Expert Systems With Applications 196 (2022) 116574

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Open APIs recommendation with an ensemble-based multi-feature model

Junwu Chen, Ye Wang, Qiao Huang *, Bo Jiang, Pengxiang Liu

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

ARTICLE INFO

Keywords:
Open APIs
APIs recommendation
Neural networks
Machine learning
Ensemble model

ABSTRACT

Recommending the appropriate APIs from a large volume of Open APIs to application developers both accurately and efficiently has become a challenging problem. Established work usually takes only one feature of Open APIs into account, which decreases the accuracy of recommendations. In order to overcome this problem, we propose an ensemble-based approach to Open APIs recommendation with a multiple feature model, which integrates both machine learning and deep learning and synthesizes a set of multiple features to accurately make the recommendation. This approach employs One-hot, Word2vec similarity and Matrix Factorization techniques to obtain the multiple features information respectively, then concatenates the features to obtain a Multivariate Information Feature (MIF) matrix, and leverages an optimized Gradient Boosting Decision Tree (GBDT) and Gated Recurrent Unit (GRU) for feature selection. GBDT is good at processing dense numerical features, while GRU is good at processing sparse categorical features, Finally, the results are synthesized to obtain a recommendation result. We have compared our approach with other four Open APIs recommendation approaches on the Programmable Web, and verified the effectiveness of our approach in precision, recall, F1-measure.

1. Introduction

Modern enterprises need fast, accurate, and efficient optimization of their business processes to shorten development cycles and costs. Open APIs, also known as Web APIs or services, which are the basic development units of web-based applications, usually contain only one single function and are published on the Internet. Such architecture (Wang, Li, & Zhao, 2018) transforms software development from a traditional "product-centric" model to a "consumer-centric" model, and encourages users to create their own applications through Open APIs (Sumithra & Sarath, 2015), supporting content reuse, collaborative sharing, and user content compilation principles (Liu, Hui, Sun, & Liang, 2007; Majchrzak & More, 2011). Mashup mixes together two or more of Open APIs, forming an integrated application, has emerged as a novel development paradigm at present. Mashup has two-fold meanings: (1) Mashup is a temporary combination technology of Web applications, which allows users to create entirely new Mashups using Open APIs; (2) Mashup is an composite application developed via composing Open APIs. Due to the explosive growth of Open APIs published on the Internet, Mashup developers usually cannot obtain appropriate Open APIs. Therefore, a suitable Open APIs recommendation approach is needed to overcome the above problems and maintain the ecology of Open APIs on the Internet.

Open APIs recommendation approaches are recommending the Open APIs that Mashup developers (i.e., users) really need by mining their interest preferences and behavior characteristics. A typical scenario for Open APIs recommendation in Mashup development as follows: After the Open APIs are developed and registered to the Programmable Web (PWeb), the search requests for Mashup development will submit search requests to the PWeb. The request can be one sentence or several sentences writing in natural language. For example, "A realtime map of Northern Lights locations and photos. See the current aurora forecast and latest Flickr and Twitter posts related to the Aurora Borealis". The recommendation approach will obtain the best match between the request and the related Open API(s), (i.e., the most related Open APIs for the above request are Google Maps and Filckr API.)

There are some established researches that aim to improve the effectiveness of Open APIs recommendation, which can be generally divided into two classes: the functionality-based recommendation approaches (Aznag, Quafafou, & Jarir, 2014; Chen, Wang, Yu, Zheng, & Wu, 2013) and the QoS (Quality of Service)-based recommendation approaches (Qi, Hu, Song, Ge, & Lü, 2015; Shao et al., 2007). Various machine learning (e.g., topic models Aznag et al., 2014, collaborative filtering Zheng, Ma, Lyu, & King, 2011, matrix factorization Lo,

* Corresponding author.
E-mail addresses: cjwnuzx@163.com (J. Chen), yewang@zjgsu.edu.cn (Y. Wang), qiaohuang@zjgsu.edu.cn (Q. Huang), nancybjiang@zjgsu.edu.cn (B. Jiang), 404871666@qq.com (P. Liu).

https://doi.org/10.1016/j.eswa.2022.116574
Received 18 February 2021; Received in revised form 15 November 2021; Accepted 17 January 2022
Available online 8 February 2022

Yin, Li, & Wu, 2015), and deep learning techniques (e.g., Factorization Machine Cao et al., 2019; Cao, Xiao, Zhang and Liu, 2019) are used in these approaches. However, these approaches still have the following problems: (1) The implicit semantic information of natural-language(NL)-based API descriptions cannot be well modeled due to the lack of sufficient training corpus; (2) The correlation between various QoSs of Open APIs is usually ignored in the above approaches; (3) The existing Open APIs recommendation approaches fail in updating the information in time; (4) There are both sparse categorical data and dense continuous numerical data as important features for Open APIs. For example, the similarity between Mashups and similarity between Open APIs can be transformed to dense continuous numerical data whereas the Open APIs ID and the Mashups ID can be transformed to sparse categorical data.

In order to address the above issues, in this paper, we propose a novel approach — recommending Open APIs based on an ensemble-based multiple feature model. This approach takes the textual description of the Open APIs and Mashups as input (including both functionalities and QoSs) to construct a Multivariate Information Feature (MIF) matrix, and then enters this matrix into an optimized Gradient Boosting Decision Tree (GBDT Ji, Yang, & Zhou, 2018) and Gated Recurrent Unit (GRU Tan et al., 2020) for feature selection respectively. Because GBDT is good at handling continuous numerical features, while the neural network GRU is good at handling sparse categorical features. By synthesizing GBDT and GRU results, we obtain a final recommendation score for each Open API, and recommend the Open APIs with highest scores to Mashup developers.

The main contribution of our work is as follows:

- We propose a new approach which takes the advantages of both GBDT and GRU, leading to its capability of dealing with both dense numerical data and sparse categorical data.
- We compare our approach with other four common recommendation approaches based on the real data set of the Programmable Web, and the results show that our approach performs better than other approaches. The comprehensive effect of our method is the best, and the best result is in the case of TOP_3. The precision, recall and F1-measure are 3.8%, 6% and 4.1% higher than the best baseline approach respectively.

The rest of the paper is structured as follows. Section 2 introduces the related work of Open APIs recommendation approaches. Section 3 presents our approach. Section 4 compares our approach with other four approaches. Section 5 shows the experiment results. Section 6 draws the threats to validity. Section 7 shows the future research directions and Section 8 shows the conclusion.

2. Literature review

This section presents the related work of intelligent Open APIs recommendation approaches. this paper uses the Open API as a unified description for Web API and Web Service.

2.1. Functionality-based Open APIs recommendation

The functionality-based Open APIs recommendation approaches mainly calculate text similarity through Open APIs text content, and make recommendations through similarity ranking.

The topic model can model topics on the Open API description documents, obtain the topic distribution vector of the descriptions, and use the trained topic distribution to explore the implicit relationships between Open APIs and Mashups. There is a large amount of the recommending work related to the topic model. Yao, Sheng, Segev, and Yu (2013) proposed a latent variable model to explore potential connections between Web APIs through a common call pattern between Web APIs. Chen et al. (2013) used enhanced LDA topic model seamlessly integrates user-contributed tagging data and WSDL description documents to improve the efficiency of the Open API search engine. Aznag et al. (2014) proposed a method based on non-logical Correlated Topic Model (CTM). CTM can extract the topic of Open API semantic description, and model the correlations between extracted topics to improve the accuracy of Open API recommendation. Cao et al. (2016) proposed a two-level topic model based on Web API content and network to improve the effectiveness of Web API clustering and achieve better recommendation results. Rodriguez-Mier, Pedrinaci, Lama, and Mucientes (2016) proposed a graph-based Open APIs composition approach, which calculates text similarity of a set of semantically relevant I/O request APIs. Zhang, Wang, and Ma (2020) proposed a method for mining domain knowledge of Web API goals from API description documents. This method consists of two parts: (1) extracting the API target from the API textual description based on linguistic analysis; (2) merging the domain API target with semantically similar API targets in the domain to construct the request.

2.2. QoS-based Open APIs recommendation

Some Open APIs are similar in the functionalities, but differs from each other in terms of qualities. Only considering functionality cannot make accurate Open APIs recommendation. QoS-based Open APIs recommendation is more about taking a set of qualities of Open APIs into consideration, such as historical calling rules, response time, geographical location of visits, and so on. QoS-based Open APIs recommendation is generally divided into two types according to their techniques, namely collaborative filtering (CF) and matrix factorization (MF). CF is based on the idea that infers the current user preferences based on the information of other correlated users and Open APIs, whereas MF is based on the idea that mines the potential factors that affect QoSs via the historical user information.

