Cold-Start Developer Recommendation in Software Crowdsourcing: A Topic Sampling Approach

Yu Yang, Wenkai Mo, Beijun Shen†, Yuting Chen
School of Electronic Information and Electrical Engineering
Shanghai Jiao Tong University, Shanghai, China
Email: {monkeydeking, jirachikai, bjshen, chenyt}@sjtu.edu.cn

***Abstract*—Recently, software crowdsourcing platforms, which provide paid tasks for developers, become attractive to both employers and developers. Developers expect to find tasks that match their interests and capabilities via crowdsourcing platforms, and thus recommender systems play important roles in these platforms. However, we still face several challenges when building a recommender system for a crowdsourcing platform. A major challenge is how to recommend tasks to *cold-start* developers whose task interaction data is not available. This paper presents a novel, topic sampling approach to tackling with the cold-start developer recommendation problem. First, it employs a general method for modeling developers and tasks, which solves the data heterogeneous issue across different platforms. After that, it casts the cold-start developer recommendation problem into a multi-optimization problem, and takes a topic-sampling based genetic algorithm to recommend tasks. More specifically, our approach is different from traditional solutions in that it leverages task descriptions and popularity-to-be, allowing new tasks to be recommended to cold-start developers. To evaluate the effectiveness of the proposed approach, we have conducted experiments on a large dataset crawled from three real-world software crowdsourcing platforms. Compared with other state-of-the-art recommendation solutions, the experimental results show that the proposed approach improves 75% of precision and recall on average.**

***Keywords*—Cold-Start Problem, Software Crowdsourcing, Topic Sampling, Developer Recommendation.**

I. Introduction

Software crowdsourcing has attracted great attentions from both industry and academia recently. An increasing number of software companies have turned to find online developers in software crowdsourcing platforms, such as OsChina[1] and Zhubajie[2], to accomplish various types of software development tasks, including architecture design, component development, testing and bug fixing. On a crowdsourcing platform, developers hunt for suitable tasks, and employers also search for qualified developers. Inappropriate developer-task matching may decrease the quality of the software deliverables. Thereafter, a recommender system needs to be designed and integrated into a crowdsourcing platform, helping users find the best fitting developers or tasks effectively.

For a software crowdsourcing recommender system, a major challenge is how to recommend tasks to *cold-start* developers (i.e., new developers) whose task interaction data is not available. Even worse, unlike movies or music items in traditional recommender systems, tasks in software crowdsourcing platforms may be insufficient in their data accumulations, since they usually have short lifetime period. Available developer behavior history information becomes rare, making the cold-start developer problem severe.

In this paper, we focus on solving the cold-start developer problem in software crowdsourcing recommender systems, i.e. recommending tasks to cold-start developers with little online behavior information. Traditional solutions conduct some questionnaire surveys [1] on cold-start developers and then build models based on results or simply recommend popular tasks to them [2]. However, it requires a lot of expert efforts to prepare good questionnaires, while developers may still forget to fill up them. Furthermore, these methods cannot recommend new tasks whose popularity are relative low to developers, even though these tasks are potentially popular in the future.

To tackle these problems, we propose a topic sampling based approach to recommend tasks to cold-start developers. Our approach builds the models of developers and tasks in a general way, casts the cold-start developer recommendation problem into a multi-optimization problem, and takes a topic-sampling based genetic algorithm to recommend tasks. More specifically, it leverages task descriptions and popularity-to-be, and can recommend even new tasks to cold-start developers, which makes it significantly different from previous solutions. We conducted several experiments to evaluate the proposed approach. Using three real datasets from software crowdsourcing platforms, we can conclude that our approach has a significant enhancement, 75% on both precision and recall in average, at the recommendation.

