Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

RepoLike: a multi-feature-based personalized recommendation approach for open-source repositories[*]

Cheng YANG[†], Qiang FAN, Tao WANG, Gang YIN, Xun-hui ZHANG, Yue YU, Hua-min WANG

Key Laboratory of Parallel and Distributed Computing, National University of Defense Technology, Changsha 410073, China

[†]E-mail: delpiero710@126.com

Received Feb. 13, 2017; Revision accepted July 20, 2017; Crosschecked Feb. 15, 2018

Abstract: With the deep integration of software collaborative development and social networking, social coding represents a new style of software production and creation paradigm. Because of their good flexibility and openness, a large number of external contributors have been attracted to the open-source communities. They are playing a significant role in open-source development. However, the open-source development online is a globalized and distributed cooperative work. If left unsupervised, the contribution process may result in inefficiency. It takes contributors a lot of time to find suitable projects or tasks from thousands of open-source projects in the communities to work on. In this paper, we propose a new approach called "RepoLike," to recommend repositories for developers based on linear combination and learning to rank. It uses the project popularity, technical dependencies among projects, and social connections among developers to measure the correlations between a developer and the given projects. Experimental results show that our approach can achieve over 25% of hit ratio when recommending 20 candidates, meaning that it can recommend closely correlated repositories to social developers.

Key words: Social coding; Open-source software; Personal recommendation; GitHub
https://doi.org/10.1631/FITEE.1700196 **CLC number:** TP391.7

1 Introduction

The comprehensive integration of social media (Boyd and Ellison, 2007) and software development tools (Begel et al., 2010; Storey et al., 2010) has considerably changed software development in the Internet age. Social media and mechanisms, such as star, watch, and @, provide extremely convenient and low-cost communication channels for large-scale collaborative developments. The crowds' continuous participation and contribution become the key factors for the success of open-source projects. Therefore, many open-source software development communities have shifted their focus from software projects to developers. For example, GitHub proposed a new collaborative development mechanism called "fork," which has created a new social programming concept (Dabbish et al., 2012; Begel et al., 2013). Contributors in such communities can use the provided social tools to discover interesting open-source projects, track code development activities, or comment freely on source codes contributed by others. Numerous software users or open-source enthusiasts are attracted to join the open-source communities. The social and transparent form of collaboration helps them achieve considerable software productivity. As of April 2016,

[*] Project supported by the National Natural Science Foundation of China (Nos. 61432020, 61472430, and 61502512) and the National Key R&D Program of China (No. 2016YFB1000805)
ORCID: Cheng YANG, http://orcid.org/0000-0002-4782-1645

GitHub had hosted more than 35 million software code repositories, attracted more than 14 million online developers to participate, and generated more than 12 million code-merging requests (Yu et al., 2016).

However, such crowd-based software creation behavior (Wang H et al., 2014) is usually conducted in a fully open manner. The massive developers are distributed around the world and are driven by interest in participating in specific development tasks, such as bug fixes, code testings, and documentation improvement. These developers have different personality traits, educational backgrounds, and expertise levels. Furthermore, their participation in open-source projects often does not match with their interests and expertise. Therefore, if we cannot properly guide this mass of social development activities to achieve the best match between public intelligence resources and open-source development tasks, the open-source development will be affected adversely. On one hand, a massive number of contributors may need to spend too much time and effort in finding proper and interesting open-source projects. On the other hand, the migration and withdrawal of developers often increase staff training and project management costs.

We propose a personalized recommendation method called "RepoLike" to recommend open-source projects for developers based on multidimensional features. We quantify the potential correlations between developers and open-source projects from three different dimensions: the popularity of the open-source project, the technical relevance of the project, and the social relevance between the developers. On the basis of these aspects, we propose a learning-to-rank-based approach to conduct personalized recommendation. Specifically, the main contributions of the study are as follows:

1. A technical-interest measurement model is proposed, which quantitatively measures the developers' personal interests in joining repositories from technical and social dimensions.

2. A personalized recommendation algorithm using two different approaches including linear combination and learning-to-rank approach is designed based on the above-mentioned metrics.

3. Extensive experiments were conducted using the GitHub dataset containing 16 249 public contributors and 801 228 repositories. The results verified the effectiveness of RepoLike in qualitative and quantitative ways.

2 Motivation

Online contributors in open-source communities often need to spend much time and effort in selecting proper projects to which they will contribute from numerous candidates. A personalized recommendation system can reduce the cost of this process effectively. However, to achieve this goal we need to solve two problems. First, high-quality projects should be identified efficiently because open-source projects differ significantly in quality. Second, the personal technical interest of contributors should be measured effectively to provide personalized recommendations. In this section, we demonstrate the motivations of our research through an actual case in the GitHub community.

In the GitHub community, the massive open-source projects and developers are interrelated in different ways, such as the relationships "watch and star" (between developer and project repository) and "fork" (between repositories). This kind of correlation information provides an important clue in identifying high-quality repositories. In the groups of software project based on rails (Fig. 1), we can infer that the Diaspora and Paperclip projects should be of high quality as they are socially close to the rails repository. Therefore, the recommendation system may recommend the above-mentioned two projects to amateurs who are interested in projects based on rails.

Apart from the relationship between projects, developers often communicate with each other through community-based social media. Social connections can help personalize recommendations in a different manner.

As shown in Fig. 2, Lain Mitchell and Gemma Cameron are friends as can be inferred from their communications in the open-source community. Therefore, their technical concerns or development interests may be more similar than those with others. Based on this inference, we can recommend software projects to them based on their social connection. For example, we can recommend the project trellopipes, in which Lain Mitchell is currently involved, to Gemma Cameron.

