Improving Personalized Project Recommendation on GitHub Based on Deep Matrix Factorization

Huan Yang[1], Song Sun[1], Junhao Wen[1], Haini Cai[1(✉)], and Muhammad Mateen[2]

[1] School of Big Data and Software Engineering, Chongqing University, Chongqing, China
{huanyang,sun2007song,jhwen}@cqu.edu.cn
[2] Department of Computer Sciences, Air University, Multan, Pakistan
muhammad.mateen@aumc.edu.pk

Abstract. GitHub is a hosting platform for open-source software projects, where developers can share their open-source projects with others in the form of a repository. However, as the software projects hosted on the platform increase, it becomes difficult for developers to find software projects that meet their need or interest. Considering the practical importance of software project recommendations, we propose a recommendation method based on deep matrix factorization and apply it to GitHub, which is used to recommend personalized software projects in GitHub. With the use of deep neural network, we learn a low dimensional representation of users and projects from user-project matrix in a common space, in which we can capture the user's latent behavior preference of each item, and automatically recommend the top N personalized software projects. The experiments on use-project data extracted from GitHub shows that the proposed recommendation method can recommend more accurate results compared with other three recommendation methods, i.e., UCF (user collaborative filtering), ICF (item collaborative filtering) and PPR (a personalized recommendation method).

Keywords: GitHub · Project recommendation · Personalized recommendation · User behavior · Deep matrix factorization

1 Introduction

GitHub [22] is a large and popular hosting platform for software projects. As of September 2020, GitHub reported that there are more than 56 million developers and 60 million software code repositories active on their websites around the world [1]. These large-scale software projects have increased the difficulty for developers to search for target software projects. In the actual development process, developers will spend a significant amount of time to browse popular

H. Gao and X. Wang (Eds.): CollaborateCom 2021, LNICST 406, pp. 318–332, 2021.
https://doi.org/10.1007/978-3-030-92635-9_19

software libraries or consult different resources, such as project documents, mailing lists and forums, to find software projects they are interested in [13]. What more, the traditional keywords search engines using in consulting are usually relying on text-matching like similarity measures [17–19], but a small number of keywords may not accurately describe all the characteristics of a software project.

In this situation, traditional recommendation methods seem to be hard to work, because that the previous studies mainly focused on collaborative filtering methods and content-based recommendation methods [2]. However, this kind of recommendation methods do not consider the individual need of the developers, while only rely on the similarity of the project description and the project code [17,24]. Besides, they extract the code through the suffix of the project file when calculating the code similarity, which greatly leads to the inaccuracy.

During the past few decades, deep learning has achieved great success in various fields such as computer vision, natural language processing, pattern recognition and so on [6,16]. Since the excellent performance of deep learning in non-linear relationships capturing and data representation for sparse high-dimensional vectors, its high potential in rating prediction has attracted more and more attention in the recommendation system. Besides, to support the collaborative development of software projects, GitHub has implemented various functions, such as create, fork, and star. Hence, it is achievable to capture the projects that meet user's personal preference and interest with the help of deep learning by collecting a mass of user behavior data when they use these functions [23].

In this paper, we propose a personalized recommendation method: we score user behavior based on their operations generate a user-project matrix firstly, then learn a low dimensional representation of users and projects in a common space through neural networks, finally we predict score for each pair of user and project and recommend a list of software projects with high predicted score to developers. It is worthy to note that our method does not focus on any specific software project, but analyzes the behavior data of developers. Therefore, we can avoid many errors caused by similar projects (such as the software projects with the same programming language, and the software projects with the same configuration file that is easy to misjudged as similar projects) and achieve the real personalized recommendation. In summary, our contributions are as follows:

- We propose a recommendation method based on deep matrix factorization to recommend personalized software projects for developers in GitHub. We use user behavior matrix and neural networks to predict users' potential preferences
- The experimental results show that our proposed recommendation method is more effective than other three baseline methods.

The remainder of this paper is organized as follows. In Sect. 2, we introduce some related work of software project recommendation system. In Sect. 3, we present the proposed method of architecture and details. In Section 4, we illustrate our experimental setup. In Sect. 5, we report the experimental results. In

Sect. 6, we list three types of threats to the validity of our method. Section 7 makes a conclusion and give details of the future work.

