SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

May 2018, Vol. 61 000000:1–000000:14
https://doi.org/10.1007/s11432-017-9419-x

Personalized project recommendation on GitHub

Xiaobing SUN[1,2,5*], Wenyuan XU[1], Xin XIA[3], Xiang CHEN[4] & Bin LI[1]

[1]*School of Information Engineering, Yangzhou University, Yangzhou 225007, China;*
[2]*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China;*
[3]*Faculty of Information Technology, Monash University, Melbourne 3800, Australia;*
[4]*School of Computer Science and Technology, Nantong University, Nantong 226019, China;*
[5]*Information Technology Research Base of Civil Aviation Administration of China, Civil Aviation University of China, Tianjin 300300, China*

Received 6 November 2017/Accepted /Published online 27 April 2018

Abstract GitHub is a software development platform that facilitates collaboration and participation in project development. Typically, developers search for relevant projects in order to reuse functions and identify useful features. Recommending suitable projects for developers can save their time. However, finding suitable projects among many projects on GitHub is difficult. In addition, different users may have different requirements. A recommendation system would help developers by reducing the time required to find suitable projects. In this paper, we propose an approach to recommend projects that considers developer behaviors and project features. The proposed approach automatically recommends the top-N most relevant software projects. We also integrate user feedback to improve recommendation accuracy. The results of an empirical study using data crawled from GitHub demonstrate that the proposed approach can efficiently recommend relevant software projects with relatively high precision.

Keywords software recommendation, developer behavior, GitHub, user feedback, personalized recommendation

Citation Sun X B, Xu W Y, Xia X, et al. Personalized project recommendation on GitHub. Sci China Inf Sci, 2018, 61(5): 000000, https://doi.org/10.1007/s11432-017-9419-x

1 Introduction

GitHub is a popular open-source platform that facilitates collaboration and participation in software development [1]. In GitHub, developers initiate all software repositories, i.e., they create repositories for their own projects and can participate in and search for other projects [2, 3]. When developing a project, most developers search for similar GitHub projects. By finding similar projects, software engineers can identify reusable source code, obtain help building prototypes, and identify alternative implementations and innovations [4]. Many features have been implemented in projects hosted on GitHub, and developers can simply refactor those features to satisfy their requirements and spend more time implementing functions not available on GitHub [3, 4]. Studies have shown that 42% of developers agree that an automated recommendation tool would be useful [5]. Many software projects on GitHub progress slowly because few developers know about them. In addition, similar projects are initiated and developed repeatedly by different developers, which is a waste of time and effort. An automatic recommendation system would help address these issues.

* Corresponding author (email: xbsun@yzu.edu.cn)

Generally, developers use search engines (e.g., Google or Bing) to find similar projects. However, search engines typically focus on text matching based on similarity measures [6, 7]. A small number of keywords may not fully describe the characteristics of a software project. In addition, selecting suitable keywords may be difficult [8]. Some previous studies have focused on software project recommendation systems. However, such systems tend to recommend software projects based on project descriptions or source code and do not consider personalized developer requirements [4, 6, 9]. Different developers may have different requirements; thus, the recommendation results may be inaccurate, which wastes time and causes developers to mistrust such systems. Moreover, the recommendation process is always a single click, i.e., click, which considers few about developers' own needs.

To address these challenges, we propose a personalized and interactive recommendation approach that considers developer behavior and the features of software projects. For developer behaviors, we consider various actions, i.e., create, fork, and star. We also utilize user feedback to improve recommendation accuracy. For project features, we analyze and extract terms from a project's description documents and source code [10]. The proposed approach integrates developer behavior and project features to recommend the top-N most relevant projects .

Our approach compensates the disadvantages of search engines. In the early stage of a project, developers often search for similar projects using keywords; however, the results are often inaccurate. A small number of keywords may return a huge number of results and relevant projects may be omitted. Nevertheless, developers will fork or star some projects that do not entirely satisfy their requirements. The proposed recommendation system helps developers find more relevant projects, and its content-based recommendation results typically include unpopular projects, which helps such projects receive more attention. In addition, our recommendations are personalized (i.e., developer-oriented), which differs from existing project-oriented methods.

The proposed approach recommends projects based on developer behavior and the projects in their repository rather than a particular project (on GitHub, it is called a repository). In practice, it is generally difficult for developers to find suitable projects at the beginning, and identifying suitable projects using a search engine is time-consuming because search results tend to include a vast number of projects. However, by considering the user's focus (i.e., the weight of their behavior) and the content of all projects (README files and source code), the proposed approach can return more relevant results. Moreover, many real-world projects do not comprise a single repository (e.g., a website may have a back-end repository developed in Java and a front-end repository developed using Node.js). Thus, a single repository may not fully represent a given developer's skills and experience.

We evaluate our approach with four groups of data sets which represent three different development areas and a mixed one from GitHub. We compare our approach with the content-based recommendation without user behavior and two state-of-the-art recommendation algorithms, i.e., user-based collaborative filtering (UCF) and item-based collaborative filtering (ICF) [11]. The results show that the accuracy of top 5 project recommendation in four groups is 63.16%, 71.43%, 65.85%, and 25.61%, respectively, which improves the baseline by a substantial margin.

2 Preliminaries

Here, we present the preliminaries used to support the proposed approach, i.e., term frequency-inverse document frequency (TF-IDF) and simulated annealing (SA).

2.1 TF-IDF

TF-IDF is typically used to measure the importance of a word in a document relative to a corpus.