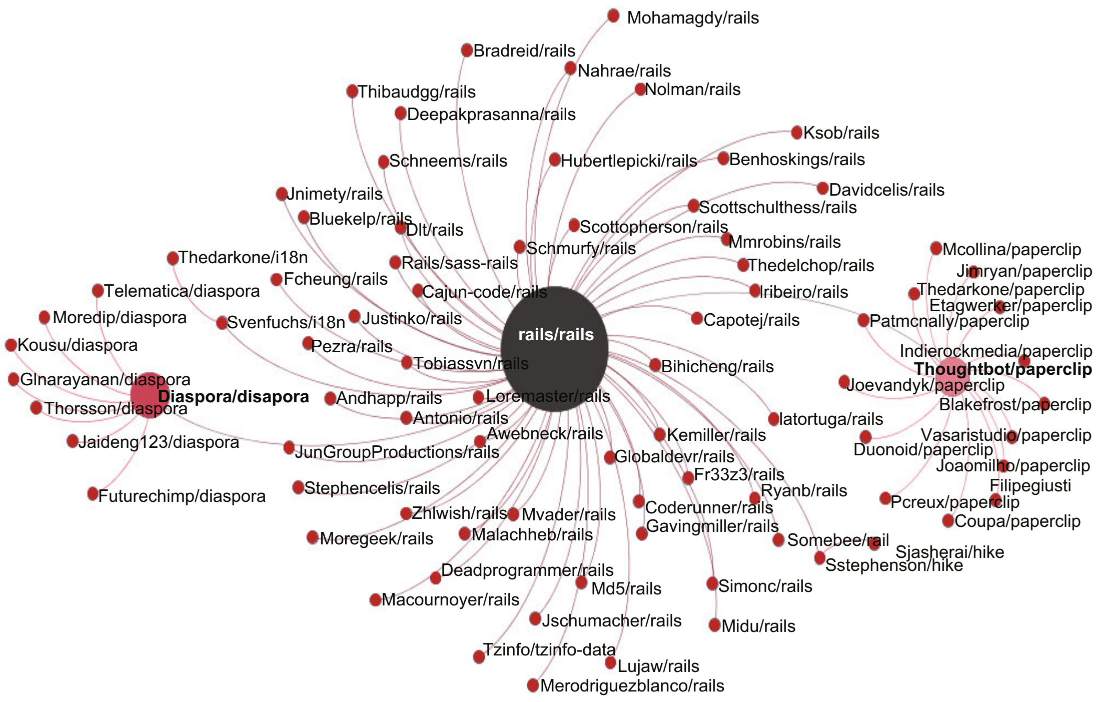

Fig. 1 Fork network of the rails family

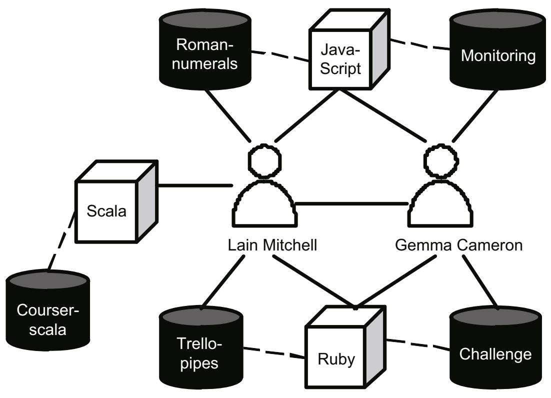

Fig. 2 Context of GitHub developers

3 Personalized recommendation approach

In this section, we present the detailed description of our personalized recommendation approach, including the overall framework, the different dimensionalities of features, and the recommendation algorithm.

3.1 Recommendation framework

Compared with the recommendation mechanisms in online commercial sites, the biggest challenge for software recommendation lies in the difficulties in modeling the developers' personal interest in programming. This is because one may take part in many different projects and the historical data are often highly scattered.

The key idea of our approach is to take full advantage of user data in GitHub, and construct an interest measurement model for each developer based on their joined projects and socially connected developers. Using three dimensions including the project itself, the technical dependency between the projects, and the social association between the developers' measurement models, we can predict the correlations between developers and the open-source projects, and then conduct personalized recommendations. Fig. 3 shows the overall framework of RepoLike, which includes the following four stages:

1. Acquisition of the historical development activities of developers

At this stage, we collect the entire GitHub dataset and aggregate the distributed development historical data of developers. The development historical data collected at this stage are divided into items including watch, issue, issue comment, pull-request, pull-request comment, and fork for

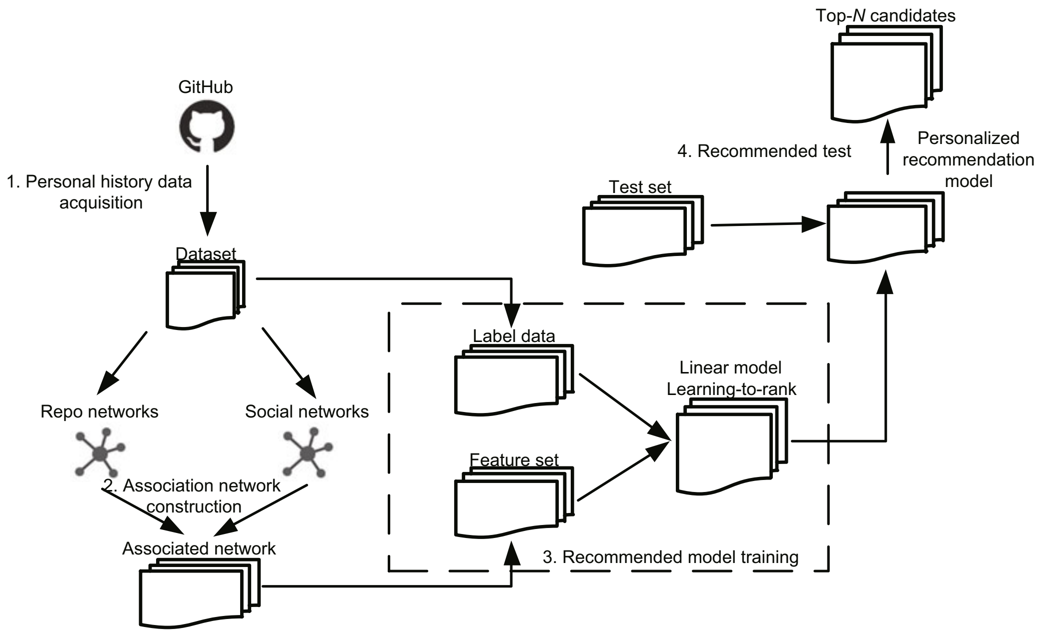
Top-N candidates
GitHub
1. Personal history data acquisition
4. Recommended test
Personalized recommendation model
Test set
Dataset
Label data
Linear model Learning-to-rank
Repo networks
Social networks
Feature set
2. Association network construction
Associated network
3. Recommended model training

Fig. 3 Overview of the personalized recommendation framework

developers. We obtain the other associated information on developers.

2. Construction of the association network

We build the association networks from the project- and developer-centric perspectives. From the project-centric perspective, we build an inter-project and technology-related network based on technical dependencies between projects. The inter-project technology association network reflects the technical relationship between projects. We also build a developer network based on social connections of the contributors in the GitHub community.

The technical concern (e.g., concern about the closely related projects or technology) of a developer is limited to a certain period of time. The project technological association network can be helpful in identifying the closely related projects in which developers may be interested.

Meanwhile, the developers in GitHub who have close interactions are more likely to have similar technical interests. Based on this intuition, we use the social networking of developers as another important factor in the personalized recommendation of projects.

3. Recommendation model training

The proposed model training consists of two steps: ranking of training data and feature extraction of training data. The ranking criterion is the engagement level of developers' participation in projects. Given that developers devote dissimilar time and efforts to different projects, this variation reflects the different preferences of developers. The labeled items can be used as the inputs for training the recommendation model. Feature extraction includes mainly the popularity of the project, technology relevance based on the project's association network, and the social connection based on the developer's association network.

Detailed feature extraction and measurement methods will be described in Section 3.2. After obtaining the order of the training data items and multi-dimensional features, we propose a personalized recommendation method. This method builds a recommendation model from the feature set of the training data, and ranks repositories in a personalized manner. Linear combination and learning-to-rank approaches are used to optimize the features between the parameters and to establish the recommendation model.

4. Recommendation result generation

After training the personalized recommendation model, we can obtain the final recommendation results of each candidate. In our experiment, we input the previously obtained testing candidates and obtain the output top-n candidates to test the performance of our recommendation model.

