Personalized API Recommendation via Implicit Preference Modeling

Wei Gao[1(✉)], Liang Chen[2], Jian Wu[1], Hai Dong[2], and Athman Bouguettaya[2]

[1] College of Computer Science and Technology, Zhejiang University, Hangzhou, China
{gw,wujian2000}@zju.edu.cn
[2] School of Computer Science and Information Technology, RMIT, Melbourne, Australia
{liang.chen,Hai.dong,athman.bouguettaya}@rmit.edu.au

Abstract. With a huge amount of APIs on the Internet, understanding users' complex needs and preferences for APIs becomes an important task. In this paper, we aim to uncover users' implicit needs for APIs and recommend suitable APIs for users. Specifically, first different similarity scores between APIs are computed according to heterogeneous functional aspects of APIs. Next, users' preferences for APIs is combined with similarities of APIs measured with different functional aspects, and matrix factorization technique is used to learn the latent representation of users and APIs for each functional aspect. Then we use a personalized weight learning approach to combine the latent factors of different aspects to get the predicted preferences of users for APIs.

1 Introduction

With the development of Web 2.0 paradigm and mobile service computing, API (Application Programming Interface) as a form of REST-ful service has become prevalent in developing Web and mobile applications. Consumers can enjoy web or mobile service by simply invoking the APIs provided by service developers. As a result, the number of services has been experiencing a dramatic increase over the past few years. And platforms like ProgrammbleWeb[1], Mashape[2], APIStore[3] that collect, host and aggregate APIs on the Internet are emerged to help users find APIs that meet their requirements in an efficient manner. However, with the huge number of APIs on the web, it is quite challenging for a consumer to find APIs that are satisfactory to her. There is a need to understand users' implicit needs and preferences for APIs, predict what APIs they are likely to use and recommend a list of suitable APIs.

A lot of methods have been proposed for service recommendation, including collaborative-filtering based method [5,10,11], content-based method [2,3,6] and

[1] www.programmableweb.com.
[2] www.mashape.com.
[3] apistore.baidu.com.

Q.Z. Sheng et al. (Eds.): ICSOC 2016, LNCS 9936, pp. 646–653, 2016.
DOI: 10.1007/978-3-319-46295-0_44

a hybrid of both [7–9]. Our work can be regarded as a hybrid recommendation method that combines collaborative filtering and content-based method. In our API recommendation model, users and APIs are connected via the follow relations representing users historical preferences for APIs. Furthermore, APIs are linked with different aspects of functional information. For instance, each API is associated with its provider, category, tags, textual information or the mashups composing it. And different types of relations exist between APIs. For instance, the Facebook API and the Twitter API are linked to each other as they belong to the same Social category, and they are composed together by some mashup. As a result, there exist heterogeneous relations between APIs and these rich source of relations can be utilized in conjunction with user preference relations for hybrid API recommendation.

In this paper, we explore the multiple aspects of API functionalities and their effects on users' preferences, and propose a novel API recommendation approach. We identify several functional aspects of APIs that could impact the adoption of APIs for users, including APIs provider, category, developer, tag, textual description and mashups it composes. We propose to measure the similarities of APIs according to different functional aspects. Then we combine the users historical interaction data with different aspects of API similarities by propagating the user-API relations under different aspects of functionality. Matrix factorization technique is applied on the propagated user preference data and latent factor representation for users and APIs is calculated for each functional aspect of API accordingly. Then we combine these latent factors of different aspects with different weights personalized for each user. A weight learning method is proposed to learn a personalized recommendation model for each user. The final predicted scores of users' preferences for APIs are computed with the learned weights and top scoring APIs are used for recommendation. Our contributions in this work are summarized as follows.

1. We study the problem of discovering users' implicit preferences for APIs and make personalized recommendations by mining rich heterogeneous functional information of APIs with various aspects.
2. We combine matrix factorization based collaborative filtering as well as different types of functional aspects of APIs with personalized weights to perform API recommendation.
3. We conduct extensive experiments on the ProgrammableWeb dataset and the result demonstrate the effectiveness of our recommendation approach.

