Hindawi
Scientific Programming
Volume 2021, Article ID 1013682, 9 pages
https://doi.org/10.1155/2021/1013682

Research Article

Service Composition Recommendation Method Based on Recurrent Neural Network and Naive Bayes

Ming Chen[1], **Junqiang Cheng**[2], **Guanghua Ma**[3], **Liang Tian**[4], **Xiaohong Li**[3], **and Qingmin Shi**[3]

[1]*College of Software Engineering, Zhengzhou University of Light Industry, Zhengzhou 450000, China*
[2]*Europe-Asia Hi-tech and Digital Technology Company Limited, Zhengzhou 450000, China*
[3]*Key Laboratory of Data Analysis and Financial Risk Prediction, Xinxiang University, Xinxiang 458000, China*
[4]*Institute of Computer and Information Engineering, Xinxiang University, Xinxiang 458000, China*

Correspondence should be addressed to Liang Tian; tianliang@xxu.edu.cn

Received 10 September 2021; Accepted 19 October 2021; Published 29 October 2021

Academic Editor: Tongguang Ni

Due to the lack of domain and interface knowledge, it is difficult for users to create suitable service processes according to their needs. Thus, the paper puts forward a new service composition recommendation method. The method is composed of two steps: the first step is service component recommendation based on recurrent neural network (RNN). When a user selects a service component, the RNN algorithm is exploited to recommend other matched services to the user, aiding the completion of a service composition. The second step is service composition recommendation based on Naive Bayes. When the user completes a service composition, considering the diversity of user interests, the Bayesian classifier is used to model their interests, and other service compositions that satisfy the user interests are recommended to the user. Experiments show that the proposed method can accurately recommend relevant service components and service compositions to users.

1. Introduction

With the rapid development of Web 2.0, users gradually participate in the creation of web content. However, it is becoming more and more difficult to meet users' complex needs with the single service. Thus, users begin to combine different services for the generation of their own service composition [1–3]. Service compositions refer to several services in a certain logical order to form an integrated application. For example, IFTTT (If This Then That) was used to customize smog SMS. IFTTT has started a new trend, shifting users from creating content to creating service composition. The traditional service system, however, is too complicated and has poor scalability. It is difficult for users to combine services. Therefore, the lightweight WEB-API has become the future direction of service composition, owing to its easy access, extensibility, and easy development.

The user-oriented lightweight service composition allows users to drag and drop service components on a lightweight service composition platform to generate a new service sequence. It can thus fulfill users' individual needs. Generally speaking, the platform tools of lightweight service composition can support graphical components encapsulated by third parties, such as RSS/Atom feeds, web services, and various programming APIs (Google Maps, Flickr). Users can create service compositions through a visual operation interface without programming skills. Both the industry and the academia have shown great interest in this user-oriented lightweight service composition method.

Although the service composition tools are recognized by users, they still need strategic guidelines when combining lightweight services [4]. These guidelines include initial user guidance and user interest extraction. For the initial user guidance, at the beginning phase of the service selection, when a user selects a service component, other service components that effectively associate with the selected service should also be added in the recommendation list and be recommended to the user because of their potential

unknown interests. For user interest extraction, owing to the diversity of user interests, the current user interest scenario should be modelled according to the user's selection. Other service compositions similar to the user interests should be recommended to the user.

In view of the above reasons, the paper puts forward a service composition recommendation method based on recurrent neural network and Naive Bayes. The method is divided into two stages: (1) When a user's initial interests are unknown, according to the user's selection, the method firstly recommends n service components with the highest correlation to the user by the RNN algorithm. (2) When the user completes a service composition, considering the diversity of their interests, the Bayesian classifier is used to model these interests, and other service compositions suitable for the user interests are recommended to the user. In the current research, the RNN algorithm recommends related services to users, which is likely to alleviate the problem of service mismatch. The Naive Bayes algorithm provides users with other service compositions that can satisfy their interests. It not only can meet the diversity of user interests but also can create excellent service compositions in the template with reused library. Experiments show that the proposed method is able to accurately recommend service components and service compositions to users.

2. Related Works

Previous researchers mainly utilized the topic model to obtain the latent topics for the improvement of the recommendation accuracy, for instance, the Latent Dirichlet Allocation (LDA) [5]. However the training of topic model was time-consuming. Subsequently, the matrix factorization was widely applied in service recommendations [6]. As the matrix factorization was not suitable for general prediction tasks, a general predictor method named Factorization Machines (FM) [7] was proposed. By exploiting Factorization Machines, Cao et al. [8] proposed the Self-Organization Map-Based functionality clustering algorithm and the Deep Factorization Machine-Based quality prediction algorithm to recommend API services. In addition, to solve the sparsity problem of historical interactions, Cao et al. [9] used topic models to extract the relationships between mashups and to model the latent topics. Although the above methods had generated several satisfactory results, traditional service recommendation approaches usually overlooked the dynamic nature of usage pattern. Therefore, it is suggested by Bai et al. [10] to incorporate both the textual content and the historical usage to build latent vector models for service recommendation. Meanwhile, to address the cold-start problem, Ma et al. [11] proposed learning the interaction information between candidate services and mashups based on the content and the history. Then, according to interaction vectors, a multiple layer perceptron was used to predict the rank of candidate services. Through the user's historical service access records, Gao et al. [12] utilized a PLSA-based semantic analysis model to capture the user's interests and to recommend the services that meet the user's preference.