Here, assume a collection of N documents. Let TF_{ij} be the frequency (number of of occurrences) of term (word) i in document j. Then, TF_{ij} is defined as

$$\mathrm{TF}_{ij} = \frac{n_{ij}}{\sum_{k \in j} n_{kj}}, \tag{1}$$

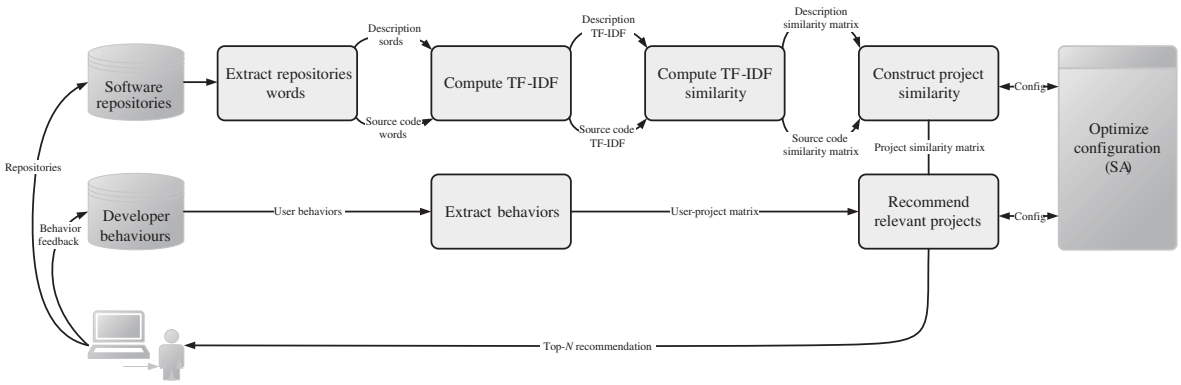

Figure 1 (Color online) Overview of the architecture of our approach.

where n_{ij} is the number of times word i appears in document j. The denominator is the sum of all words appearing in the document j.

IDF measures how much information the word provides, which means whether the word is common or rare in all documents. The IDF of a word i in the corpus N is defined as

$$\mathrm{IDF}_{iN} = \log\left(\frac{|N|}{|\{d|d \in N, i \in d\}|}\right), \tag{2}$$

where $|N|$ indicates the total number of documents in the corpus N, and $|\{d|d \in N, i \in d\}|$ is the number of documents where the word i appears.

TF-IDF is calculated as $\mathrm{TF}_{ij} \times \mathrm{IDF}_{iN}$. Based on TF-IDF results, more important words in a document can be identified.

In this study, we calculate the TF-IDF of each word in the source code files and project documents.

2.2 Simulated annealing

SA is a probabilistic technique to approximate the global optimum of a given function in a large search space [12]. At each step, the SA heuristic considers the neighboring state of the current state and probabilistically determines whether to move to the neighboring state or remain in the current state. Generally, the goal is to move to states with lower energy. This step is repeated until the system reaches a state that is sufficient for the given application scenario or until a given computation budget is exhausted.

In this paper, we use SA to automatically tune the parameters during the recommendation process.

3 Proposed approach

An architectural overview of the proposed approach is shown in Figure 1. First, we extract keywords from each repository and calculate a project similarity matrix. Then, we extract developer behaviors to form a user-project matrix. Finally, we combine the similarity and user-project matrices by leveraging SA to obtain optimal parameters to produce the top-N software project recommendations.

3.1 Project feature extraction

In GitHub, each project has its own repository, and each repository contains different types of files, such as source code and description files. For example, description files show the information and usage of a project, and source code files show a program's implementation. First, we extract project features from these files in each repository.

Before extracting features, files should be preprocessed [13] e.g., to remove noisy information, such as meaningless words (e.g., "you", "is", "are"). Description documents can be identified by their suffixes, such as "md" and "txt", which are written to introduce the project for other developers to understand.

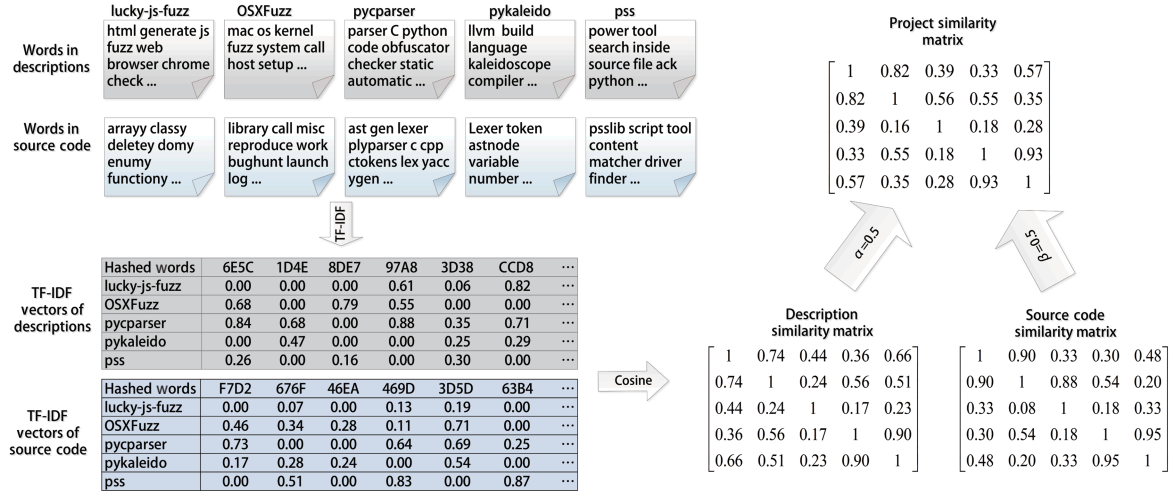

Figure 2 (Color online) Procedure to extract features and calculate similarity.

In addition, we also remove the code in these files. Finally, we obtain a list of words from the project description documents.

For source code files, we first remove numbers and escape characters because these are meaningless relative to measuring similarity between projects. We also filter out words with fewer than three letters. Then, we transform compound words (i.e., identifiers) into single words based on two common formats used to define identifiers, i.e., camel-case and underline-case. Finally, we obtain a list of words from the source code files. Source code also contains comments that help developers understand the code. We add the words in the comments to the list of source code words. After obtaining lists of words from the description documents and source code, we extract project features based on TF-IDF. Here we calculate a word vector that indicates project features. We use two vectors, i.e., a TF-IDF description vector and a TF-IDF source code vector, to represent a project's characteristics. Note that we cannot compare the similarity of vectors with different lengths. Therefore, we must use a hash table and map each word to a hash integer value to unify the length of the TF-IDF vectors.

After obtaining project features, we calculate the similarity between projects. First, we consider the project's language. For projects developed in different languages, we set the similarity to zero. Then, we calculate the similarity between projects based on the description document and source code vectors. Here, we use cosine similarity to calculate these two similarity matrices:

$$\text{similarity}(a,b) = \frac{\text{tfidf}_a \cdot \text{tfidf}_b}{\|\text{tfidf}_a\|_2 \|\text{tfidf}_b\|_2}. \tag{3}$$

Here, tfidf_a denotes the TF-IDF vector of project a and tfidf_b denotes the TF-IDF vector for project b.

