Wang XY, Xia X, Lo D. TagCombine: Recommending tags to contents in software information sites. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 30(5): 1017–1035 Sept. 2015. DOI 10.1007/s11390-015-1578-2

TagCombine: Recommending Tags to Contents in Software Information Sites

Xin-Yu Wang [1] (王新宇), Xin Xia [1,*] (夏 鑫), *Member, CCF, ACM, IEEE*, and David Lo [2], *Member, ACM, IEEE*

[1] *College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China*
[2] *School of Information Systems, Singapore Management University, Singapore, Singapore*

E-mail: {wangxinyu, xxia}@zju.edu.cn; davidlo@smu.edu.sg

Received March 20, 2015; revised July 9, 2015.

Abstract Nowadays, software engineers use a variety of online media to search and become informed of new and interesting technologies, and to learn from and help one another. We refer to these kinds of online media which help software engineers improve their performance in software development, maintenance, and test processes as software information sites. In this paper, we propose TagCombine, an automatic tag recommendation method which analyzes objects in software information sites. TagCombine has three different components: 1) multi-label ranking component which considers tag recommendation as a multi-label learning problem; 2) similarity-based ranking component which recommends tags from similar objects; 3) tag-term based ranking component which considers the relationship between different terms and tags, and recommends tags after analyzing the terms in the objects. We evaluate TagCombine on four software information sites, Ask Different, Ask Ubuntu, Freecode, and Stack Overflow. On averaging across the four projects, TagCombine achieves recall@5 and recall@10 to 0.619 8 and 0.762 5 respectively, which improves TagRec proposed by Al-Kofahi *et al.* by 14.56% and 10.55% respectively, and the tag recommendation method proposed by Zangerle *et al.* by 12.08% and 8.16% respectively.

Keywords software information site, online media, tag recommendation

1 Introduction

Online media has changed the way people communicate, collaborate, and share information with one another. Online media is playing a more and more important role in the whole life cycle of software engineering[1-2]. There are various forms of online media that are regularly used by software engineers. Stack Overflow① is a popular Q&A (Question and Answer) site which focuses on technical questions about software development. SourceForge② and Freecode③ are two popular project information sites which allow users to share information about their projects. We refer to these kinds of online media which help software engineers to improve their performance in software development, maintenance, and test processes as software information sites.

In software information sites, tags are popular. They provide a form of metadata applied to software objects such as questions in Stack Overflow, projects in SourceForge and Freecode. They can be used to search, describe, identify, and bookmark various software objects. For software development, tags also help to bridge the gap between social and technical aspects[3-4].

Regular Paper
Special Section on Software Systems
This research was partially supported by China Knowledge Centre for Engineering Sciences and Technology under Grant No. CKCEST-2014-1-5, the National Key Technology Research and Development Program of the Ministry of Science and Technology of China under Grant Nos. 2015BAH17F01 and 2013BAH01B01, and the Fundamental Research Funds for the Central Universities of China.
A preliminary version of the paper was published in the Proceedings of MSR 2013.
*Corresponding Author
①http://stackoverflow.com/, July 2015.
②http://sourceforge.net/, July 2015.
③http://freecode.com/, July 2015.

Most software information sites allow users to tag various objects with their own words, and users increasingly use tags to describe the most important features of their posted contents or projects. The flexibility of tags makes them easy to use and tagging becomes popular among users. However, we notice that not all objects are well tagged. Some objects are not sufficiently tagged with descriptive words. Also, as tagging inherently is a distributed and uncoordinated process, some similar objects are tagged differently.

Some software information sites require users to add tags after they post an object. Selecting appropriate tags is not an easy task if users are not familiar with the site. If we could have a method which would recommend some tags according to the object a user posts and the previous tags of objects that other users have already posted, then the user could add appropriate tags easier, and the tag synonyms problem can also be avoided.

In this paper, we address the following research question: how to recommend appropriate tags for objects in software information sites? We propose TagCombine, which analyzes software objects in software information sites to improve the performance of tag recommendation. We mainly consider the text information in these software objects. TagCombine is a composite method, which has three different components: multi-label ranking component, similarity-based ranking component, and tag-term based ranking component. In multi-label ranking component, we consider the tag recommendation problem as a multi-label learning problem[5], where each tag maps to a label. We infer the appropriate label sets (tags) using multi-label learning algorithms, and rank the tags according to their likelihood scores. In similarity-based ranking component, we search similar software objects of the untagged objects, and recommend tags from the similar objects. In tag-term based ranking component, we first compute the affinity scores between tags and terms based on the historical tagged software objects. For an untagged object, we compute the ranking scores of various tags using the terms in the object and the pre-computed affinity scores.

We evaluate our solution on four software information sites, Ask Different, Ask Ubuntu, Freecode, and Stack Overflow, which contain 13 351, 37 354, 39 231, and 47 668 text documents, respectively, and 153, 346, 243, and 437 tags, respectively. Experimental results show that for Ask Different, our TagCombine achieves recall@5 and recall@10 scores of 0.674 9 and 0.821 4, respectively; for Ask Ubuntu, our TagCombine achieves recall@5 and recall@10 scores of 0.568 6 and 0.727 3, respectively; for Freecode, it achieves recall@5 and recall@10 scores of 0.639 1 and 0.777 3, respectively; for Stack Overflow, our TagCombine achieves recall@5 and recall@10 scores of 0.596 4 and 0.723 9, respectively. We compare our work with two similar work in the literature: 1) Al-Kofahi *et al.* proposed a tag recommendation system for software work item system such as IBM Jazz, which is based on fuzzy set theory[6]; 2) Zangerle *et al.* proposed a tag recommendation system for Twitter short messages, which recommends tags according to the tags of similar short messages[7]. We apply their tag recommendation systems to our problem. Averaging over Stack Overflow and Freecode results, for recall@5 and recall@10 scores, we improve TagRec proposed in [6] by 14.56% and 10.55% respectively, and the tag recommendation method proposed in [7] by 12.08% and 8.16% respectively.

This paper extends a preliminary study published as a research paper in a conference[8]. It extends the preliminary study in various ways: 1) we investigate several alternative algorithms of TagCombine which combine two of the three components, and evaluate their performance; 2) we add two more datasets, i.e., Ask Different and Ask Ubuntu, to further evaluate the performance of TagCombine; 3) we investigate the effect of TagCombine with respect to different evaluation criteria (ECs).

The main contributions of this paper are as follows.

1) There are limited studies on tag recommendation in the software engineering literature, especially for software information sites. Our research fills this gap.

2) We propose TagCombine, an accurate, automatic tag recommendation algorithm which analyzes tag recommendation problem from three different views, using three different components.

3) We evaluate TagCombine on four popular software information sites, Ask Different, Ask Ubuntu, Freecode, and Stack Overflow. The experimental results show that TagCombine achieves the best performance compared with state-of-the-art methods, i.e., TagRec and Zangerle *et al.*'s method[7].

The remainder of the paper is organized as follows. In Section 2, we present the background and elaborate the motivation of our work. In Section 3, we propose TagCombine, which contains three different components, to automatically recommend tags in software information sites. In Section 4, we report the results of

our experiment which compares TagCombine with the algorithms proposed by Al-Kofahi *et al.*[6] and Zangerle *et al.*[7] In Section 6, we present related studies. Finally, in Section 7, we conclude this paper and mention future work.

2 Background and Motivation

In this section, we first briefly introduce tags in software information sites, and then we elaborate the motivation of tag recommendation.

2.1 Tags in Software Information Sites

Tags are popular in software information sites. They are used as a form of metadata to describe the most important features of various software objects. Figs.1 and 2 present two software objects from two software information sites, Stack Overflow and Freecode, respectively.

In Fig.1, a user posts a question about string conversion in ASP.Net, which has three tags, i.e., "c#", "asp.net", and "null". These three tags describe the question in the following ways: "c#" and "asp.net" inform that the question is about the programming language C# and ASP.Net; "null" informs that the question is related to null. Terms "c#" and "null" both appear in the description of the question; "asp.net" does not appear in the description of the question, but developers who are familiar with C# can infer that the question is about ASP.Net.