In recent years, a few researchers have begun to pay attention to service recommendation from the perspective of Quality of Service (QoS) [9,13–19]. By focusing on the network resource consumption, Zhou et al. [13] used an integer nonlinear programming to solve microservice mashup problems, and an approximation algorithm was designed to solve the NP-hard problem. Xia et al. [14] offered to determine each service's virtual cost according to the service's attributes and the user's preference. As a result, the service composition with the least total virtual cost was recommended to users. In addition, in terms of service function, by weighing the relationship between maximizing service cousage, maximizing functional diversity, and maximizing functional matching, Almarimi et al. [20] provided the nondominated sorting genetic algorithm to extract an optimal set to create a mashup. Shin et al. [21] proposed a service composition method based on functional semantics, and Shi et al. [22] employed a Long Short-Term Memory- (LSTM-) based method with a functional attention mechanism and a contextual attention mechanism to recommend services. In terms of semantic relevance, Ge et al.[23] suggested to effectively use the existing service composition and semantic association to expand the scope of service recommendation. In the research of Duan et al. [24], the integration of the probabilistic matrix factorization (PMF) model and the probabilistic latent semantic index (PLSI) model were adopted to recommend services to users. In the present study, the PLSI model was used to train user access records. By mining historical execution trajectories, He et al. [25] discovered potential behavior patterns based on the context and user characteristics. And context-related and preference-related user activity selection probability models were established. This potentially supported the construction and the recommendation of optimized personalized mashup.

In summary, when a user clicks on the service "Flickr," combining the user has clicked the service "Facebook" before, it can thus recommend other services linked to the sequence Facebook- > Flickr. After the user finishes a service process, because of the diversity of user interests, it is necessary to recommend other service processes that are of potential interests. However, most schemes in previous studies only focus on one point, weakening the user's experience. In addition, the service component recommendation based on association rules ignores the relevance between word orders; thereby, it has a relatively low recommendation accuracy. The service composition recommendation based on QoS mainly focuses on the nonfunctional needs of users. In the Mobile Internet and the 5G era, users, however, pay more attention to their functional requirements. Therefore, this paper proposes a service composition recommendation method based on the RNN and Naive Bayes. The RNN is used to ensure the relevance between word orders. Naive Bayes is adopted to identify users' potential interests according to the component function and provides users with excellent service processes in the template library.

3. Algorithm Description

3.1. Service Component Recommendation Based on Recurrent Neural Network. In this section, service compositions are first sent to RNN for training. The training is divided into two phases: forward propagation and back propagation. Then the error losses of the output layers at different times are obtained through forward propagation. Subsequently, according to the cross entropy of error losses, weight increments $\nabla U, \nabla V, \nabla W$ are calculated through back propagation. Finally, the weights U, V, W are updated by a gradient descent method.

3.1.1. Preprocessing of Service Process Call Records. To train a suitable RNN, it is needed to preprocess the service process call records as the input data and the predefined output data. Here, for each service composition in the training set, the last service component is deleted, and other service components are inserted into the list x_data as a list element. For each service composition, the first service component is deleted, and other service components as the corresponding list element are inserted into the list y_data. For example, if two service compositions are Facebook- > Flickr- > GoogleMaps and Time- > Weather- > Text, where - > represents the linked sequence between service components, then x_data = [["Facebook", "Flickr"], ["Time", "Weather"]], y_data = [["Flickr", "GoogleMasp"], and ["Weather", "Text"]]. Here x_data_j will be used as the input data of forward propagation, and y_data_j will be used as the predefined output data of back propagation. x_data_j and y_data_j need to be converted to a one-hot vector before training. In other words, there are L words in the dictionary; if the position of a service in the dictionary is j, then the service can be represented as an L-dimensional vector, where the j^{th} dimension is set to 1, and the remaining dimensions are all set to 0.

3.1.2. Forward Propagation. The forward propagation process of RNN is shown in Figure 1. Here U represents the weight between the input layer and the hidden layer. V represents the weight between the hidden layer and the output layer. W represents the weight of the adjacent hidden layers.

At time t, x_t is the input value, and s_t is the state in the hidden layer, which is related to the input value x_t and the state s_{t-1} of the previous hidden layer. The mathematical expression is $s_t = f(Ux_t + Ws_{t-1})$. f represents an activation function in the hidden layer. In the paper, f = tanh.

$\hat{y}_t$ is the output value at time t. The mathematical expression is $\hat{y}_t = g(Vs_t)$. In the output layer, g represents an activation function. In the paper, $g()$ = softmax().

3.1.3. Back Propagation. RNN uses the back propagation to add up the error losses of the output layers at different times

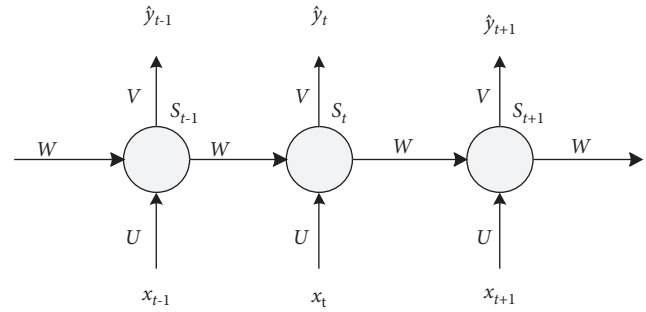

Figure 1: The forward propagation process of RNN.

to obtain the total error loss E and then calculates the gradient of each weight U, V, W to obtain weight increments $\nabla U, \nabla V, \nabla W$ and finally employs a gradient descent method to update each weight U, V, W.