Our main contributions are summarized as follows:

1) *Task Popularity-to-be Estimation*. Instead of using existing task popularity, we propose an estimation method to get task popularity-to-be as one of key properties of task model. It adopts regression, a machine learning technique, to estimate whether a cold-start task may become popular in the future.
2) *General Resources Modeling*. We propose a general modeling method for modeling developers and tasks. The method abstracts the key properties of resources from heterogeneous data, modeling developers from

† Corresponding Author
[1]http://www.oschina.net/
[2]http://www.zbj.com/
DOI reference number: 10.18293/SEKE2017-104

their profiles and behaviors, while modeling tasks from their descriptions by natural language processing technologies. Then it transfers the features from warmed developers (also called non-cold start developers) to cold-start developers, through classification and clustering.

3) *Cold-start Developer Recommendation Algorithm*. With task model and developer model, a semi-supervised algorithm is designed to recommend tasks to cold-start developers, called *Topic Sampling-based Recommendation Algorithm*, which designs the recommendation problem as a new multi-optimization problem and employs the genetic algorithm to solve it.

II. Related Work

Recommendation System. The traditional approaches on recommendation are collaborative filtering (CF) approaches [3], such as singular value decomposition (SVD) [4], where the user gets items based on other items with similar patterns. However, an inherent prerequisite of CF is to have historical user-item interactions. Thus, CF suffers from the cold-start problem.

Some researchers proposed content-based approaches for building a recommender system. For example, Takacs et al. constructed content similarity matrix and used neighbor based approach to recommendation, where users ratings are affected by their nearest neighbors [5]. But it is difficult for content-based approaches to construct proper profiles for cold-start developers.

Cold-Start Problem. In this paper, we focus on the user cold-start problem in recommender systems. To model the preferences of cold-start users, previous work usually obtain related information by short interviews [6] [7]. During interviews, the users are asked to rate several items from carefully constructed seed sets, which are constructed based on popularity, contention, and coverage [8]. However, usually only a few people will fill out the questionnaires, and badly written questions limit the usefulness [9].

One of the classical approaches to addressing cold-start problem is popularity-based approach. Miao et al. jointly predicted the rating and popularity for cold-start items by sentinel user selection [10]. Arapakis et al. characterised the online popularity of news articles by two different metrics [11]. But popularity-based approaches are not good at personalizing recommendation.

Another is bandits-based approach. Li et al. modeled personalized recommendation of news articles as a contextual bandit problem [12], and solved it by using user-click feedback to maximize total user clicks. In [3], CF was combined with exploration-exploitation strategies for content recommendation. Instead of cold-start users, it is better at solving cold-start item problem.

Developer Recommendation. There are also some studies of recommendation technologies for software crowdsourcing platform. Zhu extracted skill and location features from the textual descriptions, and adopted *learning to rank* technology to perform recommendation [13]. Lee et al. proposed a dynamic planning algorithm to recommend tasks [14]. Yuen et al. pointed out that the past task preference and user performance could be utilized for task preference model [15]. However all of these work did not address the cold start problem.

III. Problem Definition and Approach Overview

We give a formal definition of cold-start developer recommendation problem, and then present an overview of our approach.

A. Cold-Start Developer Recommendation Problem

There are two basic resources on a software crowdsourcing platform: tasks denoted as T, and developers as U. Once an employer has requested a task, the developers can bid for, win, and then deliver the task. So we express the relationship between one developer u and one task t as the set $R = \{bidding, winning, delivering\}$. Thus, data on the software crowdsourcing platform can be expressed as a relationship triples: $(u, t, r) \in U \times T \times R$.

Definition 1: ***Developer Activity Percentile.*** We design a new metric, developer u's current activity percentile $Act\,(u)$, to measure how often the developer interact with tasks, which is defined as the number of tasks u has bid for.

$$Act\,(u) = \sum_{i \in T} 1\,\{if\ u\ bids\ i\} \tag{1}$$

Then, we calculate the average current activity percentile μ_{act} and the variance of the current popularity σ^2_{act} on the platform. In this paper, we suppose that the activity percentile is consistent with the Gaussian distribution $Act \sim N\left(\mu_{act}, \sigma^2_{act}\right)$.

