API-Prefer: An API Package Recommender System Based on Composition Feature Learning

Yancen Liu and Jian Cao(✉)

Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
{LiuYancen,cao-jian}@sjtu.edu.cn

Abstract. With the exponential increase in Web Application Programming Interfaces (APIs), selecting appropriate APIs to construct a mashup is a challenging task. When multiple APIs are put together, their overall function is not just a superposition of their individual functions in many cases. Unfortunately, the approaches proposed to date do not sufficiently model the synthetical functions of the combined APIs. In this paper, an *API P*ackage *re*commender system based on composition *fe*ature lea*r*ning (API-Prefer) is proposed. API-Prefer tries to learn the composition features of an API pair. Then the composition features can be used to predict whether this API pair can be adopted by a mashup or not. Specifically, a deep neural network is designed for composition feature learning and adoption probability prediction in API-Prefer. Since there is a large amount of API pairs, API-Prefer applies a strategy to select the potential APIs first, then the API packages can be discovered based on the predicted scores over multiple API pairs. Experiments on a real-world dataset show API-Prefer is significantly better than the comparative methods.

Keywords: API package recommendation · Composition feature · Mashup · API · Neural network

1 Introduction

Web services are important components of a modern information system. As a type of Web services, the number of Web Application Programming Interfaces (or APIs, for short) is increasing exponentially on the Web. In order to help developers or non-IT professionals make use of APIs, various tools have been developed. Of them, mashups are becoming a commonly used approach, through which multiple APIs can be combined together to provide more comprehensive functions. Since the number of available APIs on the Web is huge, it is a challenging task to find the APIs we need. To better assist mashup development, we recommend multiple sets of cooperative APIs where each of them can achieve

E. Kafeza et al. (Eds.): ICSOC 2020, LNCS 12571, pp. 500–507, 2020.
https://doi.org/10.1007/978-3-030-65310-1_36

the functions of a mashup as a whole. These sets of cooperative APIs are often referred to as "*API Packages*".

We propose an *API P*ackage *re*commender system based on composition *fe*ature lea*r*ning (API-Prefer). API-Prefer extracts the *composition features* of an API pair through a neural network. The contributions of this paper are as follows:

- Based on analyzing the relationships between APIs and mashups, we design a deep learning model to learn the composition features of API pairs to support both *shallow composition relationships* and *deep composition relationships*.
- We propose API-Prefer, an API package recommender system for mashup. API-Prefer is based on composition feature learning. It also includes the strategies to avoid unnecessary calculations and generate the final packages.
- We compare API-Prefer with baselines and state-of-art models and the experimental results show that API-Prefer is significantly better than the counterparts in terms of the recall and precision.

2 Related Work

API (or Web Service) recommendation for mashups has been a popular research topic, and various methods have been proposed in recent years.

When an API is published, its name, description, or tags are often provided. These methods utilize traditional information retrieval ways to select the recommended APIs by matching the description of the mashup with the API description. For example, in [1], a vector space model is used for service retrieval. Recently, researchers began to adopt more advanced technology to extract the semantic relationship between mashups and APIs.

With the continuous development of machine learning and deep neural network in recent years, some methods combining deep learning with service recommendation have emerged. In [2], a method to extract user preference embeddings to personalize and precisely recommend APIs to mashup developers is proposed. Although these approaches try to learn more latent relationships between mashups and APIs through the deep learning model, they don't learn the synthetical functions of composed APIs. Furthermore, these approaches still recommend an API list instead of an API package.

The frequent co-occurrence set-based approach applies some traditional data mining technology to discover frequent API sets. For example, in [3], a method to mine frequent API pairs for recommendation is proposed. An information-retrieval based approach can be combined with the frequent co-occurrence set-based approach. For example, a multi-level relational network is proposed to obtain the comprehensive relationships among topics, tags and APIs [4].

Our model is also a hybrid approach. Different from the other models, our model learns the composition features of API pairs through a deep neural network, which can support both shallow composition and deep composition relationships.

3 API-Prefer: An API Package Recommender System Based on Composition Feature Learning

3.1 Composition Features of APIs

Let us give specific explanation of *composition features* through two real-world examples. The first example is a mashup with the description "*Find lyrics and karaoke videos with this mashup. Record it and publish it also*". It is based on two APIs, *YouTube API* and *LyricWiki API*. This example represents the case where we can search APIs based on the phrases in the mashup descriptions in a straightforward way, which is called a *shallow composition relationship*. The second example is *RueFind*, whose description is "*RueFind is a travel application which tracks interesting tourist attractions around the world. Users can add, rank or create lists of their favorite attractions*". *Google Map API* and *Yahoo Weather API* are used in *RueFind*. According to the descriptions, the main function of *RueFind* is to share information on tourist attractions. Although travel relates to maps and weather, it cannot be matched with them directly. This fact indicates that by combining APIs with different functions, synthetical functions can be created. This is called a *deep composition relationship*.

