2017 24th Asia-Pacific Software Engineering Conference

Transfer learning for cross-platform software crowdsourcing recommendation

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

***Abstract*—Recently, with the development of software crowdsourcing industry, an increasing number of users joined the software crowdsourcing platforms to publish software project tasks or to seek proper work opportunities. One of competitive functions of these platforms is to recommend proficient projects to developers. However, in such recommender system, there exists a serious platform cold-start problem, especially for new software crowdsourcing platforms, as they usually have too little cumulative data to support accurate model training and prediction. This paper focuses on solving the platform cold-start problem in software crowdsourcing recommendation system by transfer learning technologies. We proposed a novel cross-platform recommendation method for new software crowdsourcing platforms, whose idea is trying to transfer data and knowledge from other mature software crowdsourcing platforms (source domains) to solve the insufficient recommendation model training problem in a new platform (target domain). The proposed method maps different kinds of features both in the source domain and the target domain after a certain transformation and combination to a latent space by learning the correspondences between features. Specifically, our method is an instance of content-based recommendation, which uses tags and keywords extracted from project description in crowdsourcing platforms as features, and then set weights for each feature to reflect its importance. Then, Weight-SCL is proposed to merge and distinguish tag features and keyword features before doing feature mapping and data migration to implement knowledge transformation. Finally, we use the data from two famous software crowdsourcing platform as dataset, and a series of experiments are conducted to evaluate the performance of the multi-source recommendation system in comparison with the baseline methods, and get 1.2X performance promotion.**

Keywords*—Tranfer Learning, Software Crowdsourcing, Recommender Systems, Cold-start Problem

I. Introduction

A new upsurge in software crowdsourcing industry is in the making recently. It can not only make full use of the idle resources on the Internet and development resources which can not be gathered together for geographical reasons, but also mobilize collective intelligence to collaborate on software development. This new type of development model, on the one hand, can improve the efficiency of development; on the other hand, can also enhance the competitive advantage of enterprises, teams and programming elite. Undoubtedly, as a kind of e-commerce platforms, a software crowdsourcing platform requires a recommender system to make it easier for users of the platform to discover projects they are interested in and within their ability. However, there exists a difficult problem in the recommendation system: the platform cold-start problem. The cumulative data of a newly launched software crowdsourcing platform is so small that it is difficult to support the recommendation model training.

At present, we have not found a published research on the platform cold-start problem of software crowdsourcing recommendation. Most of existing recommendation algorithms in crowdsourcing platforms focus on the traditional recommendation problem which considers data in the platform is sufficient to train a recommendation model. These methods can be categorized to: Collaborative filtering[1][2], Content-based approaches[3] and Hybrid approaches[4]. Collaborative filtering (CF) approaches rely on past user behaviors, e.g., bidding for or completing projects in software crowdsourcing, and do not require the creation of explicit profiles, while content-based approaches make recommendation by calculating the similarity of developer and project profiles.

However, when new developers or projects sign up, the platforms have no or little knowledge on them and thus cannot make good recommendation by traditional methods, which is also called the user or item cold-start problem. To solve the cold-start problem, Singh Amit Pratap proposed a transfer learning algorithm using collective matrix factorization[5], and Li Bin proposed a rating-matrix generative model[6].

Although these methods can work well for new coming users or project, they rely on abundant data of the platform from other users or projects. For new platforms, most users and projects are new, and with less information, so those methods which can solve user or item cold-start problem cannot work anymore. This is the hardest cold-start problem, called the platform cold-start problem. As our best knowledge, there is no research on the platform cold-start problem caused by insufficient data. We consider transfer learning framework can well solve the problem.

Transfer learning is used to improve a learner of one domain by transferring information from a related domain. In the study of transfer learning, Blitzer J proposed a feature-based transfer learning algorithm named structural corresponding learning (SCL)[7]. This algorithm finds features which appear in both source domain and target domain to be the pivot features, and maps the domain-specific features to pivot features. Sinno Jialin Pan proposed a cross-domain transfer learning algorithm using feature aligning[8]. However, these algorithm can only be applied on the dataset which contains only one kind of features, and in the software crowdsourcing platform, there exist two kinds of features in a project: tag feature and project description. Tag feature is a set of some tag words, and project description is a text. These two kinds of feature have different importance in recommendation system. The existing algorithms are not well suited to this kind of data.

To solve the cold-start problem of the software crowdsourcing recommendation platform, there exist several challenges:

1) How to build a generic data model for projects and developers from different software crowdsourcing platforms and how to calculate the similarity of projects and developers?

2) In order to solve the cold-start problem, how to use source domain data which features are inconsistent with the target domain features?

3) How to apply the transfer learning algorithm on datasets with multiple kinds of features such as Tag feature and Keyword feature?

For the above challenges, we proposed a transfer learning algorithm based on the feature importance named Weight-SCL, an improvement for SCL algorithm. In our algorithm, we use tags and keywords extracted from project description in crowdsourcing platforms as features, and then find out a new representation of cross-domain word features to reduce the differences between domains. By differentiating two types of features and using words unrelated to the domain as a bridge, Weight-SCL can be applied to multi-kind features. In addition, a clustering algorithm is used to align two kinds of features into the same feature cluster, which reduces the mismatch between the two domains' features. We use these clusters to represent all the data instances and train the recommendation classifier using the new representation.

The contributions of this paper include:

1) Through analyzing the data charicteristics on the software crowdsourcing platforms, we build a uniformed model for developers and projects in different platforms. In the modeling phase, considering the special features of data from software crowdsourcing, a method of computing the similarity of projects and develpoers is proposed.

2) The transfer learning algorithm using feature mapping named Weight-SCL was designed for the software crowdsourcing platform data. This algorithm solves the problem of the inconsistency of multi-kind features, and helps to combine and distinguish two kinds of characteristics. The feature mapping is implemented on this basis.

3) The data from the famous software crowdsourcing platforms(Joint Force[1], Zhubajie[2]) were used for the experiments. The results of experiments show that the performance of the multi-source recommendation system in comparison with the basic SCL method has an improvement of 1.2X.

The rest of the paper is organized as follows. In the next section, we firstly review some related works. Then describe the problem we study and give some definitions in Section III. We describe the modeling method and dataset construction in Section IV. Then we present the idea behind our proposed feature-based transfer learning approach. The details of our solution are presented in Section V. We conduct a series of experiments to evaluate the effectiveness of our proposed solution in Section VI. Finally, we conclude our work in Section VII.