Definition 2: ***Cold-Start Developer.*** When the activity percentile of developer u is less than twice the variance of the Gaussian function formed by all developers' activity percentile on the platform, developer u is regarded as a cold-start developer, that is, $Act\,(u) < Gaussian\left(\mu_{act} - 2 * \sigma^2_{act}; \mu_{act}, \sigma^2_{act}\right)$.

Definition 3: ***Cold-Start Developer Recommendation Problem.*** For the all tasks, it is given that $\forall t \in T, t = \{desc, f\}$, where $desc$ represents description of task t and f a boolean value indicating whether task is terminated. The problem is that recommender system must learn an appropriate model to recommend a suitable task set $T^{'}$ for a cold-start developer, where $\forall t^{'} \in T^{'} \left\{t^{'}.f = 0\right\}$. Besides, the task set recommended must meet $\forall t^{'} \in T^{'} \left\{i\ does\ not\ bid\ t^{'}\right\}$, that is, without being bid by i.

B. Approach Overview

To address the above cold-start problem, a novel approach is proposed, as shown in Fig. 1. It consists of two phases, the resource modeling phase which models tasks and developers from heterogeneous data in the platform, and the recommendation phase which recommends appropriate tasks for cold-start developers with topic sampling.

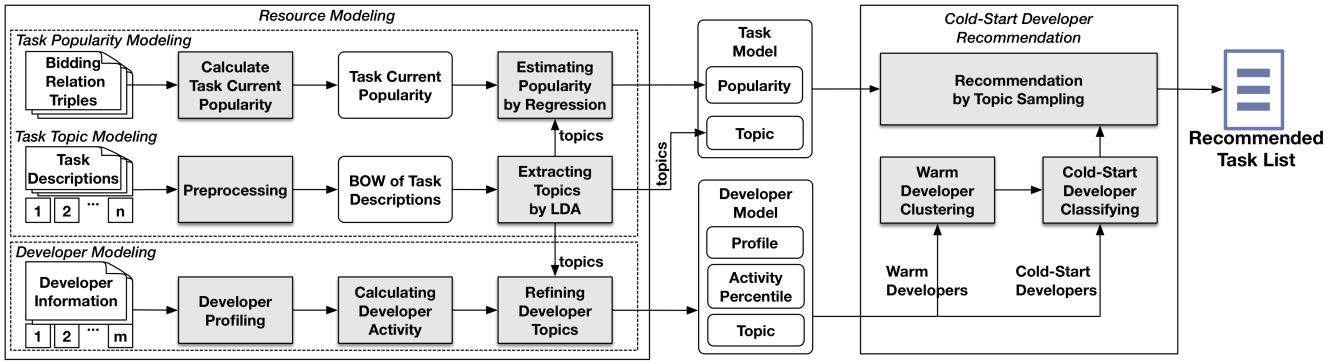

Fig. 1: Our Approach to Cold-Start Developer Recommendation.

1) Software Crowdsourcing Resource Modeling: Due to heterogeneous data from different software crowdsourcing platforms, we build the models of developers and tasks in a general way. For tasks, our approach extracts task topics by the topic model, and calculates task popularity by the regression model. For developers, it builds profile vector, topic vector and activity percentile through NLP technologies.

2) Cold-Start Developer Recommendation: Recommendation phase has three steps: (1) *Warm Developer Clustering* produces some clusters and their center topics; (2) *Cold-Start Developer Classification* assigns each cold-start developer to one cluster of warm developers; (3) *Recommendation Based on Topic Sampling* proposes a topic-based popularity sampling algorithm which applies genetic programming to perform recommendation.

IV. Software Crowdsourcing Resource Modeling

In this section, a general resource modeling method will be introduced, which can extract task and developer information from heterogeneous data in different software crowdsourcing platforms.

