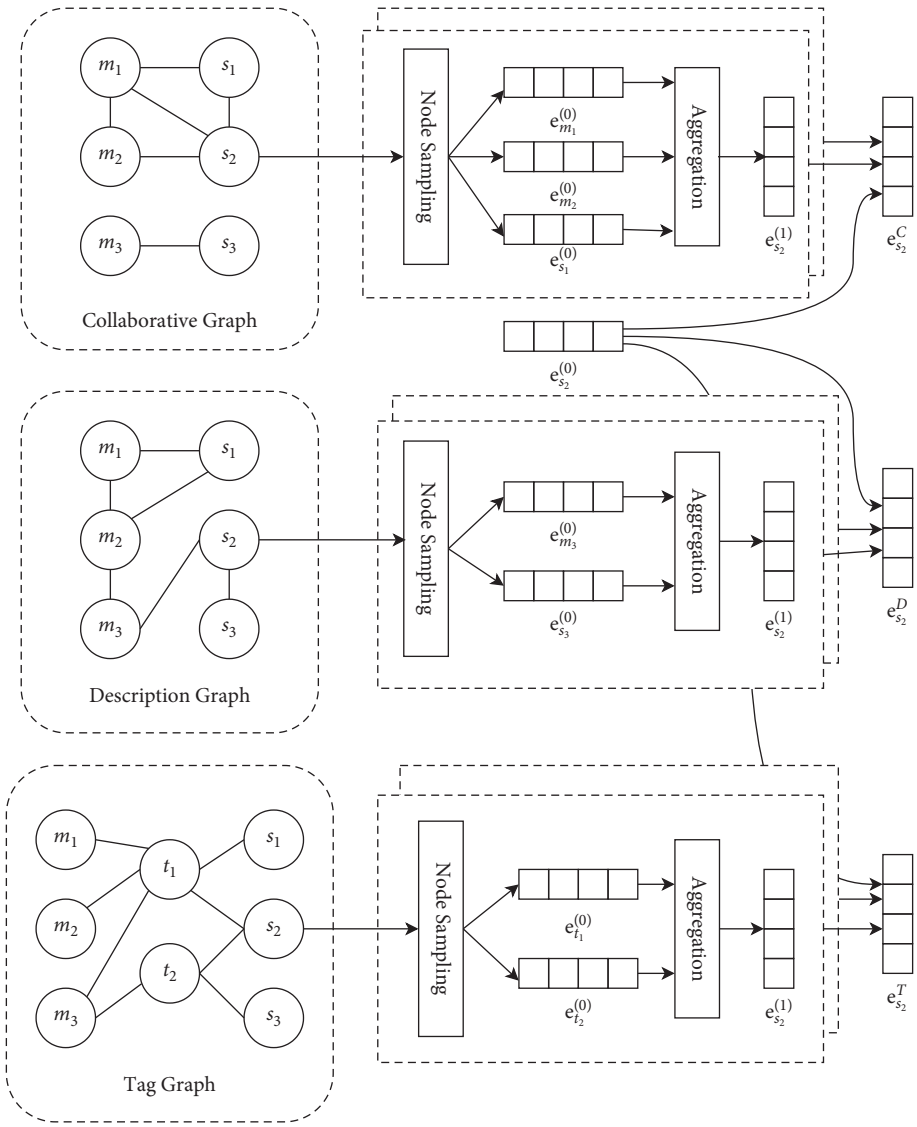


Figure 2: Overall framework of multigraph convolutional network.

document and predicts the following word. It can be seen as a self-supervised learning task where the input is the context words and the document, and the label is the target word. After training on a large number of documents, we can learn a good document embedding that captures the overall semantics of the document. It has been shown to achieve superior performance compared to traditional bag-of-words models such as LDA [35] or HDP [36]. We feed the descriptions of all mashups and services for training the Doc2vec model, and get the corresponding description embeddings after training. We calculate the cosine similarity of the learned embeddings to represent the description similarity. Specifically, given two nodes i and j that could be either mashup or service in the description graph, the similarity between them is calculated as $sim(i, j) = \mathbf{d}_i \cdot \mathbf{d}_j / \mathbf{d}_i \mathbf{d}_j$. However, including all pairs of nodes as edges brings a lot of noise as most pairs of mashup and service are irrelevant. We set a threshold τ (empirically set as 0.1) to filter out dissimilar pairs of nodes. That is, if $sim(i, j) > \tau$, there is an edge between nodes i and j. In this way, we get a description graph $\mathcal{G}^D = (\mathcal{M} \bigcup \mathcal{S}, \mathcal{E}^D)$, where $\mathcal{E}^D$ denotes the filtered edges in the description graph.

3.3.3. Tag Graph. The tag graph represents the annotated tags of mashups and services and is denoted as $\mathcal{G}^T = (\mathcal{M} \bigcup \mathcal{S} \bigcup \mathcal{T}, \mathcal{E}^{MT} \bigcup \mathcal{E}^{ST})$. Different from the collaborative graph and the description graph, it consists of three types of nodes: mashup, service, and tag. There is an edge between mashup i and tag j, i.e., $(i, j) \in \mathcal{E}^{MT}$ if $MT_{ij} = 1$. Similarly, an edge $(i, j) \in \mathcal{E}^{ST}$ exists between service i and tag j if $ST_{ij} = 1$.

3.4. Multigraph Convolutional Network. After constructing three types of graphs reflecting different relations between mashups and services, we employ the multigraph convolutional network to learn comprehensive mashup and service features from the three graphs. In particular, it consists of two steps: (1) Neighbor Sampling, which samples a fixed number of neighbors for each node in the graph; (2)

Graph Convolution, which aggregates neighboring nodes, updates node embeddings at each layer, and combines embeddings at all layers. Figure 2 illustrates an example of the whole process.

3.4.1. Neighbor Sampling. In traditional GCN [37], each node aggregates information from all neighboring nodes in the graph. However, in the case of the three constructed graphs, some nodes may be connected to a large number of nodes, resulting in a large number of neighbors. For instance, in the collaborative graph, a popular service may connect to a large number of mashups and services, as it may be composed of many mashups and cocomposed with many services. Directly aggregating all neighboring nodes may have several issues: (1) it adds the computational burden of the graph convolution process as the size of neighbors grows exponentially with the number of layers; (2) it makes the learning process excessively focus on the nodes with a large number of neighbors and ignores the nodes with only a few neighbors, causing the overfitting issue for the densely-connected nodes and the underfitting issue for the sparsely-connected nodes; (3) it causes the oversmoothing issue [38] when aggregating a large number of neighbors with weak correlation as the learned embedding for nodes and their neighbors will become indistinguishable. Hence, we adopt the neighbor sampling strategy to control the size of the neighbors before graph convolution. Specifically, for each node, we sample its neighbors with respect to the similarity between them. The sampling probability of each neighbor is computed as the normalization of their similarity: first, we get a vector of similarity scores for the target node and its neighboring nodes. Then, we normalize the vector into a probability distribution. Finally, we sample the neighbors with respect to the probability distribution. We use the standard normalization function defined as

$$\cdot norm(\mathbf{z})_i = \frac{\mathbf{z}_i}{\sum_{j=1}^{K} \mathbf{z}_j}\#, \tag{1}$$

where $\mathbf{z}$ is the vector of similarity scores. Other normalization functions can be used, such as the softmax function, although no performance gain is observed. Next, we describe the neighbor sampling process for each graph in detail. A graphical illustration of the sampling process for each graph is shown in Figure 3.

(i) Collaborative graph: for each mashup node, we keep all of its service neighbors. If the number of service neighbors is less than N^C, we sample the neighbors from $\mathcal{E}^{MM}$ according to the sampling probability distribution derived from mashup collaborative similarity, until the number of neighbors reaches N^C. For each service node, we keep all of its mashup neighbors. If the number of mashup neighbors is less than N^C, we sample the neighbors from $\mathcal{E}^{SS}$ according to the sampling probability distribution derived from service collaborative similarity, until the number of neighbors reaches N^C. However, as some popular services may be composed of a large number of mashups, if the number of mashup neighbors is greater than N^C, we sample N^C neighboring mashups with a uniform probability distribution.

(ii) Description graph: for each node, if the number of neighbors is greater than N^D, we sample N^D neighboring mashups and services according to the sampling probability distribution derived from the description similarity.

(iii) Tag graph: as mashups and services are only associated with a few tags, no sampling is needed for the mashup and the service node. For the tag node, we sample N^T nodes from its neighboring mashups and services with a uniform probability distribution.

