DevRec: A Developer Recommendation System for Open Source Repositories

Xunhui Zhang(✉), Tao Wang, Gang Yin, Cheng Yang, Yue Yu, and Huaimin Wang

National University of Defense Technology, Changsha, Hunan, China
{zhangxunhui,taowang2005,yingang,yuyue,hmwang}@nudt.edu.cn, delpiero710@126.com

Abstract. The crowds' active contribution is one of the key factors for the continuous growth and final success of open source software. With the massive amounts of competitions, how to find and attract the right developers to engage in is quite a crucial yet challenging problem for open source projects. Most of the current works mainly focus on recommending experts to specific fine-grained software engineering tasks and the candidates are often confined to the internal developers of the project. In this paper, we propose a recommendation system *DevRec* which combines users' activities in both social coding and questioning and answering (Q&A) communities to recommend developer candidates to open source projects from all over the community. The experiment results show that *DevRec* is good at solving cold start problem, and performs well at recommending proper developers for open source projects.

Keywords: Developer recommendation · Collaborative Filtering · StackOverflow · GitHub

1 Introduction

Nowadays, open source software (OSS) has formed a brand-new development paradigm and achieved its unprecedented success. Compared to the traditional software development in industry, open source software is driven by a massive number of stackholders including developers, users, managers and so on. These stakeholders involve in OSS projects by interests, and most of them have their own full-time job and only spend spare time on OSS. Although crowds may join in an OSS project occasionally, and then withdraw from it at any time, OSS has achieved great success at creating high-quality software like Linux, MySQL, Spark and so on. Nowadays, OSS is viewed as "eating the software world" by the Future of Open Source Survey [1].

On GitHub alone, one of the largest open source communities, there are more than 48 million open source projects hosted. However, according to our statistics, 95.2% of them do not received any attention by public (i.e. no watcher and forked repository) and 15.1% of them were not updated for more than one year. Even for

G. Botterweck and C. Werner (Eds.): ICSR 2017, LNCS 10221, pp. 3–11, 2017.
DOI: 10.1007/978-3-319-56856-0_1

those projects which used to experience their success will languish without the crowds' continuous contributions. Therefore, to find and attract the right developers to participate in is quite crucial for OSS.

However, due to the massive amounts of competitive OSS, the crowds are often limited by their time and energy to choose from all the related projects. An automatic approach to bridge the gap between developers and projects is useful for both the developers and projects. In this paper, we propose a hybrid recommendation system called *DevRec*, which combines the development activity (DA)[1] based approach and knowledge sharing activity (KA)[2] based approach respectively to recommend proper developers for open source projects. The main contributions of this paper include:

- We propose a DA-based recommendation approach, by mining the crowds' development activities to discover and recommend proper collaborators for a given project from all over the community.
- We combine the developers' knowledge sharing and development activities, which helps to solve the cold-start problem for newly released projects.
- We conducted experiments on a large-scale dataset containing 165,741 projects and 72,877 developers, to show the effectiveness of our approach.

2 Related Work

In the age of global and distributed software development, finding the right person to collaborate and complete the right task is of great importance. Bhattacharya et al. [2] employed the incremental learning approach and multi-feature tossing graphs to improve the bug triaging. Xuan et al. [3] combined network analyzing and text classification approaches to recommend proper developers for a specific bug. Yu et al. [4–6] analyzed the pull-request mechanism, and embedded the social factors into typical recommendation approaches of bug triaging, to recommend pull-request reviewers. Besides, there are some works related to software recommendation. Lingxiao Zhang et al. [7] recommended relevant projects according to the relationship between developers and projects. Naoki Orii combined the probabilistic matrix factorization and topic model method to recommend repositories to programmers [8]. Different from these works, our work aims to recommend developers at the granularity of repository and the range of the whole software community.

There are also many researches focusing on the interplay between Q&A and social coding communities. Bogdan Vasilescu et al. [9] did some researches about the relationship of users' activities between StackOverflow and GitHub. Wang et al. [10] proposed an approach to link Android issues and corresponding discussions. Giuseppe Silvestri et al. [11] studied whether the relative importance of users vary across social networks, including StackOverflow, GitHub and Twitter.