(1) Error Loss Function. For each time t, there will be an error loss e_t between the output value $\hat{y}_t$ of RNN and the predefined output value y_t. Assuming that the cross entropy is used as the error loss function, there is a total error loss $E = \sum_{t=1}^{N} e_t$, and $e_t = -\sum_{i=1}^{L} y_t(i) \ln \hat{y}_t(i)$, where N represents the length of x_data_j or y_data_j, L represents the length of the one-hot vector, and $x_t \in x\_\text{data}_j$, $y_t \in y\_\text{data}_j$.

(2) Gradient Calculation. For ∇V, V does not depend on the previous states; thus, it is relatively easy to obtain. However, for $\nabla W, \nabla U$, the chain derivation rule is needed to obtain them. The calculation process is as follows:

$$\begin{cases} \nabla U = \dfrac{\partial e_t}{\partial U} = \dfrac{\partial e_t}{\partial \hat{y}_t} \cdot \dfrac{\partial \hat{y}_t}{\partial s_t} \cdot \dfrac{\partial s_t}{\partial U} + \dfrac{\partial e_{t+1}}{\partial \hat{y}_{t+1}} \cdot \dfrac{\partial \hat{y}_{t+1}}{\partial s_{t+1}} \cdot \dfrac{\partial s_{t+1}}{\partial s_t} \cdot \dfrac{\partial s_t}{\partial U} \\ = \left(\dfrac{\partial e_t}{\partial \hat{y}_t} \cdot \dfrac{\partial \hat{y}_t}{\partial s_t} + \dfrac{\partial e_{t+1}}{\partial \hat{y}_{t+1}} \cdot \dfrac{\partial \hat{y}_{t+1}}{\partial s_{t+1}} \cdot \dfrac{\partial s_{t+1}}{\partial s_t} \right) \cdot \dfrac{\partial s_t}{\partial U}, \\ \nabla V = \dfrac{\partial e_t}{\partial V} = \dfrac{\partial e_t}{\partial \hat{y}_t} \cdot \dfrac{\partial \hat{y}_t}{\partial V}, \\ \nabla W = \dfrac{\partial e_t}{\partial W} = \dfrac{\partial e_t}{\partial \hat{y}_t} \cdot \dfrac{\partial \hat{y}_t}{\partial s_t} \cdot \dfrac{\partial s_t}{\partial W} + \dfrac{\partial e_{t+1}}{\partial \hat{y}_{t+1}} \cdot \dfrac{\partial \hat{y}_{t+1}}{\partial s_{t+1}} \cdot \dfrac{\partial s_{t+1}}{\partial s_t} \cdot \dfrac{\partial s_t}{\partial W} \\ = \left(\dfrac{\partial e_t}{\partial \hat{y}_t} \cdot \dfrac{\partial \hat{y}_t}{\partial s_t} + \dfrac{\partial e_{t+1}}{\partial \hat{y}_{t+1}} \cdot \dfrac{\partial \hat{y}_{t+1}}{\partial s_{t+1}} \cdot \dfrac{\partial s_{t+1}}{\partial s_t} \right) \cdot \dfrac{\partial s_t}{\partial W}. \end{cases} \tag{1}$$

Assuming the error variation of the hidden layer $\delta_t^h = \partial e_t / \partial \hat{y}_t \cdot \partial \hat{y}_t / \partial s_t + \partial e_{t+1} / \partial \hat{y}_{t+1} \cdot \partial \hat{y}_{t+1} / \partial s_{t+1} \cdot \partial s_{t+1} / \partial s_t$ and the error variation of the output layer $\delta_t^o = \partial e_t / \partial \hat{y}_t$, $\nabla U, \nabla V, \nabla W$ can be expressed as follows:

$$\begin{cases} \nabla U = \dfrac{\partial e_t}{\partial U} = \dfrac{\partial e_t}{\partial \hat{y}_t} \cdot \dfrac{\partial \hat{y}_t}{\partial s_t} \cdot \dfrac{\partial s_t}{\partial U} + \dfrac{\partial e_{t+1}}{\partial \hat{y}_{t+1}} \cdot \dfrac{\partial \hat{y}_{t+1}}{\partial s_{t+1}} \cdot \dfrac{\partial s_{t+1}}{\partial s_t} \cdot \dfrac{\partial s_t}{\partial U} \\ = \delta_t^h \cdot \dfrac{\partial s_t}{\partial U}, \\ \nabla V = \dfrac{\partial e_t}{\partial V} \\ = \delta_t^o \cdot \dfrac{\partial \hat{y}_t}{\partial V}, \\ \nabla W = \dfrac{\partial e_t}{\partial W} = \dfrac{\partial e_t}{\partial \hat{y}_t} \cdot \dfrac{\partial \hat{y}_t}{\partial s_t} \cdot \dfrac{\partial s_t}{\partial W} + \dfrac{\partial e_{t+1}}{\partial \hat{y}_{t+1}} \cdot \dfrac{\partial \hat{y}_{t+1}}{\partial s_{t+1}} \cdot \dfrac{\partial s_{t+1}}{\partial s_t} \cdot \dfrac{\partial s_t}{\partial W} \\ = \delta_t^h \cdot \dfrac{\partial s_t}{\partial W}. \end{cases} \tag{2}$$