2.2.1. CF-based Open APIs recommendation

Collaborative filtering (CF) recommendation algorithm is a well-known recommendation algorithm. The main function is prediction and recommendation.

Shao et al. (2007) proposed a user-based CF approach for predicting QoS values from data between similar service users. Zheng et al. (2011) suggested using historical data of users to predict QoS values, which uses the user-collaborative mechanism to collect the historical QoS information of different users, and then perform CF based on the collected data to obtain the QoS value. Kuang, Xia, and Mao (2012) proposed a context-aware Open APIs recommendation approach based on current users' historical APIs invocation experience. Hu, Peng, and Hu (2014) proposed a time-aware Open APIs recommendation method, which adds temporal information to the similarity calculation, and uses a hybrid personalized random walk algorithm to reduce the sparseness of the data. Cao et al. (2020) proposed a two-level topic model based on the relationship between Mashup services to explore potentially useful and novel topics. Then, based on Mashup clustering results, a Web APIs recommendation algorithm based on collaborative filtering (CF) is designed.

2.2.2. MF-based Open APIs recommendation

MF is a widely used technique in recommendation systems. It can overcome the problem of CF-based recommendation. Generally, MF first constructs a user information matrix with the existing QoS data, and then constructs the QoS record matrix to calculate the predicted value.

Qi et al. (2015) used MF to integrate neighbors with target user information and Open API neighborhood information to predict QoS values. Lo et al. (2015) proposed a Local Neighborhood Matrix Factorization (LONMF) model of the two-level selection mechanism, which can identify a set of highly relevant local neighbors for target users. It identifies highly relevant neighbors for current users, and then uses MF to make personalized QoS predictions through the integration of

geographic locations. Su, Ma, Xiao, and Zhang (2016) developed a hybrid QoS prediction method by combining CF and MF. First, a non-negative matrix factorization model was used to predict the QoS of Open APIs, secondly, then optimize the learning of the model based on expectation maximization (EM) to obtain a better QoS prediction value. Wu, Yue, Li, Zhang, and Hsu (2018) proposed a context sensitive matrix factorization (CSMF) method for QoS prediction. CSMF makes full use of the implicit and explicit context factors in QoS information by modeling users-to-services and environment-to-environment. Yin, Chen, Xu, and Wan (2018) proposed an Open APIs recommendation approach based on probability matrix factorization (PMF), which predicts QoS by merging network locations and the capabilities of users and the associated Open APIs. Nguyen, Yu, Bai, Yongchareon, and Han (2020) proposed an Attentional PMF model, which leverages a neural attention network to learn the significance of feature interactions and uses Doc2Vec technique for mining the contextual information. Yao, Wang, Sheng, Benatallah, and Huang (2021) proposed a probabilistic matrix factorization approach with implicit correlation regularization to solve the recommendation problem and enhance the recommendation diversity. MF-based recommendation can map a high dimensional matrix to the product of two low dimensional matrices. Yet, the interpretability of MF is poor, and the dimensions in its hidden space cannot correspond to concepts in reality, which decreases the training speed.

2.3. A summary of related work and motivation

The advantages and disadvantages of the aforementioned approaches are analyzed and summarized as follows: (1) The functionality-based Open APIs recommendation approaches utilize the Open API description to calculate the semantic similarity of functionalities. Yet, such approaches rely heavily on the textual description of the Open APIs, which ignores other important information of Open APIs, such as the correlations between different Open APIs; (2) The CF-based recommendation approaches only collect explicit information (e.g., the number of times a Mashup developer invoke an Open API), without exploring implicit information (e.g., the developer click behaviors and search behaviors) to make more accurate predictions; (3) The MF-based recommendation approaches can map a high-dimensional matrix to the product of two low-dimensional matrices. Yet, the interpretability of MF is quite poor, and the dimensions in its hidden space cannot correspond to concepts in reality, which decreases the training speed.

By employing the advantages of related work, our approach takes both functionalities and QoSs of the Open APIs into account. The functionalities are treated by natural language processing techniques while the QoSs are treated by calculating the co-occurrence and popularity. Moreover, our approach not only considers the explicit information (such as Open API textual description and Mashup textual description) but also considers the implicit information (such as Open API popularity). In addition, the dense data and sparse data are respectively processed by different learning models in our approach.

3. The proposed approach

In this section, we present the details of our approach and its related concepts.

3.1. The concept

- **Open API** is a remote programming interface published online that transfers data and composite Mashups, denoted as O.
- **Mashup** refers to the composite applications developed using Open APIs, denoted as M.
- **Co-occurrence** refers to the correlation between two Open APIs. When two Open APIs are invoked by the same Mashup, the co-occurrence value increases.
- **Popularity** refers to the degree to which an Open API is invoked by Mashups.

The above concepts are illustrated by formulas in Section 3.3.

3.2. The framework of the approach

Fig. 1 shows the overall framework of the approach in this paper. First, the textual description of Open APIs and Mashups as well as the Mashup information which invokes the Open APIs are processed to obtain a MIF (Multivariate Information Feature) matrix (see Section 3.3). Then the MIF matrix will be entered into GBDTLR (the integrated model of GBDT and Logistics Regression (LR Wen, Hsu, Wang, & Wu, 2012)), which is good at processing dense numerical data (see Section 3.4). Then the MIF matrix will be entered into MLPGRU (the integrated model of Multilayer Perceptron (MLP Yeung, Sun, & Zeng, 2001) and GRU), which is good at processing sparse categorical data(see Section 3.5). Finally, we synthesize these two models by linear combination to calculate final recommendation scores of Open APIs, and rank the scores for recommendation (see Section 3.6).

3.3. Constructing the MIF matrix

The original Open API information data has the problem of sparseness and clutter. Only one feature cannot represent the comprehensive information. By referring to Cao, Liu et al. (2019) and Yao et al. (2013), we have integrated a set of features correlated to Open APIs, including the Open API IDs, Mashup IDs, similarity of Open APIs, similarity of Mashups, co-occurrence of Open APIs, popularity and matrix factorization of Open APIs. We analyze all features and then construct the MIF matrix through the following steps: (1) transforming Open API IDs and Mashup IDs into one-hot vectors that can be recognized by machine learning and neural networks; (2) using Word2vec (Ray, Garain, & Sarkar, 2021) to vectorize the textual description of Open APIs and Mashups, and then calculate the similarity; (3) calculating the co-occurrence of the Open APIs; (4) calculating the popularity of the Open APIs; (5) calculating the matrix factorization of the Open APIs; (6) combining all the above features to form a MIF matrix as a whole. Each feature is described in Table 1 and will be introduced in details in the following subsections.

3.3.1. Representing Open APIs and Mashups using one-hot

As the classifier is difficult to process sparse categorical data, the continuous value IDs cannot be directly calculated and need to be converted into a one-hot vector for subsequent operations. For example, the ID set of Open APIs is $O = \{O_1, O_2, \sim, O_n\}$, where n represents the number of all Open APIs in the library, $O_i = [0, 0, \sim, 1, 0, \sim 0]_{1*n}$, and O_i represents the ith elements, and the length of O_i is also n. Similarly, the ID set of Mashups is $M = \{M_1, M_2, \sim, M_m\}$, where m represents the number of all Mashups in the library, $M_j = [0, 0, \sim, 1, 0, \sim 0]_{1*m}$, and M_j represents the jth elements, the length of M_j is also m.

3.3.2. Calculating similarity between Open APIs and the similarity between Mashups

Open APIs or Mashups with analogous textual description has similar functionality. So we adopt Word2Vec and cosine similarity to calculate the similarity between Open APIs and the similarity between Mashups. Word2Vec turns words into word vectors by Google on the Google News Corpus (Mikolov, Chen, Corrado, & Dean, 2013). This word vector can map the corresponding word to 300 dimensions with a total size of 11 GB. We averaged the corresponding word vectors of each word in each sentence, so as to get the sentence vector of each sentence. Then we calculate the corresponding similarity using the cosine similarity. The specific formula is as follows:

$$\mathrm{SenVec} = \frac{\sum_{i=1}^{n} \mathrm{word2vec}(w_i)}{n} \tag{1}$$

$$\mathrm{Sim} = \frac{\mathrm{SenVec}_1 \times \mathrm{SenVec}_2}{\sqrt{\mathrm{SenVec}_1} \times \sqrt{\mathrm{SenVec}_2}} \tag{2}$$

where *word2vec* (w_i) represents using Word2vec to vectorize a word, n represents that the number of words a sentence has. The cosine similarity of the two words is between 0 and 1.