[1]https://www.jointforce.com/

[2]http://www.zbj.com/

II. Related Work

Our work is mainly related to two lines of researches:

A. *Recommender System in Software Crowdsourcing Platform*

Wired's editors, Jeff Howe and Mark Robinson, coined the term crowdsourcing in 2005 to describe how companies use the Internet to outsource work to people[9]. Aniket Kittur and Jeffrey V. Nickerson defined the common crowdsourcing process in CSCW 2013[10]. Man-ching Y proposed to use the history of users behavior to recommend a list of jobs for users[11]. On this basis, this team further put forward a way to establish model for users [12], through the user's information such as history of completed project and search to build the model, and recommend projects to users using this model. After that, this team also applied probability matrix decomposition in software crowdsourcing recommendation[13]. Sooyoung Lee proposed to use dynamic programming for separate recommendation of projects and users[14]. Y. Zheng proposed a new algorithm using optimal selection technology, which can recommend suitable users for the software crowdsourcing projects[15]. Subsequently, researchers such as Y. Zheng proposed a software crowdsourcing project recommendation algorithm, which can recommend a quality project to the users in linear time[16]. At the same time, J. Fan et al. built a crowdsourcing platform called iCrowd [17], which evaluates the user's degree by assessing the user's historical data, thus recommend items for the recipient. In SEKE2015, Zhao S. put forward a method for modeling the projects and users respectively, which recommends items by selecting the matching partner[18]. In APSEC2015, Jiangang Zhu proposed a CRF-based model to extract criterias (i.e. skills and locations) from descriptions to match a given task to the right crowdworkers[19]. However, these algorithms have severe cold-start problems, the recommendation model cannot be fully trained when lacking historical data and the recommended effect should be improved. In APSEC2016, a social influence-based method is proposed by Ning Li to recommend suitable tasks for both active and inactive developers for user cold-start problem[20]. But there still existed no solution for platform cold-start problem.

B. *Recommendation Algorithm using Transfer Learning*

In the past years, some researches has already applied transfer learning technologies on traditional recommender system using collaborative filtering, to solve the cold-start problem. Ajit p. Singh proposed a transfer learning algorithm using collective matrix factorization[5], whose main idea is to use the entities involved in multiple domains to share the feature factors of these entities to simultaneously decompose multiple matrices. Li Bin designed a rating-matrix generative model[6], whose main idea is that the relationship between multiple domain scoring matrices can be established through a potential cluster-level scoring matrix, and then this matrix is shared to help predicting the missing field of the target domain scoring matrix. Weike Pan et al. proposed a collaborative filtering recommendation algorithm for Coordinate System Transfer (CST)[21], which also transfers knowledge of users and projects in the source domain to mitigate the sparseness of the target domain data. Bin Li et al. Proposed a Codebook Transfer (CBT)[22]. Based on a limited score in the target domain, the algorithm transfers the useful knowledge gained from the learning tasks in the source domain by establishing a bridge between the two domains scoring matrix. However, there exist several problems on recommendation algorithm using collaborative filtering: the quality of results heavily depends on other users' historical data; on sparse dataset the recommend accuracy is not high. So we need to study how to apply transfer learning on content-based recommendation systems, and how to utilize the domain-specific features in software crowdsourcing.

In the research of migration techinoloy, Blitzer J proposed a transfer learning algorithm named structural corresponding learning(SCL) [7] in 2006. This algorithm finds features which appear in both origin domain and target domain to be the pivot features, and map the domain-specific features to pivot features. And then the pivot feature is added to the original feature, or directly uses the pivot feature for training. The application of SCL algorithm is very extensive, including part of speech (PoS) tagging [23], sentiment classification[24], and so on. Quattoni et al. [25] applied structural learning to image classification in settings where little labeled data is given. Sinno Jialin Pan proposed a spectral feature alignment (SFA) algorithm to align domain-specific words from different domains into unified clusters[8], with the help of domain independent words as a bridge. In this way, the clusters can be used to reduce the gap between domain-specific words of the two domains, and train sentiment classifiers in the target domain accurately. However, these algorithms can only be applied on the dataset which contains only one kind of features, The existing algorithms are not well suited to the data of software crowdsourcing platform. We tried to build a recommendation system on software crowdsourcing platform and further more propose a transfer learning algorithm which is suited to the data of different software crowdsourcing platforms.

III. Problem Definition and Approach Overview

A. Problem Definition

We divide the data on the software crowdsourcing platform into two entities: user U and project T. User could be further devided into two categories $U = \{Dev, Emp\}$, where Dev represents the developer, and Emp represents the employer. We assume there are m developers and n projects in the platform, in another word, $|Dev| = m, |T| = n$.

There exist some relationships between the user and the project on the software crowdsourcing platform: employer will publish some projects, so the Emp and T have a publishing relationship between them; meanwhile, developers can apply for the published projects, and each project will apply one or more developers for resolving requirement issues after contract negotiation. Finally, the developers will deliver their solution and other artifacts to the employers, who will organize the acceptance testing or review. Namely, Dev and T have three kinds of relationships: $apply$, $win-bid$ and $accept$. So the relationships between project and user can be expressed as $R = \{publish, apply, win-bid, accept\}$. So we represent the data on the platform as $(u,t,r) \in U \times T \times R$. Each project has some tags and a description. So $\forall t \in T, t = \{k,d\}$, where k represents project description, and d represents project tag.

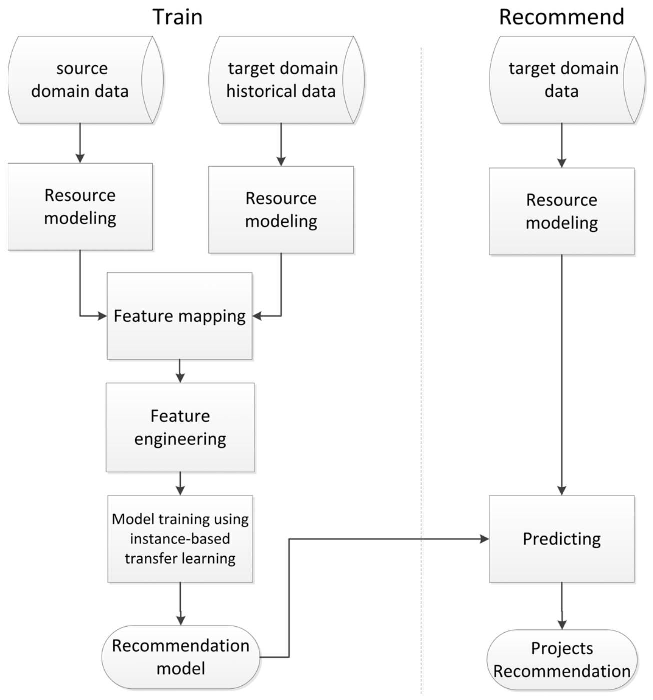


Fig. 1. Overview of Our Approach.

