2015 Asia-Pacific Software Engineering Conference

TBIL: A Tagging-Based Approach to Identity Linkage across Software Communities

Wenkai Mo*, Beijun Shen*†, Yuting Chen*, Jiangang Zhu*
*School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China.
†bjshen@sjtu.edu.cn

***Abstract*—Nowadays, developers can be involved in several software developer communities like StackOverflow and Github. Meanwhile, accounts from different communities are usually less connected. Linking these accounts, which is called identity linkage, is a prerequisite of many interesting studies such as investigating activities of one developer in two or more communities. Many researches have been performed on social networks, but very few of them can be adapted to software communities, as information of users provided in these communities has a huge difference to that in social networks. We tackle with the problem by introducing TBIL, a novel tagging-based approach to identity linkage among software communities. The essential idea of this approach is to employ skills (measured by tags), usernames and concerned topics of developers as hints, and to use a decision tree-based algorithm and another heuristic greedy matching algorithm to link user identities. We measure the effectiveness of TBIL on two well-known software communities, i.e., StackOverflow and Github. The results show that our method is feasible and practical in linking developer identities. In particular, the F-Score of our method is 0.15 higher than previous identity linkage methods in software communities.**

I. Introduction

Nowadays, software developers do not create software alone. When facing intractable problems during their development, the developers may seek questions and answers from software developer communities such as the StackExchange networks [1]. One of the most popular Q&A sites is StackOverflow[1], which owns more than 4 million registered users and 11 million questions[2]. Many developers also share and learn open-source projects in software repository platforms. One of such platforms is Github[3], in which as of 2015, there were over 9 million users and 21.1 million repositories[4]. Both of the Q&A sites and the software repository sites are important for facilitating software developers and speeding up software development.

On the other hand, developers can get involved in two or more software communities. Some interesting researches, such as cross sites data mining [2] or recommendation [3], can thus be performed for helping understand the developer behaviors and optimize these communities. Also, integrating heterogeneous data of a developer from different communities can provide hunter companies more valuable information for recruiting. Meanwhile, most of these researches rely on Identity Linkage, a technique of linking accounts among different communities [4]. Fig. 1 shows a simple example of identity linkage between two software communities, each of which contains only three users. Although six accounts have been created in the two communities, they only refer to three people, corresponding to three linkages between two communities: the user Brandon exists in community 1 and community 2, while Sam Peter in community 1 has an abbreviation (Sam P.) in software community 2, and John in community 1 and Jackson in community 2 are the same developer.

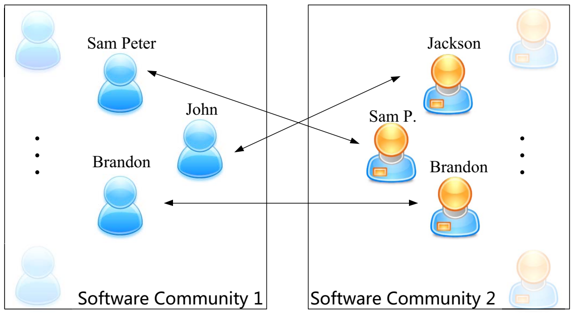

Fig. 1. An Example of Identity Linkage between Two Software Communities

Identity linkage problem has been widely studied in social networks like Facebook and Twitter [4][5][6]. The researches usually link identities by capturing user profiles (including usernames, avatars, descriptions, locations, etc.), constructing users' topological structures and as well tracking user behaviors (e.g., writing styles, uploading photos, etc.). However, these approaches are not easy to adapt to software communities, as the personnel information is not shareable and trackable. For example, a user is not allowed to upload photos in Github or StackOverflow. Thereafter, some researches in software communities, such as [7][8][9], focus on using usernames and user emails to link user identities, while it is still possible that many users choose different usernames in different communities. User emails are also not accessible in many communities for privacy reasons.

Having realized the limitation of adapting the existing approaches to identity linkage in software communities, we believe that some extra information can be mined for linking identities, because (1) unlike people in social networks, users in these communities are all developers and have specific programming skills (such as Java, JavaScript, C++, etc.), and (2) topics discussed in these communities are all related to software engineering. In this paper, we borrow some ideas from identity linkage methods in social networks, and then attempt to introduce these two special features to solve this problem in software communities. However, we will face some

[1]http://stackoverflow.com/
[2]https://en.wikipedia.org/wiki/Stack_Overflow
[3]https://github.com/
[4]https://en.wikipedia.org/wiki/GitHub

fundamental problems: (1) How to obtain a set of skills terms for different communities and how to obtain the skills of each user? (2) How to get topics that a user concerns? (3) How to link users based on various features?

To address these challenges, we propose a novel method, TBIL, to link users between different software communities. Our method consists of three steps: (1) to extract three kinds of features from usernames, user skills and user concerned topics from the text information generated by users. To obtain user skills for each user in two communities, we propose a cross-sites tagging method based on Naive Bayes and spreading activation technique [10]. To describe user concerned topics, we apply a topic model, Latent Dirichlet Allocation (LDA) [11] to get the topic distribution for each user. (2) based on these features, to apply a machine learning method, Decision Tree (C4.5 algorithm [12]), to obtain probabilities for every two users in different communities, which represents how likely they refer to a same person. (3) based on these probabilities, to use a heuristic greedy matching algorithm to link these users with one-to-one constraints, which states that no user has multiple accounts in the same community (for one user has multiple accounts, it is a different problem [13]). We measure the effectiveness of TBIL on two well-known software communities, i.e., StackOverflow and Github. The results show that our method is feasible and practical in linking developer identities. In particular, the F-Score of our method is 0.15 higher than other previous identity linkage methods in software communities. Our main contributions are summarized as follows: (1) We propose an identity linkage method for software communities by leveraging user skills and user concerned topics compared to previous methods. (2) We propose a cross-sites tagging algorithm to obtain user skills. (3) We propose a heuristic greedy matching algorithm to link users with one-to-one constraints.