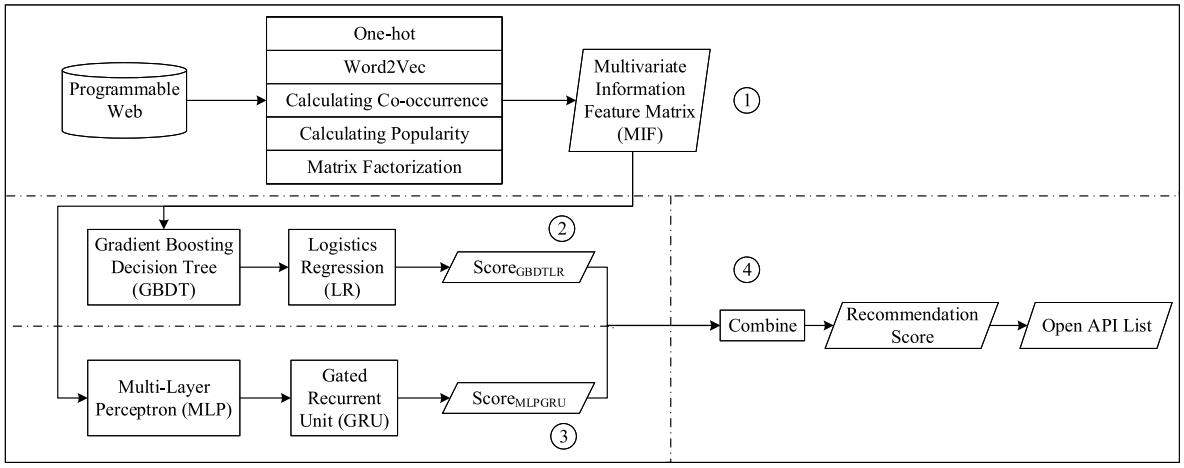

Fig. 1. The framework of the approach.

Table 1
The features of the MIF matrix.

Feature of the MIF matrix	Data Source	Technique
One-hot of Open APIs	Open API IDs	One-hot
One-hot of Mashups	Mashup IDs	One-hot
Similar Open APIs	Open APIs textual description	Word2Vec
Similar Mashups	Mashups textual description	Word2Vec
Open API Co-occurrence	Number of times Open APIs invoked by Mashups	Co-occurrence
Open API popularity	Number of times Open APIs invoked by Mashups	Popularity of TOP_K similar Open APIs
Open API factorization machine	Number of times Open APIs invoked by Mashups	Factorization machine and Mashup categories

3.3.3. Calculating the co-occurrence of Open APIs

Co-occurrence is a manifestation of the external combination relationship of Open APIs, which can indicate the degree of correlations between two Open APIs. The formula for calculating the co-occurrence is as below:

$$\mathrm{Co}\left(O_i, O_j\right) = \frac{\left|O_i \cap O_j\right|}{\left|O_i \cup O_j\right|} \tag{3}$$

where $\left|O_i \cap O_j\right|$ represents the times that the Open API O_i and the Open API O_j are invoked by the same Mashup, $\left|O_i \cup O_j\right|$ represents the total times that the Open API O_i and the Open API O_j are invoked by different Mashups and the final result is between 0 and 1.

3.3.4. Calculating the popularity of Open APIs

The more times an Open API is invoked, the more popular the Open API is. In a set of similar Open APIs, an Open API is popular if it is frequently invoked. The popularity is calculated as below:

$$\mathrm{Pop}\left(O_i\right) = \frac{\mathrm{Num}\left(O_i\right) - \mathrm{Min\,Num}\left(\mathrm{Sim}_k\left(O_i\right)\right)}{\mathrm{Max\,Num}\left(\mathrm{Sim}_k\left(O_i\right)\right) - \mathrm{MinNum}\left(\mathrm{Sim}_k\left(O_i\right)\right)} \tag{4}$$

where $Num(O_i)$ represents the times an Open API O_i has been invoked by all Mashups. $Sim_k(O_i)$ represents the TOP_K Open APIs that are most similar to the Open API O_i, and the similarity is obtained from Section 3.3.2. *MinNum(.)* represents the minimum times the Open API is invoked by all Mashups, *MaxNum(.)* represents the maximum times, and the final result is between 0 and 1.

3.3.5. Calculating the matrix factorization of Open APIs

The number of times an Open API is invoked by a certain category of Mashups also reflect the degree of Mashups' preference for the Open API. However, some Open APIs may not have been invoked by a certain category of Mashups, and the times that some Open APIs have been invoked may not be counted completely. Therefore, we first use the number of existing Mashups invoke the Open APIs as a basic data, and then use matrix factorization (MF) to fill the data. The calculation is as follows:

$$r_{n\times m} \approx p_{n\times k} \times q_{k\times m} \tag{5}$$

In the process of matrix factorization, the original scoring matrix $r_{n\times m}$ is decomposed into the product of two matrices $p_{n\times k}$ and $q_{k\times m}$. The matrix $p_{n\times k}$ represents the relationship matrix between n Open APIs and k features. Each feature is an intermediate variable, and the transpose of the matrix $q_{k\times m}$ is the matrix $q_{m\times k}$, which represents the relation matrix between m Mshups and k features. In order to obtain an appropriate k, we choose the best value of k is using the cross-validation method.

Let $\hat{r}_{ij} = p_i q_j = \sum_{k=1}^{k} p_{ik} q_{kj}$, we consider the square of the error between the original scoring matrix $r_{n\times m}$ and the reconstructed scoring matrix $\hat{r}_{n\times m}$ as the loss function. In order to make the result gain better generalization ability, we add a regular term to the loss function to constrain the parameter, adding the loss function of L2 regularization is:

$$e_{ij}^2 = \left(r_{ij} - \sum_{k=1}^{k} p_{ik} q_{kj}\right)^2 + \frac{\beta}{2} \sum_{k=1}^{k} \left(p_{ik}^2 + q_{kj}^2\right) \tag{6}$$

Then we continuously update the p and q components according to the gradient descent until the algorithm finally converges.

The matrix factorization can get the scoring situation of a certain category of Mashups for Open APIs that have not yet been invoked, and thus it can fill in incomplete data.

The resultant MIF matrix is shown in Fig. 2. The active Open API represents the Open APIs to be recommended, the target Mashup represents the recommended target object, and the similar Open API represents the text similarity between the current active Open API and other Open APIs. The similar Mashup means the text similarity between the current target Mashup and other Mashups. The co-occurrence means the co-occurrence between Open APIs, and the Matrix Factorization represents the matrix factorization scores between Open APIs. The popularity represents the popularity of Open APIs. The score indicates whether the target Mashup invokes active Open APIs.

MIF																								Y
Active Open API				Target Mashup				Similar Open API				Similar Mashup				Co-occurrence				Matrix Factorization				Popularity	Score
1	0	0	⋯	1	0	0	⋯	0	0.6	0.5	⋯	0	0.3	0.7	⋯	0	0.5	0.6	⋯	0.6	0.3	0.4	⋯	12	1
0	1	0	⋯	1	0	0	⋯	0.4	0	0.7	⋯	0	0.3	0.7	⋯	0.3	0	0.9	⋯	0.3	0.5	0.5	⋯	3	0
0	0	1	⋯	0	0	1	⋯	0.5	0.8	0	⋯	0.3	0.4	0	⋯	0.6	0.8	0	⋯	0.2	0.6	0.5	⋯	5	1
1	0	0	⋯	0	0	1	⋯	0	0.6	0.3	⋯	0.3	0.4	0	⋯	0	0	0.4	⋯	0.5	0.2	0.3	⋯	7	0
0	1	0	⋯	0	1	0	⋯	0.6	0	0.7	⋯	0.5	0	0.6	⋯	0.4	0.5	0.8	⋯	0.4	0.7	0.6	⋯	1	1
0	0	1	⋯	0	1	0	⋯	0.5	0.7	0	⋯	0.5	0	0.6	⋯	0.5	0.6	0	⋯	0.6	0.2	0.4	⋯	8	1

Fig. 2. The MIF matrix.