2 Related Work

2.1 GitHub Project Recommendations

A search of the literature reveal few studies which is about GitHub project recommendations. Some previous researches can be roughly divided into two aspects: CF-based (collaborative filtering) and network-based. The CF-based method is to determine whether the target project has similar software projects by calculating the similarity between projects. The network-based method uses the user's historical operation information [14] to build an information network depending on users and projects. Li et al. [24] proposed a method to detect similar repositories on GitHub, which mainly evaluates the similarity between two repositories by calculating star-based correlation and readme-based correlation. Koskela et al. [11] proposed a hybrid open-source software recommendation method. Specifically, they combined three different similarity measurement methods on three different feature sets to form a recommendation list. Han et al. [8] proposed an approach to predict the popularity of GitHub project, which rely on the number of stars of a project. These studies focused on the features of the project itself. They recommend projects depending on the similarity or popularity of projects, which ignored the interest of users and their real need. In contrast, our study focuses on personalized project recommendation, i.e., we consider developer behavior to recommend relevant projects.

In study [23], the authors used different user behaviors on GitHub to study which types of user behavior data are suitable for recommending related projects. On this basis, Guendouz et al. [7] proposed and discussed a GitHub repository recommendation system based on collaborative filtering, which models user behavior as a user-project matrix for calculating the similarity between users (developers) and items (repositories). Tadej et al. [14] proposed a recommendation system. They constructed a network graph relying on user data from GitHub (such as fork and pull-request), then an unsupervised learning model was used to predict the relationship of items and users. However, only considering the explicit ratings of developer behavior seems not sufficient, so we also take the developer's underlying preference into account to improve the recommendation accuracy.

2.2 Deep Learning in Recommendation Systems

The application of deep learning in recommendation systems has attracted more and more attention. Badiâa Dellal et al. [3] realized a recommendation system based on MLP deep learning adapted to data already defined by their characteristics. He et al. [9] proposed the NCF model, which is short for neural collaborative filtering. They used a multilayer perceptron to learn user-item interaction

functions. The study [15] proved that ensembling NCF and matrix factorization can be helpful. Xue et al. [21] proposed a deep matrix factorization model (DMF), which maps users and items into low-dimensional vectors, making full use of explicit ratings and implicit feedback. Lian et al. proposed CCCFNet [12], namely Cross-domain Content-boosted Software Service Recommendation Base on Collaborative Filtering Neural Network, which combines collaborative filtering and content-based filtering in a unified framework. Liu et al. [13] proposed a learning-to rank model named NNLRank by analyzing the different features of projects and expertise of developers. It extracts nine features as inputs from projects and developer's personal information respectively and predicts the project's preference based on a feedforward neural network. With the help of neural network, we take the developer behavior information as input to predict user preference instead of item preference, which is different from these studies above.

3 Proposed Methods

This section describes the design and implementation of the recommendation method we proposed in detail and Fig. 1 shows the overall architecture of the recommendation process.

3.1 Data Collection

Firstly, we download the data from January 2016 to June 2019[1] on GitHub with the use of GHTorrent[2]. Secondly, according to the entities and relationships[3] of these data, we capture user_ids for each organization from table organization_members.csv and table user.csv. With the help of the user_id, we obtain the project_ids from table projects.csv and generate the table uid-pid.csv. Then the data from GitHub are processed into a form of < user_id, project_id, create, fork, star > by using uid_pid.csv, projects.csv and watchers.csv. Finally, we get the datasets of users, projects and corresponding behavior to generate the user-project matrix. We will explain the task of this part in details in the next content.

3.2 Recommender System

In this part, we assign different scores to the user behavior and generate a user-project matrix. Then, we perform deep matrix factorization on this matrix to obtain the predicted score for each pair of user and project, which make full use of the explicit score and implicit feedback. And finally, a sorted list of projects will be recommended for developers.

[1] https://ghtorrent.org/downloads.html.
[2] https://ghtorrent.org.
[3] https://ghtorrent.org/relational.html.

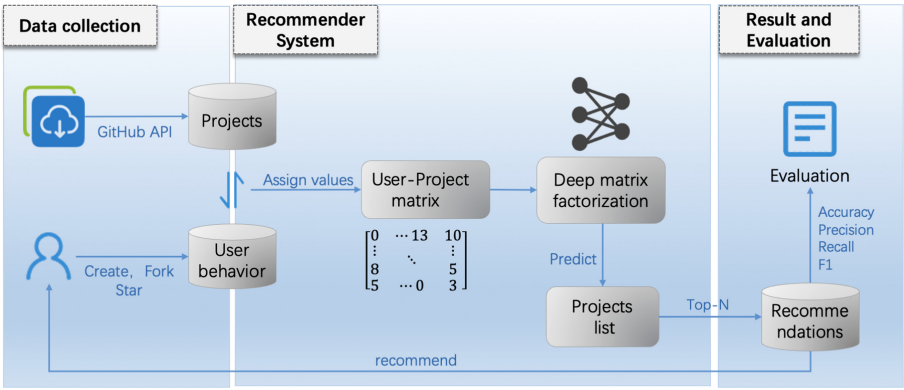


Fig. 1. The architecture of our method.