Then, we obtain two matrices. One matrix represents similarity between description documents, and the other represents similarity between the source code. We combine these two similarity values with different weights as follows:

$$\begin{aligned} &\text{SIM}(a,b) = \alpha \cdot \text{SIM}_{\text{doc}}(a,b) + \beta \cdot \text{SIM}_{\text{src}}(a,b), \\ &\text{s.t.} \quad \alpha + \beta = 1. \end{aligned} \tag{4}$$

We use α to represent the weight of description documents and β to represent the weight of the source code. Different weights show different ability of documents and source code to reflect the similarity of projects. However, the best optimal α and β values are difficult to determine. Thus, we employ an automatic configuration algorithm to address this issue (Subsection 3.4).

Figure 2 shows the procedure used to extract project features and calculate similarity. This example involves five projects, i.e., lucky-js-fuzz, OSXFuzz, pycparser, pykaleido, and pss. First, we extract the words from these projects' descriptions and source code. Then, we calculate the description and source

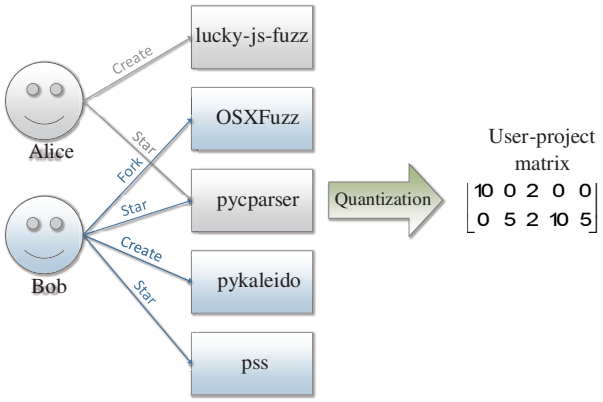

Figure 3 (Color online) Modeling user behaviors.

code TF-IDF vectors for each project. Finally, we calculate the description and source code similarity based on these vectors and sum them using different weights. Here, assume $\alpha = 0.5$ and $\beta = 0.5$. With these values, we obtain a similarity matrix for these projects shown in the bottom-right of Figure 2.

3.2 User behavior extraction

Developers can perform various behaviors on GitHub, such as create, fork, and star. The create behavior is directly related to a user's own project; thus, we consider it first. Several studies have shown that these behaviors can reflect developer demand for different projects [4, 5]. For example, starring a project may indicate that he/she is interested in this project, and forking a project indicates that the project is urgently needed. Therefore, we consider create, fork, and star behaviors when determining recommendations. We can obtain these user behaviors using the GitHub API[1)] [14]. Alternatively, we can identify created, forked, and starred projects on a user's GitHub page[2)3)].

In addition, we consider developer feedback. Developers can provide feedback with like or dislike about our recommendation results, which indicates whether the recommendation is valid. Such feedback can be used to improve future recommendation results.

Note that we assign different values to different behaviors. The create behavior is assigned the highest value (10), followed by the fork (5) and star (2) behaviors. For developer feedback, we set like and dislike behaviors to values of 1 and -3, respectively. Thus, each behavior can be quantified as the triple $\langle$user_id, project_id, value$\rangle$, where value reflects the importance of the given project (project_id) to the given developer (user_id). Collectively, these behaviors form a user-project matrix UP where $\mathrm{UP}(a, b)$ represents the behavior values of user a for project b. In GitHub, a user may perform more than one behavior for a given project.

In this case, we select the behavior with the greatest value. We use the example shown in Figure 3 to explain how user behaviors are extracted. Here, Alice created lucky-js-fuzz and stared pycparser, and Bob created pykaleido, forked OSXFuzz, stared pycparser and pss. After assigning different values, we can obtain the user-project matrix, as shown on the right side of Figure 3.

3.3 Software project recommendation

Based on the project similarity matrix and the user-project matrix, we use a demand measure to predict a user's need for unknown projects. The demand measure is defined as follows:

$$\mathrm{demand}(u, p) = \sum_{i \in \mathrm{repo}(u) \cap \mathrm{TopK}(p,k)} \mathrm{UP}(u, i) \times \mathrm{SIM}(i, p). \tag{5}$$

1) https://developer.github.com/.
2) https://github.com/⟨username⟩?tab=repositories.
3) https://github.com/⟨username⟩?tab=stars.

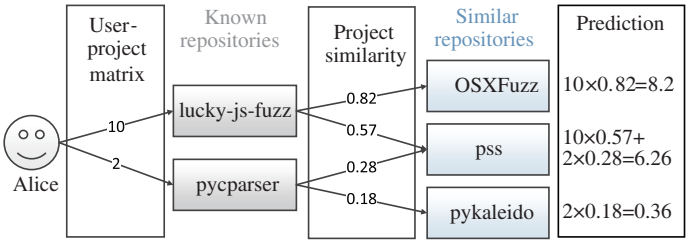

Figure 4 (Color online) Example project recommendation.

Here, $i \in \text{repo}(u)$ represents repositories that the given user knows, and $\text{TopK}(p, k)$ is a set of the top-k relevant repositories for project p. $\text{UP}(u, i)$ represents user u's demand for project i, and $\text{SIM}(i, p)$ is the similarity value between projects i and p. Thus, the sum of $\text{UP}(u, i) \times \text{SIM}(i, p)$ can be used to predict the user's need for unknown projects. We recommend top-N projects ranked by demand values to obtain personalized recommendation results.

Figure 4 shows an example of this procedure. Here, we obtain Alice's behaviors from the user-project matrix, and we set $k = 2$. From the project similarity matrix, the two projects most similar to lucky-js-fuzz are OSXFuzz and pss. In addition, pss and pykaleido are the most similar to pycparser. Then, we can predict the degrees of Alice's needs for these projects, as shown in the right rectangle in Figure 4. Finally, the top-N recommendations are given for each user. In this example, the two most similar projects for Alice are pss and pykaleido.

