2017 International Workshop on Complex Systems and Networks (IWCSN)
December 8-10, 2017, Doha, Qatar

Expert Recommendation in OSS Projects Based on Knowledge Embedding

Chenbo Fu, Mingming Zhou, and Qi Xuan
College of Information Engineering
Zhejiang University of Technology
Hangzhou 310023, China
Email: xuanqi@zjut.edu.cn

Hong-Xiang Hu
College of Mathematics
Hangzhou Dianzi University
Hangzhou 310018, China
Email: kukunan911@hotmail.com

***Abstract*—Modern Open Source Software (OSS) projects depend on the globally-distributed and synchronized software development. The online collaboration promotes more and more developers to join in OSS projects, while on the other hand, integrating new developers with teams is challenging and pivotal to the success of a project. In this paper, we propose a novel expert recommendation method, based on knowledge embedding, that realizes real-time recommendation for working developers. To capture structural information of source files in call graph, we use *node2vec* algorithm to convert file entities within projects into knowledge mappings within low-dimensional space, based on which we further propose four features to capture the work status and social relationship of developers. We then design a recommender system using random forest method to recommend appropriate experts for the developers. Experiments on 20 Apache OSS projects show that, compared with the baseline methods, our approach behaves significantly better in terms of a series of performance metrics.**

Keywords*—expert recommendation; open source software; knowledge embedding; node2vec; machine learning.

I. Introduction

The rapid growth of software development practices in recent years has been a bonanza for technology diffusion and commercial advantage. These outstanding success made in Open Source Software (OSS) has mostly involved the co-operation pattern of social-technical interaction and globally-distributed fashion.

Specifically, Apache Software Foundation (ASF) [1], Github [2], and Stack Overflow (SO) [3] are classic communities in software engineering domain. An ASF project involves a software development team aiming at creating a robust, commercial-grade, multi-functional, and freely-available source code implementation. Typically, the project is jointly managed by a group of volunteers, located around the world, who contribute ideas, code, and documentation. The working product that is contributed by committing to files would be recorded in a Git repository. In addition, hundreds of users communicate, plan, and develop the software through the Internet. As well, the history of communication information would be stored in a particular mailing list.

The synchronous development, as a kind of social synchrony [4], [5] in software engineering, is reported as highly associated with effective productivity and coordination: during the co-commit time, the project size often grows faster with less coding effort than other time [1]. From the perspective of developers themselves, Xuan *et al.* [6] found that developers with fair balance between work and talk tend to produce as much work as those putting more emphasis on work or talk. Most interestingly, the fair balance for developers is important to sustain OSS projects. Moreover, unlike traditional code ownership heuristics based on the authorship of code changes, Thongtanunam *et al.* [7] found that code review activities become more and more important to be a reference standard of code ownership and a judge criterion of developer's contribution in Modern Code Review (MCR). Their statistic result shows that 67% to 86% of developers only contribute to modules by reviewing code changes, 18% to 50% of these review-only contributors are documented core developers of the studied systems.

However, finding a list of appropriate developers to exchange knowledge and experiences and seek help is always not straightforward. Rubin *et al.* [8] gave an illustration that one participant, working in a large organization, mentioned that he has a problem to find the experts with experiences relevant to what he is trying to develop. They also reported that time differences, language differences, and the lack of physical access make coordinating people in different time zones less productive. An empirical investigation by Thongtanunam *et al.* [9] shows that 4% to 30% of reviews have code-reviewer assignment problem in OSS projects, these reviews with code-reviewer assignment problem require 12 days longer to approve code changes.

Expert recommendation is widely studied in the area of software engineering. Bayati *et al.* [10] proposed a security expert recommender for social Q&A Websites, e.g., SO, that applies security ontology, glossary, programming language and location information to find the experts in appropriate local area who answered to related posts with the highest answer count and vote value. Naguib *et al.* [11] employed Latent Dirichlet Allocation (LDA) to cluster bug reports into topics to be embedded as activity profiles for users. For a new bug report, the model can assign recommendation(s) according to the related activity profiles. Shokripour *et al.* [12] utilized a noun extraction process on several information sources to determine bug location information and a simple