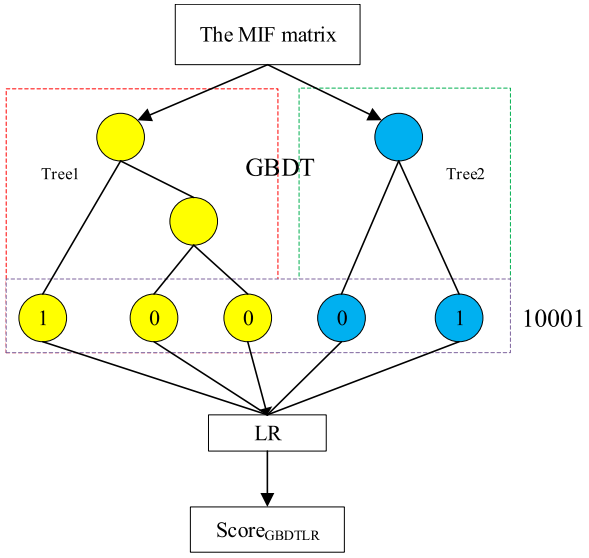

Fig. 3. The GBDTLR framework.

3.4. Machining learning with GBDTLR

Gradient Boosting Decision Tree (GBDT) is an iterative decision tree algorithm composed of multiple decision trees. First, the residual of the prediction results of the previous $k-1$ tree is used to reduce the residual of the training process. Also, there are other feature selection algorithms, such as the nature-inspired feature selection algorithms Grey Wolf Optimizer (GWO) (Mirjalili, Mirjalili, & Lewis, 2014). We tested through 200 pieces of data. We find that the performance of the two feature selection algorithms is similar, but GWO used about 2.42 s, while GBDT only ran 0.17 s. From the test, the GBDT algorithm is more suitable for the Open APIs and Mashups data set. Then iteratively construct a strong classifier from the weak classifier LR. The LR algorithm is simple, efficient, easy to parallel, and dynamically scalable, thus is widely used in machine learning. GBDT is good at processing dense features, but not good at sparse features, while LR can better make the results of GBDT output to fall in between 0 and 1.

Because the original data is too sparse, in this work, GBDT is used to automatically combine the original training data into new training data. LR is used as a classifier for the new training input data and performs weight learning on features extracted by GBDT. The details are shown in Fig. 3.

At beginning, the MIF matrix obtained in Section 3.3 is the input into the GBDT. The position of the leaf node to which the maximum probability value calculated by each tree in the GBDT belongs is recorded as 1. The idea is similar to one-hot. Each weak classifier (here we use decision tree) has only one leaf node to output the prediction result, so in a GBDT has n weak classifiers and a total of m leaf nodes. However, after GBDT extraction of the original data, the data will become sparse, and due to the large number of weak classifiers and the number of leaf nodes, the new training data may be too large, so we use the L1 regularization to reduce the risk of overfitting.

After constructing new training data using GBDT, we input the newly received training data with the original data into the LR classifier to train the final classifier, and obtain the current Mashup's score on an Open API. The final performance results of GBDT are as follows:

$$F_M(x) = F_0 + \beta_1 T_1(x) + \beta_2 T_2(x) + \cdots + \beta_M T_M(x) \tag{7}$$

where β_i is the weight, is the initial value, F_0 is the initial value, and T_i represents the decision tree constructed in the ith iteration, $i = 1, 2, \ldots, M$.

Then we use the LR formula to calculate the final probability score, and the final result is:

$$\text{Score}_{GBDTLR} = \frac{1}{1 + e^{-\theta^T F_M(x)}} \tag{8}$$

where is the parameter vector. The value of $Score_{GBDTLR}$ ranges from 0 to 1.

3.5. Deep learning with MLPGRU

Neural networks can handle sparse data well and can update data online in time, so we adopt neural networks to perform the same training on the original data. Due to the particularity of the original data (e.g., the Open API and Mashup IDs are represented as one-hot vectors that are sparse data and the processed textual description data is called dense data), we divide the original data into sparse data and dense data, and then perform an embedding lookup operation on the sparse data and a Multi-Layer Perceptron (MLP) operation on the dense data. The MLP consists of three layers: the input layer, the hidden layer, and the output layer. The layers are fully connected, that is, all neurons on the previous layer are connected to those on the next layer. Then the features are interactively connected and input to the Gated Recurrent Unit (GRU). GRU is a variant of LSTM (Long-Short Term Memory Lepori, Linzen, & McCoy, 2020) with a simpler structure and better performance in general. LSTM is Long Short-Term Memory, which is a time-ordered recurrent neural network. It uses the forget gate, input gate, and output gate to complete the feature selection. Unlike LSTM, GRU combines the forget gate and input gate into a single update gate, and passes through the activation function (turns linear features into nonlinear features) to get the final recommendation scores. In order to compare the performance of GRU and LSTM, we selected 500 pieces of data for testing. We ran 30 iterations and the results showed that LSTM needed 93 s with an accuracy of 0.908, while GRU only needed 89 s and the accuracy was reduced to 0.901. We found that the performance of GRU and LSTM was similar, but GRU ran faster than LSTM. We believe that with the increase of data volume, GRU can save more time and improve the efficiency. The MLPGRU specific process is shown in Fig. 4.

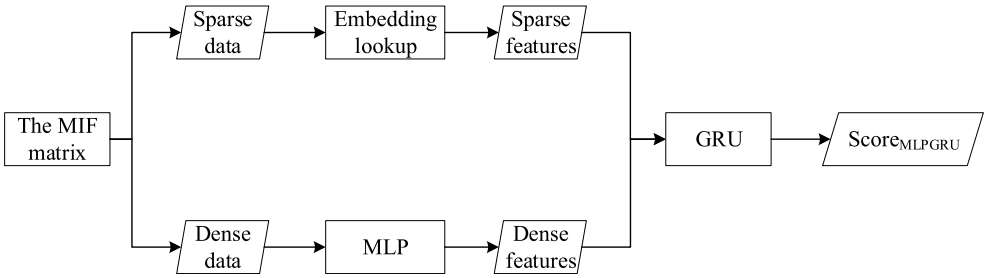

Fig. 4. The MLPGRU flowchart.

The feature information in MIF is divided into two categories, one is sparse features such as Open API and Mashup IDs after one-hot processing, and the other is dense features after the computation of similarity, co-occurrence, popularity and matrix factorization. For sparse features, we use the embedding lookup layer to transform the feature vector with dimension n. We use the embedding lookup for Open API IDs and Mashup IDs respectively. The specific formula is as follows:

$$w_{o_i}^T = e_{o_i}^T W \tag{9}$$

$$w_{m_i}^T = e_{m_i}^T W \tag{10}$$

where $e_{o_i}^T$ represents the transposition of the one-hot vector of the ith id, where the ith position is 1, and the other positions are 0, The index i represents the ith id, corresponding row vector of the embedding table $W \in R^{m \times d}$, where m represents the number of IDs, and d represents the dimension of the one-hot ID.

For dense features, they are first connected into a feature vector, and then input into the MLP to be converted into the same n dimensional feature vector. MLP is a Multi-Layer Perceptron neural network consisting of a fully connected layer and a staggered connection of activation functions.

$$w_{d_i} = MLP\left(\left[\mathrm{sim}_{o_i}; \mathrm{sim}_{m_i}; \mathrm{co}_i; mf_i; pop_i\right]\right) \tag{11}$$

$$MLP = W_k \sigma\left(W_{k-1}\sigma\left(\ldots\sigma\left(W_1 w_{d_i} + b_1\right)\ldots\right) + b_{k-1}\right) + b_k \tag{12}$$

where sim_{o_i} represents the similarity of the Open APIs in the ith row, sim_{m_i} represents the similarity of the Mashups in the ith row, co_i represents the co-occurrence of row i, pop_i represents the popularity of the ith row, weight matrix $W_i \in R^{nl \times nl-1}$, bias $b_l \in R^{nl}$ for layer $l = 1, \ldots, k$.

The embedding lookup features of Open APIs and Mashups are used to multiply the dot product of the corresponding elements, and then they are connected to the features that passed MLP to fuse all the features together.

$$F_i = concat(w_{d_i}, w_{o_i} \cdot w_{m_i}) \tag{13}$$

where w_{d_i} represents dense features, w_{o_i}, w_{m_i} represent the sparse feature of Open APIs and Mashups, respectively.