In RNN, the calculation of back propagation is from the back to the front. At each moment, weight increments $\nabla U, \nabla V, \nabla W$ are updated as follows:

$$\begin{cases} \Delta U = \Delta U + \delta_t^h \cdot \dfrac{\partial s_t}{\partial U}, \\ \Delta V = \Delta V + \delta_t^o \cdot \dfrac{\partial \hat{y}_t}{\partial V}, \\ \Delta W = \Delta W + \delta_t^h \cdot \dfrac{\partial s_t}{\partial W}. \end{cases} \tag{3}$$

(3) Weight Update. When the training of a service composition is completed, the RNN uses a gradient descent method to update U, V, W along the negative gradient direction. The updated process is as follows:

$$\begin{cases} U = U - lr * \Delta U, \\ V = V - lr * \Delta V, \\ W = W - lr * \Delta W. \end{cases} \tag{4}$$

Here the initial U, V, W are randomly generated. lr is the step length of the gradient descent method.

After the update of U, V, W is completed, the loop is repeated until the error loss E reaches the threshold. At this time, the weights U, V, W are used to predict the output results according to the input data.

3.1.4. Service Components Recommendation. When users select a service component, the service component is sent to the RNN. It uses the weights U, V, W obtained in Section 3.1.3 to compute the following prediction services and then sends the top n prediction services to the recommendation list and posts them to users.

3.2. Service Composition Recommendation Based on Naive Bayes. It is noted that service components selected by users need to be further reduced through the information gain, and then the Naive Bayes classifier is exploited to extract user interests based on the reduced service component set. Finally, similar service compositions are recommended to the user according to their interests. Bayesian can quickly and efficiently identify the user's interest according to several service components clicked by the user; and those with similar interest in the user template library can directly match the common components clicked by the user.

3.2.1. Information Gain. After users finish a service composition, we need to determine user interests based on this service composition. To decrease the interference of noncritical service components, the information gain algorithm is used to reduce the service component set. The gain value *IG(s)* of each service component in the service composition can be calculated. The service components are sorted by the gain value *IG(s)*, and the first n service components are regarded as the reduced service component set.

The process is as follows:

(1) The entropy of each service component SC_j in the service composition SC is calculated, which is $H(SC_j|SC)$.

(2) The entropy without this service component SC_j in the service composition SC is calculated, which is $H(SC_j|\overline{SC})$.

(3) The difference between the entropy $H(SC_j|SC)$ and the entropy $H(SC_j|\overline{SC})$ is the classification gain value of this service component, which is $IG(SC_j)$, as shown in the following formula:

$$\begin{aligned} IG(SC_j) &= H(SC_j|SC) - H(SC_j|\overline{SC}) \\ &= -\sum_{i=1}^{n} P(c_i).\ \log_2\ P(c_i) + \sum_{i=1}^{n} P(c_i|SC_j)\log_2\ P(c_i|SC_j) + \sum_{i=1}^{n} P(c_i|\overline{SC_j})\log_2\ P(c_i|\overline{SC_j}), \\ P(c_i|SC_j) &= \frac{n(SC_j|c_i)}{n(c_i)}, \\ P(c_i|\overline{SC_j}) &= \frac{n(\overline{SC_j}|c_i)}{n(c_i)}. \end{aligned} \tag{5}$$

Here $P(c_i|SC_j)$ represents the probability of the service component SC_j belonging to interest c_i. $P(c_i|\overline{SC_j})$ denotes the probability of the service component SC_j not belonging to interest c_i. $n(SC_j|c_i)$ means the number of service compositions including SC_j in interest c_i. $n(\overline{SC_j}|c_i)$ means the number of service compositions excluding SC_j in interest c_i. $n(c_i)$ is the number of service compositions in interest c_i. $P(c_i)$ represents the proportion of services compositions belonging to interest c_i in all services compositions.

(4) The service components are sorted according to the classification gain value, and the first n service components form a reduced service component set.

3.2.2. User Interest Modeling. According to the reduced service component set, the Naive Bayes classifier is exploited to determine the user interests.

The process is specified as follows:

(1) As discussed in Section 3.2.1, the probability of the reduced service component set belonging to each interest category is calculated by the Naive Bayes classifier, which is $P(c_i|SC)$. According to the Bayesian formula, $P(c_i|SC) = P(c_i|SC_1, SC_2, \ldots, SC_n) \propto P(SC_1, SC_2, \ldots, SC_n|c_i)P(c_i)$. Assuming that SC_1, SC_2,..., SC_n are independent, $P(SC_1, SC_2, ..., SC_n|c_i) = \prod_{j=1}^{n} P(SC_j|c_i)$. As shown in formula (6), SC represents the sequence of the reduced service components (SC_1,SC_2,...,SC_n).