II. Related Work

In this section, we first introduce methods of identity linkage in both software communities and social networks. As our method refers to tagging problem, we also present some approaches in this area.

A. Identity Linkage in Software Communities

To link users in different software communities, several methods have been proposed [7][8][9]. In these methods, each user in a community is represented as a pair (*username*, *email*). Goeminne et al. proposed a simple algorithm [9] by defining some simple rules; for example, if two pairs share at least one element, or at least one shared element has length at least in a certain threshold *minLen*, then they are the same person. This approach is robust against noisy aliases. A more advanced algorithm was proposed by Bird et al. [8], where some text similarity measures, like Levenshtein distance, are used to measure the similarity between two usernames rather than defining some simple matching rules. Both Goeminne and Bird did not consider the semantic information in the name and the email; therefore, Erik. et al. proposed a semantic-based method [7] to link users by applying Latent Semantic Analysis (LSA) [14]. However, all these three methods mentioned above only considered the username and the email of a user while many other important information is ignored such as the programming skills and the social behaviors of a user. As users in Github and StackOverflow are all developers, they may grasp different kinds of programming skills such as Python or C++. Besides, due to the privacy, some communities do not provide the user email information.

B. Identity Linkage in Social Networks

Identity Linkage has been a hot research topic in social networks like between Facebook and Twitter in recent years. A kind of method is linking users by their profile, in which there are many available tagging information such as username, avatar, location, occupation etc. [15][16][17][18][19]. Iofciu et al. proposed a method, in which each user profile is represented as a vector, and then linking users by computing the similarity between two vectors [15]. On the other hand, user-generated contents like twits, and user-behavior like writing style are also considered in some approaches [4][20][6][5]. Liu et al. proposed a framework to link social identities called HYDRA [4]. Their method considers heterogeneous behavior of users and build consistency matrix to model the structure consistency in different social networks. Unfortunately, the goal of our paper is to link users in software communities. The largest difference between software communities and social networks is that the quality and the type of contents people can generate in software communities are far less that those in social networks; for instance, users in software communities are not allowed to share their locations. Besides, some software communities lack of topological structures among users, such as people cannot follow others in StackOverflow.

C. Tagging Techniques

There are a lot of studies of tag recommendation for social media sites like Flicker and Twitter [21][22][23][24]. Also, tag recommendation in software engineering sites is in high demand [25][26][27]. Wang et al. proposed a tag recommendation system called EnTagRec [25], in which it combines the Bayesian Inference, Frequentist Inference and spreading activation technique [10] to recommend tags for StackOverflow. Xia et al. proposed a tag recommendation for software information sites like Freecode and StackOverflow based on multi-label classification [26]. All these methods will finally recommend several tags to the contents in software information sites, while our method focuses on getting a tag distribution for each users and then calculates distributions similarity between two users to measure how likely they refer to a same nature person. Tag recommendation is the closest work to ours, we thus borrow the multi-label classification idea and the spreading activation idea from previous work.

III. Problem Definition

According to [4], we define the identity linkage problem in software communities. Let ***P*** donate the set of all nature people in real life. For two software communities *C* and *C'*, let U_C and $U_{C'}$ donate the set of users in *C* and *C'* respectively. Let $g(u) = p$, where $u \in C \cup C'$, be a function, mapping a user u to a nature person p, where $p \in \boldsymbol{P}$. The identity linkage problem is to find a function $f : U_C \times U_{C'} \rightarrow \{0, 1\}$, which means for user pair $(u_C, u_{C'}) \in U_C \times U_{C'}$, if $f(u_C, u_{C'}) = 1$, then u_C and $u_{C'}$ are referring to the same nature person, namely, $g(u_C) = g(u_{C'})$; otherwise, $g(u_C) \neq g(u_{C'})$. Because our problem is to link users with one-to-one constraints, if $g(u_C) = p$, there does not exist another $u'_C \in U_C$, such that $g(u'_C) = p$. In our problem, each user is represented by a pair $u = (UN, D)$, where UN is the username and $D = \{d_1, d_2, ..., d_n\}$ is all documents generated by this user.

The key issue to solve the identity linkage problem is to learn the function $f(u_C, u_{C'})$ to predicate whether u_C and $u_{C'}$ refer to a same nature person. A straightforward approach is to find some similarities between u_C and $u_{C'}$ from information they provide or contents they generate. Previous methods in software communities only compare the similarity of usernames and the prefix of emails, while those in social networks leverage machine learning methods to combine heterogeneous information. Because we consider three kinds of features, machine learning techniques are good options to combine these features. However, classifiers in machine learning methods usually give a binary judgement, which will make many redundant links and thus lead to a poor recall. To solve this problem, we borrows the idea in [6]. Our method leverages the intermediate results of classifiers, which are the probabilities of how likely they refer to a same person, and then finds a matching through these probabilities to link users with one-to-one constraints.