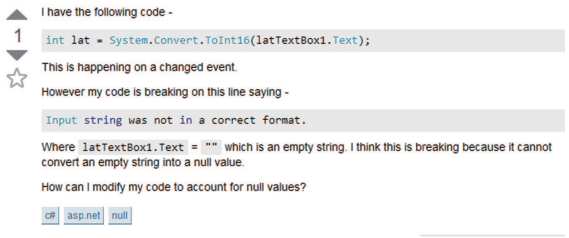


Fig.1. Question (ID = 14688802) posted in Stack Overflow about string conversion in ASP.Net.

In Fig.2, a user posts a project called "actionpoll", a simple PHP script which provides support for online voting. Four tags are given for this project, i.e., "Internet", "Web", "Dynamic Content", and "CGI Tools/Libraries". "Internet" and "Web" describe the environment that this project can be used; "Dynamic Content" describes the functionality of this project: it will capture dynamic content and analyze it; "CGI Tools/Libraries" describes the project type: it is a CGI tool and can also be used as a library. We notice that all of the four tags do not appear in the description of the project.

Fig.2. Project named "actionpoll " in Freecode.

From the above two examples, we conclude that tags help users to understand the software objects. They summarize the features of objects from different views, and users can search for appropriate objects more easily by using these tags. Tags are different from traditional keywords. Traditional keywords must appear in the object descriptions, but tags can either appear or not appear in the object descriptions.

2.2 Motivation

In this subsection, we present the motivation for automated tag recommendation in software information sites in three aspects: tag synonyms, easier posting, and better organization and search.

2.2.1 Tag Synonyms

Tag synonyms refer to tags which are syntactically different (i.e., they are different strings of symbols) but are semantically the same (i.e., they have the same meaning). For example, tags "zombie" and "zombies" both describe the zombie process in Unix; tags "xml-parser", "xml-parser", and "xmlparsing" all describe a parser of an xml file; tag "xsltproc" is an abbreviation of tag "xsltprocessor". Since there is no pre-defined tag vocabulary, and users can add tags arbitrarily, tag synonyms become an un-avoidable phenomenon in software information sites. Fig.3 presents the tag synonym page in Stack Overflow. We notice that this tag synonym list is currently maintained manually, which takes a lot of human resources.

Tag recommendation can help to avoid tag synonym phenomenon. For a new software object, tag recommendation will first learn the tags from historical objects. For synonymous tags, there will be a master

tag (the tag which more users would like to use). Tag recommendation will be likely to recommend this tag since it is supported by more training data. With the tag recommendation method employed, tag synonyms could be better avoided.

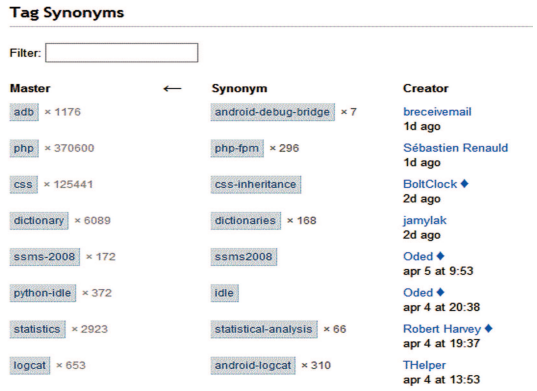


Fig.3. Tag synonyms in Stack Overflow.

2.2.2 *Easier Posting*

Some software information sites require users to add tags after they post an object. For example, in Stack Overflow, users are requested to add at least three tags after they submit a question. Choosing suitable tags is not an easy task, especially for new users. Some users just select terms in the object descriptions as the tags, but these terms might not represent the most important features of the object. There might be some latent tags which can better describe the object. From Figs.1 and 2, we note that some tags do not appear in the object descriptions. Tag recommendation makes the question posting process easier as it would recommend tags by mining historical software objects. The recommended tags could be either some of the terms in the object descriptions or some other latent terms.

2.2.3 *Better Organization and Search*

Software information sites use tags to organize the objects and help users to search for related objects in the community. For example, in Stack Overflow, users can search from the tags to see whether their question has already been posted and solved. However, the flexibility of tags (i.e., the fact that users can enter arbitrary tags) may negatively affect the organization of the information sites. For example, synonymous tags, or non-human-understandable tags are some causes of the problem. Different users would use different tags to describe a single thing. Some of the tags are much better than the others. If we could recommend high-quality tags, then the organization of the sites can be better, which would result in easier information search for end users.

3 TagCombine: A Composite Method

In this section, we first present the overall framework of our TagCombine method. Then we analyze the three components of TagCombine, i.e., multi-label ranking component, similarity-based ranking component, and tag-term based ranking component. Finally, we describe how these three components are combined.

3.1 Overall Framework

Fig.4 presents the overall framework of TagCombine. The whole framework contains two phases: model building phase and tag prediction phase. In the model building phase, our goal is to build a model from historical software objects which have known tags. In the tag prediction phase, this model would be used to predict tags for untagged software objects.

Our framework first collects historical software objects and their tags from software information sites. Then we pre-process the text information in these objects — tokenizing the text, removing stop words (e.g., "a", "the", "and", and "we"), stemming the terms, and filtering terms if their frequencies are less than a threshold (in this paper, by default, we remove terms which appear less than 20 times) (step 1). We represent these text contents of objects as "bags of words"[9].

Then we build the three components of TagCombine: multi-label ranking component, similarity-based ranking component, and tag-term based ranking component. To construct the multi-label ranking component, we first use a multi-label learning algorithm to build a multi-label classifier (step 2), and then we modify the classifier to output ranking scores for the tags given an unlabeled software object④ (step 3). To construct the similarity-based ranking component, we first transform the "bags-of-words" into TF.IDF (term frequency, inverse document frequency) vectors[9] (step 4). We calculate the similarity between two software objects by computing the cosine similarity of their TF.IDF vector representations⑤ (step 5). Next, we compute the tag-term affinity scores using historical tagged objects (step 6). We then use these affinity scores to rank

④More description is available in Subsection 3.2.

⑤More description is available in Subsection 3.3.

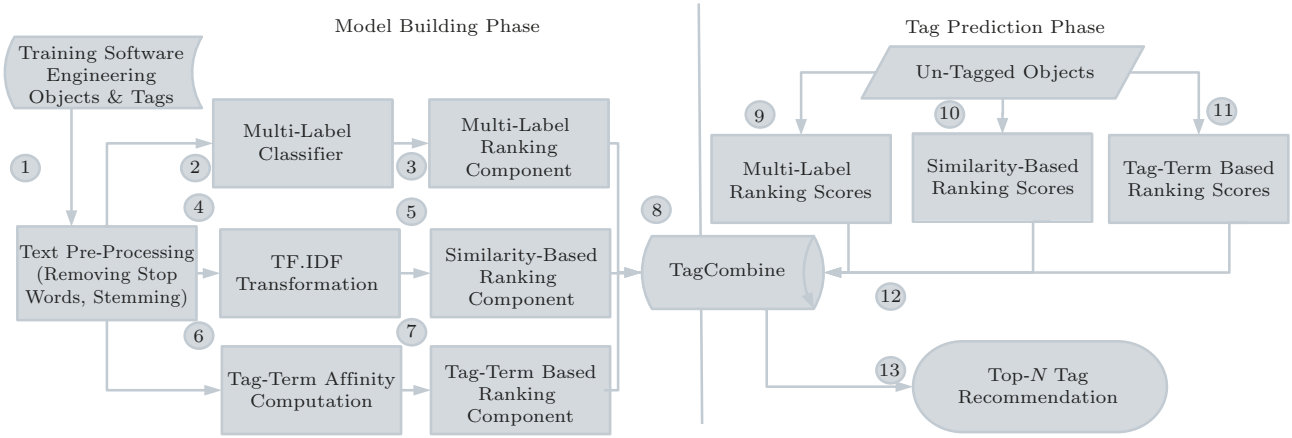


Fig.4. Overall framework of TagCombine.

tags for a given unlabeled software object⑥ (step 7). Finally, TagCombine uses these three components (step 8). It ranks tags based on the scores outputted by the three components.