User-Project Matrix. GitHub provide many functions to support the collaborative development of software, such as create, fork, watch and pull-request. When the developers use these functions, a large amount of user behavior data that record their activities around related software projects will generate. And these data reflect developers' preference and interest [23].

Users' behaviors of repositories on GitHub often contain create, fork and star, different behaviors represent the level of interest in the projects. The create behavior indicates that the user created this project, which means that the project is directly related to the user, so it has the most weight. The fork behavior used by a user implies that the project must meet the need of the user forking it. Finally, the star behavior expresses the interest of a user(s) in a particular project, but the project is not an urgent need of the users.

By scoring each project on account of different user behaviors, we build a user-project matrix. Following [17], we also use the specific construction rules: if the user creates a project, then its score is 10, forks and stars are scored by 5 and 3 points respectively. In addition, if a user creates and stars a project at the same time, the score should be the sum of the two behavior scores, which is 13 points. Similarly, if the project is forked and stared by user at the same time, the score is 8 points. It is worth noting that a project can't have three user behaviors at the same time because create and fork are always opposite and will not appear simultaneously. Therefore, all possible scores for the project are 3, 5, 8, 10, and 13 scores.

Definition: Let $U = \{u_1, u_2 \cdots u_n\}$ denote the set of all developers, $P = \{p_1, p_2 \cdots p_m\}$ denote the set of all software projects, and finally the user-project matrix is expressed as $M_{n \times m}$, where:

$$M_{ij} = \begin{cases} 3, \ if \ u_i \ \ stars \ \ p_j \\ 5, \ if \ u_i \ \ forks \ \ p_j \\ 8, \ if \ u_i \ \ stars \ \ and \ \ forks \ \ p_j \\ 10, if \ u_i \ creates \ \ p_j \\ 13, if \ u_i \ \ stars \ \ and \ creates \ p_j \end{cases}$$

Recommendations Method. In this paper, we make use of a recommendation method based on deep matrix factorization, which can recommend a list of software projects with high predicted scores to developers. Different from the traditional matrix factorization, in the deep matrix factorization model, two neural networks are used to decompose the rating matrix (User-Project matrix) obtained from the interaction information between the users and the items. By mapping the users and items to a common k-dimensional latent feature space, we can explore the relationship between users and items and learn the potential behavior preference of users in this latent space [21], so we propose the following recommendation model, as shown in the Fig. 2:

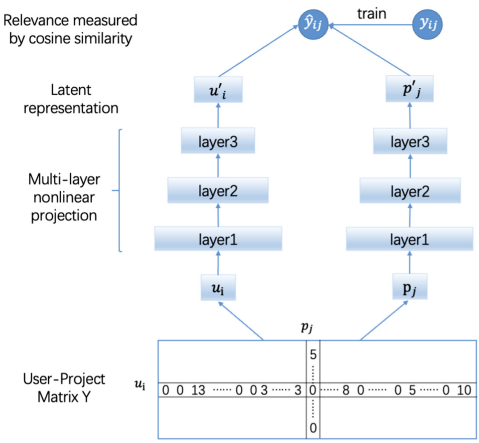


Fig. 2. Deep matrix factorization recommendation model.

From the user-project matrix Y, each user vector u_i represents the i-th user's rating of all items and each project p_j represents the ratings of all users on the j-th project p_j. Let x and y denote the input and output vectors, then the intermediate hidden layer is defined as l_i $(i = 1, 2, \cdots, N-1)$, the weight matrix of layer l_i is represented by W_i, the bias term of layer b_i is represented by bi, finally the output latent representation is represented by h:

$$\begin{aligned} l_1 &= W_1 x \\ l_i &= f(W_{i-1} l_{i-1} + b_i), i = 2, 3, \cdots, N-1 \\ h &= f(W_N l_{N-1} + b_N) \end{aligned} \tag{1}$$

In the hidden and output layers $l_i\,(i = 2, \cdots, N-1)$, we use ReLU as an activation function:

$$f(x) = max(0, x) \tag{2}$$

We use two different multi-layer networks to transform the vectors of users and items, and map them to a common low-dimensional vector space respectively, as shown in Eq. 3.