[1] development activity: users' activities in social coding communities.
[2] knowledge sharing activity: users' activities in Q&A communities.

Venkataramani et al. [12] recommended suitable experts from GitHub to StackOverflow questions, taking developers' reputation into account. However, there are few studies about recommending users in Q&A communities to OSS, which gives full play to users' expertise and helps to speed up the rate of development.

3 Recommendation Approach

3.1 Overview of Recommendation System

DevRec explores the activities in social coding and Q&A communities to measure the technical distance between a given project and the developers, and combines the two results together to rank and recommend developer candidates for OSS. The architecture of *DevRec* is shown in Fig. 1.

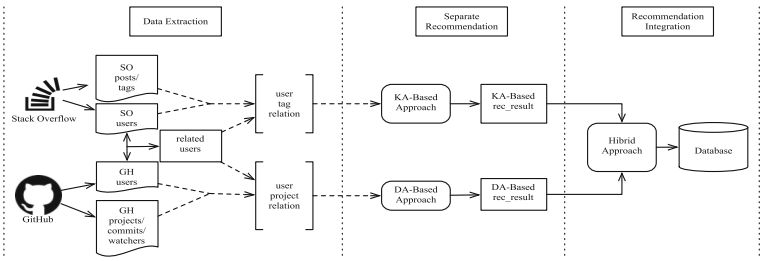

Fig. 1. Architecture of *DevRec*

Data Extraction: This step aims to gather datasets from StackOverflow and GitHub, and generate the user tag and user project association matrix.

Separate Recommendation: When a project comes, we calculate the relevance of each candidate according to the user association matrix, which is obtained from the user project and user tag association matrix.

Recommendation Integration: We combine the separate results by weight, and finally get the recommendation results for the hybrid approach.

3.2 Developer Recommendation Based on Social Coding Activities

There are kinds of activities in GitHub, including commit, fork and watch, which represent developers' interests to specific repositories. The basic intuition of this approach is that developers with similar technical interests tend to have similar development activities [7]. There are three parts in this approach.

UP Connector: This part is to create the association matrix of users and projects based on the activities in GitHub. Here we get a two-value matrix R_{u-p}, where 1 stands for participation and 0 stands for the opposite.

User Connector: This part is to calculate the association between users based on the user project association matrix using Jaccard algorithm.

Match Engine: In this part, we calculate the association between users and projects according to the user association matrix R_{u-u}. If we use $UA_p\langle u_1, u_2, ..., u_n\rangle$ to represent users that have already participated in the target project p, we can obtain the match score of each user towards project p using Eq. 1.

$$result = \sum_{i=1}^{|UA_p|} R_{u-u}[UA_p[i]] \tag{1}$$

3.3 Developer Recommendation Based on Knowledge Sharing Activities

Asking and answering technical questions are common activities among developers. Those with similar technical interests tend to focus on same posts, which are marked by same tags, and We suppose that they also tend to participate in the same kind of repositories. There are four parts in this approach.

Tag Extractor: This part is used to extract the fields that users proficient in. We consider that tags that mark users' related posts can represent user's interests or research fields.

Relation Creator: In this part, we calculate the user tag association matrix. Here we use *TF-IDF* method. If we use $U\{u_1, u_2, ..., u_n\}$ to represent users in StackOverflow, $T_u = \{t_1, t_2, ..., t_n\}$ to represent the tags that related to user u, and $C(t, u)$ to represent the number of times tag t relates to user u. Then we can calculate user tag association matrix using Eq. 2.

$$R_{u-t}(u,t) = \frac{C(t,u)}{\sum_{i=1}^{|T_u|} C(T_u[i],u)} * \log(\frac{\sum_{k=1}^{|U|}\sum_{q=1}^{|T_{U[k]}|} C(T_{U[k]}[q], U[k])}{\sum_{j=1}^{|U|} C(t, U[j])}) \tag{2}$$

User Connector: After obtaining the user tag association matrix R_{u-t}, we calculate the association of users using Vector Space Similarity algorithm.