The fusion features obtained in the previous step are entered into a two-layer GRU. The GRU selects key information in each feature and then uses the $Sigmoid$ activation function to output the corresponding results and the final score. The final result of MLPGRU is:

$$\mathrm{Score}_{MLPGRU_i} = \mathrm{Sigmoid}\left(GRU\left(F_i\right)\right) \tag{14}$$

$$\mathrm{Sigmoid}(x) = \frac{1}{1+e^{-x}} \tag{15}$$

where F_i represents the fused feature and $GRU(.)$ represents the features of GRU output.

3.6. The synthesis of GBDTLR and MLPGRU

The above machine learning score: Score_{GBDTLR} and deep learning score: Score_{MLPGRU} are aggregated in order to obtain the final Open APIs recommendation score. For each pair of a Mashup and an Open API, the final recommendation score is defined as follows:

$$\mathrm{Score} = \alpha \times \mathrm{Score}_{GBDTLR} + \beta \times \mathrm{Score}_{MLPGRU} \tag{16}$$

In the above formula, α and β represent the weights of GBDTLR and MLPGRU, respectively. We set $\alpha + \beta = 1$. The final score range is 0 to 1. The Open APIs with the highest score corresponding to a certain Mashup will be recommended to the Mashup developer for their choice.

3.7. Running examples

Table 2 shows the GBDTLR score, MLPGRU score and final score between an open API and a Mashup. Open API represents the currently recommended Open API, and Mashup represents the currently object. Score_{GBDTLR} represents the recommended score obtained by GBDTLR, and Score_{MLPGRU} represents the score obtained by MLPGRU. Through a set of experiments, we found that our approach has the best effect when the weight of GBDTLR score is setting to 0.8 and the weight of MLPGRU score is setting to 0.2.

4. Experimental setup

In this section, we describe the experimental setup that we follow to evaluate the approach. The experimental environment is python language, with NVIDIA RTX2070 GPU, running Ubuntu 16.04 LTS (64-bit).

4.1. Data collection

PWeb (Programmable Web) Corpus. We crawled the official data dump of PWeb (Maximilien & Ranabahu, 2007), which was an Open API registration platform. Open APIs on PWeb involve various domains. Data used in this work were all crawled from the PWeb platform. A total of 15,936 Open APIs were crawled. After text preprocessing and removal of 283 invalid Open APIs with no specific function description meaning, which contained meaningless Open APIs textual description and Open APIs with empty texts, as well as Open APIs with empty information sets extracted. The remaining number of Open APIs is 15,653. In the same way, 6973 Mashups and 13,613 link relationships between Open APIs and Mashups were obtained.

We need to pre-process the data through the following steps: (1) remove unreasonable special characters, including “&”,“%”,“$”,“#” and so on; (2) convert all letters to lowercase letters; (3) remove stopwords and check the correctness of all words. We split the data into the 80% training set and the 20% test set. In order to make the data more balanced, the training data were negative sampled.

Table 2
Running examples.

Open API	Mashup	$Score_{GBDTLR}$	$Score_{MLPGRU}$	Score
Google maps	Auroramap	0.4136	0.4516	0.4212
Youtube	Austin city limits music festival radio	0.06	0.4648	0.1409
Google base	Auto sector snap igoogle	0.0192	0.4132	0.098
Twitter	Backchannel us	0.16	0.4478	0.2176
Twitter streaming	Azoomed product reviews and news	0.0938	0.396	0.1543
Ebay	Avi geeks	0.569	0.3752	0.5302

4.2. Baseline approaches

We compare our approach to other four baseline approaches as listed below:

- Baseline 1 (**TF-IDF**): TF-IDF (Paik & Mizugai, 2010) is a traditional recommendation system technique. The general idea of TF-IDF is to map the textual descriptions of Open APIs and Mashups to word vectors. We use the cosine similarity to calculate the similarity between the word vector of the Open API and the word vector of Mashup textual description. Finally, the recommended score of the Open API is a combination of the score of TF-IDF and the popularity of the ith Open API.
- Baseline 2 (**LDA-CF**): LDA (Xie, Dong, & Gao, 2014) is a famous document topic generative model, which obtains the topic distribution vector of a document. The similarity between a Mashup and an Open API can be calculated by the cosine similarity, then the similarity is filled into a similarity matrix. The final similarity score was obtained using item collaborative filtering (CF) algorithm. Finally, the recommended score of the Open API is a combination of the score of LDA-CF and the popularity of the ith Open API.
- Baseline 3 (**LDA-MF**): LDA-MF adopts matrix factorization (MF) (Yin et al., 2018) to calculate the similarity, and multiply the similarity with the popularity of an Open API to get the final recommendation score. Finally, the recommended score of Open API is a combination of the score of LDA-MF and the popularity of the ith Open API.
- Baseline 4 (**MIF-FM**): FM (Factorization Machine) can learn cross features (Cao, Liu et al., 2019), decompose parameters to simulate and predict the correlation between a Mashup and an Open API. The input data of FM is the same MIF as that used in our approach. The similarity score is calculated by FM. Then we multiply the similarity with the popularity of the Open API to get the final recommendation score. Finally, the recommended score of Open API is a combination of the score of MIF-FM and the popularity of the ith Open API.

4.3. Experimental metrics

This paper uses precision, recall and F1-measure (Jiang, Zhang, Ren, & Zhang, 2017; Xiong, Wu, Li, Gu, & Hang, 2016) to evaluate experimental results, which are classical evaluation metrics for recommendation systems.

The precision indicates the proportion of the correctly recommended Open APIs to the total number of recommendations. The recall indicates the proportion of the correctly recommended Open APIs among all related Open APIs. F1-measure is the weighted harmonic average of precision and recall.

Fig. 5 shows the percentage of Mashups invoking the Open API on PWeb. From Fig. 5, we can see that 0.83% of Mashups do not invoke any Open APIs. 52.97%, 23.9%, 9.8%, 4.94%, and 2.93% of Mashup only invoked 1/2/3/4/5 Open API(s), which means that most Mashups invoke only five or less Open APIs, Therefore, our experiment is to recommend TOP _1/2/3/4/5 Open APIs.

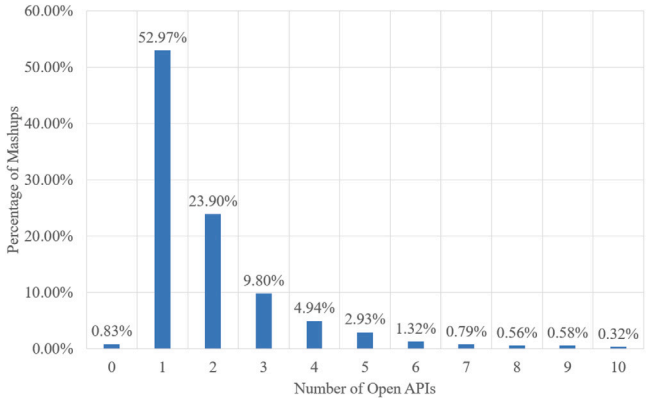

Fig. 5. The number of Mashups invoking Open APIs on PWeb.

4.4. Parameter configuration

We need to set several parameters for the experiment. For matrix factorization, we set the intermediate variable $k = 4$. For GBDT, we set the number of weak classifiers to 10. For GRU, we set the weight and bias of the Embedding lookup to meet the normal distribution, which makes the results converge faster. Similarly, the weight and bias of the MLP satisfies a normal distribution. The dropout was used before the fully connected layer and its keep rate was 0.8. The initial learning rate of Adam optimizer is 0.00001. With the increase of training times, the learning rate decreased linearly. The loss function is a binary cross entropy loss function, and calculated as below:

$$\mathrm{Loss}(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}) \tag{17}$$

where y is the true label value and $\hat{y}$ is the probability value obtained by GRU.

Our approach synthesizes the recommendation results of machine learning and deep learning. In order to obtain the best effect in our training set, we have made some experiments to select the best value of the weights θ and β. And it is verified on the test set. The results are shown in Fig. 6. It can be seen from the figures that when $\theta = 0.8$ and $\beta = 0.2$, the effect of the model is the best.

5. Experimental results

In this section, we analyze the obtained results.

5.1. RQ1: How effective is our approach? how much improvement can it achieve over the baseline approaches?

Motivation. The main purpose of our approach is to automatically recommend appropriate Open APIs for Mashup developers. Therefore, in order to evaluate the usability of this approach, we need to look at the metrics in API recommendation and compare it with existing API recommendation techniques.