3.2 Feature extraction and quantitative measurement

We measure the amateur contributors' technical preferences and their correlations with projects on three different dimensions: from project's (repository), task's, and social perspectives. The three dimensions reflect the relationship between the developer and the corresponding project from the perspectives of the project itself, the associated projects, and the associations between the developers. Feature extraction is conducted first, and then the quantification of each dimension.

3.2.1 Project's perspective

One of the key factors for a contributor to determine whether to join an open-source project and to become a long-term contributor or not is the activeness of the project (Zhou and Mockus, 2011). Developers are more likely to participate in open-source projects that are active in the community. This is because such projects may have more potential to become more flourishing than those that are less active or less interesting. Therefore, from the project's perspective, we consider mainly the project's activeness and the interest of the contributors in it.

GitHub provides a watch-and-fork mechanism. A contributor can use the watch-and-fork mechanism to track the project or create its new subbranch if he/she is interested in it. Therefore, the number of watches and forks a project obtains reflects the amount of attention developers paid to it. Specifically, we measure the popularity of projects based on the number of watches and forks.

1. Watch number

The easiest way to track an open-source project is to subscribe to it. Contributors can click on the "watch" button of the project and obtain the latest real-time information. Thus, the number of watches the project obtains can reflect the level of interest of the developers in it.

2. Fork number

Fork is a novel feature provided by GitHub. If a user is interested in a project and wants to participate, then he/she can fork the project to his/her own branch, obtain the source code, and further study or contribute to the code base. Therefore, the popularity of a project can be predicted by the number of forks it obtains.

Based on the above two factors, we quantify the popularity of an open-source project as

$$\mathrm{pp}(j) = \mathrm{nor}(\mathrm{watch}(j)) + \mathrm{nor}(\mathrm{fork}(j)), \tag{1}$$

where $\mathrm{pp}(j)$ refers to the popularity of project j ($j \in \{1, 2, ...\}$), and $\mathrm{watch}(j)$ and $\mathrm{fork}(j)$ represent the numbers of watches and forks of the open-source project, respectively. Watch and fork reflect different levels of developers' attention. Specifically, fork is more comprehensive than watch in terms of developers' participation. Therefore, for accurate measurement, we normalize the numbers of watches and forks instead of adding them directly.

3.2.2 Task's perspective

Software reuse is very common in software development, and most software relies more or less on some other projects or components. Thereby, software reuse introduces technical dependencies among projects. Such technical dependencies form a software ecosystem which reflects the technical relevance among projects.

If a developer is involved in project A and other projects which exhibit a close technical correlation with A, then the developer will possibly be interested in these technically related projects. Based on this intuition, we use a technical dependency network among projects and propose a technology-relevance measurement method based on this network to obtain a quantitative measurement.

Blincoe et al. (2015) comprehensively investigated the technical dependencies between GitHub projects. They explored the links to other projects in text items including issue, pull-request, comment, and submitting information, and viewed these links as evidence of technical dependency. In this study, we build a project-dependent network based on the research data provided by Blincoe et al. (2015). Then we measure the degree of technical relevance between the project the developer participated in and the other technically related projects. Based on this, we can quantify the correlation between the developer and the candidate projects. The specific approach is as follows:

First, we build a dependency network based on the technology dependencies between open-source projects. In this network, each node represents an open-source project. Each edge indicates that a technical dependency exists between two connected items. The weight of the edge indicates the degree of technical dependency, which can be measured by the number of connections between them.

Second, we build a sub-dependency network which focuses on the projects a specific developer used to participate in based on the dependency network. In this sub-network, the developer is the center and is connected directly to all open-source projects that he/she participated in.

For example, Fig. 4 presents a case in which the developer is involved directly in the Repo$_1$, Repo$_2$, Repo$_3$, and Repo$_4$ projects. Technical dependencies are found between Repo$_1$ and Repo$_2$. The numbers of links between the two projects are two and five, respectively. This is similar for other technology-dependent projects.

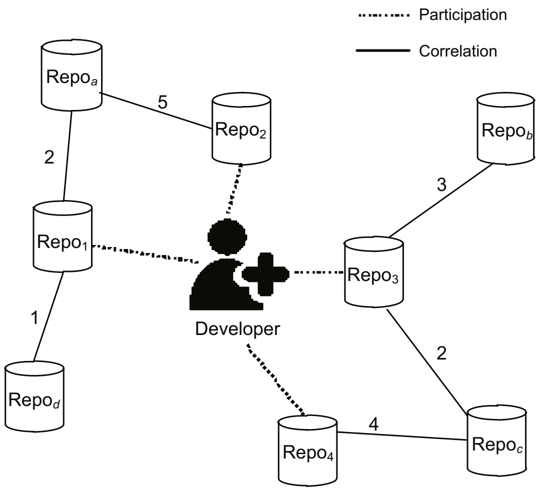

Fig. 4 Example of developer-oriented dependency sub-network

Finally, we design the following technical correlation computing formula between the developer and the candidate based on the user-oriented technology-dependent sub-network:

$$\mathrm{tr}(a,j)=\sum_i n_{\mathrm{ref}}(i,j), \tag{2}$$

where $\mathrm{tr}(a,j)$ represents the technical dependency between the given developer a and project j, and $n_{\mathrm{ref}}(i,j)$ represents the technical relevance between projects i and j, where project i refers to all the projects in which developer a participates ($i \in \{1,2,...\}$). We define the technical correlation between a developer and a candidate project as the sum of the technical correlations between the candidate project and all the projects in which the developer has been involved. Moreover, the degree of technical dependency between two projects is represented as the weight of the corresponding network edge.

For example, Fig. 4 shows the technical correlation between the developer and repositories. The developer is involved in projects 1 and 2, and the numbers of links between projects Repo$_a$ and Repo$_1$, Repo$_a$ and Repo$_2$ are two and five, respectively. Thus, we can quantify the technical correlation between the developer and project Repo$_a$ to be seven.

3.2.3 Social perspective

As a social programming site, GitHub provides social mechanisms, such as @, watch, and comments, to accelerate developer interaction and improve transparency among developer groups. Using these services, developers can easily and efficiently build relationships with other developers and interact with them. The social relationships of a developer in GitHub reflect his/her personal technical interests. In GitHub, the most common social activity is commenting on others' issues and pull-requests. Therefore, we analyze social relations based on the following comment activities:

1. Issue comment

GitHub offers a management system called "issue tracker" system that can be used by contributors to discuss deficiencies in the project or to propose new functional requirements, including discussing the issue in detail or providing a solution or recommendation.

2. Pull-request comment

Pull-request is an efficient mechanism for non-core developers to branch and be involved in project development and contribute to it. However, the pull-requests are often submitted by massive contributors and their quality usually varies a lot. Therefore, before a pull-request is merged, it is often discussed and tested in depth by a large number of relevant contributors.

For contributors, the issues or pull-requests they submit or comment on are usually those they are interested in or are familiar with. The fact that different developers participate in the same issue or pull-request indicates that they share a common interest or technical preference. A higher similarity between developers means a larger possibility that they are interested in the same project. In this study, we build a social network between developers based on contributors' participation in pull-requests. Then we measure the social association based on this network.