IV. Identity Linkage across Software Communities

As the identity linkage problem defined in Section III, for each pair of users $(u_C, u_{C'}) \in U_C \times U_{C'}$, we first extract three kinds of features to measure the similarities between them. Next we apply a machine learning algorithm, Decision Tree, to train a model from a ground-truth data set based on these features, and then assign a probability to each users pair. Finally, a heuristic greedy algorithm will be used to select proper linkages based on these probabilities with one-to-one constraints. For each user, we need two kinds of information, documents generated by this user and the username.

A. Heterogeneous Features across Communities

We totally extract three kinds of features from the information generated by a developer in both communities, which are features from usernames, feature from user concerned topics and features from user skills.

1) Features from Usernames: Because a person may have different usernames in different software communities, username cannot be the only feature to identify a user. However, usernames are more important than documents, because some users do not generate any documents but usernames are the required information for signing up their accounts.

Some people create usernames based on their names, while some do not, we thus introduce two kinds of string matching methods, the ratio of Levenstein distance and the Jaro-Winkler distance[5], to measure the similarity between two usernames. The ratio of Levenstein distance, based on the Levenstein distance, is widely used to measure the distance between two common strings, while the Jaro-Winkler distance is designed for short strings matching, especially for the person's name. The values of both distances are in the range $[0, 1]$ and higher the value, higher the similarity between two strings.

2) Feature from User Concerned Topics: We use a state-of-the-art algorithm, Latent Dirichlet Allocation (LDA) [11], to obtain the topic distribution for each user. LDA has been shown effective to process various text data in many domains, such as software engineering and social network analysis. The inputs of LDA algorithm are a set of documents and the number of topics *K*, and the outputs are the topic distribution of each input document. In our problem, some users may generate not only one document, we thus merge all documents generated by a user to D to further apply LDA.

Because LDA is a kind of bag-of-words model, before applying it, each document D must be converted to a bag-of-words vector. Firstly, we remove all stopwords[6] which are used in almost every document such as "the", "is", etc. Then we use stemming to reduce words to their root form[7]. Finally, we filter words that only appear in one community and words appearing in a low frequency (appear less than 5 times in all documents). After obtaining a text vector for each user, we set text vectors of all users from both communities as input and then use LDA to get the topic distribution for each user. On the other hand, to decide the best number of topics *K*, we use an approach proposed in [28]. The basic idea of this method is simple, which states that the best number of topics is the one with the highest log likelihood value. Therefore, we build LDA model several times with different *K*, and then choose the *K* with the highest log likelihood value. After obtaining the topic distribution for each user, we use the symmetrical KL-divergence, defined in (2), to calculate the similarity of topic distributions between two users. KL-divergence is a famous distance measure between two distributions. Eq. (1) is the definition of KL-divergence between $p(x)$ and $p(y)$. The KL-divergence satisfies $KL(p \parallel q) \geq 0$, and $KL(p \parallel q) = 0$, if and only if $p(x) = q(x)$. Besides, higher the value, lower the similarity between $p(x)$ and $p(y)$. According to the definition, KL-divergence is not symmetrical, namely, $KL(p \parallel q) \not\equiv KL(q \parallel p)$. To make it symmetrical, a symmetrical KL-divergence is defined in (2).

$$KL(p \parallel q) = -\int p(x) ln \left\{ \frac{q(x)}{p(x)} \right\} dx \tag{1}$$

$$KL_s(p, q) = \frac{KL(p \parallel q) + KL(q \parallel p)}{2} \tag{2}$$

3) Features from User Skills: Documents in some software communities are labeled with tags, such as questions are labeled by their owners in StackOverflow. These tags can be treated as skills of users as they all refer to terms in software engineering. However, some communities do not have the tagging system. While linking users in two different communities, according to the type of these communities, there are three situations: (1) If both two communities does not have the tagging system, then we can apply the cross-sites tagging method proposed in this section to label documents in these communities to tags in another community with the tagging

[5]The library we used to calculate these two distances is in https://github.com/ztane/python-Levenshtein/

[6]The stopwords list from http://www.textfixer.com/resources/common-english-words.txt

[7]The stemming package is in http://www.nltk.org/api/nltk.stem.html.

system like StackOverflow. (2) If only one community has the tagging system, we can also use the cross-sites tagging method to build a model based on labeled data in this community and then use this model to label the other community. (3) If both two communities have the tagging system, we can use the intersection of tags in these two communities. If the intersection is null set, the cross-sites tagging method still can work.

To introduce the cross-sites tagging method, we select two communities, Github and StackOverflow, as an example. Let the set of all tags in StackOverflow donate L_{SO}. The goal is to label each repository in Github based on the content of readme file and the description of the repository. The cross-sites tagging method consists of three steps: (1) The first step is to clean some unnecessary tags in StackOverflow, such as some low frequency tags. (2) The second step is training a Naive Bayes model by all labeled documents in StackOverflow and then applying this model to label repositories in Github. (3) The last step is computing co-occurrences between each two tags in StackOverflow and then applying spreading activation to obtain more associated tags.