978-1-5386-1890-5/17/$31.00 ©2017 IEEE

term weighting scheme to provide a bug report assignment recommendation. Yu *et al.* [13] combined technical semantic similarity with social comment relation network analyzing to recommend highly relevant code-reviewers for GitHub pull-requests. Steinmacher *et al.* [14] considered source code artifacts, issue tracker, mail threads and workspace information to calculate the current technical and social interest score, and then proposed a recommendation system to help newcomers find the most adequate support from other people in OSS projects. Balachandran *et al.* [15] introduced a reviewer recommendation tool, namely ReviewBot, that uses historical records of changed lines within source files to assign appropriate reviewers for VMware projects. Patanamon *et al.* [9] proposed a file location-based code-reviewer recommendation approach for modern code review, called RevFinder, that leverage file path similarity of relevant review requests. Rahman *et al.* [16] proposed a recommendation technique, namely CORRECT, to identify code-reviewers using cosine similarity of tokens (shared libraries and adopted technologies) from relevant requests in GitHub.

In this paper, we propose a novel approach for realizing real-time expert recommendation for developers in ASF projects. Inspired by Grover *et al.* [17], we first adopt a feature representation model for converting source file entities into knowledge mappings in low-dimensional space to capture file- and function-level information; then we look up relevant knowledge mappings of source files to be integrated as a working state of one developer at the commit time; finally we conduct recommendation system using machine learning techniques to predict appropriate experts for this developer.

The rest of paper is organized as follows: In Sec. II, we conduct an exploratory study on ASF project dataset. Then, we propose a novel recommendation system based on knowledge embedding in Sec. III. The evaluation metrics and the comparison with baseline methods are shown in Sec. IV and Sec. V, respectively. Finally, the paper is discussed and concluded in Sec. VI.

II. Dataset

A. Data Description

We use the same dataset as Xuan *et al.* [1], [6], [18], and mainly focus on 20 ASF projects. In each project, the data provides commit activities and email communication records for developers. For each commit, we employ the developer ID, the commit time, the source file ID. For each email communication, we employ the sender ID, receiver ID and the sending time. Note that the communication records where the sender ID is the same as the receiver ID are removed, because these records cannot be regarded as social interactions. We assume receiver IDs are experts in our experiments.

B. Exploratory Study

Before we introduce our method, we first take a glance at our data. In Fig. 1, we plot the proportion of technical communications in each project, i.e., the proportion of emails sent or received by code contributors, i.e., *developers*, over

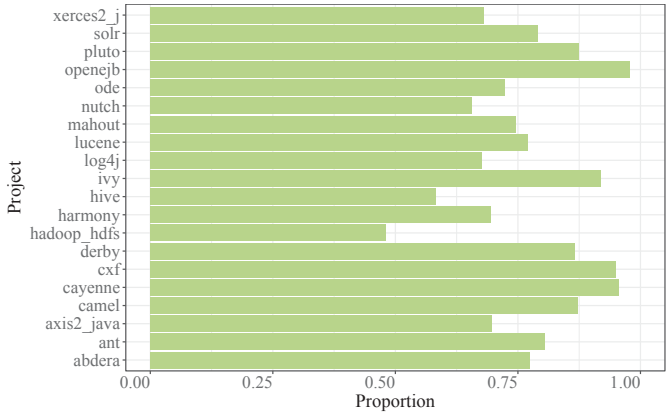

Fig. 1. The proportion of technical communications over all communications in each project.

Fig. 2. The proportions of developers sent or received emails, (blue) over all developers, and (green) over all email contacts, for each project.

all the emails in each project. It is shown that, such technical communications occupy a large portion in all email communications, especially for *cayenne*, *cxf*, *openejb*, more than 95% emails are sent or received by the developers. Furthermore, we find that the developers who also sent or received emails take up a large portion in all developers, but only occupy a relatively small portion in all email contacts, in each project, as shown in Fig. 2. This means that developers are used to share knowledge and experience by email in these projects and imply us that design of expert recommendation system should consider not only the technical characteristics based on the commits of developers on particular files but also the social communication patterns between them and other users.