Approach. To answer this research question, we named our approach as ROAMFM (Recommending Open APIs with a Multiple Feature Model) and compare ROAMFM with four kinds of baselines on the

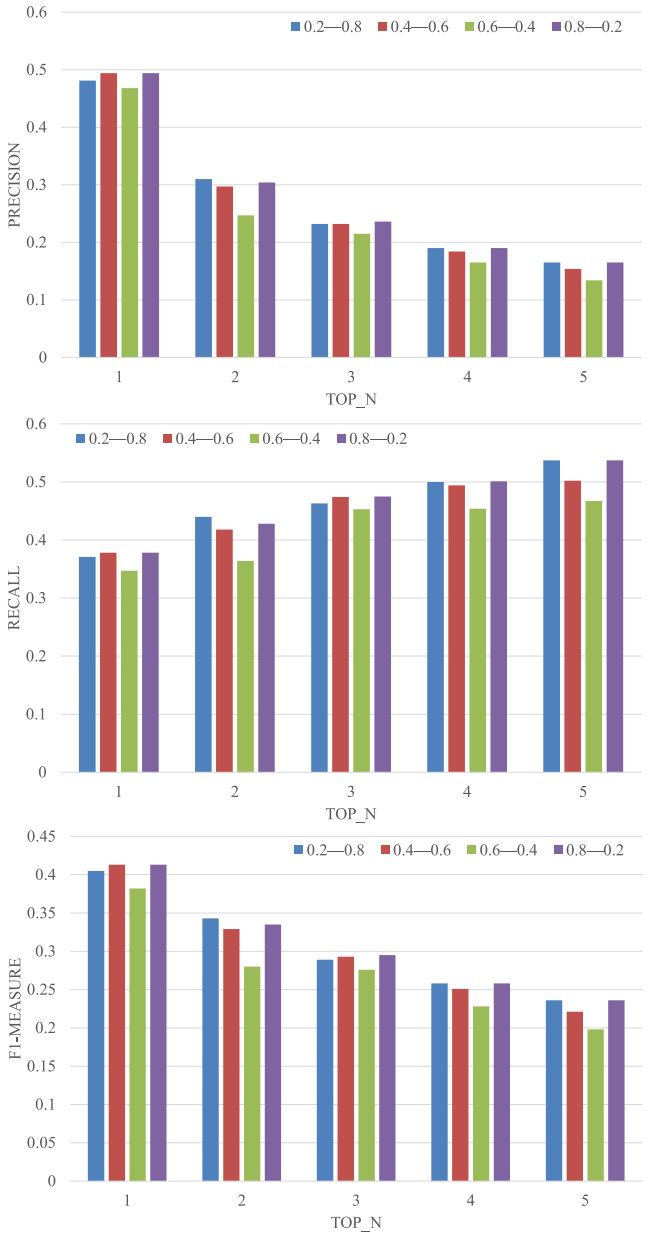

Fig. 6. Precision, Recall and F1-measure of different parameters.

Fig. 7. Precision, Recall and F1-measure of different methods.

real data set of PWeb. Specifically, first, we use the methods of One-Hot, Word2Vec, Co-occurrence, Population and Matrix Factorization to process the real data of PWeb to obtain the MIF matrix. Second we input MIF into GBDTLR and MLPGRU respectively, and finally combine the scores of the two learning models. Next we implement TF-IDF, LDA-CF, LDA-MF, MIF-FM by referring to the published source (Cao, Liu et al., 2019; Paik & Mizugai, 2010; Xie et al., 2014; Yin et al., 2018) respectively. Then we evaluate these techniques in terms of precision, recall and F1-measure.

Results. Fig. 7 shows the comparative results of precision, recall and F1-measure of ROAMFM with the other baselines respectively. We can see from the figures, that on TOP_1/2/3/4/5, ROAMFM is superior to other baselines in most cases. The following observations are made: (1) TF-IDF performs the worst; (2) LDA-CF is better than TF-IDF, but its effect is still inferior to LDA-MF, MIF-FM and ROAMFM; (3) The effect of LDA-MF is slightly better than TF-IDF and LDA-CF; (4) MIF-FM performs better than TF-IDF, LDA-MF and LDA-CF; (5) ROAMFM shows the most satisfying recommendation results. The comprehensive effect of our method is the best in the case of TOP_3. Its precision is 3.8% higher than the best baseline approach, its recall is 6% higher than the best baseline approach, its F1-measure is 4.1% higher than the best baseline approach.

5.2. RQ2: Why our approach outperforms other baseline approaches?

Motivation. We aim to investigate the reason why our approach outperforms other baseline approaches.

Approach. We evaluate the results above and analyze the characteristics of each approach one by one.

Results. The performance of TF-IDF method is the worst. Because this method only measures the similarity between Open APIs and Mashups from the word vector space model, and cannot capture the semantic information of words in a specific context, it relies heavily on corpus, and usually some low-quality corpus will affect its results. For example, in our experiment, there are two Open APIs. One API description is: “use the video search API to search the web for videos”, and the other API is described as “from their site: Photobucket is the most popular site on the internet for uploading, sharing, linking and finding photos, videos and graphics. Now you can add Photobucket

features to your application by using our new web service API". Both of them highlight video search topics, and the latter contains richer semantic information. However, in our experiment, the latter is not in the list of TOP_5 recommended by the TFIDF method.

LDA-CF and LDA-MF introduce the LDA topic model, their performance is improved compared with TF-IDF method. LDA-CF uses collaborative filtering (CF) algorithm to enhance the LDA model, to dig deeper relationship between the Open APIs and Mashups. However, LDA-CF only considers the explicit relationship between documents, and cannot make full use of the implicit information. For example, for "preciseNews" Mashup, which is described as "breaking news and colorful commentary every few minutes, 24-7", we recommend 5 Open APIs for it. One of them is called the "moreover" API, which is described as: "the search engine toolkit empowers publishers and B2B portals, vertical search engines and professional networking sites with targeted, real time content from around the Web. Access to more than 30,000 global news, industry and consumer sources through a single API can deliver fully configurable content to match any industry, profession, and interest". Its description is more obviously related to the description of "preciseNews" Mashup, which leads to its high ranking. As for another "springwidgets" API, it is described as "widget platform from Fox Interactive Media (FIM). From their site: TheSpringBox is a widget engine that allows you to run flash based widget on both the desktop and the web". This API is very related to "preciseNews" Mashup, but it is not on the recommended list due to its hidden dependencies.

LDA-MF can train the predetermined factor model by observing the explicit data. LDA-MF can also back-propagate through the loss function, which improves the precision. However, the interpretability of MF method is very poor, and the dimensions in the hidden space cannot correspond to the concepts in reality. And in the actual recommendation scenarios, only the accuracy of TOP_N results is often concerned, so it is obviously not accurate to investigate the global mean square error. And it can only handle second-order information, but cannot handle higher-order information well.

ROAMFM utilizes MIF to synthesize various features and processes dense data (e.g., the Open APIs and Mashups textual description) and sparse data (e.g., the Open API and Mashup IDs) separately by combining GBDT and GRU. What is more, the method overcomes the problem of high-order processing of FM and better learns the implicit information between features.

6. Threats to validity

Threats to internal validity relates to the parameter selection and implementation of ROAMFM and other four approaches. We freeze all parameters after training the model to ensure that the evaluation effect remains unchanged on the same test set. When implementing ROAMFM and the other four approaches, we first design pseudo-codes of different algorithms based on their references, and then find experts in the field of artificial intelligence to review the pseudo-codes. According to the pseudo codes, the learning models of the five methods are implemented by pair programming. Every code line is developed by the first author, and reviewed by the second author and the third author. All of them have rich programming experiences. The purpose of this is to ensure that the method implementation is correct.

Threats to external validity relates to the efficiency errors associated with the selection of experimental data. Although we analyzed PWeb service sharing communities which has more than 6K Mashups and more than 15K Open APIs, we cannot guarantee that our results can be generalized to other industry environments due to the differences in data set sizes and application domains among different data sources. There might be some efficiency errors in the evaluation results of different data sets. Therefore, to address the above issues, we plan to conduct new experiments using another famous data set on the MDN-Web-Docs platform and apply our approach in real industrial environments in order to evaluate our recommendation approach from a developer perspective.

