2016 23rd Asia-Pacific Software Engineering Conference

Task Recommendation with Developer Social Network in Software Crowdsourcing

Ning Li*, Wenkai Mo*, Beijun Shen*†
*School of Software, Shanghai Jiao Tong University, Shanghai, China.
†bjshen@sjtu.edu.cn

***Abstract*—Recently, crowdsourcing has been increasingly used in software industry to lower costs and increase innovations, by utilizing experiences, labor, or creativity of developers worldwide. In software crowdsourcing platforms, developers expect to find suitable tasks for their interests and abilities. So it is significant for software crowdsourcing to build a recommender system to match developers with suitable tasks. However, there are a significant number of inactive developers who have very sparse historical behavior records in the platform, and thus state-of-the-art recommendation approaches in software crowdsourcing, such as collaborative filtering, suffer from this cold-start problem. In this paper, a social influence-based method is proposed to recommend suitable tasks for both active and inactive developers. The essential idea of the novel method is (1) to construct developer social network from developer behaviors, such as browsing and bidding for tasks, (2) to calculate the influence degrees between developers using developer social network, (3) to recommend tasks to active developers using SiSVD, and (4) to recommend tasks to inactive developers by combining the recommended tasks of their friends. We have evaluated our method on a large real data set from the JointForce, a popular software crowdsourcing platform in China. The results show that our method is feasible and practical for recommendation in software crowdsourcing. In particular, the F1-Measure of our method for inactive developers with task-bidding friends is increased by 16.7% than other previous approaches averagely.**

***Keywords*—Software Crowdsourcing, Task Recommendation, Social Network, Social Influence**

I. Introduction

Recently, crowdsourcing has been widely used in many tasks, such as taxonomy construction, some question answering tasks and so on. Software crowdsourcing platforms, such as Topcoder[1], Joint Force[2] or Zhubajie[3], demonstrate the advantage of software crowdsourcing in term of cost decrease and quality increase. Here a general procedure of software development is that: (1) A task requester firstly posts a task with a budget on the platform; (2) If a developer browses this task, and thinks it is interesting and within his/her ability to finish it, he/she can bid for it; (3) A task is usually bided by many developers, then the requester will select the best developer(s); (4) If the selected developer(s) complete the task in time with high-quality, then the requester will pay for the task.

However, if the developers with proper ability cannot find their interesting tasks in time, it will bring much loss to developers, task requesters and platform itself and then make these developers and task requesters leave the platform. To attract and catch more users, a good recommender system for a software crowdsourcing platform is very important, which is also emphasized by academia. For instance, Aldhahri et al. [1] considered software crowdsourcing should address two main objectives: (1) to match developers with suitable tasks that fit their interests and abilities, and raise their rewards; (2) to raise the rates of completed tasks and the aggregated commissions accordingly, which could be realized by building an efficient recommender system.

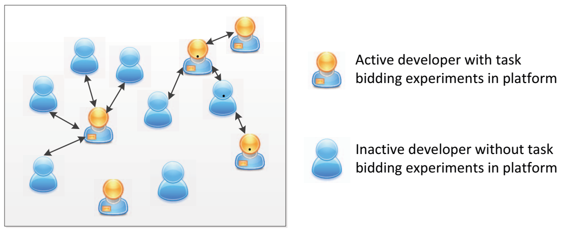

Active developer with task bidding experiments in platform
Inactive developer without task bidding experiments in platform

Fig. 1. Developer Social Networks

There are two kinds of main approaches in recommender systems, collaborative filtering [2][3] and content-based approaches [4]. Collaborative filtering (CF) approaches rely on past user behaviors, e.g., bidding for or completing tasks in software crowdsourcing, and do not require the creation of explicit profiles. However, the main drawback of CF is that it suffers from cold-start problem for developers without sufficient behaviors. Although content-based approaches can deal with cold-start problem through making recommendation by developers and tasks profiles, constructing proper profiles for recommendation is still difficult.

Some researches find that it will be effective to cope with this problem by introducing new data sources like social information between users [5][6]. Software crowdsourcing platforms also have these social information among developers, but it has unique features compared with traditional social networks according to our observation in some typical software crowdsourcing platforms. Fig. 1 illustrates a developer social network. The edges in the graph denote social influence between developer. All developers are divided into two categories, active developers and inactive developers based on whether they have made bids for tasks. Inactive developers have made no bids for tasks and they are a kind of cold-start developers. With this observation, we attempt to introduce the factor of social influence degrees to recommend tasks through constructing developer social network. However, there

[1]https://www.topcoder.com/
[2]https://www.jointforce.com/
[3]http://www.zbj.com/

are three challenges we may face: (1) How to calculate social influence degrees between developers based on developer behaviors and social information in software crowdsourcing platform? (2) How to build recommender systems by utilizing various behaviors of developers? (3) How to solve cold-start problem for inactive developers?

To address these challenges, we propose a four-steps task recommendation method. (1) Construction of Developer Social Network. All developers in software crowdsourcing platform are divided into two categories. Developers who have made bids for tasks are active developers, and the other developers are inactive developers who have no records about bidding for and completing tasks and thus could be regarded as cold-start developers. We construct a developer social network among these developers. (2) Social Influence Analysis. We define a formula to calculate social influence degrees between every two developers based on developer behaviors and social information in crowdsourcing platform; (3) Recommendation for active developers. By considering developer-task ratings and implicit feedback of their friends, a collaborative filtering approach is proposed to recommend tasks to active developers. (4) Recommendation for inactive developers. Inactive developers are recommended tasks by combining the recommended tasks of their task-bidding friends. We measure the effectiveness of our method on a real data set. The experimental results show that it is feasible and practical in task recommendation in the software crowdsourcing platform. In particular, the F1-Measure of our method for inactive developers who have task-bidding friends is increased by 16.7% than other previous approaches averagely. Our main contributions are summarized as follows:

- Our method constructs a developer social network and calculates social influence degrees between developers, which controls different degrees of implicit feedback of developers' friends.
- Our method recommends inactive developers tasks by combining the recommended tasks of their task-bidding friends, solving the cold-start problem.