The remainder of this paper is organized as follows. In Sect. 2, we give an overview of the related work of service recommendation approaches. In Sect. 3, we present details of the proposed API recommendation approach. In Sect. 4, we present and analyze our experimental results. Section 5 concludes this paper.

2 Related Work

Service recommendation can be classified into three categories: functional-based service recommendation, non-functional service recommendation and a hybrid

of both. Non-functional based service recommendation focus on predicting the non-functional features (i.e., Qos) of services and recommend services that gives the best Qos performance [5,10,11]. However, the Qos attributes of APIs are not always available due to expensive access cost to all APIs over the web. The functionality of services is usually analyzed by the functional description of services, such as structured WSDL files or free form textual descriptions. In [3], services with similar functionalities are clustered based on WSDL files and the result is used for service recommendation. [2] use LDA-based approach to model functionality of services using both WSDL files and tags. [6] propose a relational topic modeling (RTM) technique to model the functionality of mashups, services and their links. The hybrid service recommendation method combines both functionality and Qos attributes of services. Other available side information can also be utilized for recommendation. [7] use collaborative topic regression technique to integrate both users' preferences and functional features of services. [4] proposed to incorporate three heterogeneous factors to recommend APIs for mashup: the functionality of an API, the usage history of the API by mashups and the popularity of the API, and integrate different sources of information using Bayes' theorem. However, the heterogeneous functional aspects of APIs and whether it is informative to infer users' preferences is unexplored.

3 User Preference Modeling

In this section, we propose to model users' preferences for APIs in terms of different functional aspects of APIs and propose a novel personalized API recommendation model by combining heterogeneous functional information of APIs and collaborative information of other users. Specifically, with m users and n APIs, the user-API matrix $R \in \mathbb{R}^{m \times n}$ is constructed as follows: if a user u_i followed an API a_j, then the corresponding entry of R_{ij} is 1. Otherwise, it is 0. The goal is to predict preference score for unobserved user-API pairs by filling in the 0 entries. To this end, we propose the following recommendation method: First, six functional aspects of APIs, namely provider, category, developer, tag, textual descriptions and mashups, are extracted for each API and the similarities of APIs are evaluated under different aspects of API. Second, the user-API interaction matrix is constructed and is combined with API similarities measured with six aspects respectively. For each aspect of API, the user-API matrix is propagated. Then the six augmented matrices are factored into user latent factor matrices and API latent factor matrices respectively using matrix factorization technique. To predict users' preferences for APIs, the predicted score for each aspect is linearly combined with different weights for each user and a method is proposed to learn the weights from data. Finally, the predicted scores of users for APIs are calculated and APIs that receive the highest score for each user are recommended. The detailed explanation of the method is as follows:

First, we identify six aspects to model the functionalities of an API and their impact on users' preferences. We describe the similarity measuring method for each aspect as follows:

1. Provider. The provider of an API is usually a website that publishes APIs. If two APIs have the same developer, their similarity is 1, otherwise it is 0.
2. Category. Each API belongs to some categories curated by the ProgrammableWeb taxonomy. The similarity of APIs according to category is computed as the Jaccard similarity of the categories two APIs belongs to.
3. Developer. Developers are a group of people who contribute to the development process of APIs. The similarity of APIs with respect to developers is computed as the Jaccard similarity of the developers two APIs have.
4. Tag. Tagging is an effective method of annotating APIs with a list of keywords. The similarity of APIs with respect to tags is computed as the Jaccard similarity of the tags two APIs have.
5. Textual Description. Each API has a brief text describing its functions. We use Latent Dirichlet Allocation (LDA [1]) method, which is a probabilistic topic modeling technique that models the textual descriptions of APIs as a distribution on latent topics. The similarity is computed as the cosine similarity of two topic vectors of APIs.
6. Mashup. Besides functional similarity, different APIs may complement each other. Two APIs of different functionalities can be composed together to form a mashup. The complementarity of APIs is computed as the number of mashups that compose both APIs divided by the number of mashups that only compose one of them.