III. Method

The framework of this study is presented in Fig. 3. We formulate our expert recommendation as a machine learning problem, and solve it by three steps, i.e., *file embedding*, *feature extraction*, and *machine learning*. Specially, we first use

Fig. 3. The framework of our study.

node2vec method to learn the domain-specific file embeddings from large numbers of source files in ASF projects. Then, we propose four features to describe the behavioral pattern for each developer. Finally, we use machine learning method to establish a predicting model.

A. File Embedding

The traditional expert recommendation system only consider the independent source file, however, the file-file relationships also exhibit the significant influences on developer's working patterns, e.g., focus shifting pattern [18]. Thus, to better describe the developer's working patterns so as to design better recommendation system, we should take file-file relationships into consideration. Here, we use *Doxygen* tool [19] to gather the call graphs of ASF projects, based on which we further establish *File Dependency Network* (FDN). FDN is defined as a weighted undirected graph $G = (V, E, W)$, where nodes represent source files, and two nodes v_i and v_j are linked if there is a dependency relationship between the corresponding source files, *i.e.*, at least one function in file v_i (or v_j) calls a function in file v_j (or v_i). The weight w_{ij} of link represents the total number of times that the functions in v_i and those in v_j call each other.

Given the FDN, we next use the *node2vec* proposed by Grover *et al.* [17] to capture the rich systematic and functional features of source files in a project. *node2vec* is a new algorithm for mapping nodes in a network to the feature vectors in a low-dimensional space, preserving the neighborhood information of these nodes. In our study, by utilizing this method, we map the source files in the underlying call graph to domain-specific file embedding. In particular, given a source file v_i, the *node2vec* algorithm seeks the optimization of objective function which maximizes the log conditional probability of neighborhood set $V_N(v_i)$ for a source file v_i:

$$\max_f \sum_{v_i \in V} \log P(V_N(v_i)|f(v_i)), \tag{1}$$

where f is the mapping function from network representation to feature representation, *i.e.*, $f : V \to \Re^d$, and d is the dimension of feature space. With the assumptions of conditional independence and symmetry on feature space, the objective function can be simplified to:

$$\begin{aligned} & \sum_{v_i \in V} \log P(V_N(v_i)|f(v_i)) \\ = & \sum_{v_i \in V} \sum_{v_{n_i} \in V_N(v_i)} \log P(v_{n_i}|f(v_i)) \\ = & \sum_{v_i \in V} \sum_{v_{n_i} \in V_N(v_i)} \log \frac{\exp(f(v_{n_i}) \cdot f(v_i))}{\sum_{v_u \in V} \exp(f(v_u) \cdot f(v_i))}, \end{aligned} \tag{2}$$

where v_{n_i} is the node in the neighborhood set $V_N(v_i)$. Calculation of Eq. (2) is very expensive in large network. To solve this problem so as to improve the algorithm efficiency, *node2vec* proposed a second order random walk sampling strategy with a parameter q controlling the balance between *Breadth-First Sampling* (BFS) and *Depth-First Sampling* (DFS). In particular, starting from the source file v_i, we simulate a random walk with length l. Then the ith file in the walk path is generated by the following distribution:

$$P(c_i = v_x | c_{i-1} = v_u, c_{i-2} = v_t) = \begin{cases} \pi_{ux}/Z & \text{if } (v_u, v_x) \in E \\ 0 & \text{Otherwise} \end{cases} \tag{3}$$

where $\pi_{ux} = \alpha(v_t, v_x) \cdot w_{ux}$ is the unnormalized transition probability, with the parameter

$$\alpha(v_t, v_x) = \begin{cases} \frac{1}{p} & \text{if } d(v_t, v_x) = 0 \\ 1 & \text{if } d(v_t, v_x) = 1 \\ \frac{1}{q} & \text{if } d(v_t, v_x) = 2 \end{cases} \tag{4}$$

w_{ux} being the link weight between nodes v_u and v_x, and $Z = \sum_{v_x} \pi_{ux}$ being the normalized constant. In Eq. (4), $d(v_t, v_x)$ is the shortest path between nodes v_t and v_x, p and q are called return and in-out parameters, controlling the walk direction, respectively. Here, we implement 10 random walks of the fixed length $l = 80$ starting from every node in a call graph, both p and q are set to 1. After sampling, the source files in the call graph are mapped into node sequences. Feeding them into *skip-gram* architecture with the objective function Eq. (2), we finally obtain the file embedding vectors, with the dimension set to 32.