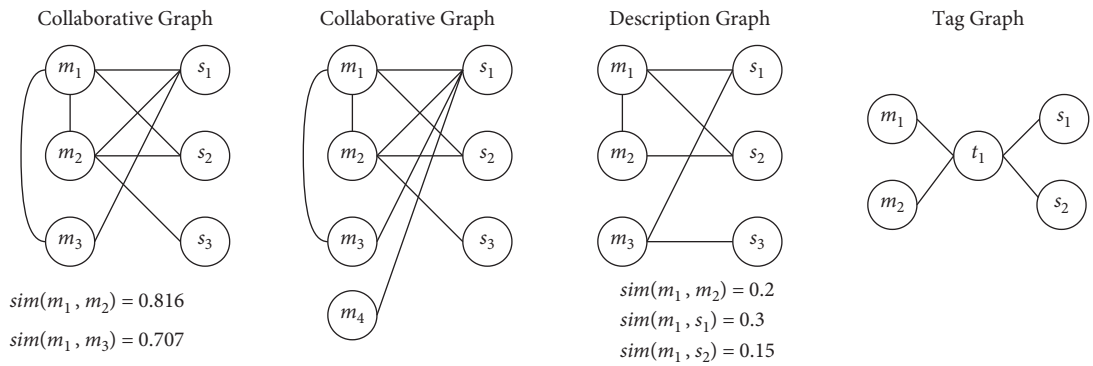
3.4.2. Graph Convolution. The core idea behind graph convolution is to iteratively aggregate feature information from the local neighbors of each node. A single layer of convolution aggregates feature information from the node's direct neighbors, and by stacking multiple layers, feature information can be propagated across long ranges and the node features can be enhanced with sufficient higher-order neighbor information. The graph convolution process consists of three steps: neighbor nodes aggregation, node embedding update, and layer combination. For each graph, after sampling the neighbors for each node, the next step is to aggregate the features of the node neighbors to obtain the neighbor embedding, and the embedding of the node is updated by fusing the neighbor embedding. Denote the center node to be updated as h and the set of neighbor nodes as $\mathcal{N}_h$, the neighbor aggregation and updating process for node h in the l-th layer can be abstracted as

$$\mathbf{e}_h^{(l+1)} = f\left(\mathbf{e}_h^{(l)}, \left\{\mathbf{e}_i^{(l)} : i \in \mathcal{N}_h\right\}\right), \tag{2}$$

where $\mathbf{e}_h^{(l)} \in \mathbb{R}^k$ represents the feature embedding of node h in the l-th layer, and $f(\cdot)$ is the aggregation function which is the core of graph convolution. In this paper, we adopt a simple aggregation function $f(\cdot)$ that drops the feature transformation and nonlinear activation function used in the original graph convolution, which has been shown to achieve better performance with less model complexity [39]. The graph convolution operation is defined as

$$\mathbf{e}_h^{(l+1)} = \sum_{i \in \mathcal{N}_h} \frac{1}{\sqrt{|\mathcal{N}_h||\mathcal{N}_i|}} \mathbf{e}_i^{(l)}. \tag{3}$$

The aggregation function is a weighted sum of the neighbor embeddings, and $1/\sqrt{|\mathcal{N}_h||\mathcal{N}_i|}$ is the normalization term to avoid the scale of embeddings increasing with graph convolution. By stacking L propagation layers defined in (3), each node is capable of receiving features from nodes within L-hops away, and the higher-order relation between mashups and services can be explored. After L layers of



Target Node	m_1	s_1	m_1	t_1
Threshold	$N^C = 3$	$N^C = 3$	$N^D = 2$	$N^T = 2$
Neighboring Nodes	$[s_1, s_2, m_2, m_3]$	$[m_1, m_2, m_3, m_4]$	$[s_1, s_2, m_2]$	$[m_1, m_2, s_1, s_2]$
Sampling Distribution	$P(m_2) = 0.536$ $P(m_3) = 0.464$	$P(m_1) = P(m_2)$ $= P(m_3) = P(m_4)$ $= 0.25$	$P(m_2) = 0.308$ $P(s_1) = 0.461$ $P(m_2) = 0.231$	$P(m_1) = P(m_2)$ $= P(s_1) = P(s_2)$ $= 0.25$
Sampled Nodes	$[s_1, s_2, m_2]$	$[m_1, m_3, m_4]$	$[s_1, m_2]$	$[m_1, m_2]$

Figure 3: An illustration of the node sampling process.

graph convolution, we have $L + 1$ embeddings for node h, and we average them to obtain the final embedding:

$$\mathbf{e}_h = \frac{1}{L+1} \sum_{l=0}^{L} \mathbf{e}_h^{(l)}. \tag{4}$$

Note that the only model parameters in the simplified graph convolution are the embeddings at the 0-th layer, i.e., $\Theta_{MGCN} = \left\{\mathbf{e}_h^{(0)}\right\}$. Figure 2 illustrates an example of learning the embedding of service s_2 with graph convolutional networks in the three graphs. All three graphs share the same initial node embedding for a mashup and service, but since different graphs capture different types of relations, the final embedding of a mashup and service after the graph convolution will capture the unique characteristics specific to that relation. We denote the node embeddings obtained from the collaborative, description, and tag graphs as $\mathbf{e}^C$, $\mathbf{e}^D$, and $\mathbf{e}^T$, respectively.

3.5. Neural Factorization Machine. After learning the embeddings of mashups and services that comprehensively model different types of relations, we use a neural factorization machine model to capture both higher-order and nonlinear interactions between mashups, services, and tags. Figure 4 shows the overall model of the neural factorization machine. Given a pair of mashup and service as well as their corresponding tags and textual descriptions, NFM models both the linear interactions between each pair of features like the traditional factorization machine, and higher-order feature interactions in a nonlinear way. The input of the NFM consists of one-hot encoding of the mashup, one-hot encoding of the service, and multihot encoding of tags that are annotated by the mashup and service. We concatenate the three encoding vectors, and the sparse feature vector is denoted as $\mathbf{x} \in \{0, 1\}^{M+N+T}$, where $\mathbf{x}_i = 1$ means the i-th feature exists in the input. The linear regression part of NFM is given as

$$l(\mathbf{x}) = \mathbf{w}_0 + \sum_{i=1}^{M+N+T} \mathbf{w}_i \mathbf{x}_i, \tag{5}$$

where $\mathbf{w}_0$ is the global bias and $\mathbf{w}_i$ is the weight of feature i. To incorporate the functional semantics in the textual descriptions, we use the dense embeddings of the mashup and service learned from Doc2vec. However, since the features of mashup, service, and tags are sparse, we need to transform them into dense representations. Different from the original neural factorization model in [10], where they build an embedding lookup table which is a fully connected layer that projects each feature to a dense embedding, the multigraph convolution network is used to project the feature to its embeddings from different graphs. In this way, the mashup and service have three embeddings, each representing information from different sources, and the feature interaction can be performed in a fine-grained manner. Moreover, a cold-start mashup or service will have a meaningful embedding which is learned from the description graph and/or tag graph. For tag features, it is also beneficial to learn the tag embeddings from the tag graph, as long-tailed tags will have a more meaningful embedding from the higher-order convolution process of multihop neighbors. With the enhanced feature embeddings, the feature interactions in NFM can be learned more effectively.

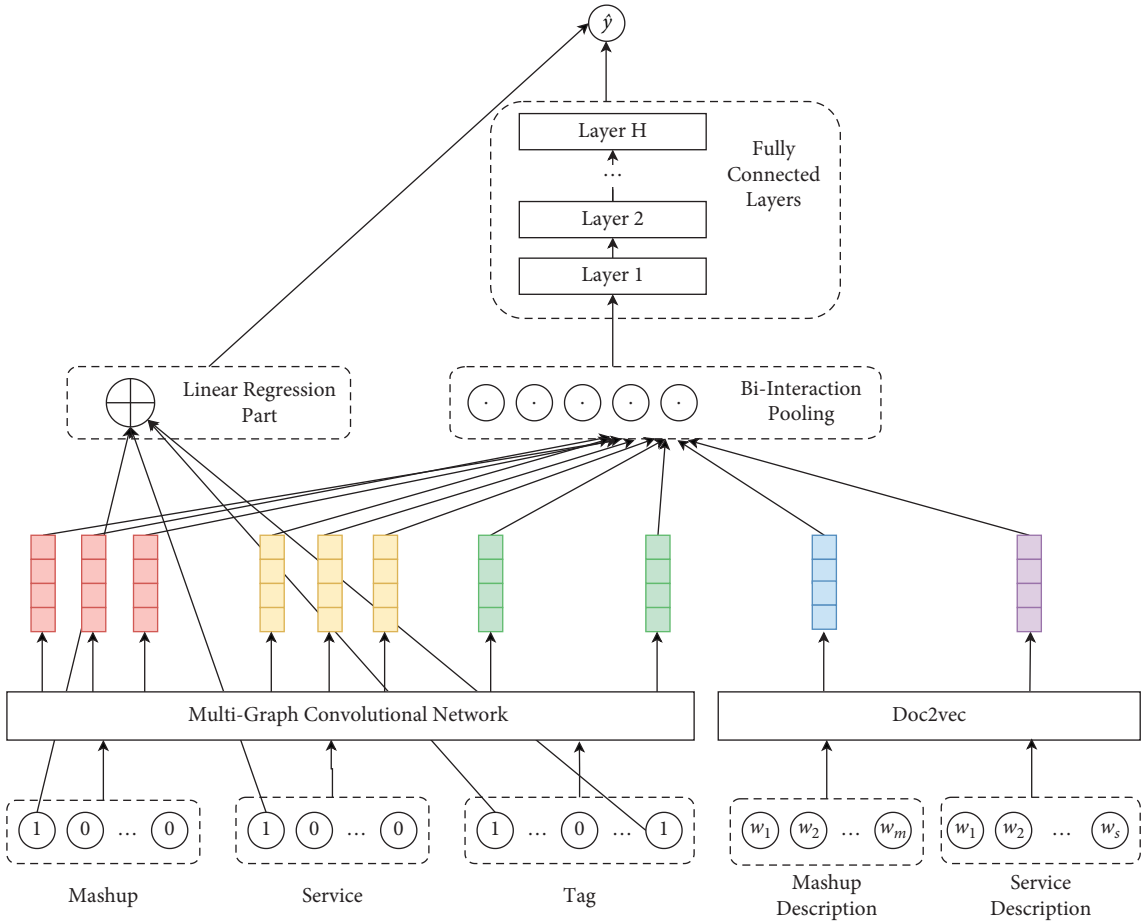


Figure 4: A framework of the neural factorization machine model.