Because most labels only appear a few times in StackOverflow, labels in L_{SO} cannot be used directly. Interestingly, we find that 20% labels, donated by L^*, can cover all questions. Therefore, the first step is to process the rest 80% labels. For each label in the $L_{SO} - L^*$, we can either abandon it directly, or use some other similar labels which appear in L^* to rewrite it. In following situations, we will rewrite the label $l_0 \in L_{SO} - L^*$:

(1) If l_0 contains the version information, then it will be filtered by some regular expressions. For example, ***ios-4.2*** or ***ios5.1*** will be rewritten to ***ios***.

(2) If l_0 is synonyms[8] to another tag $l_1 \in L^*$, then l_0 will be rewritten to l_1. For example, ***js*** will be rewritten to ***javascript***.

(3) If l_0 is the combination of labels $\{l_1, l_2, ..., l_n\}$, where $l_1, l_2, ..., l_n \in L^*$, with "-", then we split l_0 by "-", and use $\{l_1, l_2, ..., l_n\}$ to represent label l_0. For example, ***google-maps-sdk-ios*** will be rewritten to $\{$***google-maps***, ***sdk***, ***ios***$\}$. (Noted that ***google-maps***$\in L^*$).

After cleaning invalid tags, the next step is to label each repository in Github based on the texts in readme file and the description of it. As the each question or answer (an answer can link to a labeled question) in StackOverflow is labeled, we can set these labeled data as training data to train a Naive Bayes model [29], which is widely used in text categorization with word frequencies as the features. Naive Bayes model is based on the Bayes' theorem shown in (3), where $y \in L^*$, and $\mathbf{x}$ is the input document. After training the labeled data in StackOverflow, we can obtain a Naive Bayes model which is then applied to get a tag distribution TD for each repository in Github.

$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}, y)}{P(\mathbf{x})} \tag{3}$$

[8]The synonyms relations are in http://stackoverflow.com/tags/synonyms

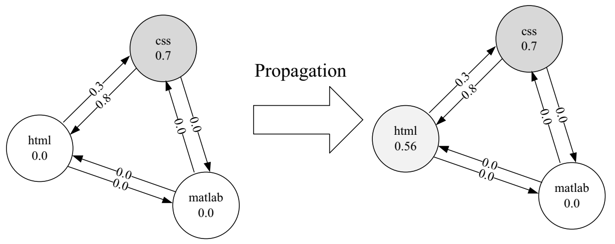

Fig. 2. An Example of Spreading Activation in One Iteration

To get the final tag distribution, we apply a technique named spreading activation [10] to infer associated tags. The inputs of spreading activation are a similarity matrix S, where each element $s_{i,j}$ in S is the similarity between the tag_i and the tag_j, and the initial state of tags which is TD in our context. During the process of spreading activation, for each tag in TD, in one iteration, it propagates its weight to other tags with corresponding similarity. Fig. 2 shows an example of an iteration of spreading activation. Supposed we have a document labeled by $\{$***css***: 0.7, ***html***: 0.0, ***matlab***: 0.0$\}$ and the universal set is $\{$***css***, ***html***, ***matlab***$\}$ with the similarity shown in Fig. 2 between every two tags. After an iteration, the ***css*** propagates to ***html*** as they have 0.8 in similarity, and thus the vector becomes $\{$***css***: 0.7, ***html***: 0.56, ***matlab***: 0.0$\}$.

To apply this technique, the most important step is to construct the similarity matrix. Each element of the similarity matrix is the similarity between two tags, and we consider that the similarity of $(tag_i \parallel tag_j)$ can be unequal to $(tag_j \parallel tag_i)$. For instance, for tags ***browser*** and ***chrome***, if a user asks a question in StackOverflow about ***chrome***, ***browser*** is also a reasonable candidate tag, while if the question is about ***browser***, then the probability of the question referring to ***chrome*** should be lower than the previous one. Thus we use co-occurrences between tags to measure this similarity. Because in StackOverflow, most questions are labeled with more than one tag, we firstly count the co-occurrence for each two tags $(tag_i, tag_j) \in L^* \times L^*$ and then calculate the joint probability $P(tag_i, tag_j)$ based on these counts. Next, we calculate $P(tag_i)$ for each $tag_i \in L^*$ also by simply counting. Finally, the condition distribution $P(tag_i|tag_j)$ is calculated by the Bayes' theorem (3), where y refers to tag_i and $\mathbf{x}$ refers to tag_j. Besides, to prevent over propagating, if $P(tag_i|tag_j) < \alpha$ ($\alpha = 0.3$ in our method), then it will be set to 0.

After obtaining the similarity matrix, the next process is to propagate initial tags to get more relevant tags. Algorithm 1 is the spreading activation algorithm. The inputs are a similarity matrix $M(n \times n)$, the initial tag distribution TD $(1 \times n)$ and the number of iteration t (In our method, t is set to 1.). The output is a new tag distribution TD', in which there are some other associated tags. In each iteration, the TD' is updated by multiple similarity matrix M (line 3). After t times propagation, we can get a new distribution TD', then we normalize it (line 5) before returning.