Following is the definition of transfer learning problem for cross-platform software crowdsourcing recommendation: In the source domain, given user $U_s = \{Dev_s, Emp_s\}$, project T_s, relationship between project and user R_s; in the target domain, given user $User_t = \{Dev_t, Emp_t\}$, project T_t, relationship between project and user R_t. The recommendation system needs to learn a model through $U_s \times T_s \times R_s$ and $U_t \times T_t \times R_t$. The model will output a set of projects $T' \subseteq T$ which will be recommended to U' after inputting $U' \in Dev_s$ into the model.

B. Approach Overview

After analyzing the data of software crowdsourcing platform, we propose a novel cross-platform software crowdsourcing recommendation approach using transfer learning to solve the insufficient data issue. The overview of our approach is shown in Figure 1.

The whole approach consists of two phases:

1) Training phase: the target of this phase is to establish recommendation model. First of all, we model for the projects and developers of source domain and target domain. Then we use a transfer learning algorithm proposed by us named Weight-SCL algorithm on the word features of different domain. Finally, we train the recommendation model using linear regression which applied instance-based transfer learning algorithm Tr-Adaboost with simularity matrix, and get the final recommendation model. We will introduce the Weight-SCL in section V, and describe how to get model and then get the simularity matrix in section IV.

2) Predicting phase: the main idea of this phase is to recommend projects for developers using the model we trained in the training phase. First of all, we model for the projects

and developers of target domain, then we apply the parameter trained in Weight-SCL on features. Finally we get the recommendation result using the recommendation model trained in the training phase.

IV. Resource Modeling and Feature Engineering

The resources on the software crowdsourcing platform are divided into projects and developers. In this section, we will examine how these two types of resources are modeled, and then construct similarity matrix.

A. Project modeling

In different software crowdsourcing platform, all of the projects have some project tags and one project description, where the project tag is a word and the project description is a text. So we could directly convert tags belonging to a project into a word vector, meanwhile we need to choose an algorithm to extract some words from project description to establish a word vector.

For combining these two kinds of word vectors in our algorithm, we need to establish a common directory for all words consisted in the tags and the words of descriptions. We use $T_k = \{t_1, t_2, \ldots, t_n\}$ to represent all the words that appear in the description text, and use $T_t = \{t_1, t_2, \ldots, t_m\}$ to represent all tag words. The number of all keywords is m, and the number of all tag words is n. After performing set union on T_k and T_t , we get the directory $D = \{t_1, t_2, \ldots, t_l\}$ which consists of all words. The size of this directory is l. For mapping each word to vector space, we define a function $\phi(w)$, whose input is a word w, and output is the index of corresponding directory location.

Because the project tag feature is a set of meaningful words, we could convert tags of a project into a word vector directly. Since there are only two status for each tag in a project: marked or not marked, so we use BOW(Bag-of-Words) method to convert word set into a binary valued vector, whose component value is either 0 or 1. We initiate the vector as all zero vector at the beginning, then check the tag word set. If a word w appears in the tag word set, then the $\phi(w)'$th value in the vector will be replaced by 1. After project tag vectorization, we get a matrix which expresses the tag information of all projects on the platform, and we use TM_t to represent this matrix.

For representing a text, we could map each text to word vector space. The final target is to use a keywords vector $d_j = (w_{1j}, w_{2j}, \ldots, w_{nl})$ to represent a text, where w_{ij} is the weight of i'th word in text j, and the weight shows the importance of the word. In this way, we convert the problem into how to calculate the component of vector d_j. We use $TermFrequency-InverseDocumentFrequency(TF-IDF)$ method to describe a text[26]. The method of obtaining the TF-IDF value of k'th word in j'th text is showed as equation (1).

$$TF-IDF(t_k, d_j) = TF(t_k, d_j) * \log \frac{N}{n_k} \tag{1}$$

$TF-IDF(t_k, d_j)$ represents the frequency of the k' th word in the text j. n_k is the number of the texts which contain the k' th word. After project description vectorization, we get a matrix which expresses the keyword information extracted from description of all projects on the platform, and we use KM_t to represent this matrix.

After calculating vectors of all project, we get the project content matrix $DM_t = \{TM_t, KM_t\}$. The solution on how to get the matrix DM_t will be described in section V.

When building similarity vector of the negative cases, we need to sort the project according to project visibility, where the project visibility is the possibility that the project could be discovered by developer. So we need to establish the model for project visibility.

First of all, we will yuintroduce the definition of the project popularity which is given in equation(2).

$$Pop_r(j) = \sum_i (apply(i, j)) \tag{2}$$

In the above mentioned definition, the function $apply(i, j)$ returns 1 while there exists an application relationship between the i'th developer and the j'th project, returns 0 otherwise. As shown in this definition, the project popularity is represented by the number of developers who apply for this project, which means that the higher the number of application, the higher the project popularity.

It is easy to observe that there exists a logarithmic relationship between the project retention time and the number of application usually. The number of application usually gets a rapid increase at the beginning of the project publishment, and the growth rate of application slows down later significantly. For balancing the projects with different retention time, we propose a time factor to gain the true visibility. Considering that the project which has a longer retention time should gain a higher visibility than a new project, so we propose another time factor to compensate for a project which has a longer retention time after balancing.

The definition of the project j is shown as equation(3)

$$Pop(j) = \alpha(t_s, t_c) * Pop_r(j) + \beta(t_s, t_c) \tag{3}$$

where $\alpha(t_s, t_c), \beta(t_s, t_c)$ are two time weight factors. We set t_s as the publishment time of the project. If someone's bidding has been accepted by the project, afterward set t_c as the project win-bid time, otherwise, set t_c as data collection time. According to the analysis as above-mentioned, we propose $\alpha(t_s, t_c), \beta(t_s, t_c)$ as equation(4) and equation(5).

$$\alpha(t_s, t_c) = \frac{1}{\ln(t_c - t_s + 1)} \tag{4}$$

$$\beta(t_s, t_c) = \ln(t_c - t_s + 1) \tag{5}$$

B. User modeling

In this subsection, we will model for developers. To establish the tag model and keyword model for developer u, we use the vectors of the projects gained from project modeling which have relationships with the developer u.