The social network reflects the social correlations between developers. In this network, each node represents a developer, and the edge is the number of co-occurrences between developers in the same issue or pull-request comments. The greater the number of co-occurrences, the more similar the preferences between the two developers.

Based on this social network, we can measure the social association between a developer and a specific project through the media of a socially connected developer. We design a method to calculate the degree of association as follows:

$$\mathrm{sr}(a,j)=\sum_{b} n_{\mathrm{int\,inacy}}(a,b)\cdot n_{\mathrm{participation}}(b,j), \quad (3)$$

where $n_{\mathrm{int\,inacy}}(a,b)$ represents the degree of social association between developers a and b, $n_{\mathrm{participation}}(b,j)$ represents the degree to which developer b participates in project j, and b is the set of all other developers that are socially associated with developer a.

Fig. 5 shows an example. The edge between developers indicates that a social relationship exists between them, and its weight is the number of co-occurrence participants in the issue or pull-request. The edge between the developer and the project indicates that the developer is involved in the project, and its weight indicates the depth of the developer's participation in the project. Consequently, we can obtain the total number of participants, the total number of pull-requests, and the total number of issues and pull-request comments in which the developer is involved.

In Fig. 5, the central developer and the other four (i.e., D_1, D_2, D_3, and D_4) co-appear in the issues or pull-requests of several projects. Both developers D_1 and D_2 are involved in project Repo$_a$. Bridging through these two developers, we can calculate the association between the central developer and Repo$_a$ as 42.

3.3 Multi-dimensional feature based recommendation algorithm

The three different perspectives from project, task, and social connection reflect the relationship between a developer and the recommended candidate from different dimensions. Combining them to

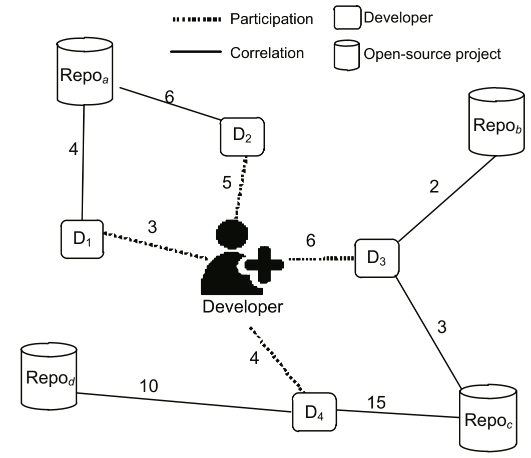

Fig. 5 Social connections between developers and repositories

fully and accurately measure the correlation between the developer and the project is the key to conduct personalized recommendation. We propose two different approaches in this subsection.

3.3.1 Linear combination based recommendation

A basic approach to aggregate the three dimensions of features is to combine them linearly. However, as the three perspectives are not in the same dimension, we need to normalize them first. As a result, each feature operates at the same order of magnitude such that parameter adjustment is facilitated. The linear combination can be carried out as

$$\begin{aligned}\mathrm{score}(a,j) &= \alpha\,\mathrm{nor}(\mathrm{pp}(j)) + \beta\,\mathrm{nor}(\mathrm{tr}(a,j)) \\ &\quad + \gamma\,\mathrm{nor}(\mathrm{sr}(a,j)), \quad (4)\end{aligned}$$

where $\mathrm{nor}(x) = x/\max(x)$, α, β, and γ are weight coefficients, and $\mathrm{pp}(j)$, $\mathrm{tr}(a,j)$, and $\mathrm{sr}(a,j)$ in the nor function represent the project level, the task level, and the social level, respectively.

First, we normalize the value of the three dimensions and then multiply each of them by different weights before summation. We adjust the parameters experimentally based on the results of feedback for optimization. Then the correlation score between a candidate project j and the developer can be obtained. Finally, we rank all candidate recommendations and recommend the top-N candidates to the developers according to their scores.

3.3.2 Learning-to-rank based recommendation

Learning-to-rank (LTR) is a machine learning method based on a given set of features. This method uses a training dataset construction model to learn the final sort function. The LTR method has been used to analyze and sort software in software engineering (Zhu et al., 2015; Chen et al., 2016). In this study, we analyze the historical data of users, extract multi-dimensional features, and construct the target ranks based on users' participation in projects. Based on this set of items, we carry out supervised learning and establish the final sorting model. The whole procedure can be divided into three stages as follows:

1. Recommended candidates filtering

At this stage, we aim to find the recommended candidates and the training dataset for a given developer. Based on the sub-project-dependent network around the developer, we can obtain all the technically correlated projects depending on the projects in which the developer is already involved; based on the developer's social network, we can obtain all the socially connected developers and all the projects in which these developers have participated. Based on the results, we view these projects in which the given developer has not participated as the recommended candidates, and the projects in which he/she is already involved as the LTR's training dataset.

2. Rank marking

Ranking is the target value for the LTR method to learn. It reflects the importance of data records. A higher ranking means a higher importance. The historical data of users show that if the user conducts a large number of activities on a project, then this user is highly interested in it. Thus, we select the actual frequency of participation in the project (i.e., the number of activities per project) as the target value and use them as the marked value for the final LTR model training and testing.

3. Model training

The datasets for LTR training can be obtained by extracting the feature sets and ranking the items. In the dataset, each record reflects the relationship between a user and a target as recommended items. The eigenvalues are the characteristics of the three dimensions between the user and the project. The target value is the user participation in the marking process. Based on these data, we can train an LTR-based recommendation ranking model. Specifically, we use the RankLib package to build the LTR model.

4 Experimental settings

In this section, we introduce the experiment dataset and the evaluation metrics.

4.1 Experiment datasets

We used the dataset provided by GHTorrent (released in June 3, 2016), which was also adopted in our previous work (Yang et al., 2016). We first found some representative developers as experimental samples. Then we analyzed and evaluated our recommendation method using the historical data of these developers. A developer could be chosen only if one of the following conditions is fulfilled: (1) the status had experienced a change from being inactive to active in the community; (2) the developer had contributed to more than one project; (3) the developer had communicated with other developers in the GitHub community. Based on the aforementioned principles, we selected 16 249 developers as candidates for the project. These developers had participated in at least five open-source projects and followed 5 to 50 developers. The extracted behavior information of these developers includes issue, pull-request, and corresponding comments, as well as the fork relationship between projects, the watch relationship between users and projects, and the "follow" relation among users. The behavior information in the dataset involved 144 543 developers who intersected with the selected candidates and covered 1 705 841 related open-source projects.

In the final experiment, we excluded projects with no fork or watch, and obtained 801 228 open-source projects. In constructing the dependency network between projects, we used the inter-project technical dependency data in Blincoe et al. (2015).

Afterwards, we divided the whole dataset into two parts: one for the training model, and the other for evaluation.

4.2 Evaluation metrics

We established a set of scientific indices to evaluate the effectiveness of our method. We grouped the datasets based on the creation time of the corresponding activity. The projects in which the developer had been involved during the most recent year were used for testing. The previous data were used to train our recommendation model. During the evaluation process, the time span of the training dataset was controlled to validate the influence of different information on the recommendation model. For feature selection, we evaluated both individual and combinational features to verify the effectiveness. For the LTR model training phase, we used a cross-validation method to divide the dataset into N parts with an equal size. One was used as the test dataset, and the other parts were used as the training sets. The model was conducted via N iterations, and there were also N tests being carried out to evaluate the model.