Next, we utilize heterogeneous aspects of APIs and incorporate them into the recommendation process. We construct an enhanced matrix $\widetilde{R}$ on the basis of R by adding functional aspects of APIs. To this end, we use the following equation to infer users' preferences for each aspect:

$$\widetilde{R}_{ij} = \frac{\sum_{k=1}^{n} R_{ik}\text{sim}(a_j, a_k)}{\sum_{k=1}^{n} \text{sim}(a_j, a_k)} \tag{1}$$

where $\text{sim}(a_j, a_k)$ measures the similarity of APIs under a particular aspect. From Eq. (2) we can see that the users' implicit preferences for unknown APIs are measured by two parts: (1) The observed user-API follow relations represented by R, and (2) the similarities between APIs for a certain aspect represented by $\text{sim}(\cdot)$ function. The predicted score of an unfollowed API is the aggregation of APIs followed by the user weighted by the similarity between them and the predicted API. In this way, the users' preferences are propagated from the original user-API follow matrix R by incorporating the functional aspect of APIs represented by their similarities.

Then, for each propagated user-API matrix, we use matrix factorization method to derive low rank matrices of users and APIs. Since the propagated matrix incorporates a specific type of similarity measurement of APIs, the latent representations of users and APIs are encoded with a certain aspect of API functionality. Specifically, for each propagated matrix $\widetilde{R}^{(l)}$, a pair of user-factor matrix $\widetilde{U}^{(l)}$ and item-factor matrix $\widetilde{V}^{(l)}$ is generatd. Each pair $(\widetilde{U}^{(l)}, \widetilde{V}^{(l)})$ represents users factors and APIs factors according to a certain aspect of APIs. The users' preferences for APIs under a specific aspect is then computed as

$\widetilde{R}^{(l)} = \widetilde{U}^{(l)}\widetilde{V}^{(l)T}$. The overall predicted user-API preference score is computed by combining different aspects of APIs together. As different users may have different preference on particular aspects of APIs, we assign a weight vector for each user to measure her personal preference on different aspects of APIs. The preference score prediction of user u_i to API a_j is defined as follows:

$$\widehat{R}(u_i, v_j) = \sum_{l=1}^{6} \theta_{il} \widetilde{U}_i^{(l)} \widetilde{V}_j^{(l)T} \tag{2}$$

where θ_{il} is the weight for user i on the lth aspect of API. The weight indicates how much impact the corresponding aspect of APIs has on the particular user.

Finally, we introduce how to learn the weight parameter θ for the personalized preference prediction model. The preference score of a user to a followed API should always be higher than the preference score of a user to a non-followed API. To be specific, for each user u_i, if API a_j is followed by user previously and API a_k has never been followed, then the predicted preference score of u_i to a_j should be higher than to a_k. That is, $\widehat{R}(u_i, a_j) > \widehat{R}(u_i, a_k)$ always holds. Assuming that the user-API follow relations are organized in the form of a series of triplets (u_i, a_j, a_k), which means u_i follows API a_j and did not follow a_k. Thus for each triplet, we can define the loss function as:

$$l(u_i, a_j, a_k) = \sigma(R(u_i, a_k) - R(u_i, a_j)) \tag{3}$$

where σ is the sigmoid function to constrain the value between 0 and 1. For all the triplets generated, the overall objective function is

$$L = \sum l(u_i, a_j, a_k) + \frac{\lambda}{2} \|\theta\|^2 \tag{4}$$

where λ is a regularization parameter. By minimizing L we can learn the parameter θ from the triplets. We employ stochastic gradient descent (SGD) method to estimate the parameter θ. The time complexity of the parameter learning process is $O(mn^2)$, which is infeasible for practical implementation. In reality, we only sample a small subset of triplets from data to make the learning process more efficient.