After TagCombine is constructed, in the prediction phase, it is then used to recommend tags for a software object with unknown tags. For each such object, we first compute its multi-label ranking score, similarity-based ranking score, and tag-term based ranking score (steps 9∼11). We compute these scores using the three trained ranking components constructed at steps 3, 5, and 7. Then we input these scores into TagCombine to get the final ranking score for each tag (step 12). Finally, top-N ranked tags with the highest scores are recommended for the object (step 13).

3.2 Multi-Label Ranking Component

Formally, multi-label learning[5,10] can be defined as follows. Let χ denote the input space and let L denote the set of labels. Given the multi-label training dataset $D = \{(X_i, Y_i)\}_{i=1}^{n}$ where $X_i \in \chi$ and $Y_i = \{-1, 1\}^{|L|}$ ($Y_i = 1$ indicates that the instance is assigned the i-th label and $Y_i = -1$ indicates the instance is not assigned the i-th label), the goal of multi-label learning is to learn a hypothesis $h : \chi \to 2^{|L|}$ which is used to predict the label set for a new instance[5,10].

There are various multi-label learning algorithms, which can be divided into two categories: problem transformation methods and algorithm adaptation methods[5,10]. The problem transformation methods transform the multi-label learning task into multiple traditional classification tasks. Two popular problem transformation methods are Binary Relevance (BR) and Label Powerset (LP). The algorithm adaptation methods extend specific learning algorithms in order to handle multi-label data directly.

To adapt multi-label learning to our tag recommendation problem, we use the pre-processed term spaces in software objects as the input space χ, and the tags as the set of labels L. Multi-label learning predicts the proper label set for a new instance. We modify it such that our multi-label ranking component outputs the ranking scores for each tag, and these scores represent the confidence that a tag should be assigned to the object. Given an instance whose labels are to be predicted, multi-label learning algorithms compute likelihood scores for all the labels. If a label's likelihood score is higher than a threshold, then the multi-label learning algorithms would predict that this label belongs to the instance. We modify multi-label learning algorithms to directly output the likelihood scores. We then normalize the scores.

Definition 1 (Multi-Label Ranking Scores). *Considering a historical software object collection SE, and its corresponding tag space $TAGS$, we build a multi-label learning classifier $MultiLabel$ to train SE. For a new software object se, we use $MultiLabel$ to get the ranking score for each tag. We denote this ranking score as $MultiLabel_{se}(tag)$ for $tag \in TAGS$.*

We choose three state-of-the-art multi-label learning methods to construct the multi-label ranking component: Binary Relevance (BR) method, random k-labelsets (RAKEL)[11], and ML.KNN[12]. BR and RAKEL are problem transformation methods, and ML.KNN is an algorithm adaptation method.

⑥More description is available in Subsection 3.4.

3.2.1 Binary Relevance (*BR*)

Binary Relevance (BR) method creates $|L|$ binary datasets from the input dataset. Each of the $|L|$ binary datasets represents one label from L[5,10]. It assumes the tags in software objects are independent with one another; thus it is efficient enough for large tag and term spaces. We use multinomial naive Bayes as the base classifier for BR since it shows good performance for text classification and its computational complexity is low compared with other classification algorithms, c.f., [13]. We modify the implementation of Binary Relevance (BR) method in Mulan[14] to construct the multi-label ranking component.

3.2.2 Random k-Labelsets (*RAKEL*)

Random k-labelsets (RAKEL) is an extension of the Label Powerset (LP) method. In general, Label Powerset (LP) method treats each unique label set as a new single label, and then it applies a multi-class classification method to learn the suitable new single label for a software object. For example, given a set of labels L which contains a total of $|L|$ labels, there would be potentially $2^{|L|}$ unique label sets, which correspond to $2^{|L|}$ new single labels. By this way, LP transforms the multi-label learning problem into a multi-class classification problem, where an instance could only belong to one of the $2^{|L|}$ new single labels. One of the limitations of the LP method is that it would suffer from the label explosion problem, i.e., with the $2^{|L|}$ labels, the training set would become extremely sparse, which would cause underfitting problem[5,10].

Random k-labelsets (RAKEL) reduces the label explosion problem by constructing an ensemble of LP classifiers. The steps of using RAKEL in our framework are as follows.

1) In the model building phase, RAKEL randomly selects k labels (i.e., tags) from a total of L labels (i.e., tags), and constructs an LP classifier to output the label sets of the k labels by using a training set of software engineering objects with known tags. We repeat the process n times, and thus in total, RAKEL would build a set of n LP classifiers. For the i-th LP classifier, we denote it as LP_i.

2) In the tag prediction phase, for a new untagged software engineering object se, we apply each of the n LP classifiers to predict the labels for se. The multi-label ranking score for a label (i.e., tag) tag for se, denoted as $MultiLabel_{se}(tag)$, is defined as the number of the LP classifiers which predict that the label tag is assigned to se divided by n.

For example, suppose there are four labels (i.e., tags), i.e., $\{tag1, tag2, tag3, tag4\}$. We randomly select $k = 3$ labels and $n = 2$ times from the four labels to construct $n = 2$ LP classifiers, e.g., $\{tag1, tag2, tag3\}$ and $\{tag2, tag3, tag4\}$. For a new software engineering object se, let its predicted labels by the two LP classifiers be $\{tag2\}$ and $\{tag2, tag3, tag4\}$. Since between the two LP classifiers, only the second LP classifier predicts that $tag3$ is assigned to se, then $MultiLabel_{se}(tag3) = 1/2$. Since both of the two LP classifiers predict that $tag2$ is assigned to se, $MultiLabel_{se}(tag2) = 2/2 = 1$. Similarity, $MultiLabel_{se}(tag1) = 0$, and $MultiLabel_{se}(tag4) = 1/2$.

Similar to the BR method, RAKEL could also be paired with different base classifiers, such as KNN, multinomial naive Bayes, C4.8 decision tree, and SVM[15]. In this paper, we use multinomial naive Bayes as the base classifier — similar to the setting we use for the BR method presented in the previous subsection.

3.2.3 ML.KNN

ML.KNN is one of algorithm adaptation methods[16]. For a new software engineering object se, ML.KNN first gets its k-nearest neighbors $knn(se)$ from the training software engineering objects. For a label (i.e., tag) tag in the label set L, it would compute the number of instances (i.e., software engineering objects) in $knn(se)$ with the label tag. We denote the number of objects assigned label tag as $C_{se}(tag)$.

Next, based on the above count, ML.KNN computes the estimated probability of se to belong to the label tag (denoted as $H_1^{tag}(se)$) and the estimated probability of se to NOT belong to label tag (denoted as $H_0^{tag}(se)$). These two estimates do not necessarily sum up to 1. The above two estimated probabilities are computed for every label tag in the label set L. The multi-label ranking score for the label (i.e., tag) tag for se, denoted as $MultiLabel_{se}(tag)$, can be computed as:

$$MultiLabel_{se}(tag) = \frac{H_1^{tag}(se)}{H_1^{tag}(se) + H_0^{tag}(se)}.$$

3.3 Similarity-Based Ranking Component

We represent the tags of the i-th software object as $tagSet_i = \{t_{i,1}, t_{i,2}, ..., t_{i,l}\}$. The value of $t_{i,j}$ is either 0 or 1; $t_{i,j} = 1$ denotes that the j-th tag belongs to the i-th object, and $t_{i,j} = 0$ denotes the j-th tag does not belong to the i-th object. Following vector space modeling[9], we represent the text in the i-th software object as a vector of term weights denoted

by $\boldsymbol{se}_i = (w_{i,1}, w_{i,2}, ..., w_{i,v})$. The weight $w_{i,j}$ denotes the TF.IDF (i.e., term frequency.inverse document frequency) score for the j-th term in the i-th object, which is computed as follows:

$$w_{i,j} = \frac{tf_{i,j}}{\text{number of terms in } Obj_i} \times \log\left(\frac{\text{number of objects}}{df_j}\right).$$

In the above equation, Obj_i denotes the i-th object in the collection, $tf_{i,j}$ denotes the term frequency of the j-th term in the i-th object, df_i denotes the document frequency of the j-th term. Term frequency $tf_{i,j}$ refers to the number of times the j-th term appears in the i-th object. Document frequency of the j-th term refers to the number of objects in which the j-th term appears. We measure the similarity between two objects by computing the cosine similarity[9] of their vector representations $\boldsymbol{se}_m$ and $\boldsymbol{se}_n$ as follows:

$$SimScore(\boldsymbol{se}_m, \boldsymbol{se}_n) = \frac{\boldsymbol{se}_m \cdot \boldsymbol{se}_n}{|\boldsymbol{se}_m||\boldsymbol{se}_n|}. \quad (1)$$