In actual situations, taking part in a repository takes a lot of time. Thus, users cannot join many repositories in a limited time. In our dataset, the average count of joining repositories per user in one year is 19.5; however, the scope of our recommendation reached 337.8 repositories out of 801 228.0, making it difficult to obtain an ideal result and to effectively evaluate our method. Thus, we used our own evaluation metric.

We used the hit ratio as the criterion for evaluation. The number of available developers to the recommendation results can be obtained accurately. We set bounds for the number of recommended items (i.e., top 5, 10, 15, and 20) to obtain the recommendation results that different numbers of projects can recommend. The hit ratio was defined as the percentage of developers who actually were participating in the recommended projects over all developers:

$$\text{ratio}_{\text{hit}} = n_{\text{hit}}/n_{\text{total}}. \tag{5}$$

The quality of the recommendation results is an important aspect in evaluating the performance. We used mean reciprocal rank (MRR) as our evaluation model in comparing the LTR method with the linear method. MRR is a general evaluation metric for a sorting algorithm and can reflect the rationality of the ranks of recommendation results. MRR can be formulated as

$$\text{MRR} = \frac{1}{|Q|}\sum_{i}^{|Q|}\frac{1}{\text{rank}_i}, \tag{6}$$

where $|Q|$ is the result of all hits, and rank_i is the ranking of hit result i in the recommendation list.

5 Experimental results and analysis

To evaluate the results of our recommendation approach, we first compare the performance of the linear combination approach using the combination of different dimensions of features and different time intervals in Section 5.1. Then we analyze the LTR-based recommendation approach in Section 5.2. In Section 5.3, we present a practical case to demonstrate the results.

In the experiments, we tested various kinds of combinations of three dimensional features. The results were affected by parameters including α, β, and γ. We determined these parameters through iterative experimentation on a given dataset, and they were set to $\alpha = 0.50$, $\beta = 0.55$, and $\gamma = 0.45$. If one type of feature is not combined in the experiment, then the corresponding factors will be set to 0.

5.1 Recommendations based on linear combination model

5.1.1 Recommendation performance using different dimensions of features

In this experiment, we analyzed mainly the impacts of the three feature dimensions, namely the project popularity (PL), the project technical dependence (TL), and the developer social relevance degree (SL). Then we discussed how to improve the performance of the model.

First, we built the model using each type of feature to verify their validity in recommendation. In this process, only the social network and project-dependent network can limit the scope of project selection. Accordingly, we compared the recommendation model using only TL and SL.

As shown in Table 1, TL and SL performed similarly in single-dimension experiments. They obtained the best results for the top-20 recommendations with 11.77% and 11.60% hit rates, respectively. Their hit rates are 6.04% and 6.24% for the top-5 recommendations. The results suggest that using TL or SL alone to build the recommendation model can achieve a certain effect; however, the overall effect can still be improved.

We then compared the recommendation performance using multi-dimensional information, which is the information of the combination of any two of the three dimensions (PL, TL, and SL). In particular,

Table 1 Comparison of recommendation performance based on different dimensions of features

Feature dimension	Hit rate (%)			
	Top 5	Top 10	Top 15	Top 20
TL	6.04	8.52	10.45	11.77
SL	6.24	8.47	10.40	11.60

TL: technical dependence; SL: social relevance degree

we used the linear combination of multi-dimensional features. Fig. 6 presents the detailed results.

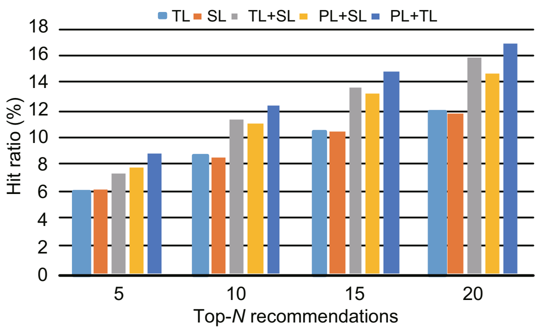

Fig. 6 Comparison of recommendation performance for combinations of two-dimensional information

TL: technical dependence; SL: social relevance degree; PL: project popularity

As shown in Fig. 6, the results of all the two-dimensional information recommendation models are better than those of the single-dimension model. The PL+TL combination is better than the TL+SL and PL+SL combinations. The hit rate of PL+TL combination model reaches 16.70%, which is 4.93% higher than the best result of the single-dimensional model.

Fig. 7 shows a comparison of the results of a recommendation model constructed with three dimensions and a combination of any two dimensions. The results of the combination of three dimensions are better than those of the combinations of two dimensions. The former combination reaches a 19.38% hit rate for the top-20 recommendations. The performance is nearly twice that of the single-dimensional model for the top-5 recommendations. Comparison results show that better personalized recommendations can be achieved when different dimensions of features are used in building the recommendation model.

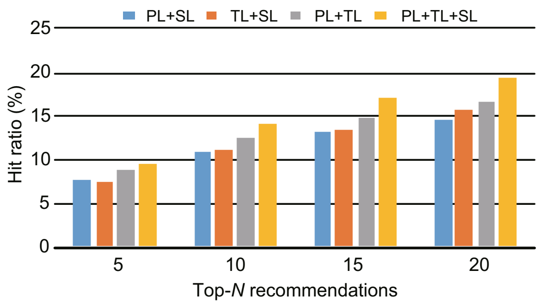

Fig. 7 Comparison of recommendation performance for combinations of multi-dimensional information

TL: technical dependence; SL: social relevance degree; PL: project popularity

5.1.2 Influence of test time interval on recommendation results

Developers usually require a lot of time and effort to participate in an open-source project. On one hand, developers are cautious when choosing to join an open-source project. On the other hand, developers can contribute to only a limited number of projects because of limited time. Therefore, the actual developer participation in projects is limited over a given period of time. This situation does not mean that developers are not interested in other projects, or that the recommendations of projects in which they have not actually participated are inaccurate. The reason can be that no recommendation has been provided to the developer such that this developer may not notice the corresponding project or does not have enough time to be involved. The project may still be attracted to the developer given sufficient time. In this subsection, we analyze the influence of different test time intervals on the recommendation results.

We divided the activity data in GitHub into three different time intervals as the test datasets, including 3, 6, and 12 months. The projects in which the given developer actually participated during the test time interval were viewed as the ground-truth. We adopted a linear combination model with multi-dimensional features to compare the test results. Fig. 8 shows the specific experimental results.