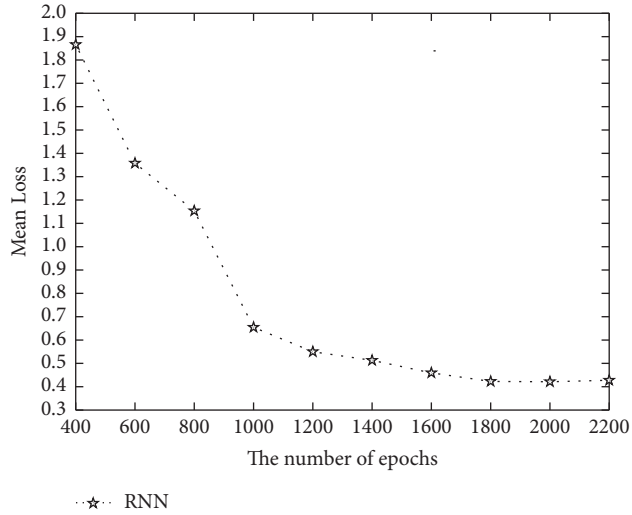

Figure 2: The number of RNN's iterations.

$$P(c_i|SC) = P(c_i|SC_1, SC_2, \ldots, SC_n) \propto P(SC_1, SC_2, \ldots, SC_n|c_i)P(c_i),$$
$$P(SC_1, SC_2, \ldots, SC_n|c_i) = \prod_{j=1}^{n} P(SC_j|c_i). \quad (6)$$

(2) According to formula (6), $P(c_i|SC) \propto P(c_i)\prod_{j=1}^{n} P(SC_j|c_i)$. This paper selects the interest category with the highest probability as the user interest; therefore, formula (7) is feasible.

$$\arg\max P(c_i|SC) \propto \arg\max\left[P(c_i)\prod_{j=1}^{n} P(SC_j|c_i)\right]. \quad (7)$$

3.2.3. Service Compositions Recommendation. According to the reduced service component set, the Naive Bayes classifier is exploited to determine the user interests.

The N-gram distance is used to compute the distance between different service compositions, and the service compositions are recommended to the user based on the similarities from high to low.

The process is specified as follows:

(1) In the service composition data set, service compositions consistent with the user interests are selected.

(2) The N-gram distance is used to compute the distance between the selected service compositions and the reduced service component set. Depending on the distance, service compositions that are most similar to the reduced service component set are recommended, as shown in the following formula:

$$\text{distance}(SC^p, SC^q) = GN(SC^p) + GN(SC^q) - 2 \times |GN(SC^p) \cap GN(SC^q)|. \quad (8)$$

(3) Here, $GN(SC^p)$ denotes the number of service components in service composition SC^p. $GN(SC^q)$ denotes the number of service components in service composition SC^q. $GN(SC_l) \cap GN(SC_p)$ is the number of the same service components in two service compositions. The similarity between two service compositions increases with the decrease in their distance.

4. Experiments

Experiments in this paper attempt to verify the effectiveness of RNN and Naive Bayes. Section 4.1 describes the data set used in Sections 3.1 and 3.2. Section 4.2 depicts the linked prediction performance of RNN, including the number of RNN's iterations, the precision, and the time comparison with the traditional algorithms (Apriori and N-gram).

Section 4.3 reports the classification performance of Naive Bayes. Section 4.4 explores the recommendation performance of N-gram distance.

4.1. Dataset. This paper uses service process call records and the service composition data set from the ProgrammableWeb website to conduct experiments. Service process call records include 20035 users' records. To improve the precision of experiments, the paper eliminates records of inactive users. In particular, users who call service processes less than 3 are regarded as inactive users; thus, there are 11730 service process call records used for our experiments. The service composition data set includes 13,082 service processes, and there are 24 types of the classification labels of service processes.

4.2. The Linked Prediction Performance of RNN

4.2.1. The Number of Iterations. The mean loss is given as follows:

$$\text{mean loss} = \frac{\sum E}{\text{the number of iterations}}. \quad (9)$$

This paper adopts the free-running mode for training. The training results are shown in Figure 2 and the mean loss is shown in formula (9). Here E represents the loss of each round iteration. It can be seen that as the number of iterations increases, the mean loss of each epoch gradually decreases. When the number of iterations reaches 2000, the convergence of the RNN algorithm is achieved.

4.2.2. Algorithm Comparison. This section compares the RNN algorithm with the traditional Apriori algorithm and the N-gram algorithm. The Apriori algorithm is a common association rule algorithm in data mining, mainly used in recommendation systems. The N-gram algorithm is also used in recommendation systems, but it can effectively reduce the recommendation space through learning the context. The comparison results demonstrate the feasibility of the RNN algorithm.

(1) Comparison of the Recommendation Precision between RNN(1), Apriori(1), and N-Gram(1). RNN(1) represents the recommendation algorithm RNN after the user calls a service component. Apriori(1) represents the recommendation algorithm Apriori after the user calls a service component. N-gram(1) represents the recommendation algorithm N-gram after the user calls a service component.

$$\text{precision} = \frac{L(\text{Rec}(sc_1 \ldots sc_i) \cap sc_{i+1})}{L(\text{Rec}(sc_1 \ldots sc_i))}. \quad (10)$$

As shown in Figure 3, the ordinate represents the recommendation precision of service components, as shown in formula (10). Here, $L(\text{Rec}(sc_1 \ldots sc_i))$ represents the number of recommended service components for the called service component sequence $sc_1 \ldots sc_i$. sc_{i+1} represents the actually linked service component for the called service component sequence $sc_1 \ldots sc_i$. $\cap$ represents the intersection.