A. Task Modeling

One of the basic elements in software crowdsourcing is the task. Each task has its own task description including a title and a body. Although there are other properties to describe tasks in some specific platforms like tags, etc., without loss of generality, we only set task descriptions as the input of modeling phase.

1) Task Topic Modeling: Task descriptions are written in natural language by the employer. We use a popular topic modeling algorithm, Latent Dirichlet Allocation (LDA) [16], to obtain the topic distribution to represent tasks descriptions. Before applying LDA, the descriptions need to be preprocessed. We first concatenate the title and the body of a task description to a big text, and remove stop words and words whose part-of-speech tags are not noun, adjective, or verb. Then each text is represented by a bag-of-words (BOW) vector. We set the number of topics to 150, and thus, each description is finally transformed into a 150-dimensional topic vector. Distinctly our method can transform heterogeneous task data from different platforms into an isomorphic vector.

2) Task Popularity Modeling: In traditional popularity-based recommendation systems, an item's current popularity Pop_c is defined as the average rating of all users to item or how many people rate/bid it. However, it cannot reflect the popularity well especially for new tasks. A new task may be popular in the future, but at the beginning, it cannot attract enough developers in short time, and thus its current popularity is low. To better model tasks, we propose a new concept called *popularity-to-be* Pop_e, which is calculated by a regression model and extremely beneficial for new tasks.

Given the training dataset $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}, x \in R^k, y \in R$, where y_i is current popularity of a task which has been bid and delivered, and x_i the corresponding task topic vector. Regression algorithms can fit a model $\hat{f}(x) \rightarrow R$ from these historical data. Then using this model, for a task, its popularity-to-be can be estimated by the task topic vector.

There are many regression algorithms, such as Classification and Regression Tree (CART) [17], linear regression [18], Gaussian mixture model [19], etc. In the experiment, the classical CART algorithm achieves the best performance and thus we apply it to estimate task's popularity-to-be.

It is meaningful and effective to extract information from the task description for estimating popularity because the task popularity-to-be can indirectly reflect the distribution of topic. However, the longer tasks are released, the less credible the estimated popularity will be. In order to avoid the inaccuracy issue, we propose a time-weighted method ultimately to calculate the popularity of task j.

$$Pop(j) = \alpha(t_s, t_c) * Pop_c(j) + \beta(t_s, t_c) * Pop_e(j) \quad (2)$$

Both $\alpha(t_s, t_c)$ and $\beta(t_s, t_c)$ are two time weighting factors, calculated by the task's release time t_c and the current time t_s. For $\alpha(t_s, t_c)$, it should be increased with $t_c - t_s$ since the reliability of current popularity is growing higher as released time goes by. Meanwhile, the reliability of estimated popularity decreases. So $\alpha(t_s, t_c)$ and $\beta(t_s, t_c)$ are defined as:

$$\alpha(t_s, t_c) = ln(t_c - t_s + 1), \quad \beta(t_s, t_c) = \frac{1}{ln(t_c - t_s + 1)} \quad (3)$$

B. Developer Modeling

We build the model of developers from three aspects.

1) Developer Profiling: When registering on different software crowdsourcing platforms, a new developer is required to fill in different individual information that can be converted into a unified vector to denote his profile model. The individual information generally contains three kinds of data type: numeric, string, and enumerations. Numeric data can be added directly to the profile vector, as well as enumeration after relatively simple discrete processing. For a string, it is generally accepted by converting it to a BOW vector so that the string can be appended to the profile vector.

2) Calculating Developer Activity Percentile: The developer activity percentile is calculated by his/her online behaviors, just according to the definition 1.

3) Refining Developer Topic: The relationship between developer and task is denoted by triples, representing whether a developer has bid, won, or delivered a task. In previous work [10] [11], only the bidding relation triples are used. In this paper, we use all relations to build developer topic model to keep more valuable information:

$$vu = \lambda_B \sum_{t \in B_u} v_t + \lambda_W \sum_{t \in W_u} v_t + \lambda_D \sum_{t \in D_u} v_t \tag{4}$$

where λ_B, λ_W and λ_D represent weights of bidding, winning and delivering, while B_u, W_u and D_u means sets of the task that developer u has bid, won and delivered. Delivering of the task reflects one developer's ability with the maximum influence, followed by winning and bidding, and thus we have $\lambda_B < \lambda_W < \lambda_D$. The more the tasks are delivered, won and bid, the stronger developer's ability is.