$$\begin{aligned} u'_i &= f_N^U\left(\cdots f_3^U\left(W_{U_2} f_2^U\left(W_{U_1} u_i\right)\right)\cdots\right) \\ p'_j &= f_N^P\left(\cdots f_3^P\left(W_{P_2} f_2^P\left(W_{P_1} p_j\right)\right)\cdots\right) \end{aligned} \tag{3}$$

Where U represents the user input matrix and P represents the project input matrix. W_{U_i} and W_{P_j} represent the weight matrix of the user and item in each of the network layer.

Finally, $\widehat{Y}_{ij}$ represents the user's predicted score for the item:

$$\widehat{Y}_{ij} = consine(u'_i, p'_j) = \frac{{u'_i}^T {p'}_j}{\|{u'}_i\| \left\|{p'}_j\right\|} \tag{4}$$

In the choice of loss function, because square loss focuses on explicit ratings and cross-entropy loss focuses on implicit ratings, so we use a new optimization function in the deep matrix factorization model [21] to incorporate explicit ratings into cross-entropy loss. Therefore, we can combine explicit and implicit information for optimization. As shown in Eq. 5.

$$\begin{aligned} L = -\sum_{(i,j)\in Y^+\cup Y^-} & \left(\frac{Y_{ij}}{max(R)} log\widehat{Y}_{ij}\right. \\ & \left. +\left(1 - \frac{Y_{ij}}{max(R)}\right) log\left(1 - \widehat{Y}_{ij}\right)\right) \end{aligned} \tag{5}$$

Although Eq. 5 links explicit and implicit feedback and perform the scale normalization, but it ignores the large difference of rating in user behavior matrix, which will affect the accuracy of model prediction. Therefore, we propose a function to improve this problem as shown in the Eq. 6:

$$S(x) = \frac{1}{1 + e^{-x+3}} \tag{6}$$

By this no-linear function, the score will be mapped compactly into the range from 0.5 to 1, and the gap between the rating is also compressed greatly. On this basis, combined with Eq. 5, a new improved loss function of Eq. 7 is proposed:

$$\begin{aligned} L = -\sum_{(i,j)\in Y^+\cup Y^-} & \left(S(Y_{ij}) log\widehat{Y}_{ij}\right. \\ & \left. +\left(1 - S(Y_{ij})\right) log\left(1 - \widehat{Y}_{ij}\right)\right) \end{aligned} \tag{7}$$

Due to the cross-entropy loss function, the prediction score $\widehat{Y}_{ij}$ may be negative, so we need to correct the predictions with the aid of Eq. 6. When the loss

function is less than a predetermined value, it will be equal to a certain minimum value, following [21], we set the minimum value $\mu = 1.0e^{-6}$.

$$Y_{ij}^{o} = max(\mu, Y_{ij}^{o}) \tag{8}$$

3.3 Result and Evaluation

To evaluate the performance of our recommendation method, following the literature [5,10], we select all software projects that have not interacted with each user, joint with the test items(interacted with users), to compose the predicted objects, and then a topN recommendation list will be generated for each user depending on the predicted score. We believe that the test items represent the ones users like, so if these items rank higher than others in the test stage, the recommendation shows better performance. For the large data set, since it is too time-consuming to sort each item in the evaluation process, we adopted a common strategy [9], which is to randomly select 30% of the items that users have not interacted with to involve in the predicted objects. The evaluation indicators will be described in detail later.

4 Experimental Setup

4.1 Datasets

Following [17], we extract user behavior and repositories into four groups as our datasets. The first three groups are extracted from three organizations on GitHub: Vim-jp[4], Formidable[5], and Harvestq[6]. In the last group, we extract 2663 active GitHub users and 75417 related repositories.

In GitHub, users have multiple behaviors towards public software projects. We start by extracting all developers of each organizations and get the user_ids, then we extract the project_ids for each user, the result of this operation is a table uid_pid.csv in a form of < user_id, project_id >. Finally, we can get the behavior data(i.e., create, fork, and star) through user_id and project_id. These behavior data are described as < user_id, project_id, create, fork, star >, where the create, fork and star describes the behavior of the user (user_id) for project (project_id). Then, following [17], we randomly selected 60% of the user behavior data as input, and used the remaining 40% of the data for evaluation. The detailed information of each group is shown in Table 1.