4 Experiments and Evaluation

We conduct a set of experiments to evaluate our proposed approach for API recommendation. We crawled all the APIs available on ProgrammableWeb up until March 2016 and their corresponding followers to construct user-API matrix. We also crawled each API's names, secondary categories, providers, textual descriptions, developers and mashup composition. In summary, our dataset contains 792 users and 9,650 APIs. And there are a total of 1,486 APIs used in 7,066 mashups. We split 80 % of the APIs each user followed into training set and the rest of the APIs as test set. We use precision, recall and F-score to evaluate

the prediction accuracy when recommending top K APIs for each user and the metrics are averaged over all users to get the overall performance. We compare our method with the following baseline methods:

1. Neighborhood-based Collaborative Filtering (NBCF). It performs APIs recommendation based on users with similar preferences.
2. Matrix Factorization (MF). The user-API follow matrix R is directly factorized for learning and prediction.
3. Matrix Factorization + Single (MFS). We evaluate the recommendation performance of each of the six aspects one by one.
4. Matrix Factorization + Uniform Weight (MFU). Each aspect is aggregated with the same weight. That is, θ_l is a constant equals to 1/6.
5. Matrix Factorization + Weight Learning (MFW). Different weights for each aspect of APIs are learned. However the weight vector of each user are all the same.
6. Matrix Factorization + Personalized Weight (MFPW). It is the personalized API recommendation method proposed in the paper with weights different for each user.

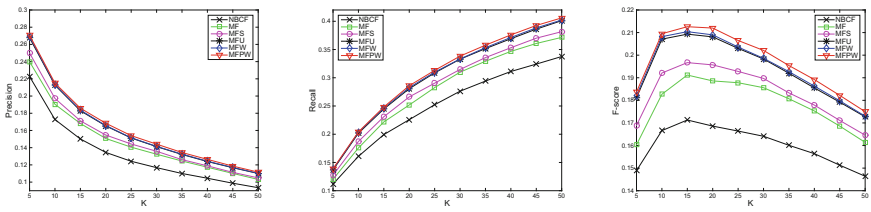

Fig. 1. Performance comparison of competitive methods

Figure 1 shows the recommendation accuracy of all the competitive methods as the number of recommended APIs K ranges from 5 to 50. For method MFS, we only plot the aspect that achieves the best performance to save space. We can see that MFPW consistently achieves the best recommendation performance among all the other methods, which demonstrates the effectiveness of our approach on API recommendation. For other baseline algorithms, Matrix factorization method outperforms neighborhood based method, which shows that latent factor model can capture user API preference relations more accurately. Furthermore, incorporating heterogeneous functional information of APIs can enhance the recommendation performance as MFS method outperforms MF method. And MFU achieves a better recommendation result than MFS, which proves that the idea of combining different aspects of APIs is more effective than only considering a single aspect. Still, MFU method can not yield a better result over MFW since the uniform combination method is too simple to model the difference of impact of API factors. MFPW learns a personalized weight for each user and model users' preferences at a finer granularity, as evidenced by the superior

performance compared to MFW. In summary, the experimental results validate our theoretical analysis and show its efficacy in API preference modeling.