More concretely, let $\boldsymbol{se}_m = (w_{m,1}, w_{m,2}, ..., w_{m,v})$, and $\boldsymbol{se}_n = (w_{n,1}, w_{n,2}, ..., w_{n,v})$. The numerator of (1) which is the dot product of the two vectors is computed as follows:

$$\boldsymbol{se}_m \cdot \boldsymbol{se}_n = w_{m,1} \times w_{n,1} + w_{m,2} \times w_{n,2} + ... + w_{m,v} \times w_{n,v}.$$

$|\boldsymbol{se}_m|$ and $|\boldsymbol{se}_n|$ in the denominator of (1), denote the sizes of the two vectors respectively. The size of a vector $\boldsymbol{se}_m$ is computed as follows:

$$|\boldsymbol{se}_m| = \sqrt{w_{m,1}^2 + w_{m,2}^2 + ... + w_{m,v}^2}.$$

The tag recommendation steps in the similarity-based ranking component are as follows:

1) represent each software object as a TF.IDF score vector;

2) for a new untagged software object se, use (1) to compute the similarity scores between se and other software objects se_{history} in historical data;

3) retrieve the top-K objects with the highest similarity scores. By default, we set $K = 50$. We extract the tags that appear in the top-K objects. For each of such tags, we compute the number of objects in the top-K list that are tagged by these objects; let us denote this count for tag t as $vote_t$. The likelihood of tag t, in tag space $TAGS$, belonging to se is then computed as follows:

$$\frac{vote_t}{\sum_{t' \in TAGS}(vote_{t'})}.$$

Definition 2 (Similarity-Based Ranking Scores). *Considering a historical software object collection SE, and its corresponding tag space $TAGS$, we build a similarity-based ranking classifier $SimRank$. For a new software object se, we use $SimRank$ to get the ranking score for each tag. We denote this ranking score as $SimRank_{se}(tag)$ for $tag \in TAGS$.*

3.4 Tag-Term Based Ranking Component

In tag-term based ranking component, we first consider the relationship between tags and terms. For each term and tag, the number of co-occurrences of the tag and term represents the importance of the term with respect to the tag. In this paper, we consider two different tag-term affinity scores: basic tag-term affinity score and fuzzy tag-term affinity score. Basic tag-term affinity score computes the fraction of the number of software engineering objects where the tag and the term both appear and the number of objects where the tag appears. Fuzzy tag-term affinity score leverages fuzzy set theory to compute the affinity score[6]. The definitions of these two tag-affinity scores are as follows.

Definition 3 (Basic Tag-Term Affinity Score). *Consider a historical software engineering object collection SE, and its corresponding tag space $TAGS$. For each tag $tag \in TAGS$, and term $t \in SE$, the basic tag-term affinity score of tag and t, denoted as $Aff^B(tag, t)$, is computed as follows:*

$$Aff^B(tag, t) = \frac{n_{t,tag}}{n_{tag}},$$

where $n_{t,tag}$ denotes the number of software objects where term t and tag tag both appear, and n_{tag} denotes the number of objects that tag appears.

Definition 4 (Fuzzy Tag-Term Affinity Score). *Consider a historical software engineering object collection SE, and its corresponding tag space $TAGS$. For each tag $tag \in TAGS$, and term $t \in SE$, the fuzzy tag-term affinity score of tag and t, denoted as $Aff^F(tag, t)$, is computed as follows:*

$$Aff^F(tag, t) = \frac{n_{t,tag}}{n_{tag} + n_t - n_{t,tag}}, \quad (2)$$

where $n_{t,tag}$ denotes the number of software objects where term t and tag tag both appear, n_{tag} denotes the number of objects that tag appears, and n_t denotes the number of objects that term t appears.

To illustrate the basic and fuzzy tag-term affinity scores, we take as an example the software engineering objects shown in Table 1, which have four terms and

three tags. In the table, the value 1 in a cell corresponding to the i-th object, the n-th term or the m-th tag means that the term appears in or the tag is assigned to the i-th software engineering object. On the other hand, 0 means the term does not appear in or the tag is not assigned to the object. The object with identifier "Test" is the object whose tags are to be predicted. For term 1 and tag 1, we notice that tag 1 appears in two objects (objects 1 and 3), i.e., $n_{tag_1} = 2$; term 1 appears in three objects (objects 1, 3 and 4), i.e., $n_{term_1} = 3$; and objects 1 and 3 have term 1 and tag 1, i.e., $n_{term_1,tag_1} = 2$. Thus, the basic tag-term affinity score for tag 1 and term 1 would be $n_{term_1,tag_1}/n_{tag_1} = 1$, and the fuzzy tag-term affinity score would be $n_{term_1,tag_1}/(n_{tag_1} + n_{term_1} - n_{term_1,tag_1}) = 0.67$. Similarly, the basic and fuzzy tag-term affinity scores for tag 1 and term 4 would be $n_{term_4,tag_1}/n_{tag_1} = 0$, and $n_{term_4,tag_1}/(n_{tag_1} + n_{term_4} - n_{term_4,tag_1}) = 0$, since $n_{term_4,tag_1} = 0$.

Table 1. Example of a Dataset with 5 Objects, 4 Terms and 3 Tags

Object ID	Term 1	Term 2	Term 3	Term 4	Tag 1	Tag 2	Tag 3
1	1	0	1	0	1	0	1
2	0	1	1	1	0	1	1
3	1	1	0	0	1	1	0
4	1	0	1	0	0	0	1
Test	1	0	0	1	?	?	?

Definition 5 (Tag-Term Based Ranking Scores). *Consider a historical software object collection SE, and its corresponding tag space $TAGS$. For a new software object se, we compute the tag-term based ranking score of tag $\in TAGS$, denoted as $TagTerm_{se}(tag)$, as follows:*

$$TagTerm_{se}(tag) = 1 - \prod_{t \in se}(1 - Aff^*(tag, t)).$$

In the above equation, $Aff^*(tag, t)$ could be the basic (i.e., $Aff^B(tag, t)$) or fuzzy (i.e., $Aff^F(tag, t)$) tag-term affinity score. We refer to a tag-term based ranking component which uses basic tag-term affinity score as basic tag-term based ranking component, and fuzzy tag-term affinity score as fuzzy tag-term based ranking component.

In Table 1, since the test object only has terms 1 and 4, its basic tag-term based ranking score for tag 1 would be:

$$TagTerm_{test}(tag_1) = 1 - (1 - 1) \times (1 - 0) = 1,$$

and the fuzzy tag-term based ranking score for tag 1 would be:

$$TagTerm_{test}(tag_1) = 1 - (1 - 0.67) \times (1 - 0) = 0.67.$$

3.5 TagCombine

As shown in Subsections 3.2, 3.3 and 3.4, we can get multi-label ranking scores, similarity-based ranking scores, and tag-term based ranking scores for a new software object se. In this subsection, we propose TagCombine, which is a method that combines all of the three components. A linear combination of the scores of the three components is used to compute the final TagCombine scores.

Definition 6 (TagCombine Scores). *Consider a new software object se, and a tag tag $\in TAGS$. The TagCombine score $TagCombine_{se}(tag)$ of tag tag with respect to object se is given by:*

$$\begin{aligned} &TagCombine_{se}(tag) \\ &= \alpha \times MultiLabel_{se}(tag) + \beta \times SimRank_{se}(tag) + \\ &\quad \gamma \times TagTerm_{se}(tag), \end{aligned}$$

where $\alpha \in [0, 1]$, $\beta \in [0, 1]$, and $\gamma \in [0, 1]$ represent the different contribution weights of multi-label ranking score, similarity-based ranking score, and tag-term based ranking score to the overall TagCombine score of tag, respectively.

To automatically produce good α, β, and γ weights for TagCombine, we propose a sample-based greedy method. Algorithm 1 presents the detailed algorithm to estimate good α, β, and γ weights. Due to the large size of historical software object collection SE, we do not use the whole collection to estimate α, β, and γ weights. Instead, we randomly sample a small subset of SE. In this paper, by default, we set the sample size as 10% of SE.