4.2 Evaluation and Metrics

We use evaluation Metrics such as Accuracy, Recall, Precision, and F1 [7] to verify the effectiveness of our recommendation method. Accuracy indicates what

[4] https://github.com/vim-jp.
[5] https://github.com/FormidableLabs.
[6] https://github.com/harvesthq.

Table 1. Details of four groups of GitHub data

Group name	Users	Projects	Development areas
Vim-jp	47	6262	Vimscript
Formidable	47	2321	Web
Harvesthq	31	944	Android
Large	2663	75417	–

percentage of sample predictions are correct. Recall indicates how many samples among all true samples are predicted to positive. Precision indicates how many predictions are correct in the sample where your prediction is positive. F1 is used to integrate precision and recall as an evaluation indicator. In this paper, the positive sample means the project our method recommends. U represents all users in the test data, and N represents the sum of the number of software projects we recommend for each user. Test represents the number of software projects in the test set that have interacted with users. TopN represents the number of software items recommended for each user. In Table 2, we give the equations of these four metrics.

Table 2. Evaluation metrics

Metric	Formula
Accuracy	$\lvert\{u \mid u \in U, test \cap topN \neq \varnothing\}\rvert / \lvert U\rvert$
Precision	$\lvert test \cap topN\rvert / N$
Recall	$\lvert test \cap topN\rvert / \lvert test\rvert$
F1	$2 * precision * recall / (precision + recall)$

4.3 Statistic Test

When we compare the performance of methods, if the average performance values of multiple methods are different, we still cannot believe that the performance of the higher average method is better than the lower average performance [20]. In this case, we perform Frideman test and Nemenyi post-hoc test, which is widely used in previous studies [20] to analyze the performance difference of the method pairs (i.e., our method and each comparison method) at significant level 0.05. The Frideman test is a non-parametric approach, which compares the overall performance of k algorithms over N data sets. If the p-value [4] calculated by the Frideman test is less than 0.05, the null hypothesis is rejected, that is, all methods perform equally, in other words, there is a significant difference between the methods.

Table 3. The precision and recall of our method, UCF, ICF and PPR

		Precision			Recall		
		Top3	Top5	Top10	Top3	Top5	Top10
Harvesthq	UCF	3.23%	4.52%	3.87%	0.68%	1.53%	2.79%
	ICF	2.15%	3.23%	1.94%	0.45%	1.09%	1.40%
	PPR	5.38%	5.16%	4.52%	3.23%	5.16%	9.66%
	Our	**17.20%**	**26.45%**	**26.45%**	**4.89%**	**12.54%**	**25.08%**
Formidable	UCF	6.82%	5.45%	3.64%	0.98%	1.31%	1.74%
	ICF	2.96%	0.91%	2.67%	0.32%	0.16%	0.97%
	PPR	6.38%	5.53%	6.81%	1.66%	2.44%	5.78%
	Our	**23.40%**	**26.81%**	**27.45%**	**3.60%**	**6.87%**	**14.07%**
Vim-jp	UCF	21.99%	20.43%	15.32%	1.03%	1.60%	2.40%
	ICF	4.96%	3.40%	2.55%	0.17%	0.20%	0.31%
	PPR	5.67%	3.83%	3.40%	0.49%	0.53%	0.96%
	Our	**31.21%**	**34.04%**	**31.49%**	**1.50%**	**2.72%**	**5.04%**
Large	UCF	3.66%	3.07%	2.34%	0.79%	1.10%	1.68%
	ICF	0.55%	0.46%	0.35%	0.12%	0.16%	0.25%
	PPR	4.11%	3.93%	3.91%	0.98%	1.56%	3.10%
	Our	**8.57%**	**9.72%**	**8.26%**	**2.06%**	**3.89%**	**6.61%**