Let TM represent the project tag vector gained from the method described in subsection IV.A. So the developer tag vector is shown as following equation(6).

$$TM_{u,:} = \lambda_A \sum_{t \in A_u} TM_{t,:} + \lambda_B \sum_{t \in B_u} TM_{t,:} + \lambda_F \sum_{t \in F_u} TM_{t,:} \tag{6}$$

Let KM represent the project keyword vector gained from the method described in subsection IV.A. So the developer keyword vector is shown as following equation(7).

$$KM_{u,:} = \lambda_A \sum_{t \in A_u} KM_{t,:} + \lambda_B \sum_{t \in B_u} KM_{t,:} + \lambda_F \sum_{t \in F_u} KM_{t,:} \tag{7}$$

In the equation(6) and equation(7), parameters $\lambda_A, \lambda_B, \lambda_F$ represent the apply weight, win-bid weight and accept weight respectively. A_u, B_u, F_u represent apply project set, win-bid project set and accept project set of developer u respectively. After calculating vectors of all developers, we get the developer content matrix $DM_u = \{TM_u, KM_u\}$. The solution on how to get the matrix DM_u will be described in section V. Considering the value of the weights, it goes without saying that the feature of the projects which are completed by developer could represent developer ability better than the win-bid projects. While applying projects shows the least. So $\lambda_A > \lambda_B > \lambda_F$. $\lambda_A = 0.3, \lambda_B = 0.8, \lambda_F = 1.0$ are set in our experiment.

C. Feature engineering

In the establishing of recommendation system, we need to define the similarity of project and developer, and then we could match the project and developer according to the similarity.

To establish an instance s of similarity matrix SM, first of all, it is required to establish the similarity vector p of a project t and a developer u. As equation (8)(9) shown, for the i'th entry p_i of p, using the absolute distance of the i'th entry $f_{u,i}$ of developer feature vector $f_u \in DM_t$ and i' th entry $f_{t,i}$ of project feature vector $f_t \in DM_u$ to represent the similarity. Since there exists a negative correlation between absolute distance and similarity, we use a constant value m to minus absolute distance to gain the similarity value. The constant value m is named the maximum value of homogenization.

$$d = |f_{u,i} - f_{t,i}| \tag{8}$$

$$p_i = m - d \tag{9}$$

We have made an in-depth analysis of the following four situations:

Situation 1: a feature exists in the project vector, but doesn't exist in user vector.

Situation 2: a feature exists in the user vector, but doesn't exist in project vector.

According to equation (9), the situation 1 and situation 2 get same similarity value. But considering the real world experiment, the similarity of situation 1 is lower than situation 2. And the reason is obvious: the requirement of project should be more important than user's ability. So while situation 1 happens, equation (10) is used instead of equation (9).

$$p_i = m - a * d \tag{10}$$

where a is a factor bigger than 1. We could use a to gain a lower similarity of situation 1. The factor a is named negative feature factor.

Situation 3: a feature exists in both project and user vectors.

Situation 4: a feature doesn't exist in any project and user vector.

According to equation (9), the situation 3 and situation 4 get same similarity value. But considering the real world experiment, the similarity of situation 4 is higher than situation 3. The situation 3 is rare and worthy of more attention, it shows that the project requirement could be satisfied by the developer. Meanwhile the situation 4 is more common, it shows the phenomenon that the project doesn't have a demand meanwhile the developer doesn't have the ability is not important. So while situation 3 happens, using equation (11) instead of equation (9).

$$p_i = m - d - z \tag{11}$$

where z is a factor bigger than 0. We could use z to gain a lower similarity of situation 3. The factor z is named zero feature factor.

Through the analysis shown above, we could get the i'th feature's similarity between a developer and a project, then we could get the similarity vector p through the same way. We use this similarity vector as the instance's features consisted in similarity matrix SM, and treat apply condition as instance's label. Provided that there is an application relation between project and developer of an instance, then we set instance's label as 1, and this instance is called a positive instance; otherwise, we set instance's label as 0, and this instance is called a negative instance. For an instance which is consisted of a developer and a project, there are corresponding feature vector p and tag l, $s = \{(p,l)|l = \{0,1\}\}$. The vector s form the matrix SM. We set the ratio of positive and negative instances to k, $k = \frac{|S_1|}{|S_0|}$, where $|S_1|$ represents the number of positive instances, and $|S_0|$ represents the number of negative instances.

For positive instance establishment, we directly select the developer and project with apply relationship. For negative instance establishment, we need to choose a developer and some of the projects which aren't applied by this developer to calculate the similarity. We preferentially use a highly exposed project with no application. Compared to random project, a project with higher visibility makes it less likely to come true that a developer doesn't apply for a project just because the developer hasn't seen the project before. In this way, we could get the negative interest better. So we use the method which is proposed in subsection IV.A to calculate the visibility $Pop(t)$ of project t, and sort projects according to this value. Then, according to the sequence of the projects, we check whether the application relationship exist between the developer and the project. If so, we continue to check the next one; if not, we use current project and developer to build a negative instance.

According to the analysis shown above, we need to assign maximum value of homogenization m, negative feature factor a, zero feature factor z and ratio of positive and negative instances k. We choose $m = 10, a = 2, z = 8, k = 1.2$ assignment in following experiment for good performance.

V. Weight-SCL : a Transfer Learning Algorithm with Feature Mapping

After data analysis and data modeling, we improve the existing transfer learning algorithm SCL to make it adapt to the data of software crowdsourcing platform, then he Weight-SCL algorithm is formed. In this algorithm, we divided the features into two kinds: important feature and unimportant feature. Specific to software crowdsourcing platform data, the important feature is tag feature, and unimportant feature is keyword feature. The importances of the two kinds of features are different. In general, the important features of source domain and target domain will not be all the same, there will exist quite a few features which are domain-specific. And we observed that some features between source domain important feature and target domain unimportant feature are same, and some features between source domain unimportant feature and target domain importance feature are same. So if we can transform and combine the important feature and unimportant feature, then we could improve the number of important feature, and improve the number same feature between source domain and target domain. Therefore, we focus on answer following questions: how to distinguish the feature importance, how to combine the important features with unimportant features, and how to transfer knowledge. The Weight-SCL algorithm we proposed is a solution for those challenge.