The rest of paper is organized as follows: Section II reviews the related work. Section III is about problem definition and approach overview. Section IV constructs a developer social network and analyzes social influence degrees between developers. Section V introduces the details about recommendation approaches. In section VI, we conduct experiments in a real world data set. The conclusion and future work are shown in Section VII.

II. Related Work

Our work is mainly related to three lines of researches:

A. Previous Social Influence Analysis in Social Networks

Much effort has been made for social network analysis and a large number of work has been done [7]. As for social influence analysis, some scholars utilized probabilistic model to analyze social influence. For example, Anagnostopoulos et al. [8] proposed methods to qualitatively measure the existence of influence. Singla et al. [9] analyzed users' social influence based on a probabilistic model over users and their attributes and relations. Crandall et al. [10] developed techniques for identifying and modeling the interactions between social influence and selection, using data from online communities where both social interaction and changes in behavior over time can be measured. In recent times, some approaches with efficient distributed learning algorithms have been proposed. Tang el al. [11] proposed Topical Affinity Propagation (TAP) to model the topic-level social influence on social networks. TAP is designed based on Map-Reduce framework.

The approaches above only use social information to analyze social influence between users. In software crowdsourcing platforms, there are developer behaviors of browsing, bidding for and completing tasks. Our method could incorporate social information and developer behaviors and thus makes a more precise calculation on social influence degrees between every two developers than the approaches above.

B. General Approaches for Recommendation

The traditional approaches on recommendation are collaborative filtering approaches, such as singular value decomposition (SVD) [12][13][14], where the user gets items based on other items with similar patterns of selected user. However, SVD only uses the latent low-dimension factor models and ignores the contents of items and users, which causes SVD suffers from cold-start problem. Some scholars proposed content-based approaches for building a recommender system [15][16][17]. Takacs et al. [18] constructed content similarity matrix and used neighbor based approach on recommendation, which thinks users' ratings are affected by their nearest neighbors. Williams et al. [19] combined a cluster approach into recommender systems, which considers the influence of users in the same cluster. But it is difficult for content-based approaches to construct proper profiles for recommendation.

In recent years, social information has been widely used in recommender systems. Specially, Ma et al. [20] proposed a social regularization approach by considering the constraint of trust mechanism in social information. The idea is to share a common user-feature matrix factorized by ratings and trust. Yang et al. [5] proposed a hybrid method (TrustMF) that combines both a truster model and a trustee model from the perspectives of trusters and trustees, that is, both the users who trust a user and those who are trusted by a user will influence the user's ratings on unknown items. But these approaches above just utilize trust relations in social information, which do not analyze social information about users deeply.

C. Recommendation Approaches in Software Crowdsourcing Platforms

Recommendation in software crowdsourcing platforms is between developers and tasks, like users and items in general recommender systems. In particular, Ambati et al. [4] proposed a content-based recommendation approach for software crowdsourcing, which combines developer performance and interests. And for content-based approaches, it is different to construct proper developer profiles. In addition, there are some research papers to utilize approaches in natural language processing for recommendation in software crowdsourcing platforms. Zhu et al. [21] proposed a conditional random field (CRF) approach to learn task characteristics from their descriptions and developer characteristic distributions from their historical tasks. Xie et al. [22] proposed DRETOM. It models developers interest in and expertise on task resolving activities based on topic models that are built from their historical task resolving records. Wu et al. [23] proposed DREX to developer recommendation based on K-Nearest-Neighbor search with task similarity and expertise ranking with various metrics, including simple frequency and social network metrics. The approaches above in natural language processing only recommend to developers the tasks which have rich content information in platforms. Additionally, Lin et al. [24] proposed a recommendation approach based on collaborative filtering for crowdsourcing, but the approach ignores social information. Our recommendation approach optimizes CF approaches by considering influence degrees between different developers, calculated by developer behaviors and social information.

III. Problem Definition and Method Overview

A. Problem Definition

Supposed that we denote all developers in a software crowdsourcing platform as U and all tasks as I, we use $F \subseteq U \times U$ to denote the friend relations between developers. For a developer $u \in U$, F_u is the set of developers who are friends with the developer. And from the records in platform, there are three kinds of relations between developers and tasks: (1) $B \subseteq U \times I$ to denote the relations that developers have browsed tasks; (2) $A \subseteq U \times I$ to denote the relations that developers have made bids for tasks; (3) $C \subseteq U \times I$ to denote the relations that developers have completed tasks. Consequently, for a developer $u \in U$, B_u, A_u, C_u are the set of tasks that the developer has browsed, bid for and completed, respectively. Since there is no direct ratings that developers make to tasks. We define the ratings using the records about bidding and completing relations. Bidding relations indicate that developers are interested in tasks, and completing relations indicate that developers are interested in and also competent in tasks. So we define the following rules:

- *If a developer u bids for a task i, then $r_{u,i}$ is 3;*
- *If a developer u completes a task i, meaning the developer u has bid for the task i previously, then $r_{u,i}$ is 5.*