Table 4. The accuracy and F1 of our method, UCF, ICF and PPR

		Accuracy			F1		
		Top3	Top5	Top10	Top3	Top5	Top10
Harvesthq	UCF	10.34%	20.00%	37.93%	1.12%	2.28%	3.24%
	ICF	6.90%	16.67%	20.69%	0.75%	1.63%	1.62%
	PPR	12.90%	25.81%	29.03%	4.03%	5.16%	6.15%
	Our	**29.03%**	**45.16%**	**58.06%**	**7.62%**	**17.01%**	**25.75%**
Formidable	UCF	17.78%	20.00%	24.44%	1.71%	2.11%	2.35%
	ICF	6.82%	4.44%	18.18%	0.58%	0.28%	1.42%
	PPR	17.02%	21.28%	38.30%	2.64%	3.39%	6.25%
	Our	**36.17%**	**38.30%**	**48.94%**	**6.24%**	**10.94%**	**18.60%**
Vim-jp	UCF	44.68%	**55.32%**	**63.83%**	1.97%	2.96%	4.15%
	ICF	12.77%	14.89%	17.02%	0.34%	0.38%	0.55%
	PPR	17.02%	17.02%	31.91%	0.89%	0.93%	1.50%
	Our	**46.81%**	46.81%	53.19%	**2.86%**	**5.05%**	**8.69%**
Large	UCF	10.07%	12.67%	17.47%	1.30%	1.62%	1.96%
	ICF	1.46%	2.15%	3.29%	0.29%	0.24%	0.20%
	PPR	9.04%	11.35%	13.79%	1.58%	2.23%	3.46%
	Our	**16.50%**	**16.63%**	**17.48%**	**3.32%**	**5.56%**	**7.35%**

Table 5. The precision and recall in different dimension

	Precision				Recall			
	k = 50	k = 100	k = 150	k = 200	k = 50	k = 100	k = 150	k = 200
Harvesthq	20.65%	26.45%	23.23%	26.45%	19.57%	25.08%	22.02%	25.08%
Formidable	23.83%	27.45%	23.83%	25.53%	12.21%	14.07%	12.21%	13.09%
Vim-jp	33.83%	31.49%	38.94%	32.55%	5.42%	5.04%	6.23%	5.21%
Large	7.56%	8.26%	7.46%	6.56%	6.05%	6.61%	7.05%	6.35%

Table 6. The accuracy and F1 in different dimension

	Accuracy				F1			
	k = 50	k = 100	k = 150	k = 200	k = 50	k = 100	k = 150	k = 200
Harvesthq	41.94%	58.06%	51.61%	58.06%	20.09%	25.75%	22.61%	25.75%
Formidable	36.17%	48.94%	36.17%	40.43%	16.15%	18.60%	16.15%	17.30%
Vim-jp	57.45%	53.19%	59.57%	55.32%	9.34%	8.69%	10.75%	8.98%
Large	16.16%	17.48%	16.14%	15.16%	6.72%	7.35%	6.72%	6.45%

If the null hypothesis is rejected, it indicates that there are significant differences among the algorithms, then we can continue to conduct follow-up tests. In this paper, Nemenyi post-hoc test is used to find out which algorithms have statistical differences in performance, and the difference of average ranking of each algorithm is correlated with a certain domain value Critical Difference (CD) [4]. If the difference is greater than this domain value, it means that the algorithm with high average ranking is statistically superior to the algorithm with low average ranking; otherwise, there is no statistical difference between the two.

5 Experimental Results

5.1 RQ1:Does the Proposed Method Perform Better Than Other Comparison Methods?

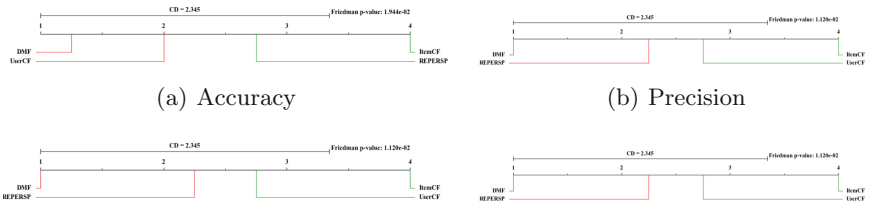
Method: We compare our recommendation results with those of ICF [17,23], UCF [23], and one state-of-the-art recommendation method, a personalized project recommendation method (PPR) [17]:

ICF is project-centric method and it contains two steps: First, it calculates similarity between projects through user behavior. Then, ICF will recommend a list of software projects to users based on the similarity of each items and their historical behavior.

UCF is similar to ICF, UCF is user-centric method and its recommendation process is fallowing: First, it determines the users having similar behavior patterns with the candidate user, which we call similar neighbors, then calculate the candidate user's predicted value for the projects based on the behavior of similar neighbors.

PPR mainly considers the combination of software project characteristics and user behavior. First, it calculates the similarity of readme file and code in each project to obtain the similarity matrices of two projects. Then PPR constructs a user-item matrix, which denotes the historical behavior of users. Finally, a recommendation list is generated for candidate users by examining item similarity matrix through the user behavior.

In addition, we perform statistic significant test to identify whether the differences in performance between our method and other 3 baseline methods are randomness or statistically significant.