A. SCL Algorithm

First of all, we need to introduce the origin SCL algorithm[7]. Given the labeled data from the source domain and unlabeled data from both source and target domains, SCL first selects a set of m pivot features that appear frequently in both domains. Then, training linear predictors for every pivot features to predict occurrences of each pivot in the unlabeled data to model the correlations between the pivot features and other features [27]. The predictor for the l'th pivot feature is characterized by its weight vector w'; positive entry in the weight vector means that the non-pivot feature is related to the corresponding pivot feature highly. The weight vectors of each pivot can be arranged into a matrix $W = [w_l]_{l=1}^{n}$. Let $\theta \in R^{k \times d}$ (d represents the number of total features) be the top k left singular vectors of W. These vectors are the major predictors for our weight matrix. If we select our pivot features well, then we expect these predictors to distinguish between total features in both domains. At training and testing phase, suppose there is a feature vector x. We use the projection θ_x for k real valued features. Now we obtain a predictor for an augmented instance $\{x, \theta_x\}$. The predictor will perform well in both source and target domains when θ contains meaningful correspondence.

B. Example

In this subsection, we will use an example to introduce the problem and challenge we face in multi-domain recommendation system.

First of all, we divide the word features into tag feature and keyword feature. The tag feature is carefully designed by the system developer, the number of tag feature is less, and the tag feature has higher importance in project classification and the project recommendation. And the keyword feature is automatically extracted from the project description, the number of keyword features is large, so there is a certain degree of error rate. The words in keyword feature have different effects on project classification, some of these words are valid for classification problems, while others are not. In this algorithm, we treat the tag feature as the important feature, and keyword feature as unimportant feature. We will pay more attention to important features, and use unimportant features as a secondary data.

TABLE I. CROSS-DOMAIN CLASSIFICATION EXAMPLES

Domain	Tag	Description
Source Domain	Java	...To carry out related web development work as required...
Source Domain	UI, interface design	...An interface design optimization scheme is proposed for system...
Target Domain	web development	...The Web is developed using Java...
Target Domain	HTML5, front-end	...Make new HTML5 games based on the game template...

TABLE II. CROSS-DOMAIN CLASSIFICATION EXAMPLES

Source domain		Target domain	
Tag	Keyword	Tag	Keyword
Java	web development	web development	Java
UI,interface design	interface design	HTML5,front-end	HTML5

Consider the example presented in TABLE I. Each instance in the source and target domain has some tag words and a project description. By using the algorithm described in the previous subsection, we can extract some keywords from the project description. TABLE II describes the tags and keywords in the source and target domain instances.

In the source domain, there are "java", "web development", "UI" in tag word set, and there are "web development", "interface design", "java" in keyword set which are extracted from project description. In the target domain, there are "html5", "web development", "front-end" in tag word set, and there are "html5", "java" in keyword set.

Consider using SCL algorithm on such data. Because there is not multiple kinds of features in SCL, if we only use the data of tag features, it goes without saying that it will cause data waste. If we apply the SCL algorithm on the two kinds of features separately, we cant gain any pivot feature because of no feature word which appears on both two domains. But we could observe that there are some unimportant features which appear on important feature set. For example, "web development" appears in both tag set of source domain and keyword set of target domain, "java" appears in both tag set of target domain and keyword set of source domain. So if we could combine the tag features and keyword features in some ways, we could get more pivot features. But if we directly combine two features, the importance of tag feature will be diluted by lots of unrelated words in keyword set. And for the word which appears in both important feature and unimportant feature, such as the word "HTML5" in target domain, how to combine such word value is also a question to be answered. In addition, some of the pivot features extracted from all features may have representative meaning, such as "UI" and "interface design", we can find out such words and cluster them. It is helpful for using more representative words to represent different kinds of projects.

C. Weight-SCL Algorithm

To solve the challenge shown above, we propose Weight-SCL based on SCL algorithm, as Algorithm 1 shown. Weight-SCL consists of 4 steps: SelectPivots, MergeFeature, TrainPivotPredictors and ComputeSVD. We will explain them in the following subsections.

D. SelectPivots

We need to select pivot features from all features in source domain and target domain. The pivot feature needs to

Algorithm 1 Weight-SCL algorithm

```
Input:
    the unimportant features set from source domain $KM_S$;
    the unimportant features set from target domain $KM_T$;
    the important features set from source domain $TM_S$;
    the important features set from target domain $TM_T$;
    Parameter $m, k, \varphi, \gamma_:, w$
Output:
    predictor f

    function SELECTPIVOTS($KM_S, KM_T, TM_S, TM_T, \varphi, \gamma_:, m$)
        $P`_k, P`_t = TransferFeature(KM_S, KM_T, TM_S, TM_T, \varphi)$
        $P_t = EliminateFeature(P`_t, \gamma_t)$
        $P_k = EliminateFeature(P`_k, \gamma_k)$
        $Cluster(P_k, P_t, m)$
    end function
    function MERGEFEATURE($KM_i, TM_i, w$) $i \in \{S, T\}$
        $D_i = KM_i + TM_i \times w$
    end function
    function TRAINPIVOTPREDICTORS($D_i, P$ )
        $D' = \{(Mask(x, p), In(x, p)) | x \in D_i\}$
        for l=1 to m do
            $w = \arg\min(\sum_{(x,y)\in D'} L(wx, y) + \lambda||w||^2)$
        end for
        $W_i = [w'_1] \dots [w'_m]$
    end function
    function COMPUTESVD($W_i, k$)
        $[UDV^T] = SVD(W)$
        $\theta_i = U^T_{[1:k,1:|V|]}$
    end functionreturn $\theta_i$;
```

satisfy two conditions: 1) The pivot features must appear in both the source and target domains and appear frequently. 2) Pivot features need to have a greater impact on the project recommendation.

In software crowdsourcing platform, there are two different kinds of features, and there are overlapping between the features. To gain pivot features better, we need to transfer some unimportant features to important features.

$P`_k, P`_t = TransferFeature(KM_S, KM_T, TM_S, TM_T, \varphi)$

We use $V_{S,K}$ represents source domain important feature word set, $V_{T,K}$ represents target domain important feature word set, $V_{S,T}$ represents source domain unimportant feature word set and $V_{T,T}$ represents target domain unimportant feature word set separately. We choose a subset $V'_{K,S} = V_{K,S} \cup V_{T,T} - V_{K,S}$. And we also select a subset $V'_{K,T} = V_{K,T} \cup V_{T,S} - V_{K,T}$. The meaning of this step is to select the feature subset of the non-important features that appear in the important features of the opposite domain. This word subset is a part of the important feature words of the opposite domain, and word subset can be used to represent the characteristics of a project well. So we extract this part of the unimportant features into important feature set. We add the word set $V'_{K,S}$ into the word set $V'_{K,S}$, and add the word set $V'_{K,T}$ into the word set $V'_{K,T}$. Finally, we get new word sets V of keyword and tag from both two domains.