After transforming sparse features of the mashup, service and tags into dense embeddings, they are fed into the bi-interaction pooling layer to convert multiple embedding vectors into one vector. Specifically, the input of the bi-interaction pooling layer consists of the set $\mathcal{V} = \{\mathbf{e}_m^C, \mathbf{e}_m^D, \mathbf{e}_m^T, \mathbf{e}_s^C, \mathbf{e}_s^D, \mathbf{e}_s^T, \mathbf{e}_t, \mathbf{d}_m, \mathbf{d}_s\}$, where $\mathbf{e}_t = \{\mathbf{e}_{t_1}, \mathbf{e}_{t_2}, \ldots\}$ is a list of vectors converted from the multihot tag encoding. The bi-interaction pooling operation is defined as

$$f_{BI}(\mathcal{V}) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathbf{v}_i \odot \mathbf{v}_j, \tag{6}$$

where n is the number of embedding vectors in $\mathcal{V}$, and $\cdot$ denotes the element-wise product of two vectors. The output of the bi-interaction pooling layer is a k-dimension vector that encodes the second-order interactions between features. It is worth noting that in the case of cold-start mashup or service, they are represented as the initial node embeddings, i.e., $\mathbf{e}_m^C$ or $\mathbf{e}_s^C$ equals $\mathbf{e}_m^{(0)}$ or $\mathbf{e}_s^{(0)}$, because there is no connection in the collaborative graph.

Above the bi-interaction layer is a stack of fully connected layers, which learn higher-order interactions between features:

$$\mathbf{z}_H = ReLU\left(W_H\left(\cdots ReLU\left(W_1 f_{BI}(\mathcal{V}) + \mathbf{b}_1\right)\cdots\right) + \mathbf{b}_H\right), \tag{7}$$

where H denotes the number of hidden layers, W_l denotes the weight matrix, and $\mathbf{b}_l$ denotes the bias vector in layer l. ReLU is the activation function. Finally, the output of the last layer $\mathbf{z}_H$ is transformed into the final score:

$$f(\mathcal{V}) = \mathbf{h}^T \mathbf{z}_H, \tag{8}$$

Overall, NFM estimates the target of an instance as the sum of the first-order linear regression part and the higher-order nonlinear feature interaction part. Since the target label is in {0,1}, with 1 indicating that the service is a component of the mashup and 0 otherwise, we transform the final output with the sigmoid function to output the probability of the service being recommended to the mashup. The sigmoid function, defined as $\sigma(x) = 1/(1 + e^{-x})$, constrains the value between 0 and 1. To summarize, the formulation of NFM is

$$\hat{y} = \sigma(l(\mathbf{x}) + f(\mathcal{V})), \tag{9}$$

with parameters $\Theta_{NFM} = \{\mathbf{w}_0, \{\mathbf{w}_i\}, \{W_l, \mathbf{b}_l\}, \mathbf{h}\}$.

3.6. Model Training and Prediction. To train the MGCN-NFM model, we define the loss function using binary cross-entropy:

$$L(\Theta) = -\sum_{i \in \mathcal{T}\cdot} \left[y^{(i)} \log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right) \right] + \lambda \|\Theta\|_2^2, \tag{10}$$

where $\mathcal{T}\cdot$ is the set of training instances, each instance i includes mashup, service, tags, mashup description, and service description. $y^{(i)} \in \{0, 1\}$ is the label indicating whether the mashup and service in instance i have been composed in the record, and $\hat{y}^{(i)} \in (0, 1)$ denotes the composition probability in instance i predicted by the model. As the mashup-service composition record is extremely sparse, the number of negative instances where the label $y^{(i)} = 0$ far exceeds the positive instances where the label $y^{(i)} = 1$. Therefore, for each positive instance, we sample 5 negative instances with the same mashup to mitigate the data imbalance problem. λ controls the strength of the L_2 regularization to prevent overfitting. In addition, dropout is used in NFM to prevent overfitting, where we randomly drop ρ percent of the dimensions in the fully connected layers when training. $\Theta = \{\Theta_{MGCN}, \Theta_{NFM}\}$ denotes all trainable model parameters. We use Adaptive Moment Estimation (Adam) [40], a variant of stochastic gradient descent, to optimize the model parameters.

Algorithm 1 presents the overall training process of the MGCN-NFM framework. In line 1, three graphs are constructed based on the input data. Line 2 constructs the training instance set $\mathcal{T}\cdot$ including both positive and sampled negative labeled instances. Each instance includes a mashup, a service, their tags, and their descriptions. Line 5–7 is a forward-propagation process to make a prediction for an instance. The embeddings of the mashup, service, and tags are learned through the steps of neighbor sampling and graph convolution. The embeddings of the descriptions are obtained from Doc2vec. They are used as the input to NFM to compute the recommendation probability. Line 8 performs back-propagation and updates the model parameters based on the loss function in Equation (10).

Once the model parameters Θ are learned, which contain the initial embeddings for mashups, services, and tags, a forward-propagation pass is performed on the multigraph convolutional network to obtain the final embeddings of all mashups, services, and tags in the three graphs, which act as a fixed embedding lookup table. Then, the mashup to be recommended and each candidate service are used to construct the input instances of the NFM, with their embeddings retrieved from the lookup table. The NFM outputs the probability of each candidate service being composed by the mashup. We rank these prediction values and recommend the top-K services with the largest values to the mashup.

We analyze the time complexity of MGCN-NFM for model training and prediction. The construction of the collaborative, description, and tag graph takes $O(nnz(Y) + M^2 + N^2)$, $O((M+N)^2)$, and $O(nnz(MT) + nnz(ST))$ respectively, where *nnz* denotes the number of nonzero entries in the matrix. The main time cost of the MGCN model lies in the graph convolution process, and the time complexity is $O(L \cdot k \cdot (|\mathcal{G}^C| + |\mathcal{G}^D| + |\mathcal{G}^T|))$, where L is the number of graph convolution layers, k is the embedding size. $|\mathcal{G}^C|$, $|\mathcal{G}^D|$ and $|\mathcal{G}^T|$ are the number of edges in the collaborative, description, and tag graph respectively. Since we perform neighbor sampling for each node in the three graphs, the number of edges is upper bounded by $N^C \cdot (M+N)$, $N^D \cdot (M+N)$, $N^T \cdot (M+N+T)$ for the collaborative, description, and tag graph respectively. Therefore, the training cost of MGCN linearly scales with the number of nodes in the graph. For NFM in both training and prediction, the bi-interaction pooling layer can be efficiently computed in $O(nk)$ time with a reformulation of (6). The main time cost lies in the hidden layers of the neural network, and the time complexity is $O(\sum_{l=1}^{L} k_{l-1} k_l)$, where k_l denotes the dimension of the l-th hidden layer.

4. Experiments

In this section, we conduct a series of experiments to evaluate our proposed model in service recommendation and present the empirical performance. All experiments were developed in Python and carried out on a personal PC with Intel Core i7 CPU with 2.5 GHz and 16 GB RAM, running on macOS High Sierra. We aim to answer the following research questions:

(i) How does the proposed MGCN-NFM model perform compared with the state-of-the-art factorization machine-based service recommendation methods?

(ii) How much do the collaborative graph, description graph, and tag graph influence the performance of the proposed model?

(iii) How much do the various hyperparameters affect the experiment results of the proposed model?

4.1. Dataset Description. The experimental dataset is collected from ProgrammableWeb (PW), the largest online web service and mashup repository. For each mashup, we crawl its name, description, tags, and composed services. For each service, we crawl its name, description, and tags (including the primary and secondary categories). We remove duplicate services, and the mashups and services without textual descriptions and tags. The dataset contains 6,300 mashups and 21,474 web services, of which only 1,609 services are composed of at least one mashup. Table 1 shows the detailed statistics of the dataset.