(a) Accuracy (b) Precision

(c) Recall (d) F1-score

Fig. 3. Comparison of our method(DMF) against UCF, ICF and PPR with Friedman test and Nemenyi post-hoc test in terms of all 4 indicators.

Result: From the experimental results in Table 3 and Table 4, we can see that our method has better accuracy, precision, recall and F1 than other comparative methods. The main reasons are as follows:

First, the sparsity of the data set is very high, which means that the amount of user behavior is scarce compared with the number of items. So, it is difficult for UCF to accurately find the similar users, and ICF can not make good use of the user behavior data to calculate the similarity of items either. However, on groups of Formidable and Vim-jp, as the number of users and projects increase, we observe a corresponding increase in the accuracy of UCF. The reason behind this correlation may be that the increase of user behavior leads to an increase in user relevance, which makes it easy for UCF to find more similar users.

Second, compared with UCF and ICF, the performance of PPR is improved to a certain extent, but this method may be affected by the same type of suffix when computing the similarity of items through description and source code. Besides, the sparsity of the user-item matrix causes that there are few user behavior data available, good accuracy cannot be achieved in the face of a large amount of data. However, by the aid of neural network, the proposed method can predict unknown user behavior scores based on existing user behavior data, and it alleviate the problem caused by sparsity.

Third, Fig. 3 visualizes the results of Friedman test and Nemenyi post-hoc test for our method and the 3 baseline methods in terms of the 4 indicators. The p-values (all less than 0.05) of Friedman test above the sub-figures show that there exist significant differences among the 4 methods on all indicators. The

Nemenyi test results show that our method significantly performs better than the other baseline methods on all indicators.

To summary, the experimental results show that the proposed method is better than other 3 baseline methods. This means that our method can provide more effective recommendation.

5.2 RQ2:What Is the Effect of the Dimension of the Low-Dimensional Vector and the Number of Recommended Lists on the Performance of the Proposed Method?

Method: To show the experimental results more clearly, we define k as the dimension of the latent vector, and topN as the number of recommended lists, and we recorded the best precision and recall for evaluation in each experiment. We carry out the experiments where k is assigned to 50,100,150 and 200 respectively with fixed value topN = 10 and topN is assigned to 3,5 and 10 respectively with fixed value k = 100.

Result: Table 3 and Table 4 show the performance affected by the number of recommendation list. On the one hand, it can be clearly seen that accuracy, precision, recall and F1 are growing with the increasing of topN. Obviously, when the number of recommended items increases, the target items in the test set are more likely to appear in the recommended list. On the other hand, as the number of items in the data set increases, the recall will decrease due to that we have to recommend topN items. Table 5 and Table 6 show the performance affected by the dimension of latent vectors. When dim increase, there is a slight volatility about accuracy, precision, recall and F1, which may caused by the problem of overfitting.

6 Threats to Validity

6.1 Internal Validity

Internal validity pays attention to the possible faults in the implementation of methods. To minimize this kind of validity, we implement our method by modifying the open source code about DMF model shared by other authors to adapt to our score prediction task. Regarding the reproduction of the baseline methods, we carefully implement them by following the corresponding studies and use the third-party open source code to implement these comparative methods. However, our implementation may not be able to fully reproduce the original methods which may lead to a bias in the comparison between our method and the baseline methods.

6.2 External Validity

External validity focuses on the generalization of the experimental conclusions to other datasets. Our experimental results are derived from user behavior data of

four GitHub groups. The first three groups represent three different development areas, which have a relatively small number of users. The fourth group, which include more than 2663 active GitHub users and 75417 related repositories, is less affected by this threat. To further improve the generalization of experimental results, we plan to include more developers and projects.

7 Conclusions

In this paper, we proposed a neural collaborative filtering recommendation method based on user behavior. We extracted users' actions on each related project from different GitHub organizations and generated a user-project matrix by scoring the different actions. Then we utilized a deep neural network to capture user's latent preference, and obtained an abstract data representation from the sparse vector. Finally we computed the predicted scores based on the representation vector, and automatically recommended the topN software projects to developers. The experimental results illustrated that our method obtained better results compared with the other three baseline methods, i.e., UCF, ICF and PPR.

In the future, we plan to extend more user behavior features, such as watch and pull-request, to optimize our method. In addition, we will also consider the impact of time factors.

Acknowledgments. This study was supported by National Natural Science Foundation of China (NSFC): Research on service recommendation of trusted sharing and heterogeneous data fusion in the mobile crowd sensing environment (Grant no.62072060).