Algorithm 1 accepts input criterion EC as an input. We can set this input criterion EC as example-based recall@$k^{[5,10]}$ defined in Definition 7.

Definition 7 (Example-Based Recall@k). *Suppose there are m software objects. For each object se_i, let the actual tags be Tag_i. We recommend the top-k ranked tags $Rank_i$ for se_i according to our method. The example-based recall@k for the m software objects is given by:*

$$recall@k = \frac{1}{m}\sum_{i=1}^{m}\frac{|Rank_i \cap Tag_i|}{|Tag_i|}.$$

Algorithm 1. EstimateWeights(SE, $TAGS$, $SampleSize$, EC)

Inputs:
SE: historical software object collection
$TAGS$: tags space for SE
$SampleSize$: sample size
EC: evaluation criterion
Outputs: α, β, and γ
Method:
$\alpha = 0, \beta = 0, \gamma = 0$;
Build multi-label ranking component using SE;
Build similarity-based ranking component using SE;
Build tag-term based ranking component using SE;
Sample a small subset $Samp_{SE}$ of SE of size $SampleSize$;
for all object $se \in Samp_{SE}$ **do**
  for all tag $tag \in TAGS$ **do**
    Compute $MultiLabel_{se}(tag)$ according to Definition 1;
    Compute $SimRank_{se}(tag)$ according to Definition 2;
    Compute $TagTerm_{se}(tag)$ according to Definition 5;
  end for
end for
for all α from 0 to 1, every time increase α by 0.1 **do**
  for all β from 0 to 1, every time increase β by 0.1 **do**
    for all γ from 0 to 1, every time increase γ by 0.1 **do**
      for all object se in $Samp_{SE}$ **do**
        Compute $TagCombine_{se}(tag)$ according to Definition 6 ;
      end for
      Evaluate the effectiveness of the combined model based on EC;
    end for
  end for
end for
Return α, β, and γ which give the best result according to EC

For example, suppose there are two software objects, and three tags are given to the objects. For object 1, the actual tags are {1,2,3}, and for object 2, the actual tags are {1}. The top-2 ranked tags are {1,2} and {1,3} for objects 1 and 2, respectively. Then the example-based recall@2 is:

$$recall@2 = \frac{1}{2}\left(\frac{|\{1,2\} \cap \{1,2,3\}|}{|\{1,2,3\}|} + \frac{|\{1,3\} \cap \{1\}|}{|\{1\}|}\right) = \frac{1}{2}\left(\frac{2}{3} + \frac{1}{1}\right) = \frac{5}{6}.$$

4 Experiments and Results

We evaluate our TagCombine method on the Stack Overflow and Freecode. We compare our method with TagRec proposed in [6], and the tag recommendation method proposed in [7]. The experimental environment is a Windows 7 64-bit, Intel® Xeon® 2.53 GHz server with 24 GB RAM. We first present our experiment setup and four research questions (Subsection 4.1). We then present our experimental results that answer the four research questions (Subsections 4.2, 4.3, 4.4, and 4.5). Finally, we describe some threats to validity (Subsection 5.2).

4.1 Experimental Setup

Table 2 presents the statistics of the four datasets, i.e., Ask Different, Ask Ubuntu, Freecode, and Stack Overflow. The columns correspond to the number of objects collected (# Obj.), tags extracted from these objects (# Tags), and terms extracted from these objects (# Terms). After filtering the tags appearing less than 50 times and terms appearing less than 20 times,

Table 2. Statistics of Collected Software Objects in the 4 Software Information Sites

Community	# Obj.	# Tags	# Terms	# Final Obj.	# Final Tags	# Final Terms
Ask Different	14 196	984	5 965	13 351	153	1 816
Ask Ubuntu	39 354	2 064	21 313	37 354	346	3 206
Freecode	45 470	7 163	23 146	39 231	243	2 995
Stack Overflow	50 000	9 616	28 456	47 668	437	4 007

we get the final number of object (# Final Obj.), final tags extracted from the objects (# Final Tags), and final terms extracted from the objects (#Final Terms) which will be used to evaluate our TagCombine method.

For the Ask Different and the Ask Ubuntu datasets, we download their data dump files from Stack Exchange⑦. For the Freecode dataset, we use the same dataset used by [17]. For the Stack Overflow dataset, we parse the challenge data published in MSR 2013 mining challenge site⑧[18]. MSR challenge data contains Stack Overflow data from 2008 to 2012, and it is 12 GB in size. We extract the first 50 000 questions and their corresponding tags. These questions are originally posted between July 2008 and December 2008. We intentionally pick questions that have been published for a long time to ensure that the set of tags assigned to the questions has stabilized (i.e., no new tags are likely to be added).

We use WVTool[19] to extract terms from these software objects. WVTool is a flexible Java library for statistical language modeling, which is used to create word vector representations of text documents in the vector space model. We use WVTool to remove stop words, do stemming, and produce "bags-of-words" from the objects. We remove the terms which appear less than 20 times since we consider these terms do not have significantly contributions for tag recommendation. Moreover, we remove tags which appear less than 50 times in the collections since these tags are rare. Rare tags are less important and less useful to serve as representative tags to be recommended to users. There are not many people who use rare tags and thus recommending these tags does not help much to mitigate the synonym problem that is addressed in this paper. After we filter terms and tags, we get 13 351 objects, 153 tags, and 1 816 terms for Ask Different, 37 354 objects, 346 tags, and 3 206 terms for Ask Ubuntu, 39 231 objects, 243 tags and 2 995 terms for Freecode, and 47 668 objects, 437 tags, and 4 007 terms for Stack Overflow.

Stratified 10-fold cross validation is used to evaluate TagCombine, i.e., we randomly divide the dataset into 10 folds, and we use nine folds to train the model, while the remaining one fold is used to evaluate the performance. We repeat the process 10 times and compute the mean and the standard deviation. The distributions of tags in the training and test folds are the same as that of the original dataset. For the evaluation metric, we use recall@k described in Definition 7.

We reimplement the TagRec method proposed by Al-Kofahi *et al.*[6], and use it to recommend tags in tag space. For Zangerle *et al.*'s method[7], we implement the method with similarity metric called "SimRank", which was shown to achieve the best performance. We set the number of most similar objects to 50 which is the same setting as the similarity-based ranking component of TagCombine.

Notice that in TagCombine, we propose three multi-label methods (i.e., binary relevance (BR), random k-labelsets (RAKEL), and ML.KNN) to construct the multi-label ranking component, and two methods (basic tag-term affinity score and fuzzy tag-term affinity score) to construct the tag-term based ranking component. In total, there are six different versions of TagCombine. By default, we choose Binary Relevance (BR) method to construct the multi-label ranking component, and basic tag-term affinity score method to construct the tag-term based ranking component.

We are interested in answering the following research questions.

*RQ*1. How do recall@5 and recall@10 of TagCombine compare with those of TagRec and Zangerle *et al.*'s method?

*RQ*2. What is the effect of using the six different methods in the two components on the performance of TagCombine?

*RQ*3. What is the effect of varying the number K in the similarity-based ranking component on the performance of TagCombine?

*RQ*4. What is the effect of optimizing TagCombine with respect to different evaluation criteria (ECs) (see Algorithm 1)?

The first research question is the most important one. The answer would shed light on the effectiveness of our approach when compared with state-of-the-art solutions. In the subsequent research questions, we would like to investigate the effect of using different methods in the two components, varying the parameter K in the similarity-based ranking component, and employing different evaluation criteria (ECs) to the performance of TagCombine.

4.2 RQ1: Recall@k of TagCombine

Table 3 and Table 4 present the experimental results of the comparison of TagCombine with TagRec and Zangerle *et al.*'s method[7]. For Ask Different, Ask

⑦http://stackexchange.com/, Aug.2015.

⑧http://2013.msrconf.org/challenge.php, July 2015.

Ubuntu, Freecode, and Stack Overflow, recall@5 of TagCombine is 0.674 9, 0.568 6, 0.639 1, and 0.594 6 respectively, and recall@10 is 0.821 4, 0.727 3, 0.777 3, and 0.723 9, respectively.