A five-fold cross-validation is performed to evaluate the effectiveness of the model. We randomly split the mashup-service invocation records into five folds, and in each round, one fold is used as the test set while the remaining four folds are used as the training set. The results of the five rounds are averaged as the final result. In addition, to test models under different data sparsity levels, we use 1/2/3/4 folds in the training set, which corresponds to 20/40/60/80% of the mashup-service invocation records. Figure 5(a) shows the

Input: $\mathcal{M}$, $\mathcal{S}$, $\mathcal{T}$, Y, $\mathcal{D}_M$, $\mathcal{D}_S$, MT, ST
Output: parameter set Θ
(1) Construct graphs $\mathcal{G}^C$, $\mathcal{G}^D$, $\mathcal{G}^T$;
(2) Construct training instances $\mathcal{T}r$;
(3) for epoch = 1, ..., p do
(4) **for** each instance i composed of mashup m, service s, tag t in $\mathcal{T}r$ do
(5) Compute $\mathbf{e}_m^C, \mathbf{e}_m^D, \mathbf{e}_m^T, \mathbf{e}_s^C, \mathbf{e}_s^D, \mathbf{e}_s^T, \mathbf{e}_t$ with Equation (4);
(6) Obtain $\mathbf{d}_m, \mathbf{d}_s$ from Doc2vec model;
(7) Compute $\hat{y}^{(i)}$ with Equation (9);
(8) Update Θ to minimize $\mathcal{L}$ in Equation (10) with Adam;
(9) **end for**
(10) **end for**
(11) sreturn Θ

Algorithm 1: Training algorithm of MGCN-NFM.

Table 1: Dataset statistics.

Statistics	Value
Number of mashups	6,300
Number of services	21,474
Number of services composed by mashup	1,609
Number of mashup-service invocation	13,219
Average number of services in mashup	2.07
Sparsity of mashup-service composition matrix	99.87%
Number of tags	312
Average number of tags in mashups	3.62
Average number of tags in services	2.93

statistics of the service distribution of mashup in the full training set, and Figure 5(b) shows the statistics of the service distribution of mashup when only 20% of the dataset is used for training. We can see that for the full training dataset, 52.3% mashups compose one service, and 91.2% mashups compose less than 3 services. When only 20% of the dataset is used for training, 68.6% mashups do not have any component services, which poses a significant challenge for the cold-start problem.

4.2. Evaluation Metrics. We adopt two metrics, namely, recall and Normalized Discounted Cumulative Gain (NDCG), to evaluate the accuracy of the top-K service recommendation.

Recall@K is the proportion of the number of services in the top-K recommendation list that is composed by the mashup to the number of services in the mashup. It is defined as:

$$Recall@K = \frac{1}{|M|} \sum_{m \in M} \frac{|rec(m) \cap truth(m)|}{|truth(m)|}, \quad (11)$$

where M is the set of mashups in the test set, rec(m) is the recommendation list of services of size K for mashup m. truth(m) is the ground truth list of services that are composed by the mashup m in the test set.

NDCG@K considers the ranking position of the recommended services, and assigns different weights to each service in the top-K recommendation list. The higher ranked service is assigned with a larger weight if it is composed by the mashup. It is defined as:

$$NDCG@K = \frac{1}{|M|} \sum_{m \in M} \frac{\sum_{i=1}^{K} 2^{I(i)} - 1/log_2(i+1)}{IDCG@K}, \quad (12)$$

where $I(i)$ indicates whether the service at position i of the ranking list is in truth(m). IDCG@K is the ideal DCG score of the top-K services that can be achieved.

4.3. Baseline Methods. We compare MGCN-NFM with the following baselines that are related to our work:

(i) RTM-FM [7]: this method combines RTM and FM for service recommendation. It uses RTM to mine latent topics of mashups and services by link prediction. In addition, it exploits the co-occurrence and popularity features of services in FM.

(ii) TR-FM [19]: this method integrates tag, topic, co-occurrence, and popularity factors which are modeled by FM for service recommendation. It uses the enriched tags and the derived topic information

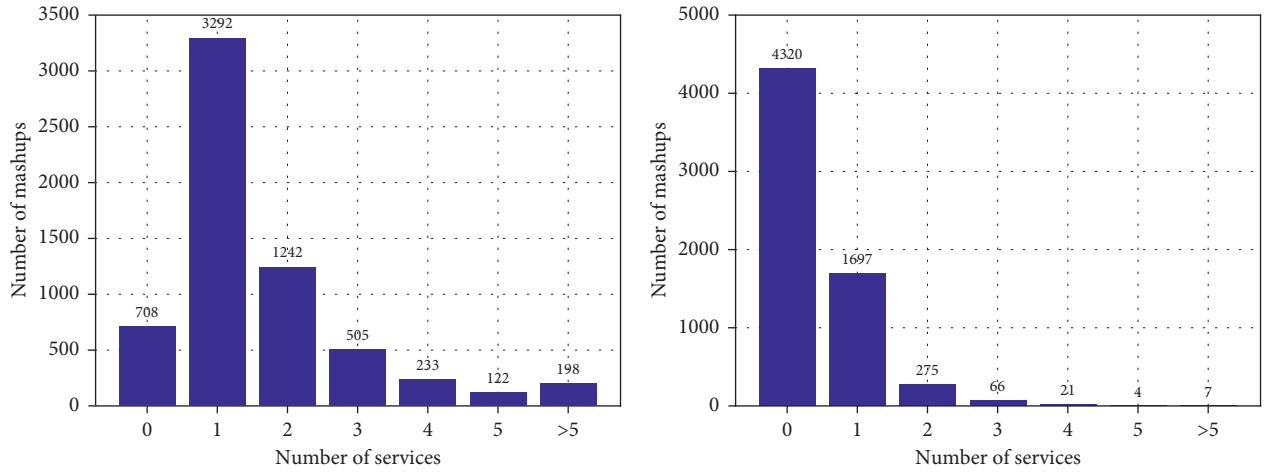


Figure 5: Service distribution of mashups. (a) Full dataset. (b) 20% of the dataset.

from RTM to measure the similarity between services.

(iii) ATT-FM [8]: this method uses attentional FM for service recommendation. It uses functional similarity, tags, and popularity of services as features, and employs an attention mechanism on feature interactions to discriminate their importance.

(iv) NAFM [9]: this is the state-of-the-art service recommendation method based on the FM model. Besides first-order and second-order linear feature interactions, it integrates a deep neural network to capture nonlinear feature interactions and an attention network to capture important feature interactions.

(v) MGCN-FM: it is a variation of the proposed method where we use the basic factorization machine model after learning the features of mashup, service, and tags. The output of the bi-interaction pooling layer is summed directly, which is equivalent to setting $H = 0$, $\mathbf{h} = 1$ in MGCN-NFM.

4.4. Parameter Settings. To learn the representation of textual descriptions of mashups and services in Doc2vec, we aggregate the textual descriptions of all 6300 mashups and 21474 services into one big corpus and perform a series of preprocessing steps: (1) split sentences into words and transform them into lower case; (2) remove stop words and words that appear less than 5 times; (3) transform words into their root form. All preprocessing steps are done using the NLTK library (https://www.nltk.org). After preprocessing, we use the models.doc2vec API in the gensim library (https://radimrehurek.com/gensim/) to learn the document embedding. All parameters are set to the default value of the gensim API except for the dimension size, which is set to 50 as the length of the description of a mashup and service is relatively short. The default values of hyperparameters in MGCN-NFM are set in Table 2.

4.5. Performance Comparison. To assess the performance of our method and baseline methods under different data sparsity levels, we select 20/40/60/80% of the data for constructing the collaborative graph and learning the parameters of the model. Table 3 shows the comparisons of different methods when the number of services recommended for a mashup $K = 3/5/10$. It can be seen that our model MGCN-NFM consistently achieves the best performance under all percentages of training data in terms of both Recall and NDCG. TR-FM achieves better performance than RTM-FM, because it considers the tagging information to refine the similarity measures of services. ATT-FM achieves better performance than the two methods that use the vanilla factorization model, since it models the feature interactions in the factorization machine with an attention mechanism that can discriminate the importance of each feature. NAFM attains better performance than the aforementioned three methods, since considering higher-order feature interactions can bring additional improvement. The MGCN-FM method achieves better performance than all baseline methods, but its performance is inferior to that of MGCN-NFM, especially on training data with high sparsity. This shows that introducing nonlinear feature interactions can improve the expressiveness of FM. Compared with the best baseline model NAFM, our model obtains 7.28% and 7.14% improvement in top-3 recommendations for Recall and NDCG respectively under the full training data. The performance gain becomes even more significant when the training data becomes sparser. For instance, when only 20% of the data is used for training, our model obtains 18.13% and 14.97% improvements in top-3 recommendations for Recall and NDCG compared with NAFM. As the percentage of the training data increases, both Recall and NDCG increase for all methods. However, our method consistently outperforms baseline methods when the training data becomes sparser, while the performance of the baseline methods drops to a greater extent than our method. It shows that our method is less sensitive to the sparsity of the training

Table 2: Default value of hyperparameters.

Item	Value
Number of convolution layers, L	2
Number of latent dimension, k	64
Number of hidden layers, H	2
Hidden units size, k_l	(16,16)
Dropout ratio, ρ	0.3
Initialization of parameters	$\mathcal{N}(0, 0.01)$
Learning rate	0.001
Regularization coefficient, λ	0.0001
Training epoch, p	100
Neighbor size of collaborative graph, N^C	15
Neighbor size of description graph, N^D	30
Neighbor size of tag graph, N^T	30

data compared with the baseline methods, which could be explained by the higher-order and nonlinear feature interaction modeling in graph convolution and NFM.