Based on the rules, we could construct a developer-task rating matrix. Suppose that a crowdsourcing platform include m developers and n tasks. Let $R = [r_{u,i}]_{m\times n}$ denotes the developer-task rating matrix, where each entry $r_{u,i}$ is the rating given by a developer u to a task i. And we need to find two low-rank matrices: the developer-feature matrix $P \in \Re^{f\times m}$ and the task-feature matrix $Q \in \Re^{f\times n}$ that can adequately recover the rating matrix R, i.e. For a developer $u \in U$ and a task $i \in I$, let I_u denote the set of tasks rated by the developer u, and let p_u, q_i be f-dimensional latent feature vectors of the developer u and the task i, respectively. In this regard, the main task of recommendation is to make the predicted rating $\widehat{r}_{u,i}$ as close as possible to the ground truth $r_{u,i}$. Formally, we can learn the developer-feature and the task-feature matrices by minimizing the loss function, given by the formula (1):

$$L = \sum_{(u,i)\in I_u} (\widehat{r}_{u,i} - r_{u,i})^2 + \lambda(||q_i||_F^2 + ||p_u||_F^2) \qquad (1)$$

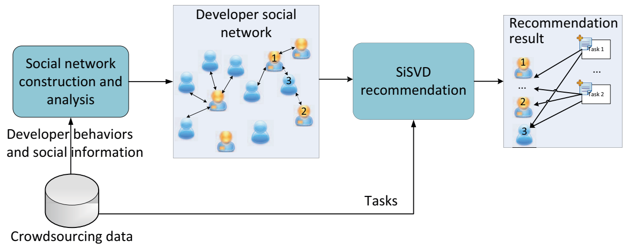


Fig. 2. Overview of Our Method.

where $||.||_F$ denotes the Frobenius norm. The constant λ controls the extent of regularization to avoid over-fitting when learning parameters, which is usually determined by cross validation.

B. Method Overview

Now, we provide an overview to explain the whole process of our proposed task recommendation method for software crowdsourcing. As shown in Fig. 2, we classify developers in the crowdsourcing platform into two categories, active developers who have made bids for tasks and inactive developers who have not. And then a developer social network is constructed, where the vertices denote all developers, and the edges denote social influence between every two developers, which are calculated by developer behaviors and social information in platform. Incorporating developer-task ratings with implicit feedback of adjacent developers in the developer social network, we propose a SiSVD approach to recommend tasks to suitable developers. For inactive developers, our approach recommends tasks by combining the recommended tasks of adjacent active developers. In this paper, the ratings are calculated by the bidding and completing relations, and the implicit feedback is calculated by the browsing relations.

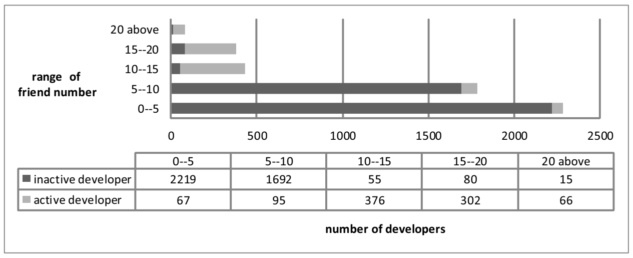
IV. Construction and Analysis of Developer Social Network

In this section, we first analyze software crowdsourcing history data from the platform and construct a developer social network. Then a formula is defined to calculate social influence degrees between every two developers.

A. Construction of Developer Social Network

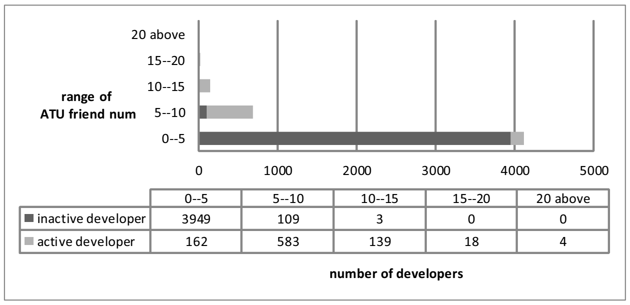
We use the data set of JointForce, a popular software crowdsoucing platform in China. The data set contains social information and developer behaviors of browsing, bidding for and completing tasks. We divide developers into two categories. ATU is the set of active developers who have made bids for tasks in the platform, and $NATU$ is the set of inactive developers who have made no bids for any tasks. We could get $U = ATU \cup NATU$. According to section III, F_u is the set of developers who are friends with the developer u. We define $AF_u = F_u \cap ATU$. And now we raise the following questions: (1) For a developer u $\in ATU$ or $NATU$, what the range is the size of F_u respectively? (2) For a developer u $\in NATU$, what the range is the size of AF_u?

By the statistical results of the data set, we could answer the questions above: (1) From Fig. 3, we could get that



	0--5	5--10	10--15	15--20	20 above
▮ inactive developer	2219	1692	55	80	15
▮ active developer	67	95	376	302	66

Fig. 3. Distribution about Ranges of Friend Number



	0--5	5--10	10--15	15--20	20 above
▮ inactive developer	3949	109	3	0	0
▮ active developer	162	583	139	18	4

Fig. 4. Distribution about Ranges of ATU Friend Number

developers in ATU have more friends, almost more than 10. But developers in $NATU$ have less friends, almost less than 10; (2) If we only focus on the number of friends who are in ATU, Fig. 4 shows the fact that most developers in $NATU$ less than 5 friends who are in ATU; (3) From Table I, for about 91.6% of developers in $NATU$, they have at least one friend who is in AF_u. By summarizing the results, we could get that for most developers, no matter active developers or inactive developers, they have rich friend relations available in the crowdsourcing platform.