Composition features are context-dependent, i.e., when a set of APIs is applied to mashups with very different functions, their composition features may be different. The approaches to learn composition features are introduced in Sect. 4.

3.2 Overview of API-Prefer

API package recommendation by API-Prefer consists of two stages, i.e., the training stage and the recommendation stage. During the training stage, a deep neural network that can predict whether an API pair can be applied to a mashup or not is trained. Specifically, this network uses a multiple-layer structure to learn the composition features of this API pair for a mashup. During the recommendation stage, given the mashup description, potential API pairs will be inputted to the network and the probability of this API pair being used by this mashup is outputted. Finally, a recommendation algorithm generates multiple API packages in terms of the adoption probability of API pairs and other information.

4 A Deep Neural Network for Predicting the Adoption Probabilities of an API Pair Based on Composition Feature Extraction

The descriptions of mashups and APIs vary in length, so we need to embed these descriptive texts into the uniform vectors. We use the latent Dirichlet allocation topic model to extract the topic feature of the text. After tokenization, the standard steps for text data pre-processing are undertaken, these being

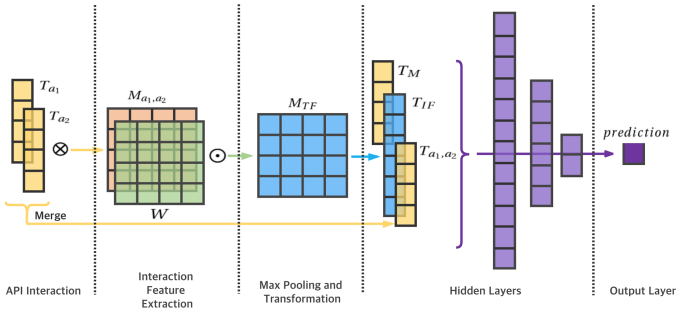

Fig. 1. The deep neural network model for predicting the adoption probability of an API pair

stemming, lemmatization and removing stop words. We transfer the descriptions of the APIs, the historical mashups and the mashup to be developed into 200-dimension LDA topic vectors (Fig. 1).

The LDA topic vector of API_1, API_2,and *mashup* is denoted by T_{a_1}, T_{a_2} and T_M respectively, which are denoted as: $T = \{t_1, t_2, t_3, ..., t_i, ...\}$.

Since we want to mine the composition features from a pair of APIs, the interactions of the features of two APIs can yield new features. Therefore, we add an interaction layer into the network, through which all the features of two APIs interact in pairs. The results are denoted by a matrix M_{a_1,a_2} as $M_{a_1,a_2} = T_{a_1} \otimes T_{a_2}$ $(M_{i,j} = t_i^{a_1} \cdot t_j^{a_2})$.

However, not all feature interactions are equally useful. Therefore, we add a weight layer W to adjust the interaction features as $M_{TF} = M_{a_1,a_2} \odot W$ $(m_{i,j}^{CF} = (t_i^{a_1} \cdot t_j^{a_2}) \cdot w_{i,j})$. We get the interaction feature matrix M_{TF}. Then we use a 10×10 max polling filter to process the matrix, and this will turn the original matrix into a 20×20 matrix. Then we transform it to a 400×1 vector, which is the interaction feature vector T_{IF}.

We merge the topic information of API_1 and API_2 to a combination feature vector T_{a_1,a_2} as $T_{a_1,a_2} = \{max(t_1^{a_1}, t_1^{a_2}), max(t_2^{a_1}, t_2^{a_2}), ..., max(t_i^{a_1}, t_i^{a_2}), ...\}$. Then we concatenate the combination features, interaction features and mashup features together $(\{T_x = T_{a_1,a_2}, T_{IF}, T_M\})$ as the input to the multiple hidden layers of the network.

After 3 hidden layers, we use a sigmoid function in the output layer to make the prediction score between 0 and 1. The loss function for the network is cross entropy, and we also add a L2 normalization to avoid the overfitting of our model.

Therefore, through a multi-layer neural network, the composition features are actually learned from the combination features, interaction features and mashup features, which are then used to make an adoption probability prediction.

5 API Package Recommendation

We select and sort the historical mashups in terms of the similarity between the descriptions of the historical mashups and the mashup to be developed. After we obtain a sufficient number of mashups, we use the APIs used by these candidate mashups to generate the API packages. As for the number of candidate APIs, on the one hand, we don't want too many APIs in the candidate set, but on the other hand, we want the candidate API set to cover as many potential APIs as it can. Therefore, we need to find an appropriate value for it.

After obtaining a set of APIs as candidates, we can predict the adoption probability $p(API_i, API_j)$ of each API pair. By regarding each API as a node, and the adoption probability of an API pair as the weight of the edge between them, we can draw a relational network of APIs. API packages can be discovered on this network.