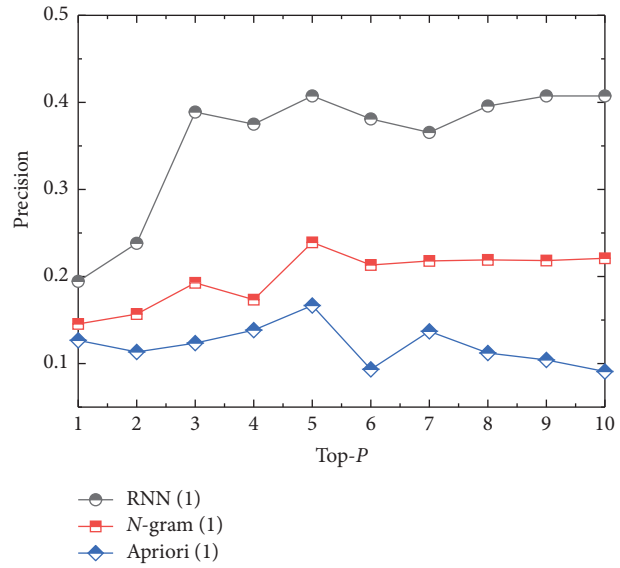

Figure 3: Comparison of the recommendation precision between RNN(1), Apriori(1), and (N)-gram(1).

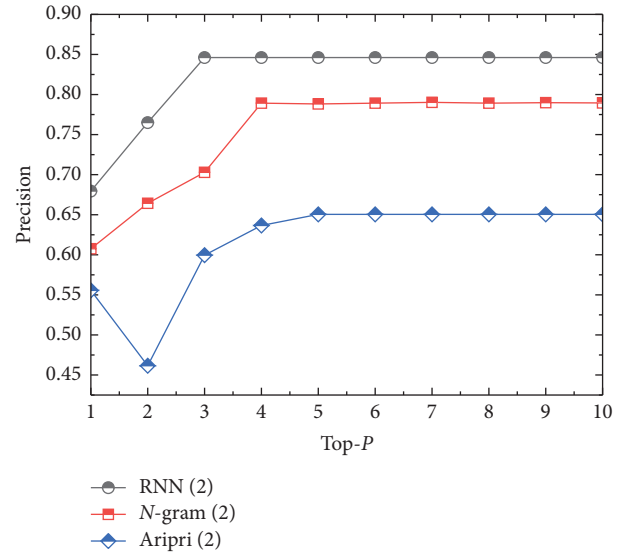

Figure 4: Comparison of the recommendation precision between RNN(2), Apriori(2), and (N)-gram(2).

$L(\text{Rec}(sc_1 \ldots sc_i) \cap sc_{i+1})$ equals 0 or 1. The abscissa Top-P represents the number of service components required to be recommended. In practice, due to the control of the predefined threshold T, $L(\text{Rec}(sc_1 \ldots sc_i)) \leq \text{Top} - P$, where $T = 0.42$. As can be seen, the precision of RNN(1) is superior to those of Apriori(1) and N-gram(1). When the Top-P is 5, RNN(1) presents the best performance. At this time, the precision of RNN(1) is 0.41, higher than 0.17 of Apriori(1) and 0.24 of N-gram(1). This is because the RNN and the N-gram can learn the linked relationships between service components through training, while Apriori can only learn the correlations between service components and cannot capture the linked order between service components. Meanwhile, due to the limitation of the Markov model, the RNN has shown superior context learning effects than the N-gram.

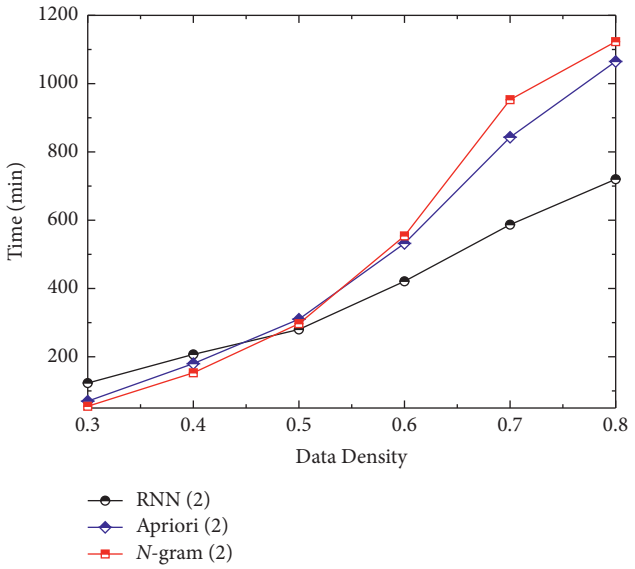

Figure 5: Comparison of the training time between RNN(2), Apriori(2), and (N)-gram(2).

(2) Comparison of the Recommendation Precision between RNN(2), Apriori(2), and (N)-Gram(2). As shown in Figure 4, RNN(2) represents the recommendation algorithm RNN after the user calls two components. Apriori(2) represents the recommendation algorithm Apriori after the user calls two components. N-gram(2) represents the recommendation algorithm N-gram after the user calls two components. When the user's initial selection of service components is greater than 1, the precision of RNN(2) is still superior to those of Apriori(2) and N-gram(2). When the Top-P is 5, the recommendation precision of RNN(2) is 0.85. The recommendation precision of Apriori(2) is 0.65. The recommendation precision of N-gram(2) is 0.79. At this time, the recommendation precisions of RNN(2), Apriori(2), and N-gram(2) are higher than those of RNN(1), Apriori(1), and N-gram(1). This is mainly because when the user selects more initial component sequences, there are fewer subsequently related service components, and the recommendation precision becomes higher.