Match Engine: The same as the match engine part in DA-based approach.

3.4 Hybrid Approach for Developer Recommendation

According to the above two approaches, we can get two recommendation results of the same repository. The combination of the two approaches takes users' activities into consideration comprehensively, and will probably improve the recommendation results. The specific steps are as follows:

Overlapped candidates' selection: Applying the aforementioned approaches and getting the top 10000 recommendation results of each project. Finding the same candidates in both sets by calculating the intersection.

Overlapped candidates' ranking: Setting a balance proportional coefficient which assign different weights for different approaches. The rank of the candidates is calculated with Eq. 3.

$$rank = W_{DA} * rank_{DA} + W_{KA} * rank_{KA} \tag{3}$$

In which $\frac{W_{DA}}{W_{KA}}$ is the proportional coefficient of DA-based and KA-based approaches, and $rank_{DA}$ and $rank_{KA}$ represent the ranks of the candidates of the two approaches. Initially, we set the coefficient to $\frac{0.75}{0.25}$.

4 Experiment

4.1 Research Questions

In order to verify the effectiveness of our recommendation approaches and explore the influence when considering about different repositories. We focus on the following two research questions.

- Q1: What is the performance difference among three approaches over repositories with different popularities?
- Q2: How will the balance coefficient between the two kinds of activities affect the recommendation performance?

4.2 Experiment Datasets

To address the above research questions and validate our approach, we use the data in GitHub and StackOverflow. Here we choose the GHTorrent[3] MySQL dump released on March 2016, and the 2015 snapshot of StackExchange[4].

We find users both active in StackOverflow and GitHub by matching email MD5. After removing fake or deleted users, 72,877 left. For projects, we get 1,355,043 that at least one of the related users participated in before point-in-time (2014-09-14) and 165,741 projects that have new users after point-in-time.

We use h.d. (history developers) to represent the number of related users who focused on projects before point-in-time, and l.d. (latent developers) means the number of users who participate in the projects after point-in-time for the first time. Here we just consider about the projects that are popular after point-in-time because these projects can be used to validate the effectiveness of the approaches. After filtering, we get 136 popular projects whose h.d. bigger than 300 and l.d. bigger than 300. Also, we get 99 unpopular projects whose h.d. less than 2 and l.d. bigger than 100.

[3] http://ghtorrent.org/downloads.html.
[4] https://archive.org/details/stackexchange.

4.3 Evaluation Metrics

Accuracy: We use the accuracy to represent the availability of approaches, which is calculated by the division of the number of matched projects and the total number of test projects.

MRR: Mean Reciprocal Rank is widely used to evaluate the performance of recommendation systems. If the correct results rank in the front, the MRR value is high. Here, we use $rank_i$ to represent the rank of result i, $P\{p_1, p_2, ..., p_n\}$ to represent the set of test repositories, and $R_p\{r_1, r_2, ..., r_m\}$ to represent correctly matched results for project p. Then the MRR value is shown in Eq. 4.

$$MRR = \frac{1}{|P|}\sum_{i=1}^{|P|}\left(\frac{1}{|R_{P[i]}|}\sum_{j=1}^{|R_{P[i]}|}\frac{1}{rank_{R_{P[i]}[j]}}\right) \tag{4}$$

5 Experiment Results

5.1 Influence of Different Activities Towards Different Projects

Figures 2 and 3 present the accuracy of three different recommendation approaches for unpopular and popular projects. From the two figures, we can see that the accuracy of DA-based approach is better than that of KA-based approach. That is to say, the development activity is more important when recommending, which is consistent with reality. Developers may focus on many techniques, but will just apply one or two when developing projects. Asking or answering a question is much easier than following a repository.