TABLE I. The Size of AF Set for Inactive Developers

AF set size	0	1	2	3	3 above
Percent	8.4%	43.4%	32.1%	11.1%	5.0%

For inactive developers, they have made no bids for any tasks and thus are cold-start developers. Based on the summary above, we have find there are closed relations between most inactive developers and active developers, which helps to recommend suitable tasks to inactive developers. In addition, for active developers, they could be influenced by the developers with closed relations when they give ratings to tasks.

In this way, we can construct a developer social network in software crowdsourcing platform.

B. Social Influence Analysis

With the developer social network, social influence degrees between developers could be analyzed from two aspects. On the one hand, there exists social influence only if developers are friends with each other. On the other hand, social influence degrees are decided by the similarity of browsing and bidding for tasks. By combining two aspects above, we define the formula (2) that calculates the social influence degree of the developer u and v.

$$\begin{aligned}D_{u,v} =&F_{u,v}(\frac{\sum_{p\in B_u\cap B_v}\frac{1}{Heat(p)}}{\sum_{p\in B_u\cup B_v}\frac{1}{Heat(p)}}\\&+\alpha\frac{\sum_{p\in A_u\cap A_v}\frac{1}{Heat(p)}}{\sum_{p\in A_u\cup A_v}\frac{1}{Heat(p)}})\end{aligned} \tag{2}$$

$$Heat(p) = 0.1 * |B_p| + 0.3 * |A_p| \tag{3}$$

In the formula (2), $F_{u,v}$ tells whether u and v are friends. If they are friends, $F_{u,v}$ would be 1. Otherwise, $F_{u,v}$ would be 0. α controls the relative proportion between the similarity of browsing tasks and the similarity of bidding for tasks. And $Heat(p)$ shows the popularity of the task p in the platform, which is calculated by the formula (3). $Heat(p)$ is similar to the IDF in TF-IDF algorithm [25] to reduce the effects of excessively popular tasks. B_p is the set of developers who have browsed the task p and A_p is the set of developers who have bid for the task p.

V. SiSVD Recommendation

Based on the developer social network, we propose a SiSVD approach for recommending suitable tasks to active developers and inactive developers with higher performance.

A. Recommendation for Active Developers

For active developers who have made bids for tasks, we recommend tasks to them using collaborative filtering (CF) approaches. CF approaches produce developer specific recommendation of tasks based on patterns of ratings without exogenous information about either developers or tasks. In order to recommend proper tasks to the developer, CF approaches need to relate two different entities: tasks and developers. We use singular value decomposition (SVD) to transform both tasks and developers to the same latent factor space [12].

1) SVD Approach: According to Ekstrand el al. [12], SVD maps tasks and developers to a joint latent factor space of dimensionality f , such that developer-task interactions are modeled as inner products in that space. The latent space tries to explain ratings by characterizing both tasks and developers on factors automatically inferred from developer feedback. Accordingly, each task i is associated with a vector $q_i \in \Re^f$, and each developer u is associated with a vector $p_u \in \Re^f$. For a given task i, the element of q_i shows the extent to which the task measures those factors. And for a developer u, the element of p_u shows the extent of interest or ability the developer has in tasks that are high on the corresponding factors. Now we could capture a interaction between developer u and task i, which could be calculated by $q_i^T p_u$, the inner products of two vectors above. The final rating is calculated by the formula (4)

$$\widehat{r}_{u,i} = \mu + b_i + b_u + q_i^T p_u \tag{4}$$

The parameters b_u and b_i indicate the observed bias of developer u and task i, respectively, from the average. And μ is the average rating over all tasks.

2) SVD++ Approach: Some scholars improve the prediction accuracy by adding the implicit feedback, which indicates developer preferences [26]. For example, we could regard developers' browsed behaviors as the implicit feedback and developers' bidding behaviors as the explicit preference. Universally, there are richer browsing behaviors than bidding behaviors for developers. Now we focus on the SVD++ approach, which improves the SVD approach by adding the implicit feedback of developers. To this end, a second set of task factors is added, relating each task i to a factor vector $y_i \in \Re^f$, to characterize developers based on the set of tasks that they have implicitly rated. The predicted rating is calculated by the formula (5), the set N_u means the tasks implicitly rated by developer u.

$$\widehat{r}_{u,i} = \mu + b_i + b_u + q_i^T(p_u + |N_u|^{-\frac{1}{2}} \sum_{j \in N_u} y_j) \tag{5}$$

3) SiSVD Approach: For an active developer u, we can enhance SVD++ by incorporating the implicit feedback of developers who are friends with the developer u. Specifically, the implicit feedback of these developers can be considered in the same form as implicit feedback of the developer u, given by formula (6):

$$\begin{aligned} \widehat{r}_{u,i} =& \mu + b_i + b_u + q_i^T(p_u + |N_u|^{-\frac{1}{2}} \sum_{j \in N_u} y_j \\ &+ \sum_{v \in F_u} |N_v|^{-\frac{1}{2}} D_{u,v} \sum_{j \in N_v} x_j) \end{aligned} \tag{6}$$

where F_u is the set of developers who are friends with the developer u. x_j is the latent factor of a friend's implicit feedback. $D_{u,v}$ given by the formula (2), denotes the social influence degree between the developer u and v. For each given rating $r_{u,i}$ and correspond predicted rating $\widehat{r}_{u,i}$, the associated prediction error is computed by $e_{u,i} = r_{u,i} - \widehat{r}_{u,i}$. From the formula (6), for active developers who have no friends, we could also recommend them tasks by SiSVD. But F_u is an empty set and the recommendation by SiSVD is equal to that by SVD++.