The *node2vec* algorithm has the following two advantages. First, it can be used to handle large-scale networks. It is known that the *Bag-of-words* model [20], another popular vector representation method, can transform the word to vector representation, in terms of words frequency, and achieves great successful in Natural Language Processing (NLP), but it is of low efficiency when dealing with big data. By comparison, using the second order random walk, *node2vec* improves the search efficiency dramatically, and shows the ability to handle large-scale networks. Second, *node2vec* uses the local topological information of nodes, making the file embedding vector capture some domain-specific characteristics. This is similar to the semantics in NLP, and thus *node2vec* may be useful in analogical reasoning tasks. For instance, in *Derby* (a database management system), the file named as LocalizedInput.java handles localized input by calling the file utilMain.java used to run database, while the file LocalizedOutput.java generates localized output by calling the file Main.java used to parsing and controlling. By mapping these files into file embedding vectors, we have f(LocalizedInput.java)-f(utilMain.java)+f(Main.java) is close to f(LocalizedOutput.java). This suggests that the relationships between source files can be indeed obtained by using basic mathematical operations on file embedding vectors.

B. Feature Extraction

The commit and communication patterns of a developer are essentially task-oriented [6], [21], and thus exhibit highly dynamic characters. This fact suggests that we should consider

the temporal features of developers when designing the recommendation system. In this study, after mapping the source files into file embedding vectors, we propose four features to capture the work pattern of each developer, as following:

- **Temporal Technical Feature (TTF)**
  Assume developer D_i sent an email to an expert at time t, then TTF for the developer is defined as the mean value of committed files in file embedding representation at commit time t_i, which is closest to t:
  $$\mathrm{TTF}_i(t_i) = \frac{1}{|F_i(t_i)|} \sum_{v \in F_i(t_i)} f(v), \tag{5}$$
  where $F_i(t_i)$ is the set of source files committed by developer D_i at time t_i, and the operator $|\cdot|$ represents the size of a set.
- **Historical Technical Feature (HTF)**
  For developer D_i, HTF is defined as the average TTF of the developer by commit time t_i.
  $$\mathrm{HTF}_i(t_i) = \frac{1}{N_c(t_i)} \sum_{\tau \le t_i} \mathrm{TTF}_i(\tau) \cdot \Delta d_\tau, \tag{6}$$
  where $N_c(t_i)$ is the number of commits by time t_i. The time decay factor is defined as:
  $$\Delta d_\tau = \exp[-(t-\tau)^2/\sigma^2], \tag{7}$$
  where τ is the commit time in history. In the present work, the regularization factor is set as $\sigma = 7$ (days).

As discussed in Sec. II, social activities should not be ignored when designing expert recommendation system. Thus, we proposed two social features as following:

- **Temporal Social Feature (TSF)**
  For developer D_i, TSF is defined as the average TTF of the last Q users that D_i had contacted through email by time t. It should be note that if the contacted user is not a developer, his/her TTF is set to zero; and for each contacted developer D_k, only the commit at the closest time to t, denoted by t_k, is considered, satisfying $t_k \le t$. Thus, we have
  $$\mathrm{TSF}_i(t) = \frac{1}{|V_i|} \sum_{D_k \in V_i} \mathrm{TTF}_k(t_k), \tag{8}$$
  where V_i is the set of contacted users of D_i.
- **History Social Feature (HSF)**
  Simply replacing TTF by HTF, we get:
  $$\mathrm{HSF}_i(t) = \frac{1}{|V_i|} \sum_{D_k \in V_i} \mathrm{HTF}_k(t_k). \tag{9}$$
  Here, for TSF and HSF, we always have $|V_i| = Q$.

C. Machine Learning Model

After extracting the four features, we adopt the *Random Forest* (RF) algorithm to design expert recommendation system. Here, in each OSS project, we divide the data into a training set and a test set in chronological order, containing 80% and 20% of the data, respectively. The RF model is generated by R package *randomForest*, and the parameters are set to *ntree*=500, *mtry*=11.