V. Cold-Start Developer Recommendation

In this section, we introduce a semi-supervised topic sampling algorithm to recommend tasks to cold-start developers on the software crowdsourcing platform. The algorithm consists of three steps: warm developer clustering, cold developer classification and recommendation by topic sampling.

A. Warm Developer Clustering

In the crowdsourcing platform, it is time-consuming to predefine the classes for all developers, which requires massive expert knowledge. In this paper, we apply clustering methods to obtain the classes automatically. Our algorithm only clusters warm developers because they have enough information left in the platform. K-means and DBScan are the effective clustering algorithms. We will compare the performance of these two algorithms in the experiments. The better one will be chosen.

After clustering by developer profiles, the cluster topic for each cluster is calculated by the formula-(5), where d is the number of developers in the cluster and $u_{i.topic}$ denotes the developer i's topic in the cluster. Thus it can be seen the cluster topic is the topic vector center of instances in one cluster.

$$Topic\,(C) = \frac{\sum_{i=1}^{d} u_{i.topic}}{d} \tag{5}$$

B. Cold-Start Developer Classification

For cold-start developers, they only have profile vector, and can be classified to warm developer clusters. To find a proper cluster for a cold-start developer, we first define cluster center for each cluster. Supposing there are d points in the cluster C, the cluster center is defined by formula-(6). Then for a cold-start developer, the distances between his profile vector and all cluster centers are calculated. The nearest cluster is the target cluster for the developer.

$$Center\,(C) = \frac{\sum_{i=1}^{d} u_{i.profile}}{d} \tag{6}$$

C. Recommendation by Topic Sampling

Accordingly, we design a recommendation algorithm based on topic sampling to make a recommendation for cold-start developers who are classified in a proper cluster.

1) Multi-objects Optimization Problem Formulation: From the profile aspect, a cold-start developer u has high similarity to developers in the same cluster. We consider that topic which the developer interests also has high similarity with those developers. However, the cold-start developer does not have the topic vector, so we use the corresponding cluster topic to represent the interested topic of the cold-start developer, denoted by $Topic_u\,(C)$. Based on $Topic_u\,(C)$, our algorithm samples l tasks which have high similarity to $Topic_u\,(C)$ and then recommends these tasks to the cold-start developer.

However, sometimes the assumption that the cluster topic can represent the cold-start developer's topic is not exactly precise since we only use developer profile to cluster and classify cold-start developer. To solve this issue, during the sampling process, we also take the task popularity into consideration.

Therefore, the cold-start developer recommendation is designed as a multi-objects optimization problem (MOP) that considers both task topic and popularity, formulated as

$$max. \frac{\sum_{i=1}^{l} Pop\,(t_i)}{l} \quad s.t. \frac{\sum_{i=1}^{l} t_{i.topic}}{l} - Topic\,(C) \leqslant \varepsilon \tag{7}$$

The constraint, $\frac{\sum_{i=1}^{l} t_{i.topic}}{l} - Topic\,(C) \leqslant \varepsilon$, tries to make the topic center of selected tasks close to the cluster topic $Topic_u\,(C)$. The optimization function $\frac{\sum_{i=1}^{l} Pop(t_i)}{l}$ ensures the overall popularity of selected tasks as high as possible. In our approach, task popularity considers both current popularity and popularity-to-be, so the recommendation algorithm can also recommend the latest tasks to developers, outperforming other popularity-based recommendation algorithms [1] [2].