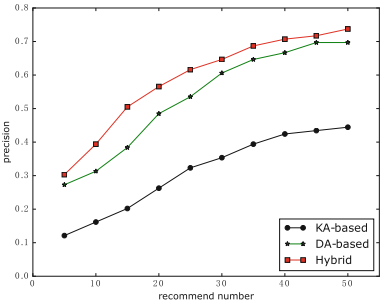

Fig. 2. Accuracy for unpopular projects

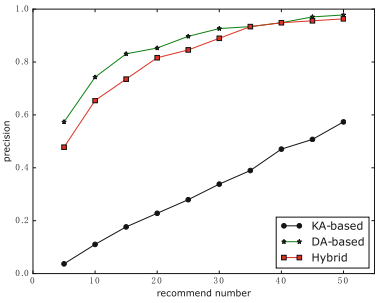

Fig. 3. Accuracy for popular projects

In Fig. 2, the hybrid approach performs the best, which means that for unpopular projects, knowledge sharing activity can help to improve the result. The reason is that the user association matrix generated from StackOverflow is more dense than that from GitHub. In Fig. 3, for popular projects, hybrid approach doesn't perform the best. This is because there are many developers focusing on

Table 1. MRR for unpopular and popular projects ($coefficient = \frac{0.75}{0.25}$)

Unpopular repositories										
	5	10	15	20	25	30	35	40	45	50
KA-based	.445	.295	.246	.229	.198	.172	.164	.155	.145	.136
DA-based	**.472**	**.323**	.267	**.206**	**.176**	**159**	**.142**	**.134**	.126	.120
Hybrid	**.503**	**.355**	.241	**.208**	**.182**	**.166**	**.148**	**.138**	.126	.112
Increase rate (%)	**6.57**	**9.91**	−9.74	**0.97**	**3.41**	**4.40**	**4.23**	**2.99**	0	−6.67
Popular repositories										
KA-based	.380	.187	.135	.113	.100	.085	.071	.063	.059	.052
DA-based	**.491**	**.321**	**.263**	**.225**	**.193**	**.167**	.149	.135	.125	.117
Hybrid	**.568**	**.362**	**.281**	**.244**	**.208**	**.170**	.148	.134	.123	.115
Increase rate (%)	**15.7**	**12.8**	**6.84**	**8.44**	**7.77**	**1.80**	−0.67	−0.74	−1.6	−1.7

the target project before point-in-time, which increases the prepared information for Collaborative Filtering algorithm.

From Table 1, we can see that hybrid approach tends to hit correct results in the front very often.

5.2 Influence of Different Coefficient Values in Hybrid Approach

Figures 4 and 5 show that coefficient value will influence the hybrid approach. In Fig. 4, the accuracy of hybrid approach increases with the decrease of coefficient value when recommending 5 to 10 developers to unpopular projects. However, for popular projects in Fig. 5, the result is opposite.

Meanwhile, Table 2 shows that for unpopular projects, the MRR of hybrid approach increases a lot when setting the coefficient to $\frac{0.25}{0.75}$, however decreases a lot for popular projects. Compare Table 2 to Table 1, when considering about MRR, small coefficient is more stable for unpopular projects because all the

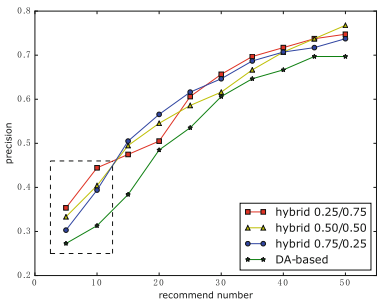

Fig. 4. Accuracy for unpopular projects with different coefficients

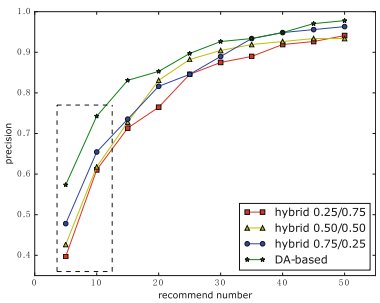

Fig. 5. Accuracy for popular projects with different coefficients

Table 2. MRR for unpopular and popular projects ($coefficient = \frac{0.25}{0.75}$)