IV. Performance Metrics

To evaluate our approach, we use *accuracy*, *mean precision*, *mean recall*, *mean F1-score* and *mean reciprocal rank* as the performance metrics. The first four metrics are all based on the top@k recommendation list. Let R be the set of all email requests of experts in the test set and D be the set of ground truth experts, we have:

- **Top@k Accuracy**
  It is defined as the proportion of correct recommendations among all the email requests in the test set:
  $$\mathrm{ACC} = \frac{\sum_{r \in R} isCorrect(r, Top@k)}{|R|}, \tag{10}$$
  where $isCorrect(r, Top@k)$ returns 1 if the email request r of the expert as the ground truth is included in the top k recommendation list, and returns 0 otherwise.
- **Top@k Mean Precision**
  For expert $D_i \in D$, the top@k precision is the proportion of recommendations that correctly recommended D_i among all the recommendations including D_i, which is defined as:
  $$\mathrm{PRE}(i) = \frac{\sum_{r \in R_i} isCorrect(r, Top@k)}{|Top@k(i)|}, \tag{11}$$
  where R_i means the set of email requests of expert D_i and $Top@k(i)$ is the number of all the $Top@k$ recommendation lists, for all the test samples, that contain expert D_i. Then, the top@k mean precision is defined as:
  $$\mathrm{PRE} = \frac{1}{|D|} \sum_{D_i \in D} \mathrm{PRE}(i). \tag{12}$$
- **Top@k Mean Recall**
  For expert $D_i \in D$, the top@k recall is the ratio of recommendations that correctly recommended D_i to the number of email requests of D_i in ground truth, which is defined as:
  $$\mathrm{REC}(i) = \frac{\sum_{r \in R_i} isCorrect(r, Top@k)}{|R_i|}. \tag{13}$$
  Then, the top@k mean recall is defined as:
  $$\mathrm{REC} = \frac{1}{|D|} \sum_{D_i \in D} \mathrm{REC}(i). \tag{14}$$
- **Top@k Mean F1-score**
  For expert $D_i \in D$, the F1-score is the harmonic mean of precision and recall, then the top@k mean F1-score is defined as:
  $$\mathrm{F1} = \frac{1}{|D|} \sum_{D_i \in D} \frac{2 \times \mathrm{PRE}(i) \times \mathrm{REC}(i)}{\mathrm{PRE}(i) + \mathrm{REC}(i)}. \tag{15}$$
- **Mean Reciprocal Rank**
  For each email request of expert, denoted by r, we define Reciprocal Rank (RR) as the multiplicative inverse of the rank of the target expert in the recommendation list, denoted by $\mathrm{RR}(r)$. If the target expert is not included in the list, the corresponding $\mathrm{RR}(r)$ will be 0. The

mean reciprocal rank for all email requests thus can be calculated by

$$\text{MRR} = \frac{1}{|R|}\sum_{r\in R}\text{RR}(r). \tag{16}$$

Based on these metrics, we evaluate the performance of our method against several traditional recommendation methods.

V. Experiments and Results

A. Traditional Recommendation Methods

Before the experiments, we first introduce two traditional recommendation methods for comparison.

1) Collaborative Filtering (CF): The basic idea of CF is grouping users or items according to similarity [22], [23]. COde Reviewer REcommendation based on Cross-project and Technology experience (CORRECT) [16] is an improved recommendation method used in software engineering, which utilizes the developers' experiences in Github to measure their similarity.

Here, we focus on ASF projects and use the committed source files to characterize the experience of a developer. Assume developer D_i sent an email request at time t, and his/her latest commit, before time t, occurred at time t_i, with the set of committed source files denoted by $F_i(t_i)$. We then get the latest h email requests sent by developers before time t, and for each of these email requests r sent by developer D_j, we obtain the latest commit of the developer before this email request. Suppose this commit occurred at time t_j, the similarity between D_i and D_j based on the two commits is thus calculated by

$$\text{Sim}(D_i, D_j) = \frac{F_i(t_i)\cap F_j(t_j)}{\sqrt{|F_i(t_i)|}\cdot\sqrt{|F_j(t_j)|}}. \tag{17}$$

We thus can rank the receivers of the h email requests and then recommend them as experts to developer D_i according to the similarity between D_i and the sender of these email requests. In our study, h is set to 5. In fact, we varied h from 5 to 30, and get the best performance when $h = 5$.