For converting some unimportant features into important features, we need to change the value of the unimportant features of instances. For an instance $d = (x_k, x_t)$ in source domain, where $x_k \in KM_S$ is d's unimportant vector, and $x_t \in TM_S$ is d's important vector, if the i'th word entry of x_t occurs in word set $V'_{K,S}$, and the value of this entry is more than the threshold value φ, we will change this value to 1. Also, for an instance $d = (x_k, x_t)$ in source domain, if the i'th word entry of x_t occurs in word set $V'_{K,T}$, and the value of this entry is more than φ, we will change this value to 1. The significance of this step is that the words that appear in the changed important features are not necessarily very important, and the important features need to represent the characteristics of the project. Therefore, we consider the entry value as an important degree. For the feature that the value is less than a certain threshold φ, we consider that it does not contribute to the characteristics of the representative project. In our experiment, we select $\varphi = 0.1$ in order to keep that we could get sufficient pivot features.

Then, we get a new data matrix after feature transformation. After this, we could get a word set $P`_k$ whose words occur at both $V_{S,K}$ and $V_{T,K}$. So $P`_k = KM_S \cup KM_T$. And we could get the word set $P`_t = TM_S \cup TM_T$ in the same way.

After feature transfer, we need to get the pivot features from all the features:

$P = EliminateFeature(P`, \gamma)$

We select a feature subset P from all the features P', including all the features that appear simultaneously in the source domain and target domain. Then we delete the features of the feature set P whose frequency are less than a threshold value γ. The residual feature in P is output as pivot feature set.

The difference between important and non-important features is that the selection of γ threshold is different. We consider that the pivot feature of the important feature should account for a large proportion. We choose the appropriate thresholds γ_t, γ_k to get enough pivot features(more than m pivots) and control 80% proportion of important features in the whole pivot features.

$Cluster(P_k, P_t, m)$

In this step, we select m pivot features from word feature set $P_k \cap P_t$. The meaning of some word features which are obtained after above steps might be very similar. Therefore, we cluster those words and get m clusters, and use the nearest word from the cluster center. We choose the k-means clustering algorithm to cluster words, and the distance d between the two vectors a and b is defined as equation(12).

$$d = \sum (a_i - b_i)^2 \tag{12}$$

E. MergeFeature

We hope that the features with higher importance can be more fully exploited when training linear classifiers. Therefore, we define a weight named importance weight w. For the important feature, we multiply the weight and feature's value. In order to obtain a combined feature data, we add important features and non-important features. The equation(13) is shown as follows. In our experiment, we use $w = 10$ as the importance weight.

$$D = KM + TM \times w \tag{13}$$

F. TrainPivotPredictors

Weight-SCL models the correlations between each pivot $p \in P$ and all other words by training linear classifiers for every pivot feature to predict whether or not pivot feature occurs in a feature vector, with the help of the other features. Due to this reason, for each pivot feature $p \in P$, a training set D' is created:

$$D' = \{(Mask(x,p), Value(x,p)) | x \in D\}$$

$Mask(x,p)$ returns a copy of x where the component associated with the p word feature is removed, which is equivalent to removing this word from the feature space. $Value(x,p)$ returns the value of the component of x associated with the word p. For each linear classifier for D which is characterized by the parameter vector w, it is trained by minimizing Equation (14) on D.

$$w = \arg\min\left(\sum_{(x,y)\in D'} L(wx,y) + \lambda ||w||^2\right) \tag{14}$$

Noted that positive entry in the weight vector means that the non-pivot feature is related to the corresponding pivot feature highly.

G. ComputeSVD

Weight-SCL identifies correlations across pivots by computing the singular value decomposition of the $|V| \times m$-dimensional parameter matrix $w = [w'_1] \dots [w'_m]$. We compute a low-dimensional linear approximation to feature of pivot feature for computation reasons.

$$\left[UDV^T\right] = SVD(W) \tag{15}$$

Noted that W encodes the covariance of pivot and non-pivot features. The rows of U identify common substructures among these multiple classifiers. We choose the rows of U associated with the largest singular values yields, then define θ=$U^T_{[1:k,1:|V|]}$ as those rows of U which are top k largest singular vectors of W.

Furthermore, $\theta_S * D_S$ is the desired mapping of low dimensional feature representation of source domain words relation, and $\theta_T * D_T$ is the desired mapping of target domain words relation. We could get a tighter representation of project features through this way. The feature matrix DM described in subsection IV.A and subsection IV.B is calculated by $\theta * D$. Then we could compute the similarity matrix SM on this new representation $\theta * D$ using approach shown in subsection IV.C between developer feature vector and project feature vector. Finally, the recommendation predictor is trained on the dataset SM, where $s = \{(p,l)|l = \{0,1\}\} \in SM$. Noted that in our experiment, the linear regression algorithm was used to be the recommendation predictor.

VI. Experiments

In this section, we first present our experimental settings and then analyze experiment results. Two experiments were conducted: 1) We verified our method in proving that with the increase of the number of pivot features, the recommended accuracy will rise; 2) We compared Weight-SCL with other approaches to recommend suitable projects to developers.

A. Data Sets

We selected two well-known software crowdsourcing platforms: Zhubajie, Joint Force. Each platform has its own data characteristics, but our algorithm used only the project description information, project tags information and developer information who has apply/win-bid/accept relationships with the project. TABLE III shows the details of the data on the software crowdsourcing platform. From the website of Zhubajie we crawled 6000 projects, 10597 developers and 48769 application relationships; from the Joint Force we crawled 2,800 projects, 5231 developers, and the number of relationships is 15498.

TABLE III. Dataset from Two Typical Software Crowdsourcing Platforms in China

Software couwdsourcing platform	The number of projects	The number of developers	The number of relationships
Zhubajie	6000	50597	487690
JointForce	2800	4131	15498

As the data of Joint Force is significantly less than the data of Zhubajie, we used the data of the Zhubajie as the source domain data and used the data of Joint Force as the target domain data, and transfered the data from the Zhubajie platform to the Joint Force platform.