Unpopular repositories										
	5	10	15	20	25	30	35	40	45	50
KA-based	.445	.295	.246	.229	.198	.172	.164	.155	.145	.136
DA-based	**.472**	**.323**	.267	**.206**	**.176**	**159**	**.142**	**.134**	**.126**	**.120**
Hybrid	**.483**	**.357**	**.297**	**.250**	**.213**	**.178**	**.161**	**.148**	**.142**	**.131**
Increase rate (%)	**2.33**	**10.5**	**11.2**	**21.4**	**21.0**	**11.9**	**13.4**	**10.4**	**12.7**	**9.17**
Popular repositories										
KA-based	.380	.187	.135	.113	.100	.085	.071	.063	.059	.052
DA-based	.491	.321	.263	.225	.193	.167	.149	.135	.125	.117
Hybrid	.394	.286	.228	.188	.159	.143	.131	.120	.109	.102
Increase rate (-%)	19.8	10.9	13.3	16.4	17.6	14.4	12.1	11.1	12.8	12.8

increasing rate are positive, however big coefficient is more suitable for popular projects with hybrid approach because the increasing rate tends to be positive.

In conclusion, using hybrid approach with small coefficient value to recommend for unpopular projects will get better accuracy and MRR value at the same time compared with DA-based approach. However, for popular projects, if the target repository considers more about the accuracy (wants to get suitable developers more probably), then choose DA-based approach. If it considers more about the MRR (wants to get suitable developers with fewer results), then use hybrid approach with big coefficient.

Acknowledgements. The research is supported by the National Natural Science Foundation of China (Grant No.61432020,61472430,61502512,61303064) and National Grand R&D Plan (Grant No. 2016-YFB1000805).

References

1. Silic, M.: Dual-use open source security software in organizations-Dilemma: help or hinder? Comput. Secur. **39**, 386–395 (2013)
2. Bhattacharya, P., Neamtiu, I., Shelton, C.R.: Automated, highly-accurate, bug assignment using machine learning and tossing graphs. J. Syst. Softw. **85**(10), 2275–2292 (2012)
3. Xuan, J., Jiang, H., Ren, Z., Zou, W.: Developer prioritization in bug repositories. In: ICSE, vol. 8543, no: 1, pp. 25–35 (2012)
4. Yu, Y., Wang, H., Filkov, V., Devanbu, P., Vasilescu, B.: Wait for it: determinants of pull request evaluation latency on GitHub. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories (MSR), pp. 367–371. IEEE (2015)
5. Yu, Y., Wang, H., Yin, G., Wang, T.: Reviewer recommendation for pull-requests in GitHub: what can we learn from code review and bug assignment? Inf. Softw. Technol. **74**, 204–218 (2016)

6. Yu, Y., Wang, H., Yin, G., Ling, C.X.: Reviewer recommender of pull requests in GitHub. In: ICSME, pp. 609–612. IEEE (2014)
7. Zhang, L., Zou, Y., Xie, B., Zhu, Z.: Recommending relevant projects via user behaviour: an exploratory study on GitHub (2014)
8. Orii, N.: Collaborative topic modeling for recommending GitHub repositories (2012)
9. Vasilescu, B., Filkov, V., Serebrenik, A.: Stackoverflow and github: associations between software development and crowdsourced knowledge. In: ASE/IEEE International Conference on Social Computing, pp. 188–195 (2013)
10. Wang, H., Wang, T., Yin, G., Yang, C.: Linking issue tracker with q and a sites for knowledge sharing across communities. IEEE Trans. Serv. Comput. **PP**, 1–14 (2015)
11. Silvestri, G., Yang, J., Bozzon, A., Tagarelli, A.: Linking accounts across social networks: the case of stackoverflow, github and twitter. In: International Workshop on Knowledge Discovery on the WEB, pp. 41–52 (2015)
12. Venkataramani, R., Gupta, A., Asadullah, A., Muddu, B., Bhat, V.: Discovery of technical expertise from open source code repositories. In: International Conference on World Wide Web Companion, pp. 97–98 (2013)