From Table 3, the improvement of TagCombine over TagRec is substantial. Averaging over information sites considered, TagCombine outperforms TagRec by 14.56% and 10.55% for recall@5 and recall@10 values, respectively. For Freecode, TagCombine achieves the highest improvements of 32.05% and 17.72% over TagRec for recall@5 and recall@10, respectively.

From Table 4, the improvement of TagCombine over Zangerle *et al.*'s method is substantial. Averaging over information sites considered, TagCombine outperforms Zangerle *et al.*'s method by 12.08% and 8.16% for recall@5 and recall@10, respectively. For Stack Overflow, TagCombine achieves the highest improvements of 30.39%, 8.79% over Zangerle *et al.*'s method for recall@5 and recall@10, respectively. From these results, we conclude that our TagCombine improves TagRec more than Zangerle *et al.*'s method.

Moreover, we notice TagCombine improves recall@5 more than recall@10. An improvement in recall@5 is more important than that in recall@10 since users of TagCombine would be more focused on the top 5 rather than the top 10 for practical purposes. Averaging over techniques compared, the improvements on recall@5 are 6.80%, 6.85%, 19.33%, and 21.82% for Ask Different, Ask Ubuntu, Freecode, and Stack Overflow, respectively, while the improvements on recall@10 are 6.96%, 8.18%, 11.82%, and 10.49%, respectively.

4.3 RQ2: Effect of Different Combinations

We propose three methods to construct the multi-label ranking component, and two methods to construct the tag-term ranking based component. In total, we have six different versions of TagCombine. We denote the combination of BR and basic tag-term affinity score as default, and that of BR and fuzzy tag-term score as BR+Fuzzy. Similarly, we denote RAKEL, with basic and fuzzy tag-term affinity score as RAKEL+Basic and RAKEL+Fuzzy respectively. And we denote ML.KNN, with basic and fuzzy tag-term affinity score as ML.KNN+Basic and ML.KNN+Fuzzy respectively. Table 5 and Table 6 present the example-based recall@5 and recall@10 scores for the different versions of TagCombine.

We notice that the differences among the different versions of TagCombine are small. On average across the four datasets, the recall@5 and recall@10 scores of different versions of TagCombine vary from 0.590 2∼0.619 8, and 0.742 1∼0.763 5 respectively. Among the six combinations, the default version achieves the best recall@5 and recall@10 scores of 0.619 8 and 0.763 5 respectively. BR+Fuzzy achieves

Table 3. Example-Based Recall@5 and Recall@10 Comparison Between TagCombine and TagRec

Dataset		TagCombine	TagRec	Improvement(%)
Ask Different	Recall@5	0.674 9±0.007 7	0.648 4±0.005 2	**4.09**
	Recall@10	0.821 4±0.005 4	0.792 0±0.006 2	**3.71**
Ask Ubuntu	Recall@5	0.568 6±0.005 1	0.522 3±0.005 1	**8.86**
	Recall@10	0.727 3±0.006 2	0.669 7±0.004 2	**8.60**
Freecode	Recall@5	0.639 1±0.006 0	0.484 0±0.004 2	**32.05**
	Recall@10	0.777 3±0.005 0	0.660 3±0.003 2	**17.72**
Stack Overflow	Recall@5	0.594 6±0.003 4	0.525 1±0.003 9	**13.24**
	Recall@10	0.723 9±0.003 3	0.645 3±0.005 0	**12.18**

Note: The result is recorded with format: mean±standard deviation. These are the means and the standard deviations of the 10 iteration results of 10-fold cross-validation.

Table 4. Example-Based Recall@5 and Recall@10 Comparison Between TagCombine and Zangerle *et al.*'s Method

Dataset		TagCombine	Zangerle *et al.*	Improvement(%)
Ask Different	Recall@5	0.674 9±0.007 7	0.616 4±0.011 5	**9.50**
	Recall@10	0.821 4±0.005 4	0.745 4±0.010 5	**10.20**
Ask Ubuntu	Recall@5	0.568 6±0.005 1	0.542 4±0.009 2	**4.83**
	Recall@10	0.727 3±0.006 2	0.675 0±0.009 1	**7.75**
Freecode	Recall@5	0.639 1±0.006 0	0.599 5±0.004 8	**3.61**
	Recall@10	0.777 3±0.005 0	0.733 9±0.005 6	**5.91**
Stack Overflow	Recall@5	0.594 6±0.003 4	0.456 0±0.026 0	**30.39**
	Recall@10	0.723 9±0.003 3	0.665 4±0.006 0	**8.79**

Table 5. Example-Based Recall@5 for Different Versions of TagCombine

Combination	Ask Different	Ask Ubuntu	Freecode	Stack Overflow	Average
Default	0.674 9±0.007 7	0.568 6±0.005 1	0.639 1±0.006 0	0.594 6±0.003 4	**0.619 8**
BR+Fuzzy	0.658 9±0.008 6	0.549 8±0.005 6	0.609 6±0.006 1	0.542 6±0.005 3	0.590 2
RAKEL+Basic	0.684 9±0.007 0	0.578 4±0.006 2	0.634 4±0.005 4	0.560 1±0.007 7	0.614 5
RAKEL+Fuzzy	0.692 5±0.007 9	0.563 8±0.006 4	0.625 6±0.006 1	0.572 4±0.004 1	0.613 6
ML.KNN+Basic	0.683 3±0.007 1	0.590 5±0.008 6	0.622 0±0.004 7	0.524 0±0.007 3	0.605 0
ML.KNN+Fuzzy	0.701 5±0.007 4	0.569 1±0.007 9	0.630 0±0.004 9	0.542 6±0.005 3	0.610 8

Note: the last column (average) shows the average example-based recall@5 scores across the 4 datasets.

Table 6. Example-Based Recall@10 for Different Versions of TagCombine

Combination	Ask Different	Ask Ubuntu	Freecode	Stack Overflow	Average
Default	0.821 4±0.005 4	0.727 3±0.006 2	0.777 3±0.005 0	0.723 9±0.003 3	**0.762 5**
BR+Fuzzy	0.817 9±0.006 9	0.704 4±0.006 2	0.763 8±0.005 2	0.682 1±0.006 6	0.742 1
RAKEL+Basic	0.817 8±0.006 5	0.725 0±0.005 8	0.771 0±0.005 8	0.719 1±0.006 7	0.758 2
RAKEL+Fuzzy	0.830 0±0.005 4	0.714 0±0.005 5	0.763 5±0.005 6	0.694 2±0.005 5	0.750 4
ML.KNN+Basic	0.829 0±0.005 6	0.727 5±0.008 2	0.770 5±0.004 5	0.710 5±0.008 6	0.759 4
ML.KNN+Fuzzy	0.835 5±0.006 1	0.702 2±0.007 1	0.765 2±0.004 5	0.682 1±0.006 6	0.746 3

the poorest performance, with recall@5 and recall@10 scores of 0.590 2 and 0.742 1 respectively. Since the default version achieves the best performance, in the following two research questions, we only investigate the effect of varying various parameters on the default version of TagCombine.

Recall that in the default version of TagCombine, we use binary relevance (BR) to construct the multi-label ranking component, and use basic tag-term affinity score to construct the tag-term based ranking component. This default option is better than the other options due to the following reasons.

1) Compared with RAKEL and ML.KNN, binary relevance (BR) builds an independent ranking model for each tag, while RAKEL builds a number of ranking models based on a subset of all the tags, and ML.KNN recommends tags for an object by considering tags assigned to its k-nearest-neighbors. Binary relevance (BR) assumes the low level of dependencies among labels (aka.tags). We manually check the tag distributions in our collected data, and we find that the co-occurrences of many different pairs of labels are low, which indicates that there are few dependencies among many of the tags. For example, in the Stack Overflow data, the tags "java" and ".net" only appear 72 times while each of the tags appears 6 221 times and 5 063 times respectively, "c++" and "windows" only appear 54 times while each of the tags appears 2 650 times and 1 315 times respectively, and "C#" and "plugins" only appear once while each of the tags appears 6 217 times and 156 times respectively. Previous studies show that binary relevance (BR) will perform better if there is a low level of dependencies among the labels[5,10].