2) Random Walk based Context-aware Friend Recommendation (RWCFR): RWCFR considers the current status of a user and provide personalized recommendations in social network [24]. In this study, we use the current commit and communication patterns to represent the current status of a developer. Specifically, assume developer D_i send an email at time t and the latest commit, before time t, occurred at time t_i, then we construct the temporal status network of developer D_i according to the following relationships.

- *Current Ego-File Relationship*: Between developer D_i and the source files committed by D_i at time t_i.
- *Current Ego-Contact Relationship*: Between developer D_i and the last n_s email contacts of D_i before time t.
- *Current Developer-File Relationship*: Assume time t_j is the time for the last commit of developer D_j in the above contact list of D_i before time t. Then the current developer-file relationship are the relationship between the source files in the last n_f commits by time t_j and their submitters.

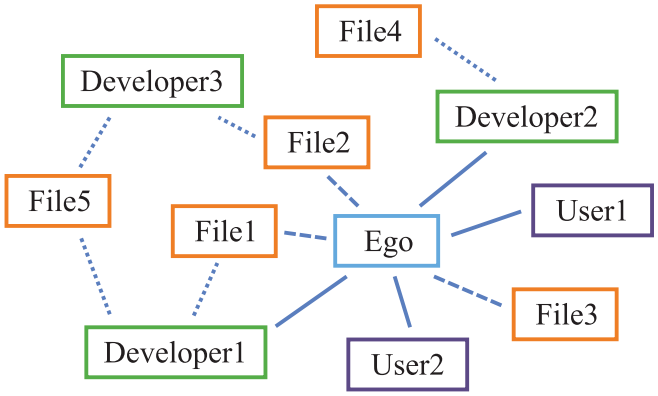

Fig. 4. Temporal status network. There have three relationships: current ego-file relationship (longdash line), current ego-contact relationship (solid line) and current developer-file relationship (dotted line).

TABLE I
Comparison between Our Approach and The Two Baseline Methods.

	ACC	PRE	REC	F1	MRR
Our Approach	**0.5631**	**0.3478**	**0.3099**	**0.3278**	**0.4156**
CF	0.4712	0.2361	0.2210	0.2283	0.3128
RWCFR	0.5202	0.2977	0.2066	0.2439	0.3002

Considering developers, users and source files as nodes and integrating the above relationships as links, we can get a temporal status network for each developer, as shown in Fig. 4. In our study, n_f and n_s are set to 5 and 15, respectively. In fact, we varied n_f from 10 to 50 and n_s from 5 to 20, and get the best performance when $n_f = 5$, $n_s = 15$. We then employ the random walk starting from developer D_i with the transition probability proportional to the weight (repetitive relationship) of the link, and the walk length is set to 100 times of network size. We calculate the frequency of visited nodes in the walk path, and the developer or user of higher frequency is more likely to become an expert.

B. Comparison

We compare our approach against two baseline recommendation methods: CF and RWCFR, on the five performance metrics. As demonstrated in TABLE I, we find that the improvement of our method over the two baseline methods is apparently: compare with baseline methods, the Top@5 Accuracy (ACC), Top@5 Mean Precision (PRE), Top@5 Mean Recall (REC), Top@5 Mean F1-score (F1) and Mean Reciprocal Rank (MRR) are at least improved by 8.2%, 16.8%, 40.2%, 34.4% and 32.9%, respectively. Note that, for the two social features, i.e.,TSF and HSF, in our approach, we set the parameter $Q = 7$, meaning that only the most recent seven contacts are considered when calculating these features. We varied the value of Q and find $Q = 7$ is an appropriate value to get reasonable performance.