2) Recommendation Algorithm Design: As an MOP, our recommendation problem is suitable to adopt genetic algorithm (GA) to search for optimal goals. GA is randomized search and optimization techniques based on the concepts of natural activity percentile of genes, individual selection, and the evolutionary process [20]. Applying the GA to the recommendation algorithm, the individual is the recommended task list, the gene is the task in the list and the population is the list of recommended lists. The optimization function, formula-(7), dictates that the next generation population will knock out

the list with a lower popularity and retain higher. The details of our recommendation algorithm are described as follows.

Algorithm: Topic Sampling Based Recommendation Algorithm

Input: all available tasks T
population size s
recommended tasks size l
cluster topic vector $Topic(C)$
selection criteria ε
iterative times t
mute rate α

Output: Rec_{list} [0]

1: Rec_{list} = **new** array(s)
2: **for** i **from** 1 **to** s:
3: Rec_{list}.append($new_instance$(Topic(C), ε, T, l)
4: Sort(Rec_{list})
5: **for** i **from** 0 **to** t:
6: **for** j **from** $\frac{s}{2}$ **to** s:
7: **if** $get_operation(\alpha)$=="mute" :
8: Rec_{list}[j] = mute(Rec_{list}[j], Topic(C), ε
9: **else**:
10: idx = $random_{int}$(0, $\frac{s}{2}$)
11: Rec_{list}[j], Rec_{list}[j+1]=cross(Rec_{list}[j], idx, Topic(C), ε)
12: j++
13: Sort(Rec_{list})

The first three lines of the algorithm initialize the initial population of random processing. Each individual is composed of two parts, gene and health degree, where the gene is the task list, the health degree is calculated using the formula (7). Then the algorithm sorts the health degree of the individual in descending order. The individuals with small health degrees in the current population are subject to mutation and cross-swapping operations (Line 6). Then the $get_{operation}$ function decides whether to perform mutation operations (Line 8) or cross-swaps (Lines 10-12) on the current individual referring to the mutation rate (Lines 7-9). Additionally, the objects of the cross-swapping are randomly selected from the high-quality individuals retained in the previous population (line 10). At the end, the most healthy individuals of the final population are returned.

In order to prevent it taking too much time in the mutate and cross functions, we can set a maximum number of mutations and the number of crossovers. When the number of mutations is greater than any of them, a new individual should be regenerated and returned.

VI. Evaluations

In order to evaluate the proposed cold-start recommendation approach, we have carried on several comparative experiments to answer four research questions.

RQ1: During task popularity estimation, which regression algorithm performs best?

RQ2: For the warm developer classification, which performs better, k-means or DBScan?

RQ3: During the task modeling, which popularity metric will more benefit the final recommendation result, Pop_c or Pop?

RQ4: Compared with other state-of-the-art recommendation approaches, how is the performance of our approach?

A. Experimental Dataset

We crawled data between Dec. 2010 to Apr. 2016 from Zhubajie, Oschina, and Witkey, which are all the largest software crowdsourcing platforms in China. There are several task categories in these platforms, and only software development tasks are considered in the experiments. In total, there are 15,375 developers and 31,255 software development tasks, as shown in Table I. Obviously, the data of Witkey is less than the other two platforms, and the relationship is relatively sparse.

TABLE I: Dataset from Three Software Crowdsourcing Platforms in China

Platfrom	Tasks	Developers	Relationship
Zhubajie	6000	10597	48769
Witkey	2800	5231	15498
Oschina	6575	15427	77925

B. Regression Experiment and Results

To answer the RQ1, we conducted the experiment to estimate task popularity by four widely used regression methods, including linear regression, classification and regression tree (CART), Gaussian mixture and artificial neural network. And three well-known metrics are used to evaluate predictive accuracy: mean absolute error (MAE), root mean square error (RMSE) and R-square.