In order to learn the parameters ($b_u, b_i.p_u, q_j, y_j$ and x_j), based on the formula (1), we define the regularized squared error of SiSVD, given by formula (7):

$$\begin{aligned} L = &\sum_{(u,i) \in I_u} (e_{u,i})^2 + \lambda(b_i^2 + b_u^2 \\ &+ ||q_i||_F^2 + ||p_u||_F^2 + ||y_i||_F^2 + ||x_i||_F^2) \end{aligned} \tag{7}$$

The constant λ controls the extent of regularization to avoid over-fitting when learning parameters, which is usually determined by cross validation.

We could minimize the loss by performed by the stochastic gradient descent (Needell el al. [27]). The stochastic gradient descent (SGD) optimization loops through all ratings in the training data. And we modify the parameters by moving in the opposite direction of the gradient. γ is the learning rate, we can make the formula (8).

$$\begin{aligned} &b_u \leftarrow b_u + \gamma(e_{u,i} - \lambda b_u) \\ &b_i \leftarrow b_i + \gamma(e_{u,i} - \lambda b_i) \\ &p_u \leftarrow p_u + \gamma(e_{u,i} q_i - \lambda p_u) \\ &q_i \leftarrow q_i \gamma(e_{u,i}(p_u + |N_u|^{-\frac{1}{2}} \sum_{j \in N_u} y_j \\ &\qquad + \sum_{v \in F_u} |N_v|^{-\frac{1}{2}} D_{u,v} \sum_{j \in N_v} y_j) - \lambda q_i) \\ &\forall j \in N_u : y_j \leftarrow y_j + \gamma(e_{u,i} q_i |N_u|^{-\frac{1}{2}} - \lambda y_j) \\ &\forall v \in F_u, \forall j :\in N_v : x_j \leftarrow x_j + \gamma(e_{u,i} q_i D_{u,v} |N_v|^{-\frac{1}{2}} - \lambda x_j) \end{aligned} \tag{8}$$

Algorithm 1 SiSVD Train.

```
Input:
    the rating matrix $R$;
    the implicit feedback matrix $N$;
    the set of active developers $ATU$;
    the social matrix $F$, the set of tasks $I$;
    the social influence degree matrix $D$;
    max iteration times $iter$, learning rates $\gamma$;
    regularization parameter $\lambda$, latent space dimension f;
Output:
    Matrix $P, Q, Y, X$
1: init $\mu$
2: for $u$ in $ATU$ do
3:     init $b_u$, $p_u$
4: end for
5: for $i$ in $I$ do
6:     init $b_i$, $q_i$, $y_i$, $x_i$
7: end for
8: for $k = 1$ to $iter$ do
9:     for $r_{u,i}$ to $R$ do
10:        $\widehat{r}_{u,i} = \mu + b_i + b_u + q_i^T(p_u + |N_u|^{-\frac{1}{2}} \sum_{j \in N_u} y_j + \sum_{v \in F_u} |N_v|^{-\frac{1}{2}} D_{u,v} \sum_{j \in N_v} x_j)$
11:        $e_{u,i} = r_{u,i} - \widehat{r}_{u,i}$
12:        $b_u \leftarrow b_u + \gamma(e_{u,i} - \lambda b_u)$
13:        $b_i \leftarrow b_i + \gamma(e_{u,i} - \lambda b_i)$
14:        $p_u \leftarrow p_u + \gamma(e_{u,i} q_i - \lambda p_u)$
15:        $q_i \leftarrow q_i \gamma(e_{u,i}(p_u + |N_u|^{-\frac{1}{2}} \sum_{j \in N_u} y_j + \sum_{v \in F_u} |N_v|^{-\frac{1}{2}} D_{u,v} \sum_{j \in N_v} x_j) - \lambda q_i)$
16:        for $j$ in $N_u$ do
17:            $y_j \leftarrow y_j + \gamma(e_{u,i} q_i |N_u|^{-\frac{1}{2}} - \lambda y_j)$
18:        end for
19:        for $v$ in $F_u$ do
20:            for $j$ in $N_v$ do
21:                $x_j \leftarrow x_j + \gamma(e_{u,i} q_i D_{u,v} |N_v|^{-\frac{1}{2}} - \lambda x_j)$
22:            end for
23:        end for
24:    end for
25: end for
26: return $P, Q, Y, X$;
```

Algorithm 1 illustrates the trained procedure of SiSVD approach. b_u and b_i indicate the observed deviations of developer u and task i, matrix P is the developer latent dimension feature matrix, matrix Q is the task latent dimension feature matrix, matrix Y is latent dimension feature matrix of the developers' implicit feedback, and Matrix X is latent dimension feature matrix of their friends' implicit feedback. q_i, y_i, x_i is the i row in Q, Y, X, and p_u is the u row in P. μ is the mean rating over all the platform. Lines 1 to 7 initialize the value of b_u, b_i, p_u, p_i, q_i, y_i, x_i and μ. Line 8 to 25 run the SGD algorithm to learn the value of variables until the max iteration times. γ is the learning rates. At last, we would get the matrix P, Q, Y, X (line 26).