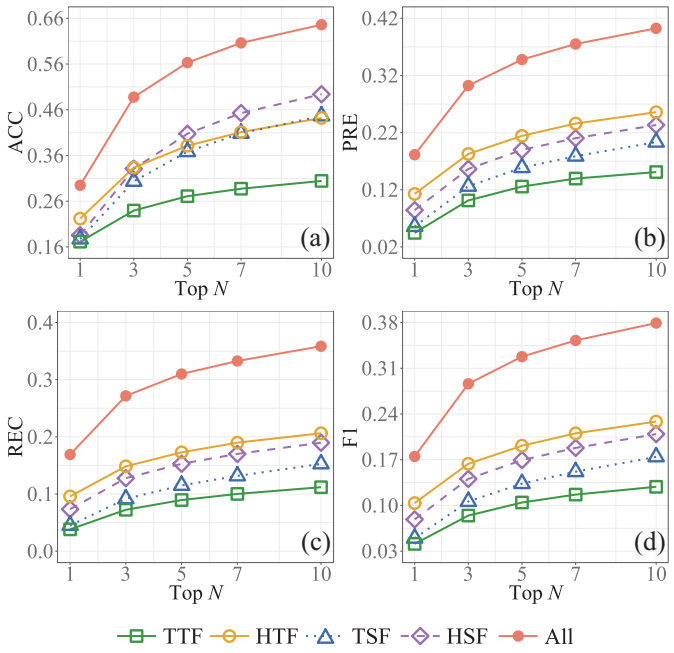

Fig. 5. The performance metrics, including ACC, PRE, REC and F1, based on each and all the four features, as functions of recommendation list length.

Besides, we also investigate the recommendation performance of our method just using one of the four features. As shown in Fig. 5, we find that the two features associated to *history*, i.e., HTF and HSF, outperform the other two in general. This phenomenon implies that, by comparison, the historical features HTF and HSF, can provide more comprehensive information to describe the status of developer. In addition, we also find that when applying all the four features for recommendation, the performance metrics have significant improvements over the single feature.

For the performance metrics PRE, REC and F1, HTF behaves best, probably because the task-driven essence of these OSS projects makes technical features more appropriate to characterize appropriate experts. On the other hand, HSF shows its advantage in ACC which mostly depends on the number of correct recommendations. As we mentioned in Sec. II, developers are used to share knowledge and experience by email. Although this part of developers only occupy small proportion of users, they are still more likely to be recommended as experts.

VI. Discussions and Conclusion

The present work is a novel extension to the previous studies of expert recommendation. Prior to our work, most of the studies on experts recommendation had been focus on independent source file, and omitted the file-file relationship. The current study, while considering the file-file relationship, we construct call graph of source file and mapping this relationship into 32 dimension space, which capturing the file- and function- level information. Considering the characteristics of online cooperation in ASF projects, the proposed features

TABLE II
The Performances of Our Approach with Different Parameter q.

		ACC	PRE	REC	F1	MRR
p=1	q=0.25	0.5248	0.2926	0.2504	0.2699	0.3763
	q=0.5	0.5247	0.2949	0.2521	0.2718	0.3750
	q=1	**0.5631**	**0.3478**	**0.3009**	**0.3278**	**0.4156**
	q=2	0.5272	0.2964	0.2553	0.2743	0.3781
	q=4	0.5253	0.2900	0.2485	0.2677	0.3758

in our work capture the social and working status of developer, and performs well in the design of recommendation system.

It is worth to noting that, in Sec. III, we have mentioned that the parameter p and q are important parameters in *node2vec*. However, compare to p, the parameter q have more clearer physical meanings, *i.e.*, $q > 1$, the random walk is approximately BFS behavior and $q < 1$ is DFS-like exploration. In this work, beside $q = 1$, we also explore different q values on the 20 OSS projects, *e.g.*, $q = 0.25, 0.5, 2, 4$. As shown in Tab. II, the results show that $q = 1$ performs best under our evaluation metrics.

In summary, we adopt the *node2vec* method to mapping the source file in call graph to file embedding vector and propose four new features, including TTF, HTF, TSF, and HSF, to capture the work status and social relationship of developers. By utilizing RF algorithm, we design a new expert recommendation approach in OSS projects. The experiments validate the effectiveness of our method. Furthermore, when the recommendation list increases, HTF dominates the other three features under the performance metrics *PRE*, *REC* and *F1*. Our work highlights that the network theory integrated with the machine learning methods can improve the predicting performance, and thus is necessary and important complement to the current recommendation methods.