4.6. Impact of Different Graphs. In MGCN-NFM, we represent different sources of information by constructing three graphs: collaborative graph, description graph, and tag graph. To demonstrate the effectiveness of utilizing the three graphs in our model, we design six comparing variants of the model: (1) CGCN-NFM, where only the collaborative graph is constructed. In this model, only $\mathbf{e}_m^C$ and $\mathbf{e}_s^C$ are computed in Step 5 of Algorithm 1; (2) DGCN-NFM, where only the description graph is constructed, and only $\mathbf{e}_m^D$ and $\mathbf{e}_s^D$ are computed; (3) TGCN-NFM, where only the tag graph is constructed, and only $\mathbf{e}_m^T$, $\mathbf{e}_s^T$ and $\mathbf{e}_t$ are computed; (4) CDGCN-NFM, where both collaborative graph and description graph are constructed, and tag graph is omitted; (5) CTGCN-NFM, where both collaborative graph and tag graph are constructed; (6) DTGCN-NFM, where only collaborative graph is omitted in the original model.

Figure 6 shows the result of the comparison among different variants. We observe that MGCN-NFM achieves the best performance, confirming that all three graphs are necessary for the model, as they are complementary to each other for enriching different aspects of relations between mashups and services. When either graph is dropped from the original model, the performance becomes worse, which also suggests that both three graphs can indeed help improve the model performance. Furthermore, incorporating two graphs into the model is better than only utilizing one graph in the model, except for CTGCN-NFM. In this case, we argue that the description graph is the most important graph for the model performance, as DGCN-NFM outperforms CGCN-NFM and TGCN-NFM, and without the description graph, CTGCN-NFM performs even worse than DGCN-NFM. Therefore, we conclude that accurately modeling the functional information of mashups and services in their descriptions is vital. In addition, the collaborative graph has a stronger influence on the model performance than the tag graph, which is exhibited by the fact that CGCN-NFM outperforms TGCN-NFM and CDGCN-NFM outperforms DTGCN-NFM.

We further discuss the reason and provide several case studies of how the three graphs alleviate the sparsity, imbalance, and cold-start problems in service recommendation. The collaborative graph connects services that are rarely composed to frequently composed ones, which makes them more densely connected, so that the latent factors can be updated more frequently. The description graph and tag graph connect cold-start services with services that are composed by mashups, so that the cold-start services can be discovered and recommended by the FM model. Table 4 shows some actual examples of service recommendation results. CityPockets Daily Deals is a mashup that aggregates all daily deals and coupons from various deal sites. Compared with DTGCN-NFM which excludes the collaborative graph, MGCN-NFM successfully recommends service 8coupons in the Top-5 list, because service Groupon and service 8coupons have been composed by different mashups a few times, and it is captured by the collaborative graph. NearPlace mashup is a free store locator and Google Maps marker. Compared with CTGCN-NFM which excludes the description graph, MGCN-NFM successfully recommends cold-start service MetaLocator in the Top-5 list. Although MetaLocator has never been composed by any mashups before, it is a mapping and locator service for stores, vendors, and ATMs. Therefore, its functionality is similar to popular services such as geocoder and Google Maps Places, which is reflected in the description graph. iEnviroWatch is a mashup that visualizes and queries environmental information in a geographical area of one's interest. The MGCN-NFM successfully recommends the EEA Discomap service in the Top-5 list, because they are both tagged with "Environment", and the tag "Sustainability" in the mashup is also close to the tag "Environment" in the tag graph.

4.7. In-Depth Analysis of Graph Construction

4.7.1. Impact of Mashup-Mashup Graph and Service-Service Graph in Collaborative Graph Construction. To demonstrate the superiority of incorporating mashup-mashup collaborative graph and service-service collaborative graph into the sparse mashup-service collaborative graph, we design three variants of the model: (1) $\mathcal{G}^{MS}$, where only the mashup-service collaborative graph is constructed; (2) $\mathcal{G}^{MS} + \mathcal{G}^{MM}$,

Table 3: Recall and NDCG comparison of recommendation methods.

	Method	Dataset density							
		Recall				NDCG			
		20%	40%	60%	80%	20%	40%	60%	80%
$K=3$	RTM-FM	0.2429	0.2940	0.3423	0.3806	0.2392	0.2803	0.3105	0.3271
	TR-FM	0.2685	0.3197	0.3628	0.4053	0.2530	0.2930	0.3238	0.3403
	ATT-FM	0.3421	0.3955	0.4225	0.4598	0.3013	0.3357	0.3623	0.3759
	NAFM	0.3733	0.4256	0.4526	0.4724	0.3127	0.3461	0.3724	0.3862
	MGCN-FM	0.4195	0.4652	0.4851	0.5035	0.3402	0.3736	0.3989	0.4116
	MGCN-NFM	0.4410	0.4798	0.4931	0.5068	0.3595	0.3859	0.4069	0.4138
$K=5$	RTM-FM	0.3807	0.4336	0.4855	0.5084	0.2785	0.3184	0.3583	0.3760
	TR-FM	0.4021	0.4548	0.5061	0.5291	0.2913	0.3309	0.3697	0.3863
	ATT-FM	0.4492	0.4969	0.5430	0.5611	0.3368	0.3710	0.4030	0.4144
	NAFM	0.4650	0.5116	0.5574	0.5755	0.3502	0.3838	0.4152	0.4260
	MGCN-FM	0.5071	0.5538	0.5994	0.6173	0.3787	0.4119	0.4432	0.4538
	MGCN-NFM	0.5336	0.5711	0.6087	0.6202	0.3979	0.4244	0.4519	0.4567
$K=10$	RTM-FM	0.4257	0.4947	0.5346	0.5579	0.2822	0.3461	0.3794	0.3965
	TR-FM	0.4472	0.5156	0.5548	0.5770	0.2977	0.3504	0.3835	0.4002
	ATT-FM	0.4969	0.5570	0.5917	0.6094	0.3463	0.3920	0.4166	0.4288
	NAFM	0.5131	0.5729	0.6073	0.6245	0.3590	0.4048	0.4292	0.4403
	MGCN-FM	0.5576	0.6173	0.6512	0.6678	0.3910	0.4366	0.4609	0.4712
	MGCN-NFM	0.5769	0.6294	0.6591	0.6721	0.4096	0.4478	0.4673	0.4750

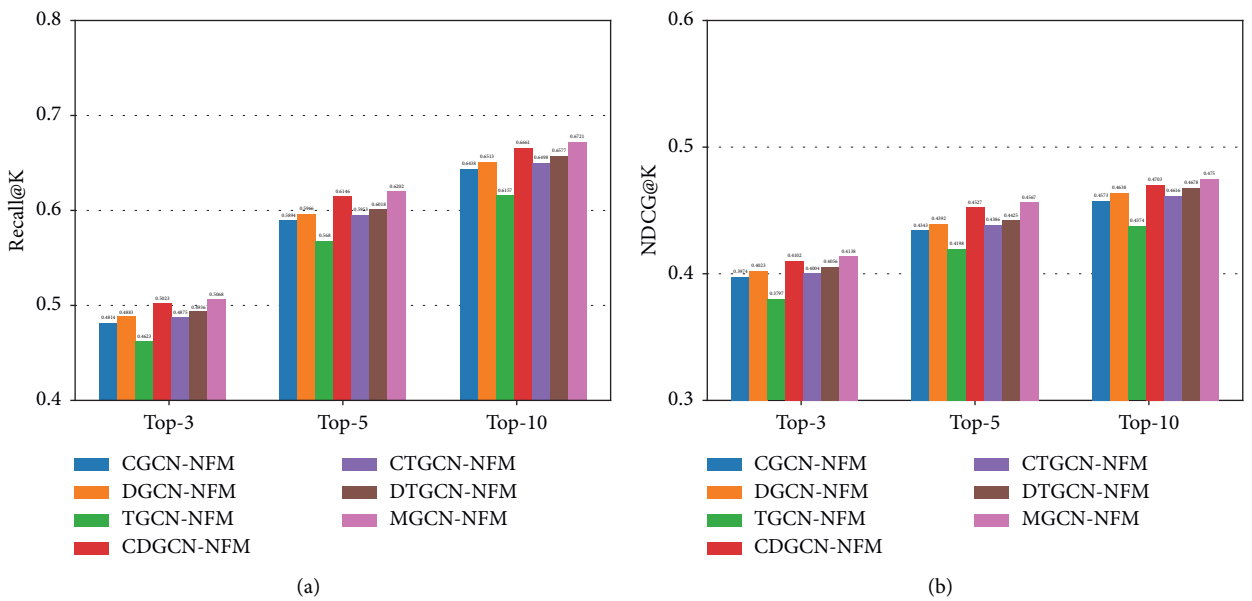


Figure 6: Comparison of different variants of MGCN-NFM. (a) Recall@K. (b) NDCG@K.

Table 4: Case-study analysis of different graphs where the hit services are marked in bold font.