Fig. 8 compares and analyzes the recommendation accuracies under different test time windows with the same number of candidates. The overall accuracy shows a rising trend with the increase of the time interval. A long test window results in a high recommendation hit rate regardless of the number of recommended candidates. With regard to the devel-

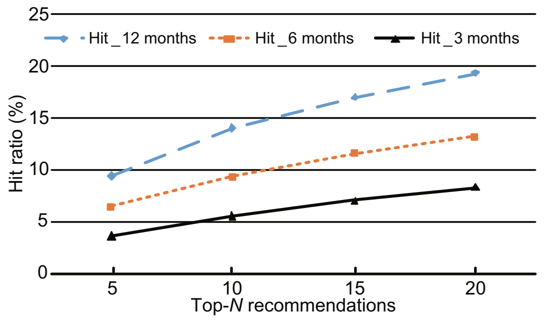

Fig. 8 Influences of different snapshots on the hit ratio

opers' actual participation, they may not participate in too many recommended projects in a relatively short period of time (such as three months), but may participate in more projects over a longer time. This suggests that, given sufficient time and energy, the developers may actually participate in many more of the projects listed in the recommendation results. Therefore, the actual participation in the project as the recommended ground-truth of the evaluation criteria is significantly strict. In other words, the actual recommendation performance should be better than the experimental results shown in Fig. 8. Developers require much time to search for open-source projects in which they may be interested. Thus, personalized recommendation can accelerate the process and promote the development of open-source software.

5.2 Recommendation based on LTR model

In this subsection, we compare the LTR model based recommendation results with those based on the linear combination model. Table 2 shows the comparison between the hit ratio and the MRR results. As we can see, the hit ratios of the top-5 to top-20 recommendations obtained using the LTR model are better than those obtained using the manual tuning linear model. In the promotion range, the top-5 and top-10 recommendations received a large increase, whereas the top-20 recommendations retained a small increase. This result indicates that LTR can better explore the relationship between features but present limited feature selection performance as the final increase is limited. Under the MRR index, the LTR results are also superior to those of the linear model. This finding further demonstrates that LTR can better prioritize the possible outcomes.

On this basis, we analyzed how the additional social connection information would affect the recommendation performance for both approaches. The promotion accuracies of recommendation are shown in Table 3.

As the number of features increases, the tuning capability of the linear model with manual tuning becomes limited. Furthermore, the enhancement in LTR becomes considerably larger than that of manual tuning. Finally, the hit rate and MRR of LTR are improved significantly.

5.3 Case study

We chose the recommendation results for the top-100 developers according to MRR. We aimed to find the reasons for error results and why correct results are ranked behind.

In our model, we neither analyzed the fine granularity of user activities, nor established the relationship between developers according to their differences.

We analyzed Dan Rasmussen (Table 4) and found that, for developer Alex Bubenshchykov who was connected to Dan, many of his related projects were recommended correctly. Even though they had commented on each other only one time, three projects were recommended at the top. However, for developer Mathias Buus who had communicated eight times with Dan on issues, three of the projects he had participated in were recommended to Dan. Because the connection between Mathias and Dan

Table 2 Comparison of recommendation performance for linear-combination model and LTR model

Recommendation	Hit ratio (%)			MRR (%)		
	Linear mode	LTR	Rate of increase	Linear mode	LTR	Rate of increase
Top-5	9.46	11.44	20.93	0.0538	0.0657	22.12
Top-10	14.00	15.28	9.14	0.0599	0.0708	18.20
Top-15	17.02	17.44	2.47	0.0622	0.0725	16.56
Top-20	19.38	19.60	1.14	0.0635	0.0734	15.59

LTR: Learning-to-rank; MRR: mean reciprocal rank

Table 3 Comparison of recommendation performance for the linear-combination model and LTR model after introducing additional social connection information

Recommendation	Hit ratio (%)			MRR (%)		
	Linear mode	LTR	Rate of increase	Linear mode	LTR	Rate of increase
Top-5	10.57	16.04	51.75	0.0609	0.1172	92.45
Top-10	15.07	20.20	34.04	0.0669	0.1227	83.41
Top-15	18.44	23.16	25.60	0.0695	0.1251	80.00
Top-20	20.99	25.04	19.29	0.0709	0.1262	78.00

LTR: Learning-to-rank; MRR: mean reciprocal rank

Table 4 Recommendation results for Dan Rasmussen

Name of Repo	Related developer	Related Repo	Join or not
Mocha-jshint	Alex Bubenshchykov	–	Yes
Nodeerrors	Alex Bubenshchykov	–	Yes
Moduleconfig	Alex Bubenshchykov	–	Yes
Node-browserify	Mathias Buus	–	No
Monu	Mathias Buus	–	No
ScreenCat	Mathias Buus	–	No
Node-webkit	–	Async	No
Jsq	Alex Bubenshchykov	–	Yes
Mongoobject	Alex Bubenshchykov	–	Yes
Errortoenglish	Alex Bubenshchykov	–	Yes
Singlequote	Alex Bubenshchykov	–	Yes

was tighter, the results ranked in front of the other four projects of Alex. We compared Alex with Mathias and found that Mathias was much more active than Alex in GitHub. Alex owned 23 repositories, had 30 stars and 23 followers, whereas Mathias owned 621 repositories, had more than 1100 stars and 2900 followers. Hence, Mathias and Dan tended to have a stronger relationship. Although Dan was not that active in GitHub, we cannot deny that Mathias' activeness had a large impact on our recommendation model.

Therefore, we believe that when calculating the relationship between developers, we should consider not only the incidence relation of issue comments, but also the number of projects in which users participate. If one is very active and has many interests, we should weaken the influence of the relationship.

For the fine granularity of project information, different projects impose different restrictions on developers.

For a developer in our recommendation list, the rails project ranked higher than the symfony project. However, the relationship between symfony and this developer was more stronger, because project symfony was mentioned twice in project monolog which the developer had participated before. Even so, rails ranks higher than symfony in the final recommendation list, because rails was much more popular than symfony.

According to our analysis, we believe that the difficulty of participation should be taken into consideration, including project scale, developer level, and the relation between developers and projects. For rails, there is a standard development process which should be abided by strictly. Meanwhile, the project scale is very large and there are many participants. This accelerates the solution of issues. Therefore, it is difficult for developers to contribute.

There is something that needs to be improved in our model. We can involve more factors that can represent the activeness of projects comprehensively. At the same time, we should differentiate different dimensions.

From the recommendation results for developer Hydai (Table 5), we found that Hydai did not participate in the recommended project ranking in a 3rd place. However, he participated in projects ranking the 4th and 16th. These three projects had no difference in our model except in project activeness. Therefore, we can see that the measurement of project activeness is not always suitable. Besides forking and watching, there are some other factors that can represent the activeness of projects, e.g., starring. Different factors have different effects and should be treated differently.

Table 5 Results of Hydai

Rank of the result	Name of repo	Fork number	Watch number	Join or not
3	Irc_log	6	36	No
4	NTHU_course	12	10	Yes
16	Tioj	3	10	Yes

In GitHub, forking, watching, and starring are all important factors reflecting the activeness of projects. Forking is to clone repositories. Watching can show users' interests because watchers can receive real-time discussion and other information about the target project. Starring is similar to voting up, which is the simplest action among the three. However, it can still represent users' interests. In our recommendation model, we should treat them as different actions and allocate them with different proportions instead of simply summing them. We will improve the model in our future work.

6 Related work

The rapid growth of open-source software not only provides considerable reusable resources, but also induces the problem of information overload. Many researchers have studied software recommendation and retrieval technology to solve this problem. According to the different types of recommendation objects, the relevant work focuses mainly on the source code, developer, and documentation of the recommendation and search engine technology.