3.4 Optimization

The recommendation process involves various parameters, such as α and β. Note that these parameters may differ for different software projects or developers. Rather than configuring parameters manually, we use the SA algorithm to configure parameters automatically.

$$\max \quad f(\alpha, \beta, k), \tag{6}$$

$$\text{s.t.} \quad 0 \leqslant \alpha \leqslant 1, \tag{7}$$

$$0 \leqslant \beta \leqslant 1, \tag{8}$$

$$\alpha + \beta = 1, \tag{9}$$

$$0 < k \leqslant 5000. \tag{10}$$

As shown in (6), the goal is to maximize recommendation accuracy. Here, f is a function to evaluate the given parameter configuration. This can be evaluated using the precision, accuracy, or a mixed measures. In this study, the goal is to achieve high precision. Here, three parameters are configured, i.e., document weight α, source code weight β, and the k most similar repositories. In addition, four constraints, Eqs. (7)–(9) are considered in the recommendation process.

Our parameter optimization algorithm takes a dataset and a default configuration as input. The output is the optimal configuration. When the algorithm is initialized, it reads the default configuration and computes the result for comparison. Note that we set the initial temperature to 10. Then, different configurations are input continuously until the temperature reaches zero. This algorithm is executed ten times for each temperature point. In each run, the algorithm first generates a new solution, which produces a new configuration that satisfies the given constraint. Then, we use an evaluation function (equivalent to an energy function) to calculate the result. As stated previously, our goal is to obtain high recommendation accuracy. If the result of the new solution is better than the old solution, the configuration will be accepted. Otherwise, the result may be accepted; however, the chance will decrease as the temperature decreases.

4 Empirical study

4.1 Research questions

In this study, we consider the following research questions:

Table 1 Statistics of four groups of GitHub data

Group name	Users	Projects	Development areas
vim-jp	22	562	Vimscript
Formidable	16	185	Web
harvesthq	43	540	Android
Large	1621	20367	/

RQ1. Does the proposed approach outperform state-of-the-art recommendation approaches?

RQ2. To what extent can the proposed approach improve the effectiveness of recommendations by considering user behaviors?

RQ3. Is it necessary to extract project features from the description documents and source code?

RQ4. To what extent can the proposed approach improve effectiveness by considering user feedback?

4.2 Empirical environment

We implemented the proposed approach using Python 2.7.9 and Apache Spark 2.1. Apache Spark was employed because there are millions of projects on GitHub and one computer cannot calculate them.

4.3 Parameter setting

The proposed approach can optimize parameters automatically, and the parameters were set such that we could address RQ1, RQ2, and RQ4. However, we had to manually set some parameters to verify that the obtained parameters values did in fact affect the experimental results. Parameters α and β represent the weight of the description documents and source code. Relative to RQ3, we set α from 0 to 1 with a step of 0.1; thus, β was set between 1 to 0 to determine the influence of these parameters on accuracy. Parameter N represents the top-N similar programs to recommend. For RQ1, RQ2 and RQ4, we set N to 3, 5, and 10, and for RQ3, we set N to 5.

4.4 Datasets

Due to the limited usage count of the GitHub API, we used a crawler to fetch and sort user behavior and repositories into four groups as our datasets from GitHub user pages. The first three groups were extracted from three organizations on GitHub: vim-jp[4)], Formidable[5)], and harvesthq[6)]. vim-jp is a vim community that primarily uses Vimscript to develop vim plugins. Formidable focuses on web application development, such as PHP and CSS development, and harvesthq focuses on Android application development. Obviously, the proposed recommendation system performs well on small-scale datasets (you can see this in Subsection 5.1). Thus, we also selected a fourth dataset to determine whether the proposed approach can be applied to different development areas among the first three groups.

We first crawled all developers in these three organizations, and then crawled the projects these developers created, forked, and starred[7)]. In the last group, we crawled 1621 active GitHub users (each user had at least 10 software repositories on GitHub) and 20367 related repositories. We used these groups to test the effect of the proposed system in a real environment. Here, user behavior was stored in the triple ⟨user_id, item_id, behavior⟩, and we randomly divided 60% of the user behavior data into a training set and 40% into a test set. Note that UCF [15], ICF [16], and the proposed approach used the same training and test sets; however, the proposed approach used project content, which UCF and ICF cannot use. Each experiment was repeated ten times, and we used the average value as the final result. The details of each group are shown in Table 1.

4) https://github.com/vim-jp.
5) https://github.com/FormidableLabs.
6) https://github.com/harvesthq.
7) The data were crawled on 6th October, 2016. The number of developers in these three organizations might be changed.

Table 2 Evaluation metrics

Metric	Formula
Accuracy	$\left\|\{u\|u \in U, R(u) \cap T(u) \neq \emptyset\}\right\| / \left\|U\right\|$
Recall	$\left\|R(u) \cap T(u)\right\| / \left\|T(u)\right\|$
Precision	$\left\|R(u) \cap T(u)\right\| / \left\|R(u)\right\|$
$F1$	$2 \cdot \text{precision} \cdot \text{recall} / (\text{precision} + \text{recall})$

4.5 Methods and evaluation metrics

4.5.1 *Methods*

For RQ1, we compared our recommendation results to those of UCF [16] and ICF [16]. UCF builds a user similarity database based on user behavior, which involves two steps. First, it identifies developers (i.e., neighbors) who share the same behavior patterns as the candidate developer. Then, the behaviors of the neighbors identified in the first step are used to calculate a prediction value for the candidate developer. ICF is similar to UCF; however, it is item-centric, which also has two steps. First, it builds an item-item matrix to compute similarity values between pairs of items based on user behavior. Then, recommendation is performed by examining the matrix and matching the given user's data.

For RQ2, we designed a control group that ignores user behaviors, i.e., we set the weight of user behaviors to 1. We then compared the results of the proposed approach, which considers user behaviors, and those obtained based on the control group.

For RQ3, we changed the value of parameter α from 0 to 1 with a step of 0.1, thus β value from 1 to 0, and observed the changes in accuracy of the proposed approach.

For RQ4, we first selected 60% of the data as a training set and 40% as a test set. Then, we simulated feedback for the recommendations obtained using the test data to obtain new recommendations in consideration of the simulated feedback. We compared the accuracy of the second recommendation results to that of the first recommendations. Since positive feedback (a like) will not appear in the recommendation results again, positive feedback would interfere with our evaluation. Therefore, we only considered negative feedback. In other words, when the first recommendation (e.g., we recommend project i to user u) does not in the test data, we judge that user u dislikes project i. Typically, a user will not give feedback on all recommendations; thus, we only simulated feedback for 80% of the recommendation results.