Mashup	Method	Top-5 recommendation of services
CityPockets daily deals	DTGCN-NFM	**Yelp fusion**, amazon product advertising, google maps, facebook, **groupon**
	MGCN-NFM	**Yelp fusion**, amazon product advertising, **groupon**, google maps, 8coupons
	Ground truth	**Yelp fusion**, **groupon**, **8coupons**
NearPlace	CTGCN-NFM	**Google maps**, microsoft bing maps, geocoder, Shopping.com, amazon marketplace
	MGCN-NFM	**Google maps**, geocoder, google maps places, Shopping.com, **MetaLocator**
	Ground truth	**MetaLocator, google maps, WordPress.org, shopify admin, WooCommerce, magento SOAP**
iEnviroWatch	CDGCN-NFM	Google maps, twitter, google latitude, facebook, WiserEarth
	MGCN-NFM	Google maps, twitter, google latitude, AMEE, **EEA discomap**
	Ground truth	**EEA discomap**

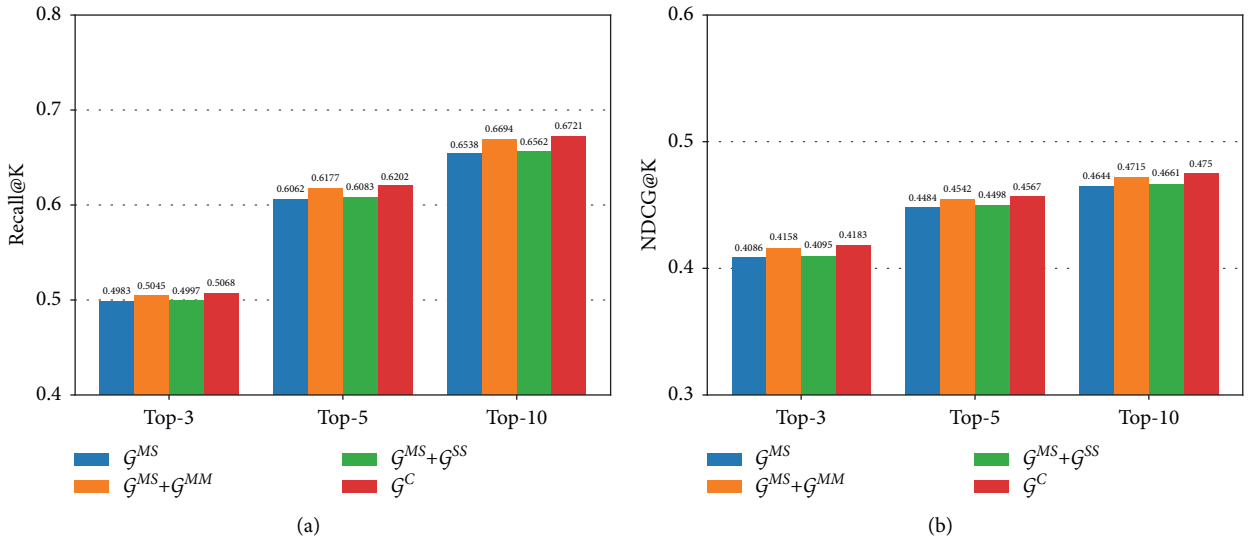


Figure 7: Comparison of different methods for collaborative graph construction. (a) Recall@K. (b) NDCG@K.

where the mashup-mashup collaborative graph $\mathcal{G}^{MM}$ is fused with $\mathcal{G}^{MS}$; (3) $\mathcal{G}^{MS} + \mathcal{G}^{SS}$, where the service-service collaborative graph $\mathcal{G}^{SS}$ is fused with $\mathcal{G}^{MS}$; (4) $\mathcal{G}^{C}$, which is the original model. Figure 7 shows the results of different variants. We observe that incorporating $\mathcal{G}^{MM}$ and $\mathcal{G}^{SS}$ both achieves better model performance than only using $\mathcal{G}^{MS}$ for the collaborative graph, which verifies the effectiveness of modeling the collaborative relations between mashups and between services. Furthermore, the collaborative relation between mashups has a more significant impact on the model performance than the collaborative relation between services. Only by combining both relations can the best result be achieved.

4.7.2. Impact of Document Embedding Techniques in Description Graph Construction. In this paper, we use Doc2vec, which learns a distributed representation of documents with a self-supervised learning paradigm. We compare Doc2vec with other document modeling techniques widely used for service description. Three variants are considered: (1) TF-IDF (https://en.wikipedia.org/wiki/Tf-idf), where the description documents are represented by the TF-IDF model; (2) LDA, where the topic probability distributions of description documents are learned by the LDA model; (3) HDP, where the topic probability distributions of description documents are learned by the HDP model; (4) Doc2vec, which is the original model. The latent dimensions of LDA and Doc2vec are set to 50. Figure 8 shows the results of different variants. We can observe that Doc2vec performs the best among all methods, indicating its superior modeling capacity for mashup and service description. HDP follows behind Doc2vec and performs better than LDA, which shows that HDP can derive better topic distribution than LDA as it can automatically infer the number of topics from data. TF-IDF performs the worst, as it only uses the term-based vector space model and lexical matching between documents.

4.8. Impact of Hyperparameters

4.8.1. Impact of the Size of Sampling Neighbors. In the neighbor sampling process, to control the size of the graph, we sample a fixed-size set of neighbors for each node. We vary the number of sampled neighbors for each node in the collaborative graph N^C, description graph N^D, and tag graph N^T from 5 to 40 with a step size of 5 to find the optimal setting. The experimental results are shown in Figure 9. We only report Recall@K as NDCG@K performs in a similar fashion. We can see that the optimal size of neighbors is different for each graph. For the collaborative graph, the optimal value of N^C is relatively small ($N^C = 15$), while for the description graph, the optimal value of N^D is larger ($N^D = 30$). When the size of neighbors further increases past the optimal value, the performance begins to decrease, which verifies the necessity of node sampling before graph convolution. For the tag graph, we observe that the performance does not change much as the size of neighbors grows. We set the optimal value of N^T to 30 considering both the training efficiency and recommendation accuracy.

4.8.2. Impact of the Number of Layers in Graph Convolution. We evaluate the effectiveness of graph convolution for learning higher-order mashup and service features. We vary the number of layers L in the range of {1,2,3,4} and evaluate the performance. The experimental results are shown in Figure 10. We can see that the optimal value of $L = 3$. Increasing layers from 1 to 3 improves the performance as

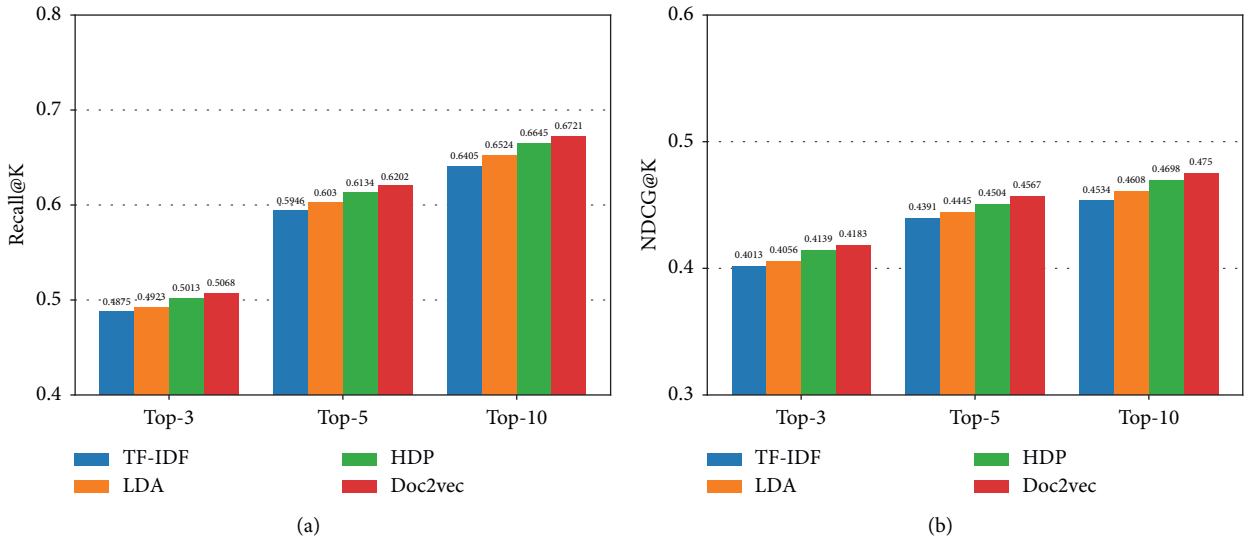


Figure 8: Comparison of different methods for description graph construction. (a) Recall@K. (b) NDCG@K.