If a community does not have the tagging system, we can apply the cross-sites tagging method to obtain a tag distribution TD' for each user from the other community with the tagging

Algorithm 1 Spreading Activation

```
Input:
    The similarity matrix $M$;
    The initial tag distribution $TD$;
    The number of iteration $n$;
Output:
    A new tag distribution $TD'$
1: $TD' = TD$
2: for $i = 1$ to $t$ do
3:     $TD' \leftarrow TD' * M$
4: end for
5: $TD' \leftarrow normalize(TD')$
6: return $TD'$;
```

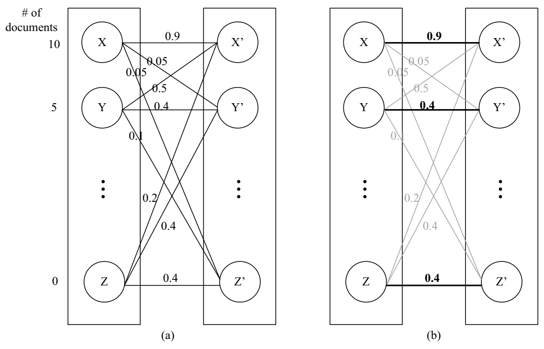

Fig. 3. An Example of Heuristic Greedy Matching Algorithm

system. Therefore, users in two communities are mapped to a same space. We then apply the symmetrical KL-divergence, defined in (2), to obtain the similarity between two users.

B. Identity Linkage

Giving two users $(u_C, u_{C'})$, we apply a machine learning algorithm, Decision Tree (C4.5 algorithm [12]), to obtain the probability of how likely they refer to the same person by combining all three kinds of heterogeneous features described in the previous section. After that, the identity linkage problem converts to find a matching in bigraph. We use a Heuristic Greedy Matching (HGM) algorithm to obtain a feasible solution. Algorithm 2 is the HGM algorithm. Obviously, the more information a user generates, the more evidences that we can use to link. Therefore, in HGM algorithm, firstly, to measure the volume of information generated by a user, we count the number of documents generated by each user in C (line 1). Secondly, HGM algorithm selects the user u_C who generates the most information (line 3), and then chooses the target user $u_{C'}$ with the highest probability with u_C (line 4). (u_C, $u_{C'}$) is two users that refer to one person (line 7). Thus, they will be removed (line 5 - line 6). HGM repeats these steps until finding a feasible solution.

Fig. 3 illustrates an example of HGM, in which each users pair is assigned a probability by decision tree. In HGM, the first step is calculate the information generation of each user, shown in the left in Fig. 3(a). Because user X generated the most information, and X' is the most proper candidate of X, as (X, X') shares the highest probability, (X, X') is the first linkage HGM selects. Then X and X' will be removed in both

Algorithm 2 Heuristic Greedy Matching for Identity Linkage.

```
Input:
    The set of users in communities $C$, $U_C$;
    The set of users in communities $C'$, $U_{C'}$;
    Probabilities between candidates across the sites,
    $\{(U_C, U_{C'}, P)\}$;
Output:
    Identity Linkage $IL$
1: Calculating the information generated by each candidate
   in $U_C$.
2: while $U_C \neq \emptyset$ do
3:     Selecting the candidate $u_C$ in $U_C$ with the most infor-
       mation generation;
4:     Selecting the candidate $u_{C'}$ in $U_{C'}$ with the largest
       probability with $u_C$;
5:     $U_C = U_C - \{u_C\}$
6:     $U_{C'} = U_{C'} - \{u_{C'}\}$
7:     $IL = IL \cup (u_C, u_{C'})$
8: end while
9: return $IL$;
```

communities before finding the next matching. Finally, Fig. 3 (b) shows all three linkages (X, X'), (Y, Y') and (Z, Z').

V. Experiments

In this section, we first present our experimental settings and then analyze the experiment results. We totally conduct four experiments to answer following questions: (1) why we select the Decision Tree method to assign the probabilities rather than SVM, Logistic Regression, etc.? (2) How much does three kinds of features contribute to the model? (3) Why do we choose the tag co-occurrence as the similarity measurement between tags rather than others? (4) How does our identity linkage method perform compared to other state-of-the-art methods?

A. Experimental Settings

We select two famous software communities, Github[9] and StackOverflow[10], to conduct the experiments. The data we use is all before January, 2015. To validate our approach, the first thing is to construct ground-truth data. Because in those two data sets, few users provide their profile urls, we consider that if the profile url of a user in StackOverflow and a user in Github both link to the same site, then they are the same person. However, we cannot simply link users by this way as in our statistics, there are less than 10% users providing their profile urls. In these 10% users, we total link 12,000 users, corresponding to 6,000 people, by profile urls. Therefore, we use these 6,000 people, which are divided into 5 groups, to adopt 5-folder cross validation to evaluate the performance of our method. The detailed information of these 6,000 people is illustrated in TABLE I. In TABLE I, there are 145,375 questions and answers posted in StackOverflow and 221,342 repositories created or attended by these 6,000 users. Among these data, there are 3,655 tags appearing in StackOverflow while only 67 in Github. Because there are far less labels in

[9]The data dumps of Github are in http://ghtorrent.org/downloads.html

[10]The data dumps of StackOverflow are in https://archive.org/details/stackexchange

TABLE I. DETAILED INFORMATION OF DATA SET

Site	Number of Documents	Documents per User	Number of Tags
StackOverflow	145,375	24.23	3,655
Github	221,342	36.890	67

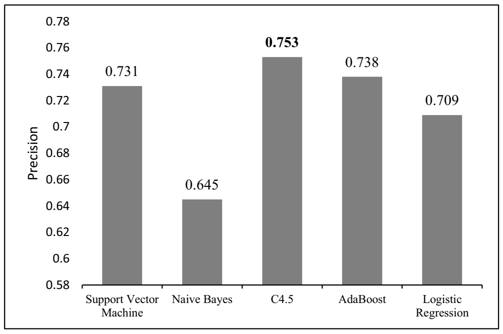

Fig. 4. The results of different algorithms when obtaining the probability

Github, we use two sets of tags, $L = L_{SO} \cap L_{GH}$ and L^* mentioned in Section IV. The number of labels in set L and L^* is 51 and 731 respectively.