B. Comparison Method and Baseline Method

We used ICBNN algorithm to simulate the existed software recommendation system. ICBNN is content-based neighbor algorithm. It uses the content matrix to calculate the similarity. Equation(16) illustrates how the similarity of developer u and project i is calculated in the algorithm.

$$ICBNNsim_{u,i} = \frac{\sum\limits_{i'\in I_k} sim_{i,i'} * U_{u,i'}}{k} \tag{16}$$

where $sim_{i,i'}$ represents the degree of similarity between projects; I_k gives the top-k projects which are most similar to project i by calculating the similarity of the cosine of the row vector in the content matrix of the corresponding projects;$U_{u,i'}$ represents whether or not developer u applied project i before. It returns 1 if applied, and returns 0 otherwise.

And we use linear regression as baseline method to compare with SCL algorithm and describe the effect of the transfer learning.

C. Evaluation Metrics

In this paper, we selected the two indicators $P@k$ and $R@k$ to evaluate the performance of the proposed algorithm. $P@k$ reflects the recommendation accuracy, that is, in the recommended top-k project, how many projects are related to the existence of applied developers; and $R@k$ reflects the recall rate, that is, in the test set how many projects which

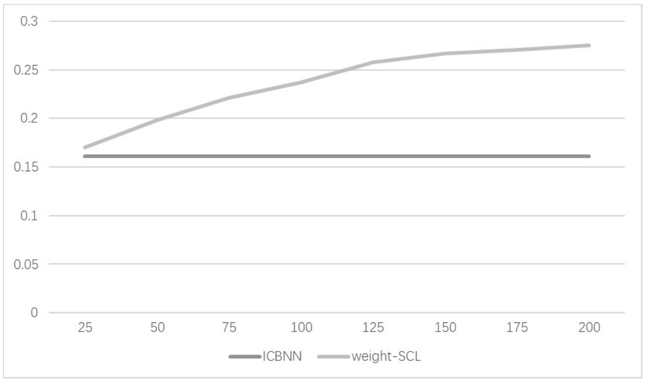


Fig. 2. P@10 Results of Weight-SCL as a function of the number of Pivots m

need to be recommended are in top-k projects recommended by algorithms.

In the recommendation algorithm, if we treat a developer u as an input, the algorithm will return a collection of projects, each of which has a corresponding training value, representing the matching degree of developer and project. The algorithm can sort the training values and obtain a recommended sequence for u.

Assuming that the sequence obtained after sorting is l_u, the function $h(k, l_u)$ is defined, which inputs a sequence that returns the projects of top-k in the list. We evaluated the recommended results using the equation(17) and equation(18).

$$P@k = \frac{1}{|Dev|} \sum_{u \in Dev} \frac{\sum_{i \in Test_u} 1\{i \in h(k, l_u)\}}{k} \tag{17}$$

$$R@k = \frac{1}{|Dev|} \sum_{u \in Dev} \frac{\sum_{i \in Test_u} 1\{i \in h(k, l_u)\}}{|Test_u|} \tag{18}$$

where $Test_u$ represents the project set corresponding to the developer in the test set; Dev represents the developer set.

D. Experiment about the Number of Pivots

The Figure 2. shows the influence of the number of pivots m on the performance of Weight-SCL. The plots show that even a small number of pivots could get a higher accuracy, which means it captures a significant amount of the correspondence between source domain and target domain. And with the increase of the number of pivot features, the recommendation accuracy will rise. And our method could help improving the number of pivots, so it could help improving the recommendation accuracy afterwards. And in the experiment of comparing Weight-SCL with other approaches, we selected $m = 150$ as the number of our pivot features.

E. Recommendation Compared with Other Approaches

First of all, we will introduce the application of three algorithms: ICBNN algorithm was trained and tested on the JointForce data set, meanwhile SCL algorithm and Weight-SCL algorithm were trained in the Zhubajie and JointForce data set, tested in the JointForce data set. We divided the data from JointForce into 10 parts, and tested all algorithms in one part; trained the ICBNN model with other 9 parts, trained SCL and Weight-SCL with whole Zhubajie Dataset and other 9 JointForce data set parts. For the SCL algorithm training ,we applied it on the two kinds of features separately.

From the data shown in Figure 3 and Figure 4, we can see that the performance of the three algorithms on P and R is basically the same, that is, the multi-source recommendation algorithm had significantly higher accuracy and recall rate than the SCL algorithm; the accuracy and recall rate of multi-source recommendation algorithm are much higher than single-source recommended algorithm. The analysis of the test result is shown as follows: the multi-source recommendation algorithm used the weight-SCL algorithm for the two kinds of feature problems, which is much better than the basic method of the SCL algorithm. Compared with the single-source recommendation algorithm, the multi-source recommendation algorithm transfered the data of Zhubajie platform, so the data set has been greatly expanded. And Weight-SCL increased the number of pivot features by transfering and combining the features, which is illustrated in the experiment about the number of pivots that with the increase of the number of pivot features, the recommendation accuracy will rise. So the effect of our algorithm had a 1.6X upgrade than ICBNN algorithm and 1.2X upgrade than SCL algorithm on accuracy and recall rate.

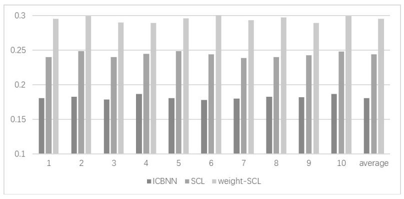
VII. Conclusion and Future Work

This paper adopts transfer learning approach to solve platform coldstart problem in software croudsourcing platform recommendation system. We try to transfer data from other software crowdsourcing platforms to expand the training dataset. We use tags and keywords extracted from project description as features in the recommender system, and design a method to calculate the similarity of projects and developers. Furthermore, Weight-SCL algorithm we proposed maps different kinds of features both in the source domain and the target domain after a certain transformation and combination to a latent space by learning the correspondences between features. The experiments were designed to verify the performance of the multi-source recommendation system in comparison with the basic SCL method, using the data from two famous software crowdsourcing platforms. The experimental results show it gets 1.2X performance promotion in accuracy and recall rate.

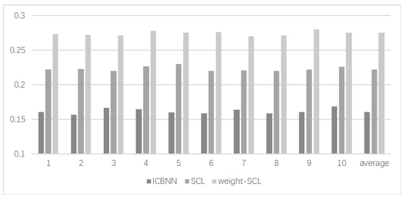
In the future work, several aspects of research is still worthy: 1) Currently we only use the content-based recommendation algorithm to construct the recommendation system, so we will further introduce the collaborative filtering algorithm to establish a hybrid recommendation system, and propose an improved transfer learning approach on it. 2) In this paper, we only utilize the data information of the recommendation system to map the features, and the next step we will research on how to map the features using the data of the existing knowledge base.