In order to discover API packages, we add a restriction on the network, i.e., only when $p(API_i, API_j) > max(p(API_i), p(API_j), \epsilon)$ is true, will an edge appear in this network. To discover all the effective edges for an API, we just get the possibilities between this API with all the other candidate APIs in the candidate set and use the equation above to get all the effective ones. This process is Function *SearchEdges*, which is used in the following Package Discovery Process. Then we try to discover the fully connected sub-graphs and the APIs represented by their nodes can compose packages. As a special case, a single API can also be a package when it is not in any fully connected sub-graph, provided its prediction score is higher than a threshold.

The discovery process starts from an API seed, and makes use of breadth-first search (BFS) to search for other members that are appropriate for a package. The API seeds are chosen based on the their popularity in the mashup candidate set from the largest to the smallest. For an API seed API_{Seed}, after we detect all its effective edges, then:

- If there is no effective edge for it, $p(API_{Seed})$ is compared with a threshold η. If $p(API_{Seed})$ is over the threshold, then API_{Seed} itself can be a package, otherwise, it is skipped.
- If multiple effective edges can be found, we adapt a BFS algorithm to find the fully connected sub-graph for this node. We maintain a *queue* Q_{API} and push the seed node into Q_{API} and the set Pkg first. Then each time we operate on the node API_{head}, which is the head of the Q_{API} until the Q_{API} is empty. For API_{head}, we traverse its effective edges to obtain an API list whose members are sorted in the descending order of their weights. If an API API_i from this list has effective edges with all nodes corresponding to the APIs in the set Pkg, that is, it can form a fully connection graph with these nodes, we then append it to Pkg and Q_{API}. Otherwise, we skip it and fetch the next API_{head}.

We try more API_{Seed}s till the number of packages meets the requirements.

6 Experiments

6.1 Dataset and Experiment Settings

We crawled 12,140 APIs and 6,976 mashups from Programmable Web, which is the largest mashup and API information sharing platform. There are 200 * 200 units in the Interaction Feature Extraction Layer, and we use a 10 * 10 filter for Max-pooling. The configuration of the 3 hidden layers is (200, 100, 20). And the L2 Regularization Strength λ is set to 0.001.

An important parameter in our method is the size of the candidate APIs. When the number reaches around 200, the mean cover rate of the final adopted APIs reaches 0.92, which is an appropriate parameter setting. As for the two thresholds in API package recommendation, after parameter tuning, we finally set $\epsilon = 0.86$ and $\eta = 0.92$.

6.2 Comparison Methods

Some baselines and state-of-the-arts methods are selected as the comparison methods.

- WVSM sorts the APIs by the product of similarity and popularity.
- WJaccard is similar to WVSM. The difference is it uses Jaccard similarity.
- Collaborative Filtering Method (CF) is based on TF-IDF between the mashup description texts is calculated to evaluate whether they are similar or not.
- ERTM [5] recommends the APIs based on an enhanced relational topic model, which leverages the potential Dirichlet distribution of the probabilistic topic model to extract the functional properties of APIs.
- TopicCF [6] combines the topic model with the collaborative filtering approach.
- SASR [7] models multi-dimensional social relationships among potential users, topics, mashups, and APIs using a coupled matrix model.
- MRN [4] captures the deep relationships among APIs on top of the latent topic, tag and API network.

6.3 Evaluation Metrics

We use precision, recall and f1-measure to evaluate our experimental results. $recall = \frac{TP}{(TP+FN)}$, $precision = \frac{TP}{(TP+FP)}$, $f1 - measure = \frac{2 \cdot recall \cdot precision}{(recall+precision)}$.

where precision represents the acceptance degree of users in relation to the recommendation results, recall represents the completeness of the recommendation results, and the f1-measure is the synthesis of the two evaluation indexes.

6.4 Results

Firstly, we compare the performances of all the approaches with the number of recommended APIs and the results are shown in Fig. 2. The performance of