After getting the b_u, b_i, P, Q, Y and X, we could predict the corresponding rating by the formula (6). We will recommend tasks with high ratings to active developers.

B. Recommendation for Inactive Developers

For inactive developers, they have no records about task bidding and thus have no ratings to tasks. We could not recommend tasks to them directly by CF approaches. But we could recommend inactive developers the same tasks as those to their friends who have made bids for tasks. Because some inactive developers have multiple task-bidding friends, we should combine the ratings of all task-bidding friends. Thus, for a developer u, $\widehat{r}_{u,i}$ is calculated by formula (9).

$$\widehat{r}_{u,i} = \frac{\sum_{v \in F_u \cap ATU} D_{u,v} \widehat{r}_{v,i}}{\sum_{v \in F_u \cap ATU} D_{u,v}} \tag{9}$$

$D_{u,v}$ is considered as the factors of social influence degrees between developers, given by the formula (2). Based on the predicted ratings, inactive developers are recommended the tasks with high ratings.

VI. Experiments

In this section, we first present our experimental settings and then analyze experiment results. Two experiments are conducted: (1) We verify our method in calculating social influence degrees between developers and find the best parameter α; (2) We compare SiSVD with other approaches to recommend suitable tasks to active developers and inactive developers.

A. Experimental Settings

We use the data set of JointForce, including 1849 tasks and 4967 developers, where 4631 developers have at least one friend. For developer behaviors, there are 65535 records for browsing behaviors, 8331 records for bidding behaviors, and 443 records for completing behaviors in the data set. Since the developer-task ratings are calculated by bidding and completing relations, there are 8331 developer-task ratings in platform. For these ratings, we use 4-fold cross-validation for training and testing. Specifically, we randomly split these ratings into four folds. In each iteration, three folds are used as the training set, and the remaining fold as the testing set. Four iterations will be conducted to ensure that all rating folds are tested and we calculate average performance. For CF approaches, the 65535 records about browsing behaviors are used as implicit feedback.

Evaluation Metrics. We use precision, recall, and F1-Measure to calculate the ranking metrics, which are defined as follows:

Precision : It is the percentage of correctly recommended tasks in all recommended tasks.

Recall : It is the percentage of correctly recommended tasks in the recommended task set of ground truth.

$$Precision = \frac{A \cap T}{A} \qquad Recall = \frac{A \cap T}{T} \tag{10}$$

where A is the task set that our approach recommends to the developer, and T is the recommended task set of ground truth. Recommended task set is composed by top 10 rating tasks.

F1-Measure (F1): F1-Measure combines the overall result of precision and recall, and is the harmonic mean of precision and recall.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{11}$$

For the rating metrics, We adopt another two well-known metrics to evaluate predictive accuracy, namely mean absolute error (MAE) and root mean square error (RMSE).

$$MAE = \frac{\sum_{u,i} |\widehat{r}_{u,i} - r_{u,i}|}{N} \; RMSE = \sqrt{\frac{\sum_{u,i} \widehat{r}_{u,i} - r_{u,i}}{N}} \tag{12}$$

Comparison Method. Three kinds of approaches are compared with our approach, SiSVD:

(1) Neighbor-based approaches (Desrosiers and Karypis [28]): developer neighbor-based (D-NB) and task neighbor-based (T-NB) approaches;

(2) Cluster approaches: Cluster approaches make all developers (D-Cluster) or tasks (T-Cluster) into several clusters and recommend tasks (Tsai et al. [29]);

(3) CF approaches: SVD++ and TrustSVD (Guo et al. [6]), which take friend information in social networks into account.

B. Experiment about α on Social Influence Degree Calculation

The experiment is conducted on all active developers. According to the formula (2) on social influence degree calculation, α controls the relative proportion between the similarity of browsing tasks and the similarity of bidding for tasks. We set the α different values and calculate social influence degrees between developers to recommend them tasks with SiSVD. Fig. 5 shows the impact of different α values on the performance. If α is 3, the MAE and RMSE reach the minimum value and our experiment gets the best performance.

C. Recommendation Compared with Other Approaches

In this experiment, we use different approaches to recommend tasks to the developers. For inactive developers, they have no records about bidding for and completing tasks. In general, they give no ratings to tasks, but SiSVD calculates the predicted ratings of inactive developers by combining the ratings of their task-bidding friends. In other words, for inactive developers, it is not the predicted ratings to make sense, but the set of recommended tasks. So we only select

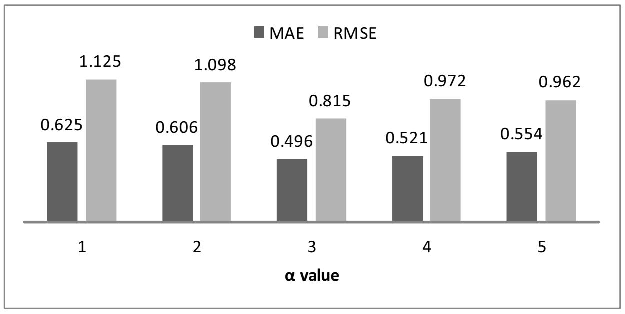


Fig. 5. MAE and RMSE of Different α Value on Social Influence Degree Calculation

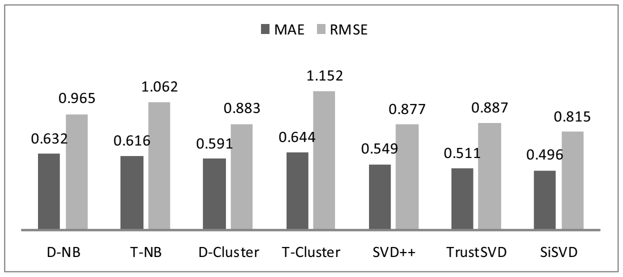


Fig. 6. Rating Performance of Different Recommend Approaches for Active Developers