2) Comparing the basic tag-term affinity score formula and the fuzzy tag-term affinity score formula, we can note that the fuzzy tag-term affinity score formula considers the number of objects that term t appears, i.e., n_t. In our collected data, n_t is much larger than n_{tag} (the number of objects that tag tag appears) — since tags are sparsely distributed. Thus, for the $n_{tag} + n_t - n_{t,tag}$ term in the fuzzy tag-term affinity score formula (see (2)), n_t has the dominant effect. In such a case, the differences between fuzzy tag-term affinity scores are small since n_t dominates; this makes the fuzzy tag-term affinity score have less discriminative power and as a result, it does not perform as well as the basic tag-term affinity score.

To summarize, in practice, we recommend users to use the default version of TagCombine.

4.4 RQ3: Effect of Varying K

The similarity-based ranking component of TagCombine chooses K most similar objects. We would like to investigate how the effectiveness of TagCombine varies for various K values. In this subsection, we choose K in [5, 250] and compute recall@5 and recall@10 of TagCombine on the Ask Different, Ask Ubuntu, Stack Overflow, and Freecode datasets. Figs. 5∼8 present the experimental results of varying K in the similarity-based ranking component. For Ask Different, recall@5 and recall@10 vary from 0.304 1 to 0.697 2 and 0.477 0 to 0.833 3, respectively. For Ask Ubuntu, recall@5 and recall@10 vary from 0.370 6 to 0.576 6 and 0.558 4 to 0.729 1, respectively. For

Freecode, recall@5 and recall@10 vary from 0.578 1 to 0.641 6 and 0.735 3 to 0.777 3, respectively. For Stack Overflow, recall@5 and recall@10 vary from 0.569 7 to 0.597 8 and 0.695 2 to 0.728 5, respectively.

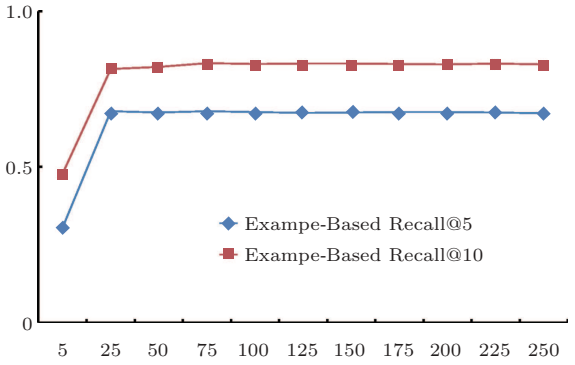


Fig.5. Example-based recall@5 and recall@10 for TagCombine with different K values ($K \in [5, 250]$) for the similarity-based ranking component when evaluated on the Ask Different project.

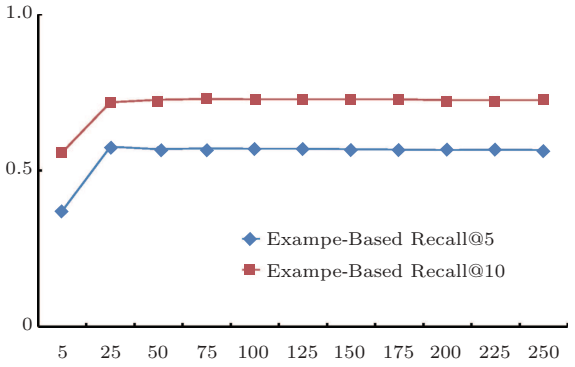


Fig.6. Example-based recall@5 and recall@10 for TagCombine with different K values ($K \in [5, 250]$) for the similarity-based ranking component when evaluated on the Ask Ubuntu project.

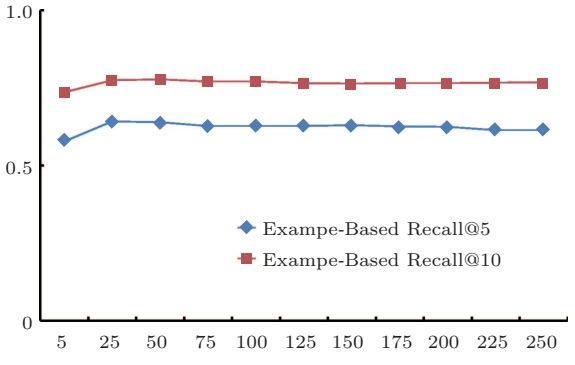


Fig.7. Example-based recall@5 and recall@10 for TagCombine with different K values ($K \in [5, 250]$) for the similarity-based ranking component when evaluated on the Freecode project.

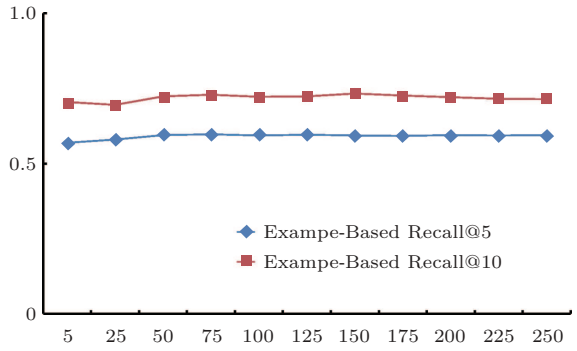


Fig.8. Example-based recall@5 and recall@10 for TagCombine with different K values ($K \in [5, 250]$) for the similarity-based ranking component when evaluated on the Stack Overflow project.

We notice that the performance of TagCombine with $K = 5$ achieves the worst performance. Since the collection is large (i.e., it contains about 40 000 objects), and the tags are imbalanced (i.e., for a particular tag, the ratio of the number of objects with the tag and the number of objects without the tag is small)[20], if we choose K values which are too small (e.g., $K = 5$), then the selected K most similar objects cannot represent the true distribution of relevant tags in the information site. For other K values (e.g., K in [25, 250]), TagCombine exhibits stable performance — the differences among different K values are small. For AskDifferent, Ask Ubuntu, and Stack Overflow, TagCombine achieves the best performance when K is set to 75. For Freecode, TagCombine achieves the best performance when K is set to 25. If we choose a large K value, the model building and prediction time will be increased. Thus, in practice, we recommend users to choose a K value in the interval of [25, 75].

4.5 RQ4: Effect of Optimizing with Different ECs

To investigate the effect of different ECs on the performance of TagCombine, we choose three ECs: recall@5, recall@10, and recall@20 evaluation criteria. Table 7 and Table 8 present example-based recall@5 and recall@10 for TagCombine with different ECs respectively. Recall@5 and recall@10 for Ask Different vary from 0.611 9 to 0.674 9 and 0.720 7 to 0.821 4, respectively. Recall@5 and recall@10 for Ask Ubuntu vary from 0.533 5 to 0.568 6 and 0.701 0 to 0.721 3, respectively. Recall@5 and recall@10 for Freecode vary from 0.628 6 to 0.639 1 and 0.768 4 to 0.777 3, respectively. Recall@5 and recall@10 for Stack Overflow vary from 0.538 9 to 0.594 6 and 0.715 4 to 0.725 3, respectively.

Table 7. Example-Based Recall@5 for TagCombine with Different Evaluation Criteria

EC	Ask Different	Ask Ubuntu	Freecode	Stack Overflow	Average
Recall@5	**0.674 9±0.007 7**	**0.568 6±0.005 1**	**0.639 1±0.006 0**	**0.594 6±0.003 4**	**0.619 8**
Recall@10	0.656 6±0.013 4	0.566 4±0.005 6	0.638 3±0.005 3	0.580 2±0.003 6	0.610 4
Recall@20	0.611 9±0.017 3	0.533 5±0.006 3	0.628 6±0.005 7	0.538 9±0.013 4	0.578 2

Table 8. Example-Based Recall@10 for TagCombine with Different Evaluation Criteria

EC	Ask Different	Ask Ubuntu	Freecode	Stack Overflow	Average
Recall@5	0.819 1±0.008 5	0.717 7±0.006 2	0.776 2±0.004 4	**0.725 3±0.002 7**	0.734 6
Recall@10	**0.821 4±0.005 4**	**0.727 3±0.006 2**	**0.777 3±0.005 0**	0.723 9±0.003 3	**0.762 5**
Recall@20	0.720 7±0.021 1	0.701 0±0.005 2	0.768 4±0.004 9	0.715 4±0.006 5	0.726 4

We notice that recall@20 EC achieves the worst performance compared with the other two ECs, i.e., recall@5 and recall@10. As stated in Algorithm 1, to estimate the best α, β, γ values, we use the training objects to select the best values which achieve the best EC values. Recall@20 EC selects too many tags, and in our experiment, we only focus on recall@5 and recall@10 values. For this reason, recall@20 EC does not perform well. Moreover, we notice that the differences in the performance of recall@5 and recall@10 ECs are small.