4.5.2 *Evaluation metrics*

We used the accuracy, recall, precision, and $F1$ metrics to answer the four research questions, respectively. The equations for these four metrics are shown in Table 2.

U represents all users in the test data, and $R(u)$ represents the number of relevant project repositories recommended to user u by the proposed recommendation approach. $T(u)$ represents the number of relevant project repositories of user u in the test data that were extracted from the remaining 40% of user behaviors.

5 Empirical results

5.1 RQ1

We first discuss the accuracy of our recommendation results and compare the results to those of the UCF and ICF methods.

The empirical results are shown in Table 3. The results show that the accuracy, recall, precision, and $F1$ values of the proposed approach are significantly higher than those of the UCF and ICF methods, both of which show poor results with the first three groups because the amount of data relative to user behavior and the number of projects is relatively small, which made the user-project matrix sparse. Thus, UCF cannot accurately find similar developers based on user behavior, and ICF cannot well use it to calculate similarity. With the proposed approach, we can calculate project similarity using descriptions and the

Table 3 The empirical results of our approach, UserCF and ItemCF

		Accuracy			Recall			Precision			$F1$		
		Top3	Top5	Top10	Top3	Top5	Top10	Top3	Top5	Top10	Top3	Top5	Top10
vim-jp	Our	**63.16%**	**63.16%**	**78.95%**	**12.44%**	**15.92%**	**25.87%**	**43.86%**	**33.68%**	**27.37%**	**19.38%**	**21.62%**	**26.60%**
	UCF	5.37%	6.46%	8.60%	0.44%	0.55%	0.82%	2.32%	1.89%	1.48%	0.73%	0.84%	1.04%
	ICF	4.26%	6.98%	8.62%	0.39%	0.63%	0.90%	1.88%	2.26%	1.69%	0.63%	0.98%	1.16%
Formidable	Our	**57.14%**	**71.43%**	**78.57%**	**18.52%**	**27.78%**	**40.74%**	**23.81%**	**21.43%**	**15.83%**	**20.83%**	**24.19%**	**22.80%**
	UCF	6.57%	6.57%	6.57%	1.28%	1.28%	1.28%	1.80%	1.80%	1.80%	1.45%	1.45%	1.45%
	ICF	7.31%	7.31%	7.31%	1.42%	1.42%	1.42%	1.90%	1.90%	1.90%	1.57%	1.57%	1.57%
harvesthq	Our	**58.54%**	**65.85%**	**68.29%**	**16.32%**	**23.01%**	**32.22%**	**32.50%**	**27.50%**	**19.25%**	**21.73%**	**25.06%**	**24.10%**
	UCF	0.00%	4.88%	7.32%	0.00%	0.84%	1.26%	0.00%	2.74%	2.29%	0.00%	1.28%	1.62%
	ICF	7.32%	7.32%	7.32%	1.26%	1.26%	1.26%	3.53%	2.16%	1.12%	1.85%	1.59%	1.19%
Large	Our	**23.10%**	**25.61%**	**26.90%**	**3.17%**	**4.21%**	**5.01%**	**13.54%**	**10.80%**	**6.43%**	**5.13%**	**6.06%**	**5.63%**
	UCF	20.03%	24.68%	24.95%	1.80%	2.53%	4.02%	9.22%	7.78%	6.21%	3.01%	3.82%	4.88%
	ICF	3.58%	4.65%	6.80%	0.31%	0.41%	0.65%	1.40%	1.11%	0.89%	0.50%	0.60%	0.75%

Table 4 Comparison of results obtained with and without considering user behavior

		Accuracy			Recall			Precision			$F1$		
		Top3	Top5	Top10	Top3	Top5	Top10	Top3	Top5	Top10	Top3	Top5	Top10
vim-jp	NoBHV	63.16%	63.16%	78.95%	12.44%	15.87%	25.37%	43.86%	33.68%	27.37%	19.38%	21.62%	26.33%
	BHV	67.63%	63.16%	78.95%	12.44%	15.92%	25.87%	43.86%	33.68%	27.37%	19.38%	21.62%	26.60%
	Gain	**7.08%**	**0.00%**	**0.00%**	**0.00%**	**0.31%**	**1.98%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**1.02%**
Formidable	NoBHV	50.00%	71.43%	78.57%	14.81%	24.07%	39.59%	19.05%	18.57%	15.55%	16.67%	20.97%	22.33%
	BHV	57.14%	71.43%	78.57%	18.52%	27.78%	40.74%	23.81%	21.43%	15.83%	20.83%	24.19%	22.80%
	Gain	**14.29%**	**0.00%**	**0.00%**	**25.00%**	**15.38%**	**2.90%**	**25.00%**	**15.38%**	**1.80%**	**25.00%**	**15.38%**	**2.11%**
harvesthq	NoBHV	51.22%	60.98%	65.85%	15.06%	21.34%	31.80%	30.00%	25.50%	19.00%	20.06%	23.23%	23.79%
	BHV	58.54%	65.85%	68.29%	16.32%	23.01%	32.22%	32.50%	27.50%	19.25%	21.73%	25.06%	24.10%
	Gain	**14.29%**	**8.00%**	**3.70%**	**8.33%**	**7.84%**	**1.32%**	**8.33%**	**7.84%**	**1.32%**	**8.33%**	**7.84%**	**1.32%**
Large	NoBHV	20.24%	22.53%	25.39%	2.78%	3.68%	4.66%	11.87%	9.46%	5.98%	4.50%	5.30%	5.24%
	BHV	23.10%	25.61%	26.90%	3.17%	4.21%	5.01%	13.54%	10.80%	6.43%	5.13%	6.06%	5.63%
	Gain	**14.13%**	**13.65%**	**5.92%**	**14.06%**	**14.22%**	**7.54%**	**14.06%**	**14.22%**	**7.54%**	**14.06%**	**14.22%**	**7.54%**

source code, as well as user behavior. This greatly improves the accuracy of our recommendations. For example, in the top 10 recommendations of the Formidable group, the accuracy of the proposed approach reached 78.95%, and, in the worst case, accuracy was 68.29%. This means that greater than two-thirds of the users received at least one useful recommendation.