Let IL_1 donate the set of true links and IL_2 donate the set of links found by our method. Because the linked step of our method is based on bigraph matching, $|IL_1| = |IL_2|$. As the definition of Precision in (4) and Recall in (5), Precision is equal to Recall. Therefore, we only list the Precisions when comparing how different features affect the result of our method, and list Precisions, Recalls and F-Scores (defined in (6)) when comparing with other methods.

$$Precision = \frac{|IL_1 \cap IL_2|}{|IL_2|} \tag{4}$$

$$Recall = \frac{|IL_1 \cap IL_2|}{|IL_1|} \tag{5}$$

$$F - Score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{6}$$

B. Obtaining Probability by Other Algorithms

Before linking users, we assign probabilities for each two users to represent how likely they refer to the same person by the Decision Tree (C4.5) algorithm. This experiment shows why we choose this algorithm rather than others. We first select other four famous classification algorithms, Support Vector Machine, Naive Bayes, AdaBoost and Logistic Regression to train four other different models, and then compare the final results of them. The results are illustrated in Fig. 4. In Fig. 4, the name of algorithm, used to construct corresponding model, is shown in the horizontal axis , while the values in vertical axis reflect the precision of corresponding model. According to Fig. 4, when these probabilities are assigned by C4.5 algorithm, the final precision is the highest, reaching 0.753, then the next is the results of AdaBoost, at 0.738. The worst situation appears in Naive Bayes, only reaching 0.645. Therefore, from the results of comparison, C4.5 algorithm is our best option.

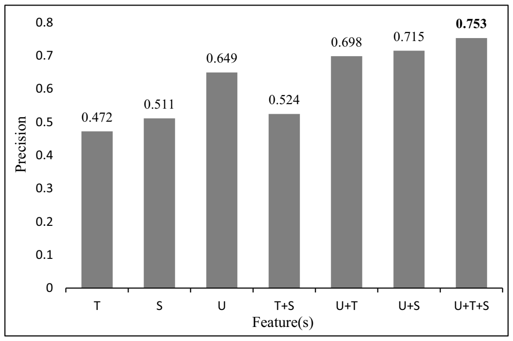

Fig. 5. The results of different models constructed by different feature(s)

C. Features' Contributions

We totally use three kinds of features (totally five features) to train the model, which are similarities between usernames (the ratio of Levenstein distance and the Jaro-Winkler distance), abbreviated by U, topics distributions, abbreviated by T, and two kinds of skills distributions (two sets of skills L^* and L), abbreviated by S. This experiment will show how much the three kinds of features donate to final results. Unlike linear classification algorithms like SVM or Logistic Regression, in which the linear weights of features reflect how importance they are, C4.5 algorithm calculates the Gain-ratio of a feature and then decides whether it should be split. We build several models with different features or feature combinations by C4.5 algorithms and then show the results of these models.

Fig. 5 illustrates how each feature or feature combination affects the model. In Fig. 5, the feature selection is shown in horizontal axis, while values in vertical axis reflect the precision of corresponding model. According to Fig. 5, when only one kind of feature is concerned, the model trained with usernames similarity features reaches the best result, at the precision of 0.649, about 0.15 larger than those trained with topics similarity feature or with skills similarity features. When two kinds of features are considered, model trained without usernames similarity features (T+S) is worse than those trained with usernames similarity features (U+T or U+S). Model trained with three kinds of features obtains the best performance among others. From the results, usernames similarity features are the most important, however, other two kinds of features also have positive contribution to final model. We consider this is because providing a username is a prerequisite for signing up an account, while users are not forced to post questions or answers in StackOverflow and to create repositories in the Github. Therefore, there are some people who do not generate any information in both sites. For these people, if we only use topics similarity or skills similarity as features, it is hard for model to do the classification; therefore, the only clue is the features obtaining from username similarity. This is why when considering the username similarity features, model can obtain a good performance.

D. Other Ways to Construct the Similarity Matrix

In our method, the conditional distribution $P(tag_i|tag_j)$ is used to measure the similarity between two tags. The

TABLE II. PRECISIONS OF OTHER WAYS TO CONSTRUCT THE SIMILARITY MATRIX

Method	**Co-occurrence**	Questions Similarity	Wikipedia Similarity
Precision	**0.753**	0.732	0.725

calculation of conditional distribution is based on tag co-occurrence. However, there exist other approaches to calculate the similarity between tags; for instance, we can compare the similarity between the descriptions of two tags in Wikipedia[11] or compare the text information of the all questions belonging to each tags. We conduct this experiment to compare in which way to construct similarity matrix can obtain the best result. When comparing the similarity between tags based on descriptions in Wikipedia or questions, we first preprocess these texts in the way mentioned in IV-A, and then calculate words *tf-idf* to construct vector model. Finally, we use KL-divergence (1) of two vectors to obtain how similar between two tags.