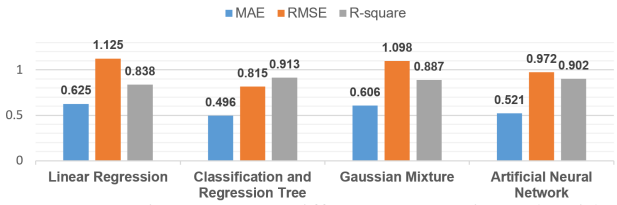

Fig. 2: Comparison among Different Regression Algorithms.

Fig. 2 shows the results, where CART achieves the best performance with the lowest MAE, RMSE and the highest R-square. Thus in our approach, we apply the CART to estimate task popularity.

C. Clustering Experiment and Results

To answer RQ2, we conducted a cold-start developer recommendation experiment using the k-means algorithm ($k = 15$ in the experiments) and DBScan algorithm for warm developer clustering in our approach. We use precision and recall as the evaluation metrics.

We experimentally set λ_B, λ_W and λ_D to 0.3, 0.8 and 1.0 in formula (4), respectively. Considering the requirements from two real software crowdsourcing platforms, we set recommended list size to 15, i.e., $l = 15$. The other settings in our algorithm are: (1) the popularity size s is 100; (2) the iterative times t is 100; (3) the mute rate α is 0.3.

TABLE II: Comparison between Two Clustering Algorithms

Platform	K-means		DBScan	
	Precision	Recall	Precision	Recall
Zhubajie	**0.099**	**0.557**	0.087	0.524
Witkey	0.069	0.531	**0.072**	**0.559**
Oschina	**0.077**	**0.481**	0.037	0.361

TABLE III: Cold-Start Developer Recommending Results

Platform	Ours with Pop_c		Ours with Pop		NRec with Pop_c		NRec with Pop		MRec with Pop_c		MRec with Pop	
	Precision	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**
Zhubajie	0.073	0.531	**0.099**	**0.557**	0.052	0.317	0.068	0.433	0.059	0.346	0.079	0.474
Witkey	0.059	0.498	**0.069**	**0.531**	0.044	0.286	0.047	0.398	0.049	0.313	0.054	0.442
Oschina	0.066	0.449	**0.077**	**0.481**	0.035	0.243	0.046	0.354	0.040	0.269	0.057	0.396

The recommendation results using two clustering algorithms are listed in Table II, which show that k-means wins on stability. In the dataset of Oschina, DBScan results in the worst recommendation recall and precision, because DBScan clusters all warm developers as one group.

D. Recommendation Experiment and Results

To answer the RQ3 and RQ4, we select two popular popularity-based recommendation approaches in [10] and [11], might called NRec and MRec, to perform the task recommendation for the cold start developers, and compare them with our approach. Each approach will adopt the two popularity metrics respectively, i.e. Pop_c and Pop. We also use precision and recall to evaluate the recommendation results.

The experimental results are shown in Table III. It can be observed that our approach outperforms MRec and NRec either with Pop_c or Pop, and improves 75% of precision and recall on average.

Even though the recommendation results of NRec and MRec with Pop are not the best one, they performs better than with Pop_c (i.e. the current popularity). Also MRec with Pop_c and NRec with Pop_c cannot recommend the latest requested task. So it is obvious that Pop is more beneficial to the final recommendation result than Pop_c.

VII. Conclusion and Future Work

In this paper, we propose a novel approach to task recommendation for cold-start developers in crowdsourcing platforms, using a topic-sampling based genetic algorithm. It builds the model of cold-start developers by leveraging the warm developers' preference and tasks' topics. Experimental results show that our approach can minimize the impact of the developer cold-start problem effectively.

For the future work, we plan to explore the task cold-start problem in depth, i.e., recommend developers to cold-start tasks, which is equally important. We will also research on the platform cold-start problem by transfer learning technologies, to build an effective recommender system for a new software crowdsourcing platform.

Acknowledgement

This research is supported by 973 Program in China (Grant No. 2015CB352203) and National Natural Science Foundation of China (Grant No. 61472242 and 61572312).