Acknowledgement

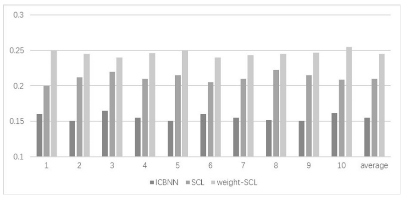
This research is supported by 973 Program in China (Grant No. 2015CB352203) and National Natural Science Foundation of China (Grant No. 61472242). Thanks JointForce for providing the experimental data set.



(a) p@5

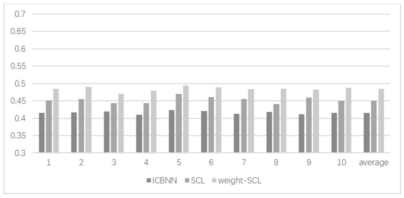


(b) p@10

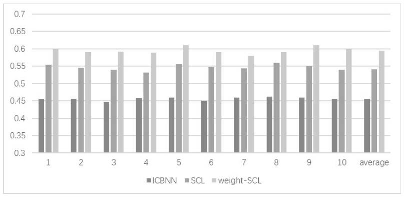


(c) p@15

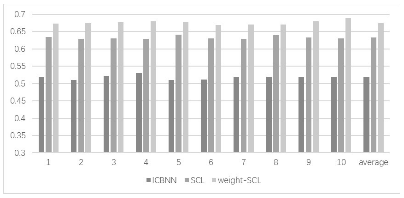
Fig. 3. Accuracy Results of Project Recommendation in Software Crowdsourcing Platform



(a) r@5



(b) r@10



(c) r@15

Fig. 4. Recall Results of Project Recommendation in Software Crowdsourcing Platform

References

[1] 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.

[2] 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.

[3] Vamshi Ambati, Stephan Vogel, and Jaime Carbonell. Towards task recommendation in micro-task markets. In *AAAI Conference on Human Computation*, pages 80–83, 2011.

[4] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.

[5] Amit Pratap Singh, Girijesh Kumar, and Rajeev Gupta. Relational learning via collective matrix factorization. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 650–658, 2008.

[6] Bin Li, Qiang Yang, and Xiangyang Xue. Transfer learning for collaborative filtering via a rating-matrix generative model. In *International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June*, pages 617–624, 2009.

[7] John Blitzer, Ryan Mcdonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *Conference on Empirical Methods in Natural Language Processing*, pages 120–128, 2006.

[8] Sinno Jialin Pan, Xiaochuan Ni, Jian Tao Sun, Qiang Yang, and Zheng Chen. Cross-domain sentiment classification via spectral feature alignment. In *International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, Usa, April*, pages 751–760, 2010.

[9] William Safire. *No uncertain terms : more writing from the popular "On language" column in The New York times magazine*. Simon & Schuster, 2013.

[10] Aniket Kittur, Jeffrey V. Nickerson, Michael Bernstein, Elizabeth Gerber, Aaron Shaw, John Zimmerman, Matt Lease, and John Horton. The future of crowd work. In *Proceedings of the 2013 conference on Computer supported cooperative work*, pages 1301–1318, 2013.

[11] Man Ching Yuen, Irwin King, and Kwong Sak Leung. Task matching in crowdsourcing. In *Internet of Things*, pages 409–412, 2012.

[12] Man-Ching Yuen, Irwin King, and Kwong-Sak Leung. Task recommendation in crowdsourcing systems. In *International Workshop on Crowdsourcing and Data Mining*, pages 22–26, 2012.

[13] Man Ching Yuen, Irwin King, and Kwong Sak Leung. *TaskRec: Probabilistic Matrix Factorization in Task Recommendation in Crowdsourcing Systems*. Springer Berlin Heidelberg, 2012.

[14] Sooyoung Lee, Sehwa Park, and Seog 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*, pages 60–65, 2014.

[15] Y Zheng, R Cheng, S Maniu, and L Mo. On optimality of jury selection in crowdsourcing. *Intl.conf.on Extending Database Technology Brussels Belgium*, 2015.

[16] Yudian Zheng, Jiannan Wang, Guoliang Li, Reynold Cheng, and Jianhua Feng. Qasca:a quality-aware task assignment system for crowdsourcing applications. In *ACM SIGMOD International Conference on Management of Data*, pages 1031–1046, 2015.

[17] Ju Fan, Guoliang Li, Beng Chin Ooi, Kian Lee Tan, and Jianhua Feng. icrowd: An adaptive crowdsourcing framework. In *ACM SIGMOD International Conference on Management of Data*, pages 1015–1030, 2015.

[18] Shixiong Zhao, Beijun Shen, Yuting Chen, and Hao Zhong. Towards effective developer recommendation in software crowdsourcing. In *The International Conference on Software Engineering and Knowledge Engineering*, pages 326–329, 2015.

[19] Jiangang Zhu, Beijun Shen, and Fanghuai Hu. A learning to rank framework for developer recommendation in software crowdsourcing. In *Software Engineering Conference*, pages 285–292, 2015.

[20] Ning Li, Wenkai Mo, and Beijun Shen. Task recommendation with developer social network in software crowdsourcing. In *23rd Asia-Pacific Software Engineering Conference, APSEC 2016, Hamilton, New Zealand, December 6-9, 2016*, pages 9–16, 2016.

[21] Pan W, Xiang E W, Liu N N, and et al. Transfer learning in collaborative filtering for sparsity reduction. In *AAAI*, pages 230–235, 2010.

[22] Bin Li, Qiang Yang, and Xiangyang Xue. Can movies and books collaborate? cross-domain collaborative filtering for sparsity reduction. In *IJCAI 2009, Proceedings of the International Joint Conference on Artificial Intelligence, Pasadena, California, Usa, July*, pages 2052–2057, 2009.

[23] Sihong Xie, Wei Fan, Jing Peng, Olivier Verscheure, and Jiangtao Ren. Latent space domain transfer between high dimensional overlapping distributions. In *International Conference on World Wide Web*, pages 91–100, 2009.

[24] John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, bollywood, boomboxes and blenders: Domain adaptation for sentiment classification. *Acl*, 31(2):187–205, 2007.

[25] C Mcdevitt, E Gilbertson, and H Muir. Learning visual representations using images with captions. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, 2007.

[26] G. Salton and Clement T. Yu. On the construction of effective vocabularies for information retrieval. *Acm Sigplan Notices*, 10(1):48–60, 1973.

[27] Rie Kubota Ando and Tong Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6(3):1817–1853, 2005.