However, in the last group, as the number of users and projects increased, user behavior also increased. On one hand, this allowed the UCF method to find similar users, which greatly improved its accuracy. On the other hand, the recommendation system must give the top-N recommendation results for each user, which greatly reduced recall. Nevertheless, our algorithm is still better, especially for the top 3 and 5 recommendations.

Therefore, the results demonstrate that, compared to the UCF and ICF methods, the proposed approach gives more accurate results for developers based on their behaviors and the characteristics of the software projects.

5.2 RQ2

Table 4 shows the results of the proposed approach (BHV) compared to recommendations that did not consider user behavior (NoBHV). As can be seen, the accuracy, recall, and precision results of NoBHV were worse than those obtained by the proposed approach, which does consider user behavior. For example, when recommending the top 3 software projects, considering user behavior greatly improved the results. This means that different developer behaviors represent their unique demands relative to the same repository.

However, because there is less user behavior information in small datasets, the improvement is limited. For large datasets, there are millions of data relative to user behavior.

Thus, we must consider different user intentions through such different behaviors. Therefore, compared to recommendations that do not consider user behavior, the proposed approach can recommend more

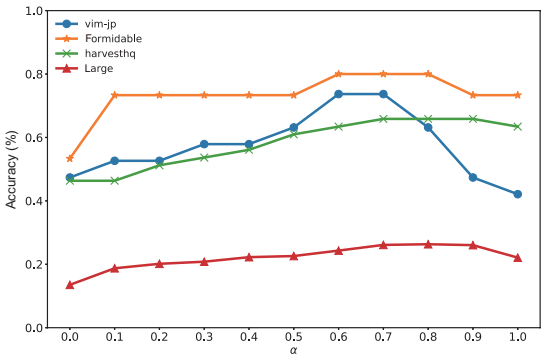

Figure 5 (Color online) Accuracy of each group.

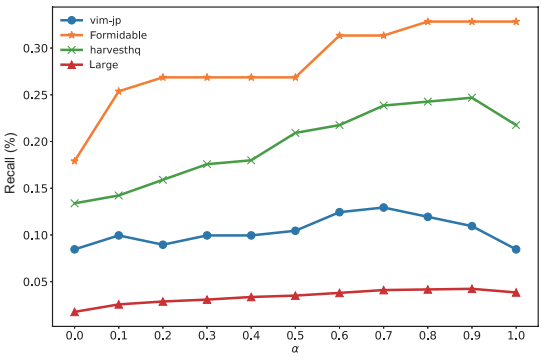

Figure 6 (Color online) Recall of each group.

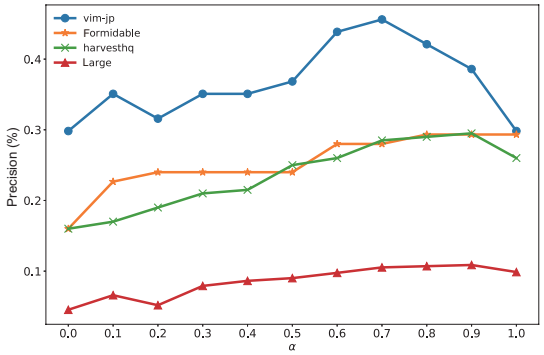

Figure 7 (Color online) Precision of each group.

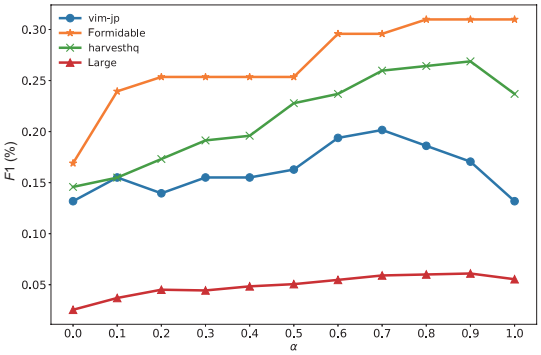

Figure 8 (Color online) $F1$ of each group.

accurate software projects to developers by considering their behavior.

5.3 RQ3

This subsection examines whether it is necessary to analyze both the description documents and source code to extract the features of a GitHub project and how to set their weights (α and β).

Figure 5 shows that α and β have little effect on accuracy. However, when $\alpha = 0$ (i.e., only the source code is considered) and $\beta = 0$ (i.e., only the description document is considered), accuracy was not the highest. This means that the features of a software project cannot be extracted from only the description or source code information alone. Based on the results shown in Figures 6–8, when $\alpha \in [0.6, 0.9]$ (i.e., $\beta \in [0.1, 0.4]$.), their values are relatively high. In addition, when α increased, these values increased initially; however, when α increased to 0.6, the rate of increase slowed. When α achieved 0.9, these values began to decline.

Based on the accuracy, precision, recall, and $F1$ results with different α and β values, we observed that both the descriptions and source code can reflect the characteristics of a project. Therefore, it is necessary to extract the features of a software project using both description documents and source code. In addition, the experimental results show that the optimal parameter configurations for different datasets differ. To better support developers, we use SA to automatically optimize our parameter configuration for automatic recommendations.

5.4 RQ4

Another advantage of the proposed approach is that we can recommend more accurate results based on the behavior of another user behaviour, i.e., user feedback. Table 5 shows that such feedback improved the recommendation results, especially for the top 3 and 5 recommendations. Relative to RQ1, we learned that the accuracy for a large group is not very high. However, after we receive feedback, our