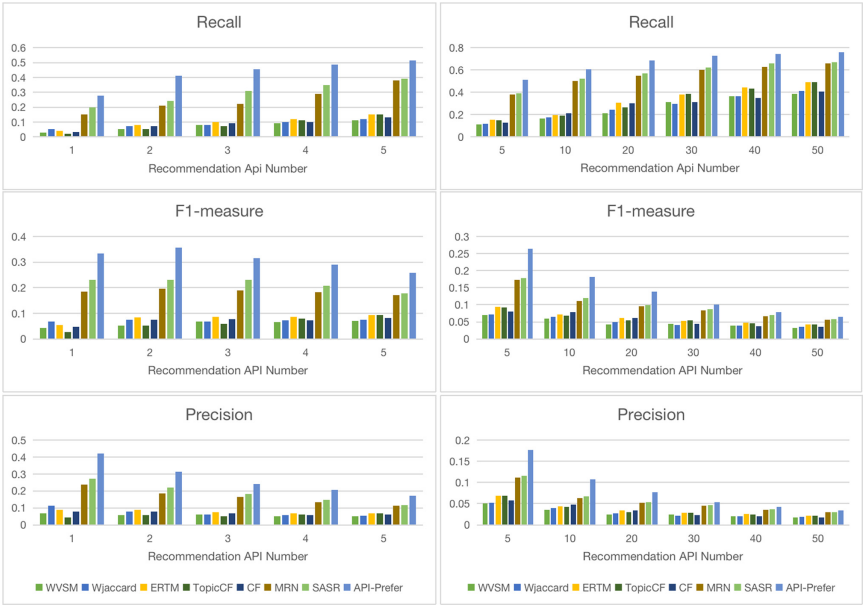

Fig. 2. Performance comparisons of all methods

the baselines is not good when either a small amount $[1, 5]$ or a large amount $[5, 50]$ of APIs are recommended. Which indicates that only considering semantic similarity and few attributes like popularity, cannot get ideal result. MRN and SASR, on the other hand, take all the features into account. Their performances are better than the previous methods. The results show that it is useful to consider multilevel relationships. However, with a recommendation number between $[1, 5]$, the performance of API-Prefer is clearly better than the others. This also reflects the effectiveness of the composition feature learning in API-Prefer.

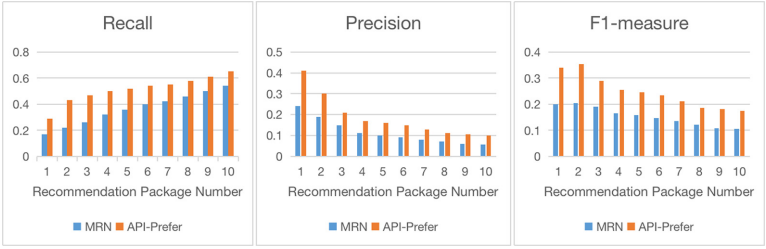

Fig. 3. Performance comparisons of API package recommendation methods

Of all the compared approaches, MRN is the only one that can recommend API packages. We compare the performances of API-Prefer and MRN with the

number of API packages to be recommended. Figure 3 shows that performances of API-Prefer are significantly better than the performances of MRN.

7 Conclusions and Future Work

In this paper, we analyze the main deficiency in the current approaches of API recommendations for mashup. Therefore, we propose API-Prefer, an API package recommender system based on composition feature learning. API-Prefer learns the composition features of an API pair based on combination features, interaction features and mashup features through a deep neural network. Then, the adoption probability can be predicted based on the description of the mashup to be developed and the composition features of API pairs. Finally, the API packages can be generated based on the adoption probabilities. The performance of API-Prefer is verified through the experiments.

Our future work will focus on improving the recommendation accuracy by using advanced textual embedding techniques and considering the composition of three or more APIs also have specific features.

Acknowledgement. This work is supported by National Key Research and Development Plan (No. 2018YFB1003800).

References

1. Platzer, C., Dustdar, S.: A vector space search engine for web services. In: Third European Conference on Web Services (ECOWS 2005), pp. 9-pp. IEEE (2005)
2. Fletcher, K.: Regularizing matrix factorization with implicit user preference embeddings for web API recommendation. In: 2019 IEEE International Conference on Services Computing (SCC), pp. 1–8. IEEE (2019)
3. Maaradji, A., Hacid, H., Skraba, R., Vakali, A.: Social web mashups full completion via frequent sequence mining. In: 2011 IEEE World Congress on Services, pp. 9–16. IEEE (2011)
4. Cao, J., Lu, Y., Zhu, N.: Service package recommendation for mashup development based on a multi-level relational network. In: Sheng, Q.Z., Stroulia, E., Tata, S., Bhiri, S. (eds.) ICSOC 2016. LNCS, vol. 9936, pp. 666–674. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46295-0_46
5. Li, C., Zhang, R., Huai, J., Sun, H.: A novel approach for API recommendation in mashup development. In: 2014 IEEE International Conference on Web Services, pp. 289–296. IEEE (2014)
6. Jain, A., Liu, X., Yu, Q.: Aggregating functionality, use history, and popularity of APIs to recommend mashup creation. In: Barros, A., Grigori, D., Narendra, N.C., Dam, H.K. (eds.) ICSOC 2015. LNCS, vol. 9435, pp. 188–202. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48616-0_12
7. Xu, W., Cao, J., Hu, L., Wang, J., Li, M.: A social-aware service recommendation approach for mashup creation. In: 2013 IEEE 20th International Conference on Web Services, pp. 107–114. IEEE (2013)