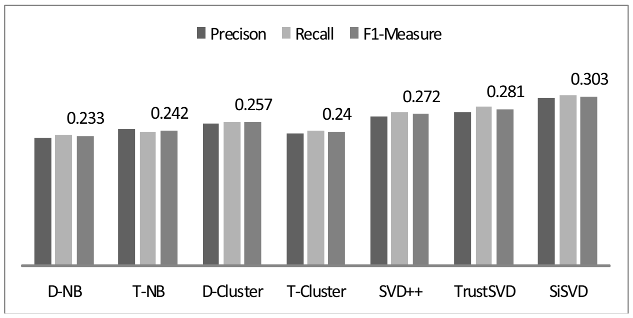


Fig. 7. Ranking Performance of Different Recommend Approaches for Active Developers

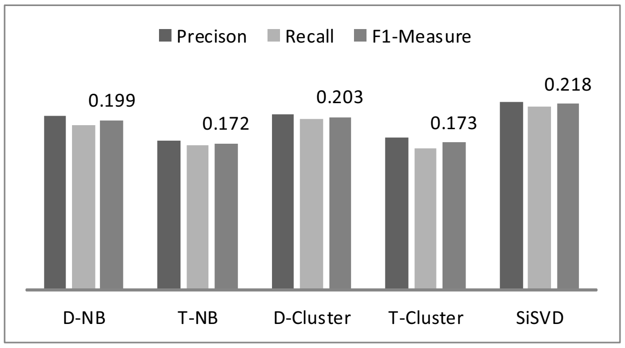


Fig. 8. Ranking Performance of Different Recommend Approaches for Inactive Developers

precision, recall and F1-measure as the evaluation metrics. We compare task sets recommended by different approaches to the recommended task set of ground truth. Since inactive developers have no ratings to tasks, so the recommended task set of ground truth is composed by the tasks that have been browsed by inactive developers. In addition, when we utilize SVD++ and TrustSVD to recommend tasks to inactive developers, the rating matrix is a zero matrix, meaning both the P matrix and Q matrix are zero matrixes. So we do not use SVD++ and TrustSVD as compared approaches to recommend tasks to inactive developers.

Parameter Settings. The optimal experimental settings for each approach are determined either by our experiments or suggested by previous work [6][28][29]. For α on social influence degree calculation, we set α = 3. Specifically, the common settings are the size of recommended tasks, which is 10 and the number of latent features f = 10 for CF approaches. The other settings are: (1) D-NB: the size of neighbors is 30; (2) T-NB: the size of neighbors is 20; (3) D-Cluster: we use the Density-Based Spatial Clustering of Applications with Noise (DBSCAN Birant et al. [30]) for clustering developers. Eps = 0.1 and $MINPTS$ = 200; (4) T-Cluster: we use the DBSCAN for clustering tasks. Eps = 0.2 and $MINPTS$ = 80.

For task recommendation to active developers, the rating performance of experiment is presented in Fig. 6, and the ranking performance is shown in Fig. 7. For all the comparison approaches, SiSVD outperforms the other comparison approaches. Since active developers have rich behaviors in software crowdsourcing platforms, CF approaches (SVD++, TrustSVD and SiSVD) mainly show better performance than content-based approaches (D-NB, T-ND, D-Cluster and T-Cluster). And TrsutSVD only uses social information about whether two developers are friends or not. Compared to SVD++, TrustSVD improves less performance than our approach does. In other words, compared to directly utilizing friend relations, we could get more precision recommendation results by considering the factor of social influence degrees between developers.

For task recommendation to inactive developers who have task-bidding friends, the experimental results are shown in Fig. 8. Previous CF approaches, SVD++ and TrustSVD, could not be used to recommend tasks to inactive developers, and content-based approaches are the compared approaches. From Fig. 8, SiSVD outperforms than compared content-based approaches, and F1-Measure of our method is increased by 16.7% than other previous approaches averagely. It means our method could recommend suitable tasks to cold-start developers who have task-bidding friends. The number of all inactive developers is 4061, where 3720 inactive developers have task-bidding friends and can get recommended tasks by our method. So our method could recommend suitable tasks to 91.6% of cold-start developers in JointForce.

VII. Conclusion and Future work

In this paper, we have proposed a novel method for task recommendation in software crowdsourcing platforms, using the developer social network. It firstly constructs a developer social network, and then combines social information and developer behaviors for social influence degree calculation between developers. Our method recommends suitable tasks to active developers, using the ratings of active developers and implicit feedback of their friends. And social influence degrees are utilized to control different degrees of implicit feedback of developers' friends. Additionally, our method recommend inactive developers tasks by combining the recommended result of their task-bidding friends. According to experiment results, our method indeed makes better performance than other approaches, especially recommending tasks to inactive developers with task-bidding friends, F1-Measure of our method is increase by 16.7% than other previous approaches averagely. It means, in some cases, our method solves cold-start problem effectively.

In this paper, our method could pay attention to recommending tasks to developers. As for future work, we will try to recommend suitable developers to tasks. And since recommendation in real crowdsourcing platforms can't suffer from the unsatisfied time consumption, we will also plan to explore the efficiency of our method for real-time recommendation.