References

[1] K. Christakopoulou, F. Radlinski, and K. Hofmann, "Towards conversational recommender systems," in *The ACM SIGKDD International Conference*, pp. 815–824, 2016.

[2] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock, "Methods and metrics for cold-start recommendations," in *International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 253–260, 2002.

[3] S. Li, A. Karatzoglou, and C. Gentile, "Collaborative filtering bandits," in *International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 539–548, 2016.

[4] O. N. Osmanli and I. H. Toroslu, "Using tag similarity in svd-based recommendation systems," in *International Conference on Application of Information and Communication Technologies*, pp. 1–4, 2011.

[5] Tak, G. Cs, I. Szy, B. Meth, and D. Tikk, "Matrix factorization and neighbor based algorithms for the netflix prize problem," in *ACM Conference on Recommender Systems (Recsys), Lausanne, Switzerland*, pp. 267–274, 2008.

[6] N. Golbandi, Y. Koren, and R. Lempel, "On bootstrapping recommender systems," in *ACM Conference on Information and Knowledge Management, CIKM 2010, Toronto, Ontario, Canada, October*, pp. 1805–1808, 2010.

[7] A. M. Rashid, G. Karypis, and J. Riedl, "Learning preferences of new users in recommender systems: an information theoretic approach," *Acm Sigkdd Explorations Newsletter*, vol. 10, no. 2, pp. 90–100, 2008.

[8] F. Hu and Y. Yu, "Interview process learning for top-n recommendation," in *ACM Conference on Recommender Systems*, pp. 331–334, 2013.

[9] R. Karimi, A. Nanopoulos, and L. Schmidt-Thieme, "Improved questionnaire trees for active learning in recommender systems," *Proceedings of the LWA 2014 Workshops*, pp. 6–11, 2014.

[10] Z. Miao, J. Yan, K. Chen, X. Yang, H. Zha, and W. Zhang, "Joint prediction of rating and popularity for cold-start item by sentinel user selection," *IEEE Access*, vol. 4, pp. 8500–8513, 2016.

[11] I. Arapakis, B. B. Cambazoglu, and M. Lalmas, "On the feasibility of predicting news popularity at cold start," in *6th International Conference on Social Informatics, Barcelona, Spain*, pp. 290–299, November 11-13, 2014.

[12] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *International Conference on World Wide Web*, pp. 661–670, 2010.

[13] J. Zhu, B. Shen, and F. Hu, "A learning to rank framework for developer recommendation in software crowdsourcing," in *Asia-Pacific Software Engineering Conference*, pp. 285–292, 2015.

[14] S. Lee, S. Park, and S. Park, "A quality enhancement of crowdsourcing based on quality evaluation and user-level task assignment framework," in *International Conference on Big Data and Smart Computing*, pp. 60–65, 2014.

[15] M. C. Yuen, I. King, and K. S. Leung, "Task recommendation in crowdsourcing systems," in *International Workshop on Crowdsourcing and Data Mining*, pp. 22–26, 2012.

[16] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

[17] M. Khandelwal, D. J. Armaghani, R. S. Faradonbeh, M. Yellishetty, M. Z. A. Majid, and M. Monjezi, "Classification and regression tree technique in estimating peak particle velocity caused by blasting," *Engineering with Computers*, vol. 32, no. 120, pp. 1–9, 2016.

[18] B. Pavlyshenko, "Machine learning, linear and bayesian models for logistic regression in failure detection problems," in *IEEE International Conference on Big Data (BigData), Washington DC, USA*, pp. 2046–2050, December 5-8, 2016.

[19] J. Ala-Luhtala and R. Piché, "Gaussian scale mixture models for robust linear multivariate regression with missing data," *Communications in Statistics - Simulation and Computation*, vol. 45, no. 3, pp. 791–813, 2016.

[20] A. H. Beg and M. Z. Islam, "A novel genetic algorithm-based clustering technique and its suitability for knowledge discovery from a brain data set," in *IEEE Congress on Evolutionary Computation*, pp. 948–956, 2016.