Table 5 Improvement with 80% feedback

		Accuracy			Recall			Precision			$F1$		
		Top3	Top5	Top10	Top3	Top5	Top10	Top3	Top5	Top10	Top3	Top5	Top10
vim-jp	Before	63.16%	63.16%	78.95%	12.44%	15.92%	25.87%	43.86%	33.68%	27.37%	19.38%	21.62%	26.60%
	Feedback	78.95%	89.47%	94.74%	14.93%	21.89%	37.81%	52.63%	46.32%	40.00%	23.26%	29.73%	38.87%
	Gain	**25.00%**	**41.67%**	**20.00%**	**20.00%**	**37.50%**	**46.15%**	**20.00%**	**37.50%**	**46.15%**	**20.00%**	**37.50%**	**46.15%**
Formidable	Before	57.14%	71.43%	78.57%	18.52%	27.78%	40.74%	23.81%	21.43%	15.83%	20.83%	24.19%	22.80%
	Feedback	64.29%	78.57%	78.57%	22.22%	37.04%	46.30%	28.57%	28.57%	17.99%	25.00%	32.26%	25.91%
	Gain	**12.50%**	**10.00%**	**0.00%**	**20.00%**	**33.33%**	**13.64%**	**20.00%**	**33.33%**	**13.64%**	**20.00%**	**33.33%**	**13.64%**
harvesthq	Before	58.54%	65.85%	68.29%	16.32%	23.01%	32.22%	32.50%	27.50%	19.25%	21.73%	25.06%	24.10%
	Feedback	73.17%	75.61%	80.49%	22.59%	29.71%	39.75%	45.00%	35.50%	23.75%	30.08%	32.35%	29.73%
	Gain	**25.00%**	**14.81%**	**17.86%**	**38.46%**	**29.09%**	**23.38%**	**38.46%**	**29.09%**	**23.38%**	**38.46%**	**29.09%**	**23.38%**
Large	Before	23.10%	25.61%	26.90%	3.17%	4.21%	5.01%	13.54%	10.80%	6.43%	5.13%	6.06%	5.63%
	Feedback	28.68%	30.83%	32.83%	3.83%	5.02%	6.24%	16.38%	12.88%	8.02%	6.21%	7.22%	7.02%
	Gain	**24.15%**	**20.39%**	**22.07%**	**20.95%**	**19.21%**	**24.58%**	**20.95%**	**19.21%**	**24.65%**	**20.95%**	**19.21%**	**24.61%**

recommendation system knows more about user preferences and can modify its recommendations to satisfy more user demands.

User feedback is useful because it reflects a user's individual needs. Even if the content of some projects is similar, users typically require different projects for practice, and such feedback can reflect this demand.

6 Validity

6.1 Internal validity

Analysis of description files. We noticed that some projects do not have description files. In such cases, the proposed approach will ignore these projects because even if these projects are recommended, it is difficult for other developers to ascertain their meaning. Note that such projects are difficult to reuse.

Time spent on SA. The parameters can be configured automatically using the SA algorithm. However, to achieve optimal or approximately optimal results, the solution space must be large and significant time is required to find an approximately optimal configuration. Recommendations and automatic configuration of parameters should be asynchronous rather than synchronized.

Simulation of the user feedback. To answer RQ4, we first selected 60% of the data as a training set and 40% as a test set. We then simulated feedback after obtaining recommendations according to the test data in order to recommend new results in consideration of this feedback. In addition, we only considered negative feedback on 80% of the recommendation results. Note that this may differ from practical recommendation processes.

Comparison of experiments. To the best of our knowledge, we are the first to propose a personalized software project recommendation method for GitHub. This makes it difficult to compare the proposed approach to state-of-the-art approaches (i.e., search algorithms). However, in our experiments, we compared the proposed approach to a method that does not consider user behavior, i.e., it calculates project similarity using only TF-IDF, which is used in many search methods.

6.2 Threats to external validity

Programming languages. GitHub contains numerous software projects written in multiple programming languages (e.g., Java, Python, PHP, and C++) or combinations of programming languages. In this study, we set up our experiments using projects with multiple languages, such as Java, Python, and C/C++. In future, we plan to perform more experiments to reduce this threat.

Experiment size. We applied the proposed approach to four GitHub groups. The first three groups represent three different development areas, and the experiment size was similar to a previous study by Zhang et al. [4] that used 50 queries and searched for similar projects from a pool of 1000 projects. However, a small number of projects will result in meaningless recommendations. The fourth group,

which included more than 1621 active users and 20367 repositories, was less affected by this threat. In future, we plan to include more developers and projects.

User behavior. We considered all developer behaviors; however, we did not consider the time at which the behaviors were generated. Therefore, there were some projects in the results that developers had been working on for long periods. Such projects may help developers refactor previous projects; however, many developers may not care about such old projects. Fortunately, they can use feedback to reject such recommendations.

Generalizability. The results obtained by the proposed approach have significant advantages compared to the UCF and ICF methods, which may not be generalizable. However, we emailed the top 10 recommendations to real developers from our large group dataset and received some positive feedback.

User 1. That's a very good list! I could see myself forking/starring elm-lang/Elm, gtk2hs/gtk2hs, Russell91/pyfi, msgpack/msgpack-haskell, ghcjs/ghcjs, tensorflow/Haskell, albertoruiz/hmatrix.

User 2.

- cojs/lock-and-yield. This is somewhat interesting to me, as I've worked on quite a few cojs projects and also haven't heard of this one yet.

- matthewmueller/unyield. This repository is super useful and I can see myself use it in the future! I hope that's of help and you will be able to create something useful for the developer community.

User 3.

- SamVerschueren/alfred-fkill. I have this installed on my system and use it regularly.

- airbrake/airbrake-ruby. I maintain an airbrake gem and while it is different from this one, they are related.

- SamVerschueren/generator-alfred. I've used this project for several of my own alfred extensions.

In future, we will conduct more user studies to demonstrate the generalizability of the proposed approach.

7 Related work

Here, we classify related work into the detection of similar repositories and GitHub studies.

7.1 Detecting similar repositories

McMillan et al. [6] designed the tool CLAN tool, which uses latent semantic indexing to measure the similarity of repositories relative to API usage. Thung et al. [9] proposed a different method that combines SourceForge tags to detect similar repositories. Zhang et al. [4] stated that GitHub users often have sufficient motivation to star a repository when they find it interesting or useful. They further improved CLAN, which can detect similar repositories in GitHub based on two data sources (i.e., GitHub stars and README files). These studies recommend projects based on a project query, i.e., a developer needs to enter a query, and these methods then recommend similar projects. Our study resolves a related but different problem, i.e., we focus on scalable, personalized, and relevant project recommendations. In other words, we consider developer behavior and project similarity to recommend relevant projects, and we do not employ a user query, while these previous tools are similar to a search engine inside GitHub. Moreover, we consider different types of user behaviors in GitHub, and we consider both the README files (documents)[8] and the source code to improve recommendation accuracy.