Acknowledgment

This work is partially supported by National Natural Science Foundation of China (11505153, 61572439, 61273212), Zhejiang Provincial Natural Science Foundation of China (LQ15A050002), and the Control Science and Engineering Discipline Prior Discipline of Zhejiang Province (20170706).

References

[1] Q. Xuan and V. Filkov, "Building it together: Synchronous development in oss," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 222–233.
[2] B. Vasilescu, K. Blincoe, Q. Xuan, C. Casalnuovo, D. Damian, P. Devanbu, and V. Filkov, "The sky is not the limit: multitasking across github projects," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 994–1005.
[3] B. Vasilescu, V. Filkov, and A. Serebrenik, "Stackoverflow and github: Associations between software development and crowdsourced knowledge," in *Social computing (SocialCom), 2013 international conference on*. IEEE, 2013, pp. 188–195.
[4] Q. Xuan, Z. Zhang, C. Fu, H. Hu, and V. Filkov, "Social synchrony on complex networks," *IEEE Transactions on Cybernetics*, 2017.
[5] Q. Xuan and V. Filkov, "Synchrony in social groups and its benefits," in *Handbook of Human Computation*. Springer, 2013, pp. 791–802.
[6] Q. Xuan, P. Devanbu, and V. Filkov, "Converging work-talk patterns in online task-oriented communities," *PloS one*, vol. 11, no. 5, p. e0154324, 2016.

[7] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, "Revisiting code ownership and its relationship with software quality in the scope of modern code review," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 1039–1050.

[8] J. Rubin and M. Rinard, "The challenges of staying together while moving fast: An exploratory study," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 982–993.

[9] P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida, and K.-i. Matsumoto, "Who should review my code? a file location-based code-reviewer recommendation approach for modern code review," in *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*. IEEE, 2015, pp. 141–150.

[10] S. Bayati, "Security expert recommender in software engineering," in *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, 2016, pp. 719–721.

[11] H. Naguib, N. Narayan, B. Brügge, and D. Helal, "Bug report assignee recommendation using activity profiles," in *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*. IEEE, 2013, pp. 22–30.

[12] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani, "Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation," in *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 2013, pp. 2–11.

[13] Y. Yu, H. Wang, G. Yin, and C. X. Ling, "Reviewer recommender of pull-requests in github," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*. IEEE, 2014, pp. 609–612.

[14] I. Steinmacher, I. S. Wiese, and M. A. Gerosa, "Recommending mentors to software project newcomers," in *Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering*. IEEE Press, 2012, pp. 63–67.

[15] V. Balachandran, "Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation," in *Software Engineering (ICSE), 2013 35th International Conference on*. IEEE, 2013, pp. 931–940.

[16] M. M. Rahman, C. K. Roy, and J. A. Collins, "Correct: code reviewer recommendation in github based on cross-project and technology experience," in *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, 2016, pp. 222–231.

[17] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 855–864.

[18] Q. Xuan, A. Okano, P. Devanbu, and V. Filkov, "Focus-shifting patterns of oss developers and their congruence with call graphs," in *ACM Sigsoft International Symposium on Foundations of Software Engineering*, 2014, pp. 401–412.

[19] C. Mao, "Structure visualization and analysis for software dependence network," in *Granular Computing (GrC), 2011 IEEE International Conference on*. IEEE, 2011, pp. 439–444.

[20] Z. S. Harris, "Distributional structure," *Word*, vol. 10, no. 2-3, pp. 146–162, 1954.

[21] Q. Xuan, H. Fang, C. Fu, and V. Filkov, "Temporal motifs reveal collaboration patterns in online task-oriented networks," *Physical Review E*, vol. 91, no. 5, p. 052813, 2015.

[22] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "Grouplens: an open architecture for collaborative filtering of netnews," in *Proceedings of the 1994 ACM conference on Computer supported cooperative work*. ACM, 1994, pp. 175–186.

[23] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*. ACM, 2001, pp. 285–295.

[24] H. Bagci and P. Karagoz, "Context-aware friend recommendation for location based social networks using random walk," in *Proceedings of the 25th International Conference Companion on World Wide Web*. International World Wide Web Conferences Steering Committee, 2016, pp. 531–536.