(3) Comparison of Training Time. As shown in Figure 5, when the training data is small, the training times of the Apriori(2) algorithm and the N-gram(2) algorithm are lesser than that of the RNN(2) algorithm. But as the training data increases, the amount of data processed by the Apriori(2) algorithm and the N-gram(2) algorithm will increase exponentially. The training time of RNN(2) is lesser than those of Apriori(2) and N-gram(2). When the data density is 80%, it costs RNN(2) 720 minutes to train, while Apriori(2) takes 1065 minutes and N-gram(2) takes 1123 minutes.

4.3. The Classification Performance of Naive Bayes. As shown in Figure 6, precision% (classfication) refers to the precision of classification prediction through the service composition data set. The predicted label is compared with the real label, and finally the classification precision of the algorithm is obtained. As can be seen, with the increase in the training data, the recommendation precision becomes higher. When the density of training data is 80% and that of the test data is 20%, the classification precision of Naive Bayes reaches 89.1%.

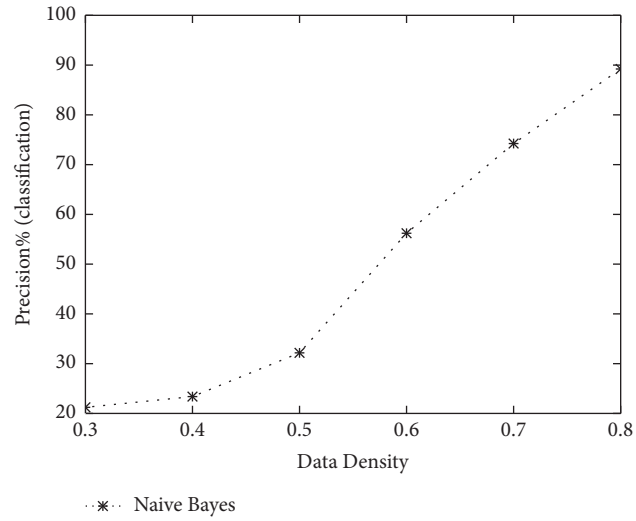

Figure 6: The classification performance of Naive Bayes.

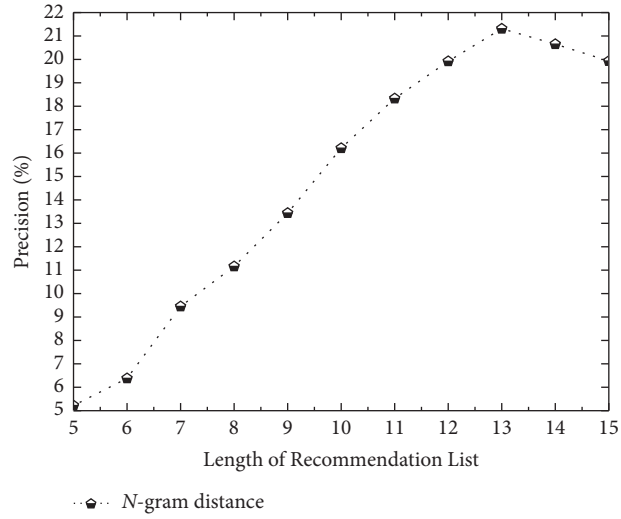

Figure 7: The recommendation performance of (N)-gram distance.

4.4. The Recommendation Performance of N-Gram Distance. Figure 7 analyzes the recommendation performance of N-gram distance. As the length of the recommendation list increases, the recommendation precision first increases and then decreases. When the length is 13, the recommendation performance is the optimal. At this time, the recommendation precision is 21.3%.

5. Conclusions

In order to optimize the assistance to users in their decision-making, this paper proposes a service composition recommendation method based on the RNN and Naive Bayes. This method has the following contributions:

(1) Different from traditional algorithms, this paper uses the context learning to reduce the recommendation space and provides users with more accurate service-linked components.

(2) To fulfill the diversity of user interests, this paper adopts the interest modeling to recommend other service processes that meet users' current interests. This can effectively promote the reuse of the template library.

It is yet worth noting that the interest modeling of Naive Bayes does not take the semantic similarity into consideration. As a result, future research would consider using the semantic analysis to model user interests.

Data Availability

The data included in this paper are available without any restriction.

Conflicts of Interest

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

Acknowledgments

This research was funded by the National Natural Science Foundation of China, Grant nos. 61975187 and 61902021, Science and Technology Planning Program of Henan Province, Grant nos. 212102210104 and 162102210214, and Philosophy and Social Science Planning Program of Henan Province, Grant no. 2019BJJ053.

References

[1] J. Sienel, A. L. Martín, C. B. Zorita, L.-W. Goix, A. M. Reol, and B. C. Martinez, "OPUCE: a telco-driven service mash-up approach," *Bell Labs Technical Journal*, vol. 14, no. 1, pp. 203–218, 2009.

[2] C. Macas, A. Rodrigues, G. Bernardes, and P. Machado, "Mixmash: a visualisation system for musical mashup creation," in *Proceedings of the Information Visualisation Conference. Fisciano*, pp. 471–477, IEEE, Fisciano, Italy, July 2018.