In our previous study [14, 17], we proposed an approach to recommend personalized software repositories to developers, which is orthogonal to the above studies. The goal of our previous study was is scalable and personalized recommendation. The current study extends that work relative to both technical and empirical perspectives. From a technical perspective, we consider the behavior of other users, i.e., user feedback, to obtain more accurate recommendations. In addition, we employ automatic parameter configuration using the SA algorithm. From an empirical perspective, we conducted the current study using more developers and software projects than our previous study.

8) Not all GitHub projects have README files.

7.2 GitHub studies

Many studies have examined user behavior [3, 5, 18] and GitHub repositories. For example, Jiang et al. [5] studied fork behavior in GitHub. They found that developers fork repositories to pull requests, fix bugs, add new features, and maintain copies. More than 42% of developers think that an automated recommendation tool would be useful to find repositories to fork. Zhang et al. [3] explored the possibilities of finding relevant projects by analyzing user behavior data. They identified four types of user behavior data, i.e., fork, watch, pull-request, and member data, that are suitable for finding relevant projects. In addition, different user behavior datasets are suitable for different recommendation purposes.

Blincoe et al. [18] investigated 800 GitHub users to understand the follow behavior. They found that popularity can be more important than contribution when developers choose to follow other developers. Ray et al. [19] studied more than 700 GitHub projects to determine the relationship between programming languages and software quality.

Differing from these studies, we did not study any data laws or phenomenon relative to GitHub, i.e., we recommend software projects to developers by considering different user behaviors based on some of the empirical results of these previous studies.

8 Conclusion and future work

In this study, we have extended our previous work [14, 17] to recommend more personalized software projects for developers in GitHub. This personalized recommendation is performed based on developer behaviors and features extracted from the description documents and source code of each project. The proposed approach also considers user feedback to improve recommendations. In addition, we optimize parameter configurations automatically using the SA algorithm to achieve higher accuracy. We evaluated the proposed approach using four groups of data from GitHub. The results demonstrate that this approach can accurately recommend software projects to developers.

Although we have demonstrated the effectiveness of the proposed approach, there is room for improvement. First, we must perform additional experiments, particularly survey-based online experiments with actual developers, to determine whether the proposed recommendation approach is actually useful. In addition, we will consider other user behaviors, e.g., search history, watch, and pull-request behaviors, to help developers find more suitable projects.

Acknowledgements This work was supported partially by National Natural Science Foundation of China (Grant Nos. 61472344, 61611540347, 61402396), Open Project Foundation of Information Technology Research Base of Civil Aviation Administration of China (Grant No. CAAC-ITRB-201704), Jiangsu Qin Lan Project, the China Postdoctoral Science Foundation (Grant No. 2015M571489), Open Funds of State Key Laboratory for Novel Software Technology of Nanjing University (Grant No. KFKT2016B21), and Natural Science Foundation of Yangzhou City (Grant No. YZ2017113).

References

1 Sun X B, Yang H, Xia X, et al. Enhancing developer recommendation with supplementary information via mining historical commits. J Syst Softw, 2017, 134: 355–368
2 Sun X B, Li B, Duan Y C, et al. Mining software repositories for automatic interface recommendation. Sci Program, 2016, 2016: 5
3 Zhang L X, Zou Y Z, Xie B, et al. 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, Hong Kong, 2014. 25–30
4 Zhang Y, Lo D, Singh K P, et al. Detecting similar repositories on GitHub. In: Proceedings of the 24rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Klagenfurt, 2017. 13–23
5 Jiang J, Lo D, He J H, et al. Why and how developers fork what from whom in GitHub. Empir Softw Eng, 2017, 22: 547–578
6 McMillan C, Grechanik M, Poshyvanyk D. Detecting similar software applications. In: Proceedings of the 34th International Conference on Software Engineering, Piscataway, 2012. 364–374

7 Sun W S, Sun X B, Yang H, et al. WB4SP: a tool to build the word base for specific programs. In: Proceedings of the 24th IEEE International Conference on Program Comprehension, Austin, 2016

8 Hu J J, Sun X B, Lo D, et al. Modeling the evolution of development topics using dynamic topic models. In: Proceedings of the 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering, Montreal, 2015. 3–12

9 Thung F, Lo D, Jiang L X. Detecting similar applications with collaborative tagging. In: Proceedings of the 28th IEEE International Conference on Software Maintenance (ICSM), Trento, 2012. 600–603

10 Yang C, Fan Q, Wang T, et al. Repolike: personal repositories recommendation in social coding communities. In: Proceedings of the 8th Asia-Pacific Symposium on Internetware, Internetware 2016, Beijing, 2016. 54–62

11 Wang J, de Vries A P, Reinders M. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Seattle, 2006. 501–508

12 Kirkpatrick S, Gelatt C D, Vecchi M P. Optimization by simulated annealing. In: Readings in Computer Vision: Issues, Problems, Principles, and Paradigms. San Francisco: Morgan Kaufmann Publishers, 1983. 671–680

13 Sun X B, Liu X Y, Hu J J, et al. Empirical studies on the NLP techniques for source code data preprocessing. In: Proceedings of the 3rd International Workshop on Evidential Assessment of Software Technologies, Nanjing, 2014. 32–39

14 Xu W Y, Sun X B, Hu J J, et al. REPERSP: recommending personalized software projects on GitHub. In: Proceedings of 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), Shanghai, 2017. 648–652

15 Zhao Z, Shang M. User-based collaborative-filtering recommendation algorithms on Hadoop. In: Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining, Phuket, 2010. 478–481

16 Sarwar B, Karypis G, Konstan J, et al. Item-based collaborative filtering recommendation algorithms. In: Proceedings of the 10th International Conference on World Wide Web, Hong Kong, 2001. 285–295

17 Xu W Y, Sun X B, Xia X, et al. Scalable relevant project recommendation on GitHub. In: Proceedings of the 9th Asia-Pacific Symposium on Internetware, Shanghai, 2017

18 Blincoe K, Sheoran J, Goggins S, et al. Understanding the popular users: following, affiliation influence and leadership on GitHub. Inf Softw Tech, 2016, 70: 30–39

19 Ray B, Posnett B, Filkov V, et al. A large scale study of programming languages and code quality in GitHub. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, Hong Kong, 2014. 155–165