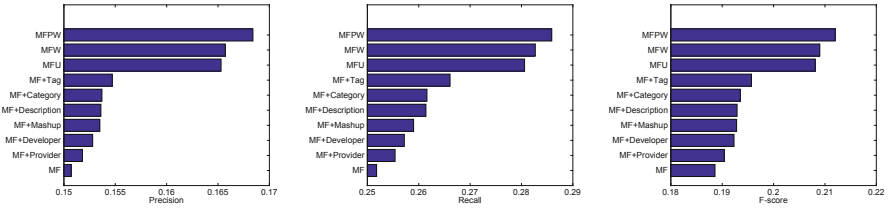

Fig. 2. Performance comparison of each aspect in MFS method

In particular, we plot the recommendation performance for each of the six aspects of APIs in the MFS method as well as other competitive methods. The result is shown in Fig. 2 with the number of recommended APIs K set to 20. Some observations can be drawn: First, compared to the basic MF method, the performance of incorporating each aspect of API all shows an increase with varying degrees. This shows that each aspect of API proposed is helpful for user preference modeling and API recommendation. In particular, different aspects of API has a different impact on users following pattern. The API tags reports the best performance among all the six aspects, which indicates tags are the most informative aspect for understanding users' preferences for APIs. And the API provider is the least informative information for characterizing users' preferences. Second, unifying each aspect of APIs together for recommendation (MFU, MFW, MFPW) performs better than only one single aspect is utilized for recommendation by a relatively large margin. This indicates that the idea of combining different aspects of APIs together is helpful in API recommendation.

5 Conclusion

In this paper, we propose a comprehensive approach for recommending APIs to users. We utilize the heterogeneous content information of APIs and explore the impact of different aspects of APIs on the preferences of users. We identify several aspects of APIs, including API's provider, category, developer, tag, textual descriptions and mashups composing them. Then users' preferences are propagated through different functional aspects of API and matrix factorization is employed for each propagated matrix to learn user and API latent factors under each aspect. To achieve personalized recommendation, latent factors of six functional aspects are combined with different weights for each user. The experimental results demonstrate that our approach outperforms other baseline approaches and proves its effectiveness for personalized API recommendation.

Acknowledgment. This research was partially supported by the Natural Science Foundation of China under grant of No. 61379119, National Science and Technology Supporting Program of China under grant of No. 2015BAH18F02, the Fundamental Research Funds for the Central Universities under grant of No. ZH2016007.

References

1. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. J. Mach. Learn. Res. **3**, 993–1022 (2003)
2. Chen, L., Wang, Y., Yu, Q., Zheng, Z., Wu, J.: WT-LDA: user tagging augmented LDA for web service clustering. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) ICSOC 2013. LNCS, pp. 162–176. Springer, Heidelberg (2013). doi:10.1007/978-3-642-45005-1_12
3. Elgazzar, K., Hassan, A.E., Martin, P.: Clustering WSDL documents to bootstrap the discovery of web services. In: IEEE International Conference on Web Services, ICWS 2010, Miami, Florida, USA, 5–10 July 2010, pp. 147–154 (2010)
4. 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). doi:10.1007/978-3-662-48616-0_12
5. Jiang, Y., Liu, J., Tang, M., Liu, X.F.: An effective web service recommendation method based on personalized collaborative filtering. In: IEEE International Conference on Web Services, ICWS 2011, Washington, DC, USA, 4–9 July 2011, pp. 211–218 (2011)
6. 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, ICWS, 2014, Anchorage, AK, USA, 27 June–2 July 2014, pp. 289–296 (2014)
7. Liu, X., Fulia, I.: Incorporating user, topic, and service related latent factors into web service recommendation. In: 2015 IEEE International Conference on Web Services, ICWS 2015, New York, NY, USA, 27 June–2 July 2015, pp. 185–192 (2015)
8. 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, Santa Clara, CA, USA, 28 June–3 July, 2013, pp. 107–114 (2013)
9. Yao, L., Sheng, Q.Z., Segev, A., Yu, J.: Recommending web services via combining collaborative filtering with content-based features. In: 2013 IEEE 20th International Conference on Web Services, Santa Clara, CA, USA, 28 June–3 July 2013, pp. 42–49 (2013)
10. Zheng, Z., Ma, H., Lyu, M.R., King, I.: Wsrec: a collaborative filtering based web service recommender system. In: IEEE International Conference on Web Services, ICWS 2009, Los Angeles, CA, USA, 6–10 July 2009, pp. 437–444 (2009)
11. Zheng, Z., Ma, H., Lyu, M.R., King, I.: Collaborative web service qos prediction via neighborhood integrated matrix factorization. IEEE Trans. Serv. Comput. **6**(3), 289–299 (2013)