[3] L. Sabatucci, S. Lopes, and M. Cossentino, "Musa 2.0: a distributed and scalable middleware for user-driven service adaptation, intelligent interactive multimedia systems and services 2017," in *Proceedings of the International Conference on Intelligent Interactive Multimedia Systems and Services*, pp. 492–501, Springer, Cham, Vilamoura, Portugal, June 2018.

[4] G. Di Lorenzo, H. Hacid, H.-y. Paik, and B. Benatallah, "Data integration in mashups," *ACM Sigmod Record*, vol. 38, no. 1, pp. 59–66, 2009.

[5] L. Chen, Y. Wang, Q. Yu, Z. Zheng, and J. Wu, "WT-LDA: user tagging augmented LDA for web service clustering," in *Proceedings of the 11th International Conference on Service-Oriented Computing*, pp. 162–176, Springer, Berlin, Heidelberg, Berlin, Germany, December 2013.

[6] L. Yao, X. Z. Wang, Q. Z. Sheng, and W. J. Ruan, "Service recommendation for mashup composition with implicit correlation regularization," in *Proceedings of the IEEE 22nd International Conference on Web Services*, pp. 217–224, IEEE, New York, NY, USA, July 2015.

[7] S. Rendle, "Factorization machines," in *Proceedings of the 10th IEEE International Conference on Data Mining*, pp. 995–1000, IEEE, Sydney, Australia, December 2010.

[8] B. Q. Cao, Q. X. Xiao, X. P. Zhang, and J. X. Liu, "An API service recommendation method via combining self-organization map-based functionality clustering and deep factorization machine-based quality prediction," *Chinese Journal of Computers*, vol. 6, no. 42, pp. 1367–1383, 2019.

[9] 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, no. 132, pp. 177–189, 2019.

[10] B. Bai, Y. Fan, W. Tan, J. Zhang, K. Huang, and J. Bi, "End-to-end web service recommendations by extending collaborative topic regression," *International Journal of Web Services Research*, vol. 15, no. 1, pp. 89–112, 2018.

[11] 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, no. 1, pp. 105–119, 2021.

[12] T. Gao, B. Cheng, J. Chen, and M. Chen, "Enhancing collaborative filtering via topic model integrated uniform Euclidean distance," *China Communications*, vol. 14, no. 11, pp. 48–58, 2017.

[13] A. Zhou, S. G. Wang, S. H. Wan, and L. Y. L. M. M. Qi, "Latency-aware micro-service mashup in mobile edge computing environment," *Neural Computing and Applications*, vol. 32, no. 32, pp. 15411–15425, 2020.

[14] Y. Xia and Z. Huang, "A strategy-proof auction mechanism for service composition based on user preferences," *Frontiers of Information Technology & Electronic Engineering*, vol. 22, no. 2, pp. 185–201, 2021.

[15] S. Wang, A. Zhou, R. Bao, W. Chou, and S. S. Yau, "Towards green service composition approach in the cloud," *IEEE Transactions on Services Computing*, vol. 14, no. 4, pp. 1238–1250, 2021.

[16] Y. Xia, Z. Q. Huang, Y. L. Zhang, M. Yuan, S. G. Wang, and Y. Zhou, "SPASC: strategy-proof auction mechanism with cost and QoS incentive for service composition," *Concurrency and Computation: Practice and Experience*, vol. 33, no. 9, 2021.

[17] Z. H. Chang, D. Ding, and Y. H. Xia, "A graph-based QoS prediction approach for web service recommendation," *Applied Intelligence*, vol. 51, no. 4, pp. 6728–6742, 2021.

[18] P. Sahu, S. Raghavan, and K. Chandrasekaran, "Ensemble deep neural network based quality of service prediction for cloud service recommendation," *Neurocomputing*, vol. 465, no. 20, pp. 476–489, 2021.

[19] V. L. Hallappanavar and M. N. Birje, "Prediction of quality of service of fog nodes for service recommendation in fog computing based on trustworthiness of users," *Journal of Reliable Intelligent Environments*, vol. 7, 2021.

[20] N. Almarimi, A. Ouni, S. Bouktif, and M. W. Mkaouer, "Web service API recommendation for automated mashup creation using multi-objective evolutionary search," *Applied Soft Computing*, vol. 85, pp. 1–13, 2019.

[21] D. H. Shin, K. H. Lee, and F. Ishikawa, "A graph-based approach enhancing correctness and speed of web services composition through explicit specification of functional semantics," *International Journal of Web and Grid Services*, vol. 10, no. 4, pp. 297–318, 2014.

[22] M. Shi, Y. Tang, and J. Liu, "Functional and contextual attention-based LSTM for service recommendation in mashup creation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 5, pp. 1077–1090, 2019.

[23] J. X. Ge, Z. H. Liu, and B. Xu, "Novel composable service recommendation based on mashup and service semantics," *Journal of Chinese Computer Systems*, vol. 11, no. 36, pp. 2434–2438, 2015.

[24] L. Duan, T. L. Gao, W. Ni, and W. Wang, "A hybrid intelligent service recommendation by latent semantics and explicit ratings," *International Journal of Intelligent Systems*, vol. 36, pp. 1–28, 2021.

[25] W. He, L. Z. Cui, G. Z. Ren, Q. Z. Li, and T. Li, "A new paradigm for personalized mashup recommendation based on dynamic contexts in mobile computing environments," *Scientia Sinica Information*, vol. 46, no. 6, pp. 677–697, 2016.