Based on these three different approaches, three different similarity matrixes are then constructed. The finally results are shown in TABLE II. It is clear from TABLE II that when constructing similarity matrix by co-occurrence, the model obtains the best result, with 0.753 precision; following by constructing matrix based on questions similarity, at 0.732. The worst results appears in the situation when constructing similarity matrix by Wikipedia similarity (0.725 precision). This can be explained by two reasons: (1) when we use vector model to calculate the similarity between two tags, the dimension of the vector is larger than 10,000 (each dimension is a word). Although we clean the stopwords before constructing vectors, there are still many unrelated words that we cannot filter. Due to these noises, these two kinds of similarity are not very accurate. (2) Tags of each question are labeled by the question asker, and these tags are also very concise without much noise. Therefore, constructing similarity matrix by tags co-occurrence can obtain a better result than constructing by other other two approaches.

E. Comparison with Other Methods

In this experiment, we compare our method with other two state-of-the-art methods, the simple method [9] and the Bird et al. method [8], to solve the identity linkage problem across software communities. As for the LSA-based method [7], we need to provide a dictionary that is a list of common names as input. Because it is too hard to collect the common names list, especially for common names in different country, we do not implement this method for comparison. In simple method and Bird et al. method, each user in a community is represented by a pair (username, email), while we cannot obtain the users' emails; therefore, we only use matching techniques mentioned to usernames in those methods between two pairs. On the other hand, in the last phase, our method links users by finding a matching with one-to-one constraints based on probabilities assigned by the classifier. However, the classifier can also provide the other kind of outputs, 0, which states that those two users do not refer to the same person, or 1, otherwise. This experiment also validates whether matching phase can improve the final performance or not.

[11]https://www.wikipedia.org/

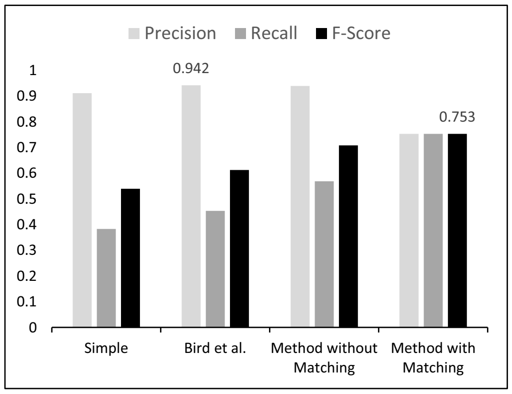

Fig. 6. The results of different algorithms when obtaining the probability

Fig. 6 illustrates the results of comparison. In Fig. 6, horizontal axis donates each method's name, in which Method without Matching is the method that uses 0-1 classification to link user and Method with Matching is the method that uses probabilities and then finds a matching. The values in vertical axis reflect the Precision, Recall and F-Score of corresponding method. From the Fig. 6, Simple method, Bird et al. method and Method without Matching show a common situation that they all achieve high precisions (around 0.94), but low Recalls and F-Scores, while despite the lowest Precision of Method with Matching, it achieves the highest Recall (0.753) and F-Score (0.753).

We explain this reason by an example illustrated in Fig. 7, in which the left side shows the user's username and corresponding skills in StackOverflow, while the right side shows those in Github. Because the username *Brandon* of the user in StackOverflow is similar to *Bradon*, *Brandon* and *Brand* in Github, method linking users only by usernames (Simple method and Bird et al. method) will generate three links. However, when the user's skills are concerned, obviously, *Brand* in Github is not the right link to *Brandon* in StackOverflow, as the skills between them have large difference. Therefore, Method without Matching will link *Bradon* and *Brandon* in Github to *Brandon* in StackOverflow. However, with one-to-one constraints, only *Brandon* in Github is linked to *Brandon* in StackOverflow, as this pairs shares a higher similarity than others. The reason why Simple method, Bird et al. method and Method without Matching can all obtain high precisions and low recalls is that they all create many redundant links compared to Method with Matching.

VI. CONCLUSION

In this paper, we have proposed an identity linkage method, TBIL, for linking users from different software communities. Two famous communities, StackOverflow and Github, are selected to validate the feasibility of our method. Differ to the previous methods in software communities and those in social networks, our method takes users' skills into account, extracting from tagging system in software communities. Besides, our method also considers similarities between usernames and those between user concerned topics. After extracting all these kinds of heterogeneous features, Our method uses a machine learning, Decision Tree, to assign a probability for each two

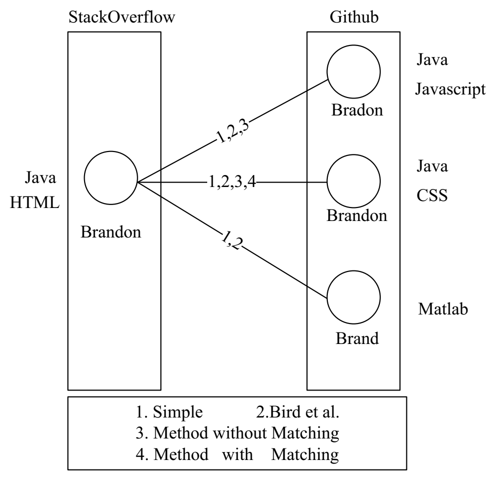

Fig. 7. Explain the Results of Different Identity Linkage Methods

users. Finally, a heuristic greedy algorithm is used to link users with one-to-one constraints.