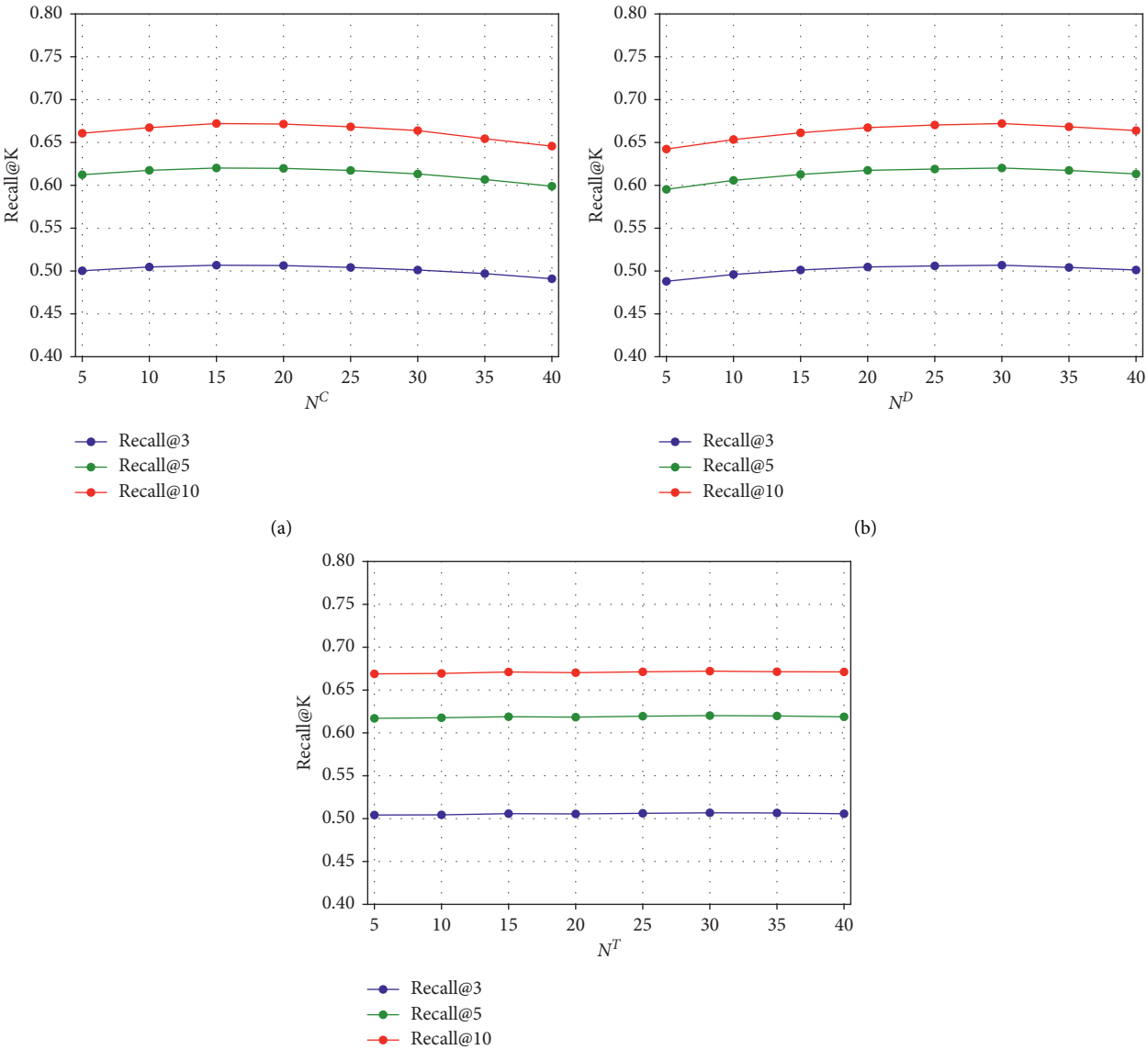


Figure 9: Impact of the size of neighbors. (a) N^C. (b) N^D. (c) N^T.

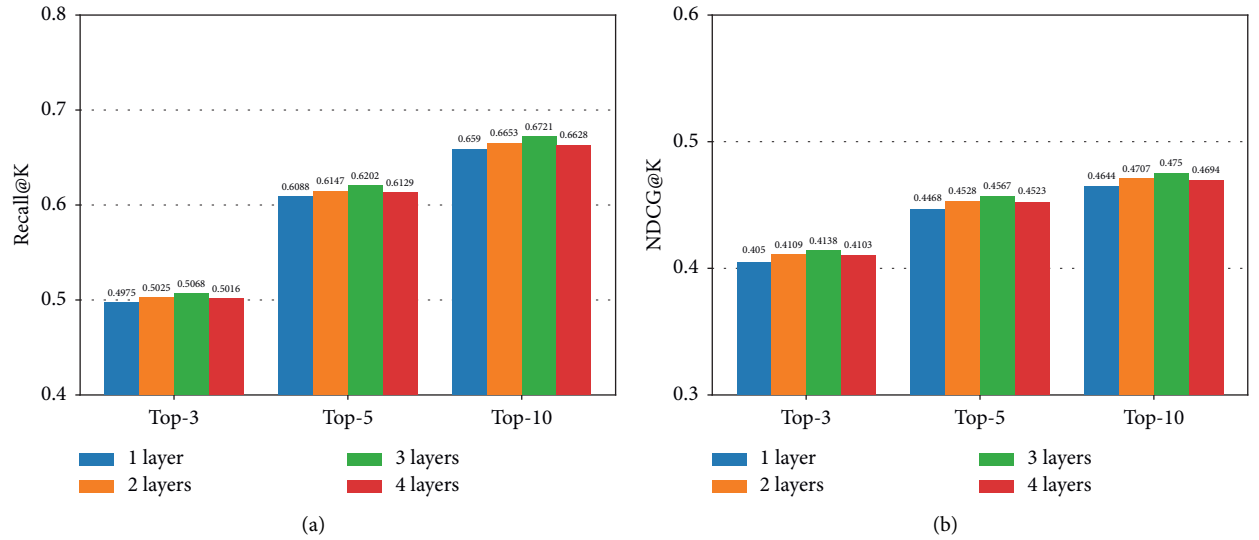


Figure 10: Impact of the number of layers in graph convolution. (a) Recall@K. (b) NDCG@K.

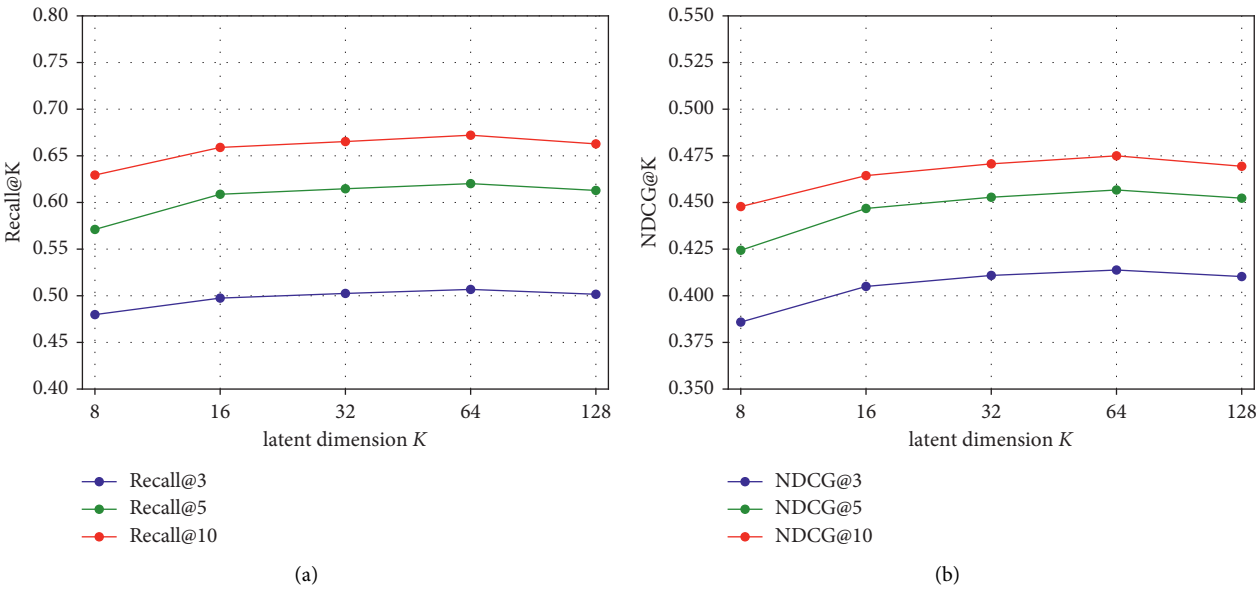


Figure 11: Impact of the latent dimension. (a) Recall@K. (b) NDCG@K.

stacking more layers helps nodes reach multihop neighbors to enrich mashup and service features. However, stacking too many layers can make the node features similar and reduce the performance.

4.8.3. Impact of the Latent Dimension. We evaluate the impact of latent dimension k on the model performance, which varies in the range of {8,16,32,64,128}. The performance is shown in Figure 11, from which we can see that the optimal value of k is 64. Setting the latent dimension too small will probably constrain the model capacity, and setting the latent dimension too large not only leads to the overfitting issue that hurts the recommendation accuracy, but also increases the model training time.