ACKNOWLEDGEMENT

This research is supported by 973 Program in China (Grant No. 2015CB352203) and National Natural Science Foundation of China (Grant No. 61472242). Thanks JointForce for providing the experimental data set.

REFERENCES

[1] Eman Aldhahri, Vivek Shandilya, and Sajjan Shiva. Towards an effective crowdsourcing recommendation system: A survey of the state-of-the-art. In *Service-Oriented System Engineering (SOSE), 2015 IEEE Symposium on*, pages 372–377. IEEE, 2015.

[2] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434. ACM, 2008.

[3] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[4] Vamshi Ambati, Stephan Vogel, and Jaime G Carbonell. Towards task recommendation in micro-task markets. In *Human computation*, pages 1–4, 2011.

[5] Bo Yang, Yu Lei, Dayou Liu, and Jiming Liu. Social collaborative filtering by trust. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 2747–2753. AAAI Press, 2013.

[6] Guibing Guo, Jie Zhang, and Neil Yorke-Smith. Trustsvd: Collaborative filtering with both the explicit and implicit influence of user trust and of item ratings. In *AAAI*, pages 123–129, 2015.

[7] Dennis Saleebey. The strengths perspective in social work practice: Extensions and cautions. *Social work*, 41(3):296–305, 1996.

[8] Aris Anagnostopoulos, Ravi Kumar, and Mohammad Mahdian. Influence and correlation in social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 7–15. ACM, 2008.

[9] Parag Singla and Matthew Richardson. Yes, there is a correlation:-from social networks to personal behavior on the web. In *Proceedings of the 17th international conference on World Wide Web*, pages 655–664. ACM, 2008.

[10] David Crandall, Dan Cosley, Daniel Huttenlocher, Jon Kleinberg, and Siddharth Suri. Feedback effects between similarity and social influence in online communities. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 160–168. ACM, 2008.

[11] Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. Social influence analysis in large-scale networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 807–816. ACM, 2009.

[12] Michael D Ekstrand, John T Riedl, and Joseph A Konstan. Collaborative filtering recommender systems. *Foundations and Trends in Human-Computer Interaction*, 4(2):81–173, 2011.

[13] Kai Yu, John Lafferty, Shenghuo Zhu, and Yihong Gong. Large-scale collaborative prediction using a nonparametric random effects model. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1185–1192. ACM, 2009.

[14] Osman Nuri Osmanli and İsmail Hakkı Toroslu. Using tag similarity in svd-based recommendation systems. In *Application of Information and Communication Technologies (AICT), 2011 5th International Conference on*, pages 1–4. IEEE, 2011.

[15] Michael J Pazzani and Daniel Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.

[16] Raymond J Mooney and Loriene Roy. Content-based book recommending using learning for text categorization. In *Proceedings of the fifth ACM conference on Digital libraries*, pages 195–204. ACM, 2000.

[17] Prem Melville, Raymond J Mooney, and Ramadass Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *Aaai/iaai*, pages 187–192, 2002.

[18] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Matrix factorization and neighbor based algorithms for the netflix prize problem. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 267–274. ACM, 2008.

[19] Elizabeth Williams, Jeff Gray, and Brandon Dixon. Mobile context recommendations from social media through geotopical clustering. *University of Alabama, SERG-2015-01*.

[20] Hao Ma, Haixuan Yang, Michael R Lyu, and Irwin King. Sorec: social recommendation using probabilistic matrix factorization. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 931–940. ACM, 2008.

[21] Jiangang Zhu, Beijun Shen, and Fanghuai Hu. A learning to rank framework for developer recommendation in software crowdsourcing. In *2015 Asia-Pacific Software Engineering Conference (APSEC)*, pages 285–292. IEEE, 2015.

[22] Xihao Xie, Wen Zhang, Ye Yang, and Qing Wang. Dretom: Developer recommendation based on topic models for bug resolution. In *Proceedings of the 8th international conference on predictive models in software engineering*, pages 19–28. ACM, 2012.

[23] Wenjin Wu, Wen Zhang, Ye Yang, and Qing Wang. Drex: Developer recommendation with k-nearest-neighbor search and expertise ranking. In *2011 18th Asia-Pacific Software Engineering Conference*, pages 389–396. IEEE, 2011.

[24] Christopher H Lin, Ece Kamar, and Eric Horvitz. Signals in the silence: Models of implicit feedback in a recommendation system for crowdsourcing. In *AAAI*, pages 908–915, 2014.

[25] Juan Ramos. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, 2003.

[26] Yancheng Jia, Changhua Zhang, Qinghua Lu, and Peng Wang. Users' brands preference based on svd++ in recommender systems. In *Advanced Research and Technology in Industry Applications (WARTIA), 2014 IEEE Workshop on*, pages 1175–1178. IEEE, 2014.

[27] Deanna Needell, Rachel Ward, and Nati Srebro. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. In *Advances in Neural Information Processing Systems*, pages 1017–1025, 2014.

[28] Christian Desrosiers and George Karypis. A comprehensive survey of neighborhood-based recommendation methods. In *Recommender systems handbook*, pages 107–144. Springer, 2011.

[29] Chih-Fong Tsai and Chihli Hung. Cluster ensembles in collaborative filtering recommendation. *Applied Soft Computing*, 12(4):1417–1425, 2012.

[30] Derya Birant and Alp Kut. St-dbscan: An algorithm for clustering spatial–temporal data. *Data & Knowledge Engineering*, 60(1):208–221, 2007.