Threats to construct validity relates to the suitability of our evaluation measures. We use precision, recall, and F1-measure, which are classical evaluation measures for information retrieval and are also widely used in previous studies (Jiang et al., 2017; Xiong et al., 2016) in Open API recommendation.

7. Future research directions

In this section, we will discuss some implications of our approach for expert systems. Four future research directions are proposed for discussion as below:

(1) **The improvement of the learning models.** Although the precision, recall and F1-measure of ROAMFM outperform the traditional approaches, these values have not achieved a satisfying level. Thus, the improvement of the learning models is still challenging for future work. There are two ways to the improvement: (a) The integration of other machine learning and deep learning techniques, such as BERT (Devlin, Chang, Lee, & Toutanova, 2019) and XLNet (Yang et al., 2019), and (b) the integration of other nature-inspired feature extraction algorithms such as Mayfly algorithms (Zervoudakis & Tsafarakis, 2020) to further improve the accuracy of Open API recommendations and reduce the time consumption.

(2) **The application of other expert system domains.** The recommendation process of ROAMFM can be applied to other expert system domains for NL-based decision making. For example, the supplier selection application is quite important for the supply chain. ROAMFM can facilitate the decision-making techniques by integrating the recommendation algorithm.

(3) **Open source of the intelligent recommendation algorithms.** Despite of various approaches proposed for Open APIs recommendation, the open source of such approaches are quite slow, which is not good for the development of the community. Opening source of the well-established approaches and tools provides a promising way for the collection of tool usage feedbacks.

(4) **The exploitation of transfer learning.** Some AI-based expert systems suffer from the problem of the small number of data set. One possible way is to exploit the domain similarity between Open APIs recommendation and other NL-based knowledge recommendation and utilize the data set of ROAMFM.

8. Conclusions

Nowadays, Mashup has emerged as a new software development paradigm. The major challenge of Mashup development is automatically and rapidly recommending Open APIs for each Mashup-to-be. This paper proposes a novel approach that integrates multiple features for effective Open APIs recommendation. This approach first extracts multiple feature information from the original recommendation data, and then synthesizes the machine learning techniques (GBDT and LR) and deep learning techniques (MLP and GRU) to get the final recommendation score of each Open API. Compared to other established recommendation approaches, our approach significantly improves the precision, recall, and F1-measure score.

Our approach (i.e., ROAMFM) has two impacts for expert systems. First, with the surging Internet applications, traditional development paradigms are no longer suitable for rapid and agile expert systems development. The Mashup developing paradigm offers one possible solution to this problem. Developers can rapidly develop an expert Mashup application by assembling a set of intelligent Open APIs, instead of developing the system from scratch. Our approach can be employed to recommend suitable Open APIs to expert Mashup applications. Second, our approach indicates the feasibility of combining the MIF matrix with both machine learning and deep learning models in NL-based knowledge recommendations (Binkhonain & Zhao, 2019). It

is recommended that our approach can be applied to other domain of expert systems such as supplier selection (Chai, Liu, & Ngai, 2013) and medical expert systems.

However, our approach still has some limitations. For example, despite of such a large number of Open APIs, only a small percentage of them are used by Mashups. Therefore, the Matthew effect (Yan, Ahmad, & Yang, 2013) appears, i.e., the frequently used Open APIs are recommended with higher priority and will be used more frequently, whereas the less frequently used Open APIs are recommended with lower priority and will be used less frequently. Another limitation of our approach is that the quality of Mashups' or Open APIs' description seriously affects the recommendation result.

To tackle the above two problems, in the future we plan to explore the effect of the invoking frequency on the recommendation accuracy by extending the MIF matrix. Besides, we plan to restrict the format of the natural language-based description of Mashups and Open APIs to improve their quality. In addition, we will explore the application of our approach into other domain of expert systems as well as other machine learning and deep learning techniques such as BERT (Devlin et al., 2019) and XLNet (Yang et al., 2019).

CRediT authorship contribution statement

Junwu Chen: Methodology, Software, Validation, Writing – original draft. **Ye Wang:** Conceptualization, Resources, Supervision, Writing – review & editing. **Qiao Huang:** Supervision, Writing – review & editing. **Bo Jiang:** Project administration, Funding acquisition. **Pengxiang Liu:** Conceptualization, Data curation, Methodology, Software.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported by Natural Science Foundation of Zhejiang Province, China (No. LY21F020011).

References

Aznag, M., Quafafou, M., & Jarir, Z. (2014). Leveraging formal concept analysis with topic correlation for service clustering and discovery. In *Proceedings of the 2014 IEEE international conference on web services*.

Binkhonain, M., & Zhao, L. (2019). A review of machine learning algorithms for identification and classification of non-functional requirements. *Expert Systems with Applications: X*, *1*, Article 100001. http://dx.doi.org/10.1016/j.eswax.2019.100001, URL: https://www.sciencedirect.com/science/article/pii/S2590188519300010.

Cao, B., Liu, X., Li, B., Liu, J., Tang, M., Zhang, M., et al. (2016). Mashup service clustering based on an integration of service content and network via exploiting a two-level topic model. In *Proceedings of the 2016 IEEE international conference on web services*. http://dx.doi.org/10.1109/ICWS.2016.35.

Cao, B., Liu, X. F., Rahman, M. M., Li, B., Liu, J., & Tang, M. (2020). Integrated content and network-based service clustering and web apis recommendation for mashup development. *IEEE Transactions on Services Computing*, *13*, 99–113. http://dx.doi.org/10.1109/TSC.2017.2686390.

Cao, B., Liu, J., Wen, Y., Li, H., Xiao, Q., & Chen, J. (2019). Qos-aware service recommendation based on relational topic model and factorization machines for iot mashup applications. *Journal of Parallel and Distributed Computing*, *132*, 177–189. http://dx.doi.org/10.1016/j.jpdc.2018.04.002.

Cao, B., Xiao, Q., Zhang, X., & Liu, J. (2019). An api service recommendation method via combining self-organization map-based functionality clustering and deep factorization machine-based quality prediction. *Chinese Journal of Computers*.

Chai, J., Liu, J. N., & Ngai, E. W. (2013). Application of decision-making techniques in supplier selection: A systematic review of literature. *Expert Systems with Applications*, *40*, 3872–3885. http://dx.doi.org/10.1016/j.eswa.2012.12.040, URL: https://www.sciencedirect.com/science/article/pii/S095741741201281X.

Chen, L., Wang, Y. L., Yu, Q., Zheng, Z. B., & Wu, J. (2013). Wt-lda: User tagging augmented lda for web service clustering. In *Proceedings of the international conference on service-oriented computing*. Berlin, Heidelberg: Springer-Verlag, http://dx.doi.org/10.1007/978-3-642-45005-1_12.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: Human language technologies, Volume 1 (Long and short papers)* (pp. 4171–4186). Minneapolis, Minnesota: Association for Computational Linguistics, http://dx.doi.org/10.18653/v1/N19-1423, URL: https://aclanthology.org/N19-1423.

Hu, Y., Peng, Q., & Hu, X. (2014). A time-aware and data sparsity tolerant approach for web service recommendation. In *ICWS'14, Proceedings of the 2014 IEEE international conference on web services* (pp. 33–40). http://dx.doi.org/10.1109/ICWS.2014.18.

Ji, F., Yang, Y., & Zhou, Z. (2018). Multi-layered gradient boosting decision trees. In *Advances in neural information processing systems 31: Annual conference on neural information processing systems 2018, NeurIPS 2018, December 3-8, 2018, Montrél, Canada* (pp. 3555–3565). URL: http://papers.nips.cc/paper/7614-multi-layered-gradient-boosting-decision-trees.

Jiang, H., Zhang, J., Ren, Z., & Zhang, T. (2017). *Proceedings of the 2017 IEEE/ACM 39th international conference on software engineering (ICSE)* (pp. 38–48). http://dx.doi.org/10.1109/ICSE.2017.12, URL: https://doi.ieeecomputersociety.org/10.1109/ICSE.2017.12.

Kuang, L., Xia, Y., & Mao, Y. (2012). Personalized services recommendation based on context-aware qos prediction. In *ICWS '12, Proceedings of the IEEE international conference on web services* (pp. 400–406).

Lepori, M. A., Linzen, T., & McCoy, R. T. (2020). Representations of syntax [MASK] useful: Effects of constituency and dependency structure in recursive lstms. In *Proceedings of the 58th annual meeting of the association for computational linguistics, ACL 2020, Online, July 5-10, 2020* (pp. 3306–3316). http://dx.doi.org/10.18653/v1/2020.acl-main.303.