6.1 Source code recommendation

In software engineering, software source code is the most frequently reused component. Thus, reusable source code retrieval and recommendation has been explored widely.

Most studies consider the software source code as a text and use text mining technology to analyze the text similarity and thus locate the source code. Ye and Fischer (2002) proposed a method, called "CodeBroker," to extract the names of methods, classes, and comments from the source code of the software. This method uses the extracted text information to represent the source code, and leverages the semantic similarity to find relevant source code.

Grechanik et al. (2010) used information retrieval and static analysis techniques to locate the required source code. McMillan et al. (2010) used the application programming interface (API) call information in the source code as a feature to compare and to discover similar software. In addition to the text information, the software source code itself is structured data. It contains valuable structured information that can be used as the feature for recommendation. Such has been investigated to retrieve and locate structure-similar software (Holmes and Murphy, 2005; Xie and Pei, 2006; Lozano et al., 2011). Xie and Pei (2006) leveraged the structure information of source code, and proposed a frequent item based approach to recommend code examples for API usage. They first gained related source code files through search engines, mined the usage patterns of API in the retrieved codes, and then recommended summarized API usage code snipets. Social media is a new source of information for source code recommendation. The Q&A communities like StackOverflow (https://stackoverflow.com) cover a large number of software-related discussion posts and code snippets. Zagalsky et al. (2012) used group discussion and voting information to measure the quality of code fragments. Based on this, they could efficiently rank the proposed code fragments and recommend high-quality code snippets to developers.

Recommendation systems usually rely on contextual information, and search engines can use the query keywords entered by users to retrieve the required resources. At present, many researchers have proposed and designed the corresponding software source code search engines. Bajracharya et al. (2009) designed and implemented a large-scale open-source code retrieval and analysis system, named "Sourceer," by fully using structured information, such as references and dependencies in the code. Kokkoras et al. (2012) provided developers with a single search interface in a search system, named "OCEAN," which aggregates search results from multiple web search engines and presents them to the user through a single portal. The OSSEAN proposed by Yin et al. (2015) is an analytical search platform for global open-source software. This platform crawls from the open-source software collaborative development community and knowledge-sharing community of massive open-source software resources. The platform also contains large-scale participants and feedbacks contained in the group wisdom on open-source software, allows ranking, and recommends high-quality open-source software. Brandt et al. (2010) incorporated the web search interface into the integrated development environment. This method allows developers to easily find sample codes on the integrated development environment (IDE) while developing the program. The analysis of most current work related to program development is mainly at the source code level. However, our work is based on the developer as the core and the number of projects from the recommended list in which the developer will be interested.

6.2 Experts' recommendation

In globalized and distributed software development, quickly and accurately finding a suitable person to solve a given problem or to complete the project development is a challenging but valuable problem. This issue has been investigated intensively.

Task assignment is the most common kind of development and management activity in software development. This activity is important in improving the efficiency and quality of software development. Anvik et al. (2006) first used a machine learning algorithm to find historical issues that are similar to a given issue based on textual information, and then sorted and assigned the developers involved in the historical issue revisions to the specified issue. Jeong et al. (2009) and Bhattacharya and Neamtiu (2010) constructed a graph based on the evolution history of a given issue and then used the classification method to find the staff for the right solution. Yu et al. (2016) investigated the overloading of pull-requests and proposed a reviewer recommendation technique based on developer social information and technical interest.

Surian et al. (2011) constructed a network called "development project language/software class" to find the right collaborator. This network defines a measure of developer social distance based on the distance and recommended potential collaborators. Canfora et al. (2012) studied the social and technical activity of developers by analyzing mailing lists and version controls to recommend a better "guide" for new entrants. The relevant research proposed by technical experts discusses mainly creating recommendations from the historical development activities and social behaviors of developers. These studies have provided enlightenment on how to measure the technical capability and technical interest of developers.

6.3 Knowledge document recommendation

Unlike commercial software, open-source software development usually lacks standardized documentation. Thus, reusing open-source software is difficult for developers. However, rich documents on the Internet have become an important source of knowledge on this type of software. For example, the StackOverflow discussion paper on the Android API covers 87% of all class files in Android; the discussion has been viewed more than 70 million times (Allamanis and Sutton, 2013). At present, many researchers focus on the issue of knowledge document recommendation for developers.

Favre et al. (2012) proposed a method based on pattern matching to locate documents corresponding to source code. Chen and Grundy (2011) combined regular expressions, keywords, and clustering methods. They also improved the recommendation accuracy of the support vector machine (SVM) model. Rigby and Robillard (2013) assisted in finding relevant discussion by extracting code judgments from posts.

Bacchelli et al. (2012) designed an Eclipse plugin that automatically links to the corresponding StackOverflow discussion posts based on the code entered by the developers in the editor. Wang T et al. (2014) focused on the automatic connection of the item issue with StackOverflow and proposed a method of combining semantic similarity and temporal locality to find and locate the discussion posts that are related closely to issues. Consequently, participants receive comprehensive information about the issue to better address it.

7 Conclusions and future work

The crowds' continuous and active participation becomes more and more important for the success of open-source software. Achieving the best match between the personal interest of developers and the technical requirements of open-source projects is the key to retain participants and to promote the sustainable and rapid development of projects. However, it is a great challenge for developers to quickly and accurately locate the desired repositories because of the large-scale and high-speed growth of open-source projects, as well as the highly decentralized activity data of their participation.

In this study, we have proposed a method called "RepoLike", which explores multi-dimensional features to measure the potential correlation between a developer and a project. We used linear combination and LTR approaches to aggregate different levels of features, and proposed a personalized recommendation method. Specifically, we considered the characteristics of the open-source project itself, the technical relevance between the open-source projects, and the social relevance degree among the developers. Finally, we conducted comprehensive experiments to validate the proposed approaches. The experimental results showed that our method can achieve a high accuracy for recommending open-source repositories to developers.

We plan to extend the present study in the following aspects: First, current technical dependencies between open-source projects consider the technical parameters in only the comments. Some other real technical dependencies from other sources are overlooked. We will further trace the source code and comprehensively examine the level of dependencies. Second, apart from direct interaction in terms of issues and pull-requests, the developers in GitHub can modify the same code file and communicate indirectly. We will consider this additional information in investigating the social association between developers. Finally, we will send the results to some of the corresponding developers in GitHub to obtain their feedback on the recommendation results. Accordingly, we can precisely evaluate the accuracy of the proposed method in terms of the recommendation results.