References

1. Github: the 2020 state of octoverse report. https://octoverse.github.com (2020)
2. Chen, L., Zheng, A., Feng, Y., Xie, F., Zheng, Z.: Software service recommendation base on collaborative filtering neural network model. In: 16th International Conference, ICSOC 2018, Hangzhou, China, November 12–15, 2018, Proceedings (2018)
3. Dellal-Hedjazi, B., Alimazighi, Z.: Deep learning for recommendation systems. In: 6th IEEE Congress on Information Science and Technology, CiSt 2020, Agadir - Essaouira, Morocco, 5–12 June 2021. pp. 90–97. IEEE (2021)
4. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. **7**, 1–30 (2006)
5. Elkahky, A.M., Song, Y., He, X.: A multi-view deep learning approach for cross domain user modeling in recommendation systems. In: Proceedings of the 24th International Conference on World Wide Web, pp. 278–288 (2015)
6. Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y.: Deep Learning, vol. 1. MIT Press, Cambridge (2016)
7. Guendouz, M., Amine, A., Hamou, R.M.: Recommending relevant open source projects on github using a collaborative-filtering technique. Int. J. Open Source Softw. Process. **6**(1), 1–16 (2015)

8. Han, J., Deng, S., Xia, X., Wang, D., Yin, J.: Characterization and prediction of popular projects on github. In: 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC) (2019)
9. He, X., Liao, L., Zhang, H., Nie, L., Hu, X., Chua, T.S.: Neural collaborative filtering. In: Proceedings of the 26th International Conference on World Wide Web, pp. 173–182 (2017)
10. Koren, Y.: Factorization meets the neighborhood: a multifaceted collaborative filtering model. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 426–434 (2008)
11. Koskela, M., Simola, I., Stefanidis, K.: open source software recommendations using github. In: Méndez, E., Crestani, F., Ribeiro, C., David, G., Lopes, J.C. (eds.) TPDL 2018. LNCS, vol. 11057, pp. 279–285. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00066-0_24
12. Lian, J., Zhang, F., Xie, X., Sun, G.: Cccfnet: a content-boosted collaborative filtering neural network for cross domain recommender systems. In: Proceedings of the 26th International Conference on World Wide Web Companion, pp. 817–818 (2017)
13. Liu, C., Yang, D., Zhang, X., Ray, B., Rahman, M.M.: Recommending github projects for developer onboarding. IEEE Access **6**, 52082–52094 (2018)
14. Matek, T., Zebec, S.T.: Github open source project recommendation system. arXiv preprint arXiv:1602.02594 (2016)
15. Rendle, S., Krichene, W., Zhang, L., Anderson, J.: Neural collaborative filtering vs. matrix factorization revisited. In: Fourteenth ACM Conference on Recommender Systems, pp. 240–248 (2020)
16. Schmidhuber, J.: Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015)
17. Sun, X., Xu, W., Xia, X., Chen, X., Li, B.: Personalized project recommendation on github. Sci. China Inf. Sci. **61**(5), 050106 (2018)
18. Xu, W., Sun, X., Hu, J., Li, B.: Repersp: recommending personalized software projects on github. In: IEEE International Conference on Software Maintenance & Evolution (2017)
19. Xu, W., Sun, X., Xia, X., Chen, X.: Scalable relevant project recommendation on github. In: Proceedings of the 9th Asia-Pacific Symposium on Internetware, pp. 1–10 (2017)
20. Xu, Z., et al.: Tstss: a two-stage training subset selection framework for cross version defect prediction. J. Syst. Softw. **154**, 59–78 (2019)
21. Xue, H.J., Dai, X., Zhang, J., Huang, S., Chen, J.: Deep matrix factorization models for recommender systems. In: IJCAI, vol. 17, pp. 3203–3209. Melbourne, Australia (2017)
22. Yu, L., Mishra, A., Mishra, D.: An empirical study of the dynamics of github repository and its impact on distributed software development. In: OnTheMove (OTM 2014) (2014)
23. Zhang, L., Zou, Y., Xie, B., Zhu, Z.: Recommending relevant projects via user behaviour: an exploratory study on github. In: Proceedings of the 1st International Workshop on Crowd-based Software Development Methods and Technologies, pp. 25–30 (2014)
24. Zhang, Y., Lo, D., Kochhar, P.S., Xia, X., Li, Q., Sun, J.: Detecting similar repositories on github. In: IEEE International Conference on Software Analysis (2017)