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] Bogdan Vasilescu, Alexander Serebrenik, Prem Devanbu, and Vladimir Filkov. How social q&a sites are changing knowledge sharing in open source software communities. In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*, pages 342–354. ACM, 2014.

[2] Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. Stackoverflow and github: associations between software development and crowdsourced knowledge. In *Social Computing (SocialCom), 2013 International Conference on*, pages 188–195. IEEE, 2013.

[3] Gregory D Linden, Brent R Smith, Nida K Zada, Jonathan O Aizen, and Groffrey B Mack. Recommendations based on cross-site browsing activities of users, March 3 2008. US Patent App. 12/041,498.

[4] Siyuan Liu, Shuhui Wang, Feida Zhu, Jinbo Zhang, and Ramayya Krishnan. Hydra: Large-scale social identity linkage via heterogeneous behavior modeling. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 51–62. ACM, 2014.

[5] Yuxiao Dong, Jie Tang, Sen Wu, Jilei Tian, Nitesh V Chawla, Jinghai Rao, and Huanhuan Cao. Link prediction and recommendation across heterogeneous social networks. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 181–190. IEEE, 2012.

[6] Xiangnan Kong, Jiawei Zhang, and Philip S Yu. Inferring anchor links across multiple heterogeneous social networks. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 179–188. ACM, 2013.

[7] Erik Kouters, Bogdan Vasilescu, Alexander Serebrenik, and Mark GJ van den Brand. Who's who in gnome: Using lsa to merge software repository identities. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 592–595. IEEE, 2012.

[8] Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. Mining email social networks. In *Proceedings of the 2006 international workshop on Mining software repositories*, pages 137–143. ACM, 2006.

[9] Mathieu Goeminne and Tom Mens. A comparison of identity merge algorithms for software repositories. *Science of Computer Programming*, 78(8):971–986, 2013.

[10] Fabio Crestani. Application of spreading activation techniques in information retrieval. *Artificial Intelligence Review*, 11(6):453–482, 1997.

[11] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[12] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[13] Indrajit Bhattacharya and Lise Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):5, 2007.

[14] Thomas K Landauer and Susan T Dumais. A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological review*, 104(2):211, 1997.

[15] Tereza Iofciu, Peter Fankhauser, Fabian Abel, and Kerstin Bischoff. Identifying users across social tagging systems. In *ICWSM*, 2011.

[16] Jennifer Golbeck and Matthew Rothstein. Linking social networks on the web with foaf: A semantic web case study. In *AAAI*, volume 8, pages 1138–1143, 2008.

[17] Yanan Qian, Yunhua Hu, Jianling Cui, Qinghua Zheng, and Zaiqing Nie. Combining machine learning and human judgment in author disambiguation. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1241–1246. ACM, 2011.

[18] Anshu Malhotra, Luam Totti, Wagner Meira Jr, Ponnurangam Kumaraguru, and Virgilio Almeida. Studying user footprints in different online social networks. In *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*, pages 1065–1070. IEEE Computer Society, 2012.

[19] Jiawei Zhang and S Yu Philip. Multiple anonymized social networks alignment. *Network*, 3(3):6.

[20] Jing Liu, Fan Zhang, Xinying Song, Young-In Song, Chin-Yew Lin, and Hsiao-Wuen Hon. What's in a name?: an unsupervised approach to link users across communities. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 495–504. ACM, 2013.

[21] Börkur Sigurbjörnsson and Roelof Van Zwol. Flickr tag recommendation based on collective knowledge. In *Proceedings of the 17th international conference on World Wide Web*, pages 327–336. ACM, 2008.

[22] Jens Illig, Andreas Hotho, Robert Jäschke, and Gerd Stumme. A comparison of content-based tag recommendations in folksonomy systems. In *Knowledge Processing and Data Analysis*, pages 136–149. Springer, 2011.

[23] Eva Zangerle, Wolfgang Gassler, and Günther Specht. *Using tag recommendations to homogenize folksonomies in microblogging environments*. Springer, 2011.

[24] Robert Jäschke, Leandro Marinho, Andreas Hotho, Lars Schmidt-Thieme, and Gerd Stumme. Tag recommendations in folksonomies. In *Knowledge Discovery in Databases: PKDD 2007*, pages 506–514. Springer, 2007.

[25] Shaowei Wang, Daniel Lo, Bogdan Vasilescu, and Alexander Serebrenik. Entagrec: an enhanced tag recommendation system for software information sites. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, pages 291–300. IEEE, 2014.

[26] Xin Xia, David Lo, Xinyu Wang, and Bo Zhou. Tag recommendation in software information sites. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 287–296. IEEE Press, 2013.

[27] Shaowei Wang, Daniel Lo, and Lingxiao Jiang. Inferring semantically related software terms and their taxonomy by leveraging collaborative tagging. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 604–607. IEEE, 2012.

[28] Thomas L Griffiths and Mark Steyvers. Finding scientific topics. *Proceedings of the National Academy of Sciences*, 101(suppl 1):5228–5235, 2004.

[29] Irina Rish. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46. IBM New York, 2001.