References

Allamanis M, Sutton C, 2013. Why, when, and what: analyzing stack overflow questions by topic, type, and code. 10th Working Conf on Mining Software Repositories, p.53-56. https://doi.org/10.1109/MSR.2013.6624004

Anvik J, Hiew L, Murphy GC, 2006. Who should fix this bug? 28th Int Conf on Software Engineering, p.361-370. https://doi.org/10.1145/1134285.1134336

Bacchelli A, Ponzanelli L, Lanza M, 2012. Harnessing stack overflow for the IDE. 3rd Int Workshop on Recommendation Systems for Software Engineering, p.26-30. https://doi.org/10.1109/RSSE.2012.6233404

Bajracharya S, Ossher J, Lopes C, 2009. Sourcerer: an Internet-scale software repository. Int Workshop on Search-Driven Development-Users, Infrastructure, Tools, and Evaluation, p.1-4. https://doi.org/10.1109/SUITE.2009.5070010

Begel A, DeLine R, Zimmermann T, 2010. Social media for software engineering. FSE/SDP Workshop on Future of Software Engineering Research, p.33-38. https://doi.org/10.1145/1882362.1882370

Begel A, Bosch J, Storey MA, 2013. Social networking meets software development: perspectives from GitHub, MSDN, Stack exchange, and Topcoder. *IEEE Softw*, 30(1):52-66. https://doi.org/10.1109/MS.2013.13

Bhattacharya P, Neamtiu I, 2010. Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. IEEE Int Conf on Software Maintenance, p.1-10. https://doi.org/10.1109/ICSM.2010.5609736

Blincoe K, Harrison F, Damian D, 2015. Ecosystems in GitHub and a method for ecosystem identification using reference coupling. IEEE/ACM 12th Working Conf on Mining Software Repositories, p.202-207. https://doi.org/10.1109/MSR.2015.26

Boyd DM, Ellison NB, 2007. Social network sites: definition, history, and scholarship. *J Comput-Mediat Commun*, 13(1):210-230. https://doi.org/10.1111/j.1083-6101.2007.00393.x

Brandt J, Dontcheva M, Weskamp M, et al., 2010. Example-centric programming: integrating web search into the development environment. SIGCHI Conf on Human Factors in Computing Systems, p.513-522. https://doi.org/10.1145/1753326.1753402

Canfora G, di Penta M, Oliveto R, et al., 2012. Who is going to mentor newcomers in open-source projects? ACM 20th Int Symp on the Foundations of Software Engineering, Article 44. https://doi.org/10.1145/2393596.2393647

Chen X, Grundy J, 2011. Improving automated documentation to code traceability by combining retrieval techniques. 26th IEEE/ACM Int Conf on Automated Software Engineering, p.223-232. https://doi.org/10.1109/ASE.2011.6100057

Chen X, Qin Z, Zhang Y, et al., 2016. Learning to rank features for recommendation over multiple categories. 39th Int ACM SIGIR Conf on Research and Development in Information Retrieval, p.305-314. https://doi.org/10.1145/2911451.2911549

Dabbish L, Stuart C, Tsay J, et al., 2012. Social coding in GitHub: transparency and collaboration in an open software repository. ACM Conf on Computer Supported Cooperative Work, p.1277-1286. https://doi.org/10.1145/2145204.2145396

Favre JM, Lämmel R, Leinberger M, et al., 2012. Linking documentation and source code in a software chrestomathy. 19th Working Conf on Reverse Engineering, p.335-344. https://doi.org/10.1109/WCRE.2012.43

Grechanik M, Fu C, Xie Q, et al., 2010. A search engine for finding highly relevant applications. ACM/IEEE 32nd Int Conf on Software Engineering, p.475-484. https://doi.org/10.1145/1806799.1806868

Holmes R, Murphy GC, 2005. Using structural context to recommend source code examples. 27th Int Conf on Software Engineering, p.117-125. https://doi.org/10.1109/ICSE.2005.1553554

Jeong G, Kim S, Zimmermann T, 2009. Improving bug triage with bug tossing graphs. 7th Joint Meeting of the European Software Engineering Conf and the ACM SIGSOFT Symp on the Foundations of Software Engineering, p.111-120. https://doi.org/10.1145/1595696.1595715

Kokkoras F, Ntonas K, Kritikos A, et al., 2012. Federated search for open-source software reuse. 38th Euromicro Conf on Software Engineering and Advanced Applications, p.200-203. https://doi.org/10.1109/SEAA.2012.55

Lozano A, Kellens A, Mens K, 2011. Mendel: source code recommendation based on a genetic metaphor. 26th IEEE/ACM Int Conf on Automated Software Engineering, p.384-387. https://doi.org/10.1109/ASE.2011.6100078

McMillan C, Poshyvanyk D, Grechanik M, 2010. Recommending source code examples via API call usages and documentation. 2nd Int Workshop on Recommendation Systems for Software Engineering, p.21-25. https://doi.org/10.1145/1808920.1808925

Rigby PC, Robillard MP, 2013. Discovering essential code elements in informal documentation. Int Conf on Software Engineering, p.832-841. https://doi.org/10.1109/ICSE.2013.6606629

Storey MA, Treude C, van Deursen A, et al., 2010. The impact of social media on software engineering practices and tools. FSE/SDP Workshop on Future of Software Engineering Research, p.359-364. https://doi.org/10.1145/1882362.1882435

Surian D, Liu N, Lo D, et al., 2011. Recommending people in developers' collaboration network. 18th Working Conf on Reverse Engineering, p.379-388. https://doi.org/10.1109/WCRE.2011.53

Wang H, Yin G, Xie B, et al., 2014. Research on network-based large-scale collaborative development and evolution of trustworthy software. *Sci Sin Inform*, 44(1):1-19. https://doi.org/10.1360/N112013-00128

Wang T, Yin G, Wang H, et al., 2014. Linking stack overflow to issue tracker for issue resolution. 6th Asia-Pacific Symp on Internetware, p.11-14. https://doi.org/10.1145/2677832.2677839

Xie T, Pei J, 2006. Mapo: mining API usages from open source repositories. Int Workshop on Mining Software Repositories, p.54-57. https://doi.org/10.1145/1137983.1137997

Yang C, Fan Q, Wang T, et al., 2016. RepoLike: personal repositories recommendation in social coding communities. 8th Asia-Pacific Symp on Internetware, p.54-62. https://doi.org/10.1145/2993717.2993725

Ye Y, Fischer G, 2002. Information delivery in support of learning reusable software components on demand. 7th Int Conf on Intelligent User Interfaces, p.159-166. https://doi.org/10.1145/502716.502741

Yin G, Wang T, Wang H, et al., 2015. Ossean: mining crowd wisdom in open source communities. IEEE Symp on Service-Oriented System Engineering, p.367-371. https://doi.org/10.1109/SOSE.2015.51

Yu Y, Wang H, Yin G, et al., 2016. Reviewer recommendation for pull-requests in GitHub: what can we learn from code review and bug assignment? *Inform Softw Technol*, 74:204-218. https://doi.org/10.1016/j.infsof.2016.01.004

Zagalsky A, Barzilay O, Yehudai A, 2012. Example overflow: using social media for code recommendation. 3rd Int Workshop on Recommendation Systems for Software Engineering, p.38-42. https://doi.org/10.1109/RSSE.2012.6233407

Zhou M, Mockus A, 2011. Does the initial environment impact the future of developers? 33rd Int Conf on Software Engineering, p.271-280. https://doi.org/10.1145/1985793.1985831

Zhu J, Shen B, Hu F, 2015. A learning to rank framework for developer recommendation in software crowdsourcing. Asia-Pacific Software Engineering Conf, p.285-292. https://doi.org/10.1109/APSEC.2015.50