5. Conclusions

In this paper, we delve into three issues, namely, feature sparsity, imbalance, and cold-start when applying factorization machine models for service recommendation, and propose a novel MGCN-NFM model to address those issues. First, three graphs are built to represent the various types of interactions between mashups and services: a collaborative graph, a description graph, and a tag graph. Next, we use the graph convolutional network to iteratively aggregate higher-order neighbors to enrich the information of the sparsely-connected nodes in each graph. The feature embeddings are used as latent factors in the neural factorization model, where it predicts the probability of composition given a pair of mashup and service, their tags, and textual descriptions.

We perform extensive experiments on the ProgrammableWeb dataset, and the results demonstrate the superiority of our proposed method. Moreover, our model is able to incorporate other information from mashups and services by defining graphs with new relations.

In the future, we intend to investigate other potentially useful attributes and social information of mashups and services, such as QoS, developers, followers, etc. In addition, other more advanced GCN models and FM models can be explored to further improve the recommendation accuracy. Finally, as the list of recommended services should be of high coverage and diversity besides high accuracy, we plan to investigate and improve the model with respect to the two metrics.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research was partially supported by National Key R&D Program of China under grant no. 2019YFB1404802, National Natural Science Foundation of China under grants nos. 62176231 and 62106218, Zhejiang Public Welfare Technology Research Project under grant no. LGF20F020013, Wenzhou Bureau of Science and Technology of China under grant no. Y2020082.

References

[1] A. Bouguettaya, M. Singh, M. Huhns et al., "A service computing manifesto," *Communications of the ACM*, vol. 60, no. 4, pp. 64–72, 2017, https://www.programmableweb.com/.

[2] D. Benslimane, S. Dustdar, and A. Sheth, "Services mashups: the new generation of web applications," *IEEE Internet Computing*, vol. 12, no. 5, pp. 13–15, 2008.

[3] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: a research roadmap," *International Journal of Cooperative Information Systems*, vol. 17, no. 02, pp. 223–255, 2008.

[4] L. Yao, Q. Sheng, A. Ngu et al., "Unified collaborative and content-based web service recommendation," *IEEE Transactions on Services Computing*, vol. 8, no. 3, pp. 453–466, 2014.

[5] S. Rendle, "Factorization machines," in *Proceedings of the IEEE International Conference on Data Mining*, pp. 995–1000, 2010.

[6] Y. Qu, H. Cai, K. Ren et al., "Product-based neural networks for user response prediction," in *Proceedings of the IEEE International Conference on Data Mining*, pp. 1149–1154, 2016.

[7] B. Cao, J. Liu, Y. Wen, H. Li, Q. Xiao, and J. Chen, "QoS-aware service recommendation based on relational topic model and factorization machines for IoT Mashup applications," *Journal of Parallel and Distributed Computing*, vol. 132, pp. 177–189, 2019.

[8] Y. Cao, J. Liu, M. Shi et al., "Service recommendation based on attentional factorization machine," in *Proceedings of the IEEE International Conference on Services Computing*, pp. 189–196, 2019.

[9] G. Kang, J. Liu, B. Cao, and M. Cao, "Nafm: neural and attentional factorization machine for Web api recommendation," in *Proceedings of the IEEE International Conference on Web Services*, pp. 330–337, 2020.

[10] X. He and T. S. Chua, "Neural factorization machines for sparse predictive analytics," in *Proceedings of the ACM SIGIR conference on Research and Development in Information Retrieval*, pp. 355–364, 2017.

[11] B. Cao, J. Liu, M. Tang et al., "Mashup service recommendation based on user interest and social network," in *Proceedings of the IEEE International Conference on Web Services*, pp. 99–106, 2013.

[12] W. Gao, L. Chen, J. Wu, and H. Gao, "Manifold-learning based api recommendation for mashup creation," in *Proceedings of the IEEE International Conference on Web Services*, pp. 432–439, 2015.

[13] T. Liang, L. Chen, J. Wu, H. Dong, and A. Bouguettaya, "Meta-path based service recommendation in heterogeneous information networks," *Service-Oriented Computing*, in *Proceedings of the International Conference on Service-Oriented Computing*, pp. 371–386, 2016.

[14] F. Xie, L. Chen, D. Lin, Z. Zheng, and X. Lin, "Personalized service recommendation with mashup group preference in heterogeneous information network," *IEEE Access*, vol. 7, no. 7, pp. 16155–16167, 2019.

[15] X. Wang, H. Wu, and CH. Hsu, "Mashup-oriented api recommendation via random walk on knowledge graph," *IEEE Access*, vol. 2018, no. 7, pp. 7651–7662, 2018.

[16] W. Xu, J. Cao, L. Hu et al., "A social-aware service recommendation approach for mashup creation," in *Proceedings of the IEEE International Conference on Web Services*, pp. 107–114, 2013.

[17] L. Yao, X. Wang, Q. Z. Sheng et al., "Mashup recommendation by regularizing matrix factorization with API co-invocations," *IEEE Transactions on Services Computing*, vol. 14, no. 2, pp. 502–515, 2018.

[18] W. Gao, L. Chen, J. Wu, H. Dong, and A. Bouguettaya, "Personalized API recommendation via implicit preference modeling," *Service-Oriented Computing*, in *Proceedings of the International Conference on Service-Oriented Computing*, pp. 646–653, 2016.

[19] H. Li, J. Liu, B. Cao et al., "Integrating tag, topic, co-occurrence, and popularity to recommend web apis for mashup creation," in *Proceedings of the IEEE International Conference on Services Computing*, pp. 84–91, 2017.

[20] B. Cao, B. Li, J. Liu et al., *Mobile Service Recommendation via Combining Enhanced Hierarchical Dirichlet Process and Factorization Machines*, vol. 2019, , 6423805, 2019.

[21] F. Xie, L. Chen, Y. Ye et al., "Factorization machine based service recommendation on heterogeneous information networks," in *Proceedings of the IEEE International Conference on Web Services*, pp. 115–122, 2018.

[22] X. Zhang, J. Liu, B. Cao et al., "Web service recommendation via combining Doc2Vec-based functionality clustering and DeepFM-based score prediction," in *Proceedings of the IEEE International Conference on Parallel & Distributed Processing with Applications*, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications, , pp. 509–516, 2018.

[23] T. Chen, J. Liu, B. Cao et al., "Web service recommendation based on word embedding and topic model," in *Proceedings of the IEEE International Conference on Parallel & Distributed Processing with Applications*, Sustainable Computing & Communications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, , pp. 903–910, 2018.

[24] 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.

[25] Y. Zhang, H. Yang, and L. Kuang, "A web API recommendation method with composition relationship based on GCN," in *Proceedings of the IEEE International Conference on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications*, pp. 601–608, Social Computing & Networking, 2020.

[26] S. Lian and M. Tang, "API Recommendation for Mashup Creation Based on Neural Graph Collaborative Filtering," *Connection Science*, pp. 1–15, 2021.

[27] W. He, C. Xia, Z. Li, X. Liu, and T. Wang, "A heterogeneous graph attention network-based web service link prediction," in *Proceedings of the International Conference on Computer Communication and the Internet*, pp. 102–108, 2021.

[28] Z. Zheng, H. Ma, M. R. Lyu, and I. King, "QoS-aware web service recommendation by collaborative filtering," *IEEE Transactions on Services Computing*, vol. 4, no. 2, pp. 140–152, 2010.

[29] Z. Zheng, L. Xiao, M. Tang et al., *Web Service QoS Prediction via Collaborative Filtering: A Survey*, IEEE Transactions on Services Computing, 2020.

[30] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He, "DeepFM: a factorization-machine based neural network for CTR prediction," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pp. 1725–1731, 2017.

[31] J. Xiao, H. Ye, X. He et al., "Attentional factorization machines: learning the weight of feature interactions via attention networks," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pp. 3119–3125, Web API Recommendation Method with Composition Relationship Based, 2017.

[32] Z. Wu, S. Pan, F. Chen et al., "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2020.

[33] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proceedings of the International Conference on Machine Learning*, pp. 1188–1196, 2014.

[34] T. Mikolov, I. Sutskever, K. Chen et al., "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems*, pp. 3111–3119, 2013.

[35] D. Blei, A. Ng, and M. Jordan, "Latent dirichlet allocation," *The Journal of Machine Learning Research*, vol. 2003, no. 3, pp. 993–1022, 2003.

[36] Y. Teh, M. Jordan, M. Beal, and D. Blei, "Hierarchical dirichlet process," *Journal of the American Statistical Association*, vol. 101, no. 476, pp. 1566–1581, 2004.

[37] T. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proceedings of the International Conference on Learning Representations*, 2017.

[38] Q. Li, Z. Han, and X. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Thirty-Second AAAI conference on artificial intelligence*, 2018.

[39] X. He, K. Deng, X. Wang et al., "Lightgcn: simplifying and powering graph convolution network for recommendation," in *Proceedings of the International ACM SIGIR conference on research and development in Information Retrieval*, pp. 639–648, Association for Computing Machinery, New York, 2020.

[40] D. Kingma and J. BaAdam, "A method for stochastic optimization," in *Proceedings of the International Conference on Learning Representations*, 2015.