Liu, X. Z., Hui, Y., Sun, W., & Liang, H. Q. (2007). Towards service composition based on mashup. In *Proceedings of the 2007 IEEE congress on services* (pp. 332–339). http://dx.doi.org/10.1109/SERVICES.2007.67.

Lo, W., Yin, J., Li, Y., & Wu, Z. (2015). Efficient web service qos prediction using local neighborhood matrix factorization. *Engineering Applications of Artificial Intelligence*, *38*, 14–23. http://dx.doi.org/10.1016/j.engappai.2014.10.010, URL: http://www.sciencedirect.com/science/article/pii/S0952197614002504.

Majchrzak, A., & More, P. H. B. (2011). Emergency! web 2.0 to the rescue!. *Communications of the ACM*, *54*, 125–132. http://dx.doi.org/10.1145/1924421.1924449.

Maximilien, E. M., & Ranabahu, A. (2007). The programmableweb: Agile, social, and grassroot computing. In *Proceedings of the international conference on semantic computing*. USA: IEEE Computer Society, http://dx.doi.org/10.1109/ICSC.2007.98.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. In *Proceedings of the 1st international conference on learning representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, workshop track proceedings*. URL: http://arxiv.org/abs/1301.3781.

Mirjalili, S., Mirjalili, S. M., & Lewis, A. (2014). Grey wolf optimizer. *Advances in Engineering Software*, *69*, 46–61. http://dx.doi.org/10.1016/j.advengsoft.2013.12.007.

Nguyen, M., Yu, J., Bai, Q., Yongchareon, S., & Han, Y. (2020). *ACSW '20, Attentional matrix factorization with document-context awareness and implicit api relationship for service recommendation*. New York, NY, USA: http://dx.doi.org/10.1145/3373017.3373034.

Paik, I., & Mizugai, H. (2010). Recommendation system using weighted tf-idf and naive bayes classifiers on rss contents. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, *14*, 631–637.

Qi, K., Hu, H., Song, W., Ge, J., & Lü, J. (2015). Personalized qos prediction via matrix factorization integrated with neighborhood information. In *Proceedings of the IEEE international conference on services computing, SCC '15* (pp. 186–193). USA: IEEE Computer Society, http://dx.doi.org/10.1109/SCC.2015.34.

Ray, B., Garain, A., & Sarkar, R. (2021). An ensemble-based hotel recommender system using sentiment analysis and aspect categorization of hotel reviews. *Applied Soft Computing*, *98*, Article 106935. http://dx.doi.org/10.1016/j.asoc.2020.106935, URL: https://www.sciencedirect.com/science/article/pii/S1568494620308735.

Rodriguez-Mier, P., Pedrinaci, C., Lama, M., & Mucientes, M. (2016). An integrated semantic web service discovery and composition framework. *IEEE Transactions on Services Computing*, *9*, 537–550. http://dx.doi.org/10.1109/TSC.2015.2402679.

Shao, L., Zhang, J., Wei, Y., Zhao, J., Xie, B., & Mei, H. (2007). Personalized qos prediction forweb services via collaborative filtering. In *Proceedings of the IEEE international conference on web services* (pp. 439–446). http://dx.doi.org/10.1109/ICWS.2007.140.

Su, K., Ma, L., Xiao, B., & Zhang, H. (2016). Web service qos prediction by neighbor information combined non-negative matrix factorization. *Journal of Intelligent & Fuzzy Systems*, *30*, 3593–3604. http://dx.doi.org/10.3233/IFS-162104.

Sumithra, R. P., & Sarath, R. (2015). Towards restful web service composition for healthcare domain. In *Proceedings of the 2015 international conference on control, instrumentation, communication and computational technologies (ICCICCT)* (pp. 845–848). http://dx.doi.org/10.1109/ICCICCT.2015.7475397.

Tan, Q. X., Ye, M., Yang, B. Y., Liu, S. Q., Ma, A. J., Yip, T. C., et al. (2020). DATA-GRU: dual-attention time-aware gated recurrent unit for irregular multivariate time series. In *The thirty-fourth AAAI conference on artificial intelligence, AAAI 2020, the thirty-second innovative applications of artificial intelligence conference, IAAI 2020, the tenth AAAI symposium on educational advances in artificial intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020* (pp. 930–937). AAAI Press, URL: https://aaai.org/ojs/index.php/AAAI/article/view/5440.

Wang, R., Li, Q., & Zhao, Q. (2018). Aircraft support information system integration based on soa and web service. *Computer Engineering*, *44*, 91–97.

Wen, C., Hsu, P., Wang, C., & Wu, T. (2012). Identifying smuggling vessels with artificial neural network and logistics regression in criminal intelligence using vessels smuggling case data. In *Lecture notes in computer science*: *vol. 7197*, *Intelligent information and database systems - 4th Asian conference, ACIIDS 2012, Kaohsiung, Taiwan, March 19-21, 2012, Proceedings, Part II* (pp. 539–548). Springer, http://dx.doi.org/10.1007/978-3-642-28490-8_56.

Wu, H., Yue, K., Li, B., Zhang, B., & Hsu, C. H. (2018). Collaborative qos prediction with context-sensitive matrix factorization. *Future Generation Computer Systems*, *82*, 669–678. http://dx.doi.org/10.1016/j.future.2017.06.020.

Xie, W., Dong, Q., & Gao, H. (2014). A probabilistic recommendation method inspired by latent dirichlet allocation model. *Mathematical Problems in Engineering*, *2014*, 1–10. http://dx.doi.org/10.1155/2014/979147, 2014, (2014-9-28).

Xiong, W., Wu, Z., Li, B., Gu, Q., & Hang, B. (2016). Inferring service recommendation from natural language api descriptions. In *Proceedings of the 2016 IEEE international conference on web services (ICWS)*. http://dx.doi.org/10.1109/ICWS.2016.48.

Yan, D., Ahmad, S. Z., & Yang, D. (2013). Matthew effect, abc analysis and project management of scale-free information systems. *Journal of Systems and Software*, *86*, 247–254. http://dx.doi.org/10.1016/j.jss.2012.08.013, URL: https://www.sciencedirect.com/science/article/pii/S0164121212002294.

Yang, Z. I., Dai, Z. H., Yang, Y. M., Carbonell, J. G., Salakhutdinov, R., & Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. In *Proceedings of the advances in neural information processing systems 32: Annual conference on neural information processing systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada* (pp. 5754–5764).

Yao, L., Sheng, Q., Segev, A., & Yu, J. (2013). Recommending web services via combining collaborative filtering with content-based features. In *Proceedings of the IEEE international conference on web services, ICWS '13*. USA: IEEE Computer Society, http://dx.doi.org/10.1109/ICWS.2013.16.

Yao, L., Wang, X., Sheng, Q. Z., Benatallah, B., & Huang, C. (2021). Mashup recommendation by regularizing matrix factorization with api co-invocations. *IEEE Transactions on Services Computing*, *14*, 502–515. http://dx.doi.org/10.1109/TSC.2018.2803171.

Yeung, D. S., Sun, X. Q., & Zeng, X. Q. (2001). Sensitivity analysis of multilayer perceptron. In *Proceedings of the seventeenth international joint conference on artificial intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001* (pp. 809–818). Morgan Kaufmann.

Yin, Y., Chen, L., Xu, Y., & Wan, J. (2018). Location-aware service recommendation with enhanced probabilistic matrix factorization. *IEEE Access*, *6*, 62815–62825. http://dx.doi.org/10.1109/ACCESS.2018.2877137.

Zervoudakis, K., & Tsafarakis, S. (2020). A mayfly optimization algorithm. *Computers & Industrial Engineering*, *145*, Article 106559. http://dx.doi.org/10.1016/j.cie.2020.106559, URL: https://www.sciencedirect.com/science/article/pii/S036083522030293X.

Zhang, N., Wang, J., & Ma, Y. (2020). Mining domain knowledge on service goals from textual service descriptions. *IEEE Transactions on Services Computing*, *13*, 488–502. http://dx.doi.org/10.1109/TSC.2017.2693147.

Zheng, Z., Ma, H., Lyu, M. R., & King, I. (2011). Qos-aware web service recommendation by collaborative filtering. *IEEE Transactions on Services Computing*, *4*, 140–152. http://dx.doi.org/10.1109/TSC.2010.52.