5 Discussion

5.1 How Can We Use the Tags Recommended by TagCombine?

Given a new question, we can use TagCombine to recommend a set of relevant tags. In this subsection, we would like to investigate how to use these recommended tags. Notice that in some software information sites such as Ask Different and Stack Overflow, a question can have a number of linked questions. For example, Fig.9 presents a question 45316 and its linked questions in Ask Different. Linked questions are similar questions that are manually identified by software information site users.

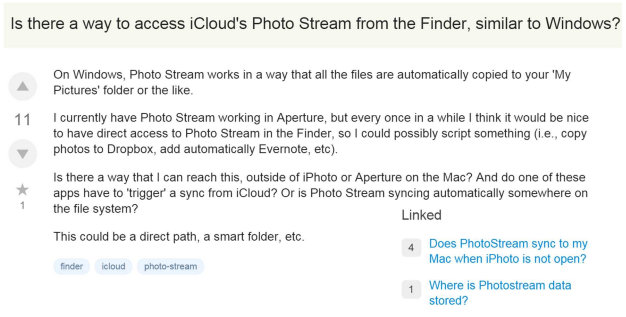


Fig.9. Question 45316 and its related questions in Ask Different.

Following advances in information retrieval, given a new question, we recommend its similar questions by computing the textual similarity (e.g., cosine similarity[9]) between the new question and the other questions, and then retrieve the top-K questions with the highest similarity scores to the new question. To compute textual similarity, we use terms in the textual contents of the questions such as words in the title and description fields of the questions. We refer to this approach as Sim^{Term}.

By leveraging TagCombine, we can recommend a set of tags to a new question. We would like to investigate whether we can improve the performance of similar question detection by using the additional recommended tags. To do this, given a new question *new*, we first use TagCombine to recommend a set of tags. In our study, we recommend five tags to the new question. Next, we merge these tags into the terms extracted from textual contents in the title and description fields of the new question. Then, we compute the textual similarity between the new question and other questions in the historical set of questions, and retrieve the top-K questions with the highest similarity scores to the new question. We refer to this approach as $\text{Sim}^{\text{TagCombine}}$. We also build a recommendation model which merges the ground truth tags of the new question with the terms extracted from textual contents, and refer to it as $\text{Sim}^{\text{Ideal}}$.

We compare $\text{Sim}^{\text{TagCombine}}$ with Sim^{Term} and $\text{Sim}^{\text{Ideal}}$. We extract 14 196 questions from AskDifferent, and sort them in chronological order of their creation time. We use the first 13 696 questions as the training set, and the remaining 500 questions as the test set. For each question in the test set, we crawl the website of Ask Different, and retrieve the questions in the "Related Question" section, and we use these questions as the ground truth. Notice that some questions do not

have the "Related Question" section, and we remove these questions from the test set.

To measure the performance of $\text{Sim}^{\text{TagCombine}}$, we use top-$K$ prediction accuracies, which follows some previous studies[21-24]. Top-K prediction accuracy is the percentage of questions in the test set where their ground truth similar questions are ranked in the top-k positions in the returned ranked lists of similar questions. In this paper, we set $K = 5$ and 10.

Table 9 presents the top 5 and the top 10 prediction accuracies for $\text{Sim}^{\text{TagCombine}}$ compared with Sim^{Term} and $\text{Sim}^{\text{Ideal}}$. $\text{Sim}^{\text{TagCombine}}$ achieves top-5 and top-10 prediction accuracies of 0.34 and 0.45 respectively, which is much better than Sim^{Term} which only considers terms in the textual contents. Moreover, we find $\text{Sim}^{\text{Ideal}}$ achieves a better performance than $\text{Sim}^{\text{TagCombine}}$. $\text{Sim}^{\text{Ideal}}$ is the ideal setting where all tags are right. The purpose of comparing $\text{Sim}^{\text{TagCombine}}$ with $\text{Sim}^{\text{Ideal}}$ is to measure the gap between $\text{Sim}^{\text{TagCombine}}$ and the ideal setting. In the future, we plan to further improve the tag recommendation approach to improve the performance of similar question detection.

Table 9. Top-5 and Top-10 Prediction Accuracies for $\text{Sim}^{\text{TagCombine}}$ Compared with Sim^{Term} and $\text{Sim}^{\text{Ideal}}$ for Ask Different Dataset

Approach	Top 5	Top 10
$\text{Sim}^{\text{TagCombine}}$	0.34	0.45
Sim^{Term}	0.27	0.38
$\text{Sim}^{\text{Ideal}}$	0.38	0.45

Besides linked question recommendation, the tags recommended by our TagCombine can potentially be used to solve other software engineering tasks. For example, we can potentially use the tags to improve the performance of duplicate question detection in Q&A web sites such as Stack Overflow, expert finding in Q&A web sites[20], and developer recommendation for a new project in Freecode or SourceForge[21]. In the future, we plan to solve these problems better by leveraging our TagCombine.

5.2 Threats to Validity

There are several threats that may potentially affect the validity of our study. Threats to internal validity relate to the errors in our experiments. We have double checked our experiments and datasets, and still there could be errors that we have not noticed. We use 10-fold cross validation[15] to evaluate the performance of our approach, which is a standard validation approach used in many previous studies[22-25].

Threats to external validity relate to the generalizability of our results. We have analyzed four popular software information sites and more than 149 000 software objects. Analyzing a large number of objects is important for the generalizability of the findings. Previous closely-related studies only investigate smaller numbers of objects[6-7]. We extend these studies by performing an evaluation based on a large number of objects. In the future, we plan to reduce this threat further by analyzing even more software objects from more software information sites, e.g., sites with a different type of user base (such as Twitter), and sites in other languages or cultures.

Threats to construct validity refer to the suitability of our evaluation measures. We use recall@5 and recall@10 as our evaluation measures. These are also used by previous studies to evaluate the effectiveness of tag recommendation[6-7,26]. Thus, we believe there is little threat to construct validity.

6 Related Work

In this section, we briefly review related studies. We first review TagRec and Zangerle *et al.*'s work[7] which are most related to our paper. We then describe studies on software information sites. Finally, we review studies on tagging in the software engineering literature.

6.1 Tag Recommendation

To our best knowledge, there is limited research on tag recommendation in the software engineering literature. TagRec is one of the most recent studies; it recommends tags in work item systems such as IBM Jazz[6]. The core technology of TagRec is based on fuzzy set theory. In this work, we consider a different problem setting, namely tag recommendation in software information sites. We have also applied TagRec in our setting and shown that our method could outperform it.

There are many tag recommendation studies in the social network and data mining fields[7,27-28]. They analyze social media sites such as Flickr, Delicious, and Twitter. The work by Zangerle *et al.* is one of the latest studies that recommend tags for short messages (aka. microblogs) in Twitter[7]. In this work, we consider a different setting, namely, the recommendation of tags in software information sites. We have applied Zangerle *et al.*'s method to our setting and shown that ours can outperform it.

6.2 Studies on Software Information Sites

A number of research studies have been performed on software information sites and social media for software engineering. Storey *et al.*[1] and Begel *et al.*[2] wrote two position papers to describe the outlook of research in social media for software engineering. They proposed a set of research questions around the impact of social media for software engineering at team, project, and community levels. Hong *et al.* compared developer social networks and general social networks and examined how developer social networks evolve over time[29]. Surian *et al.* employed graph mining and graph matching to find collaboration patterns in SourceForge.Net[30]. Surian *et al.* collected information in SourceForge.Net, and built a large-scale developer collaboration network to recommend suitable developers, using random walk with restart (RWR) method[21].

Bougie *et al.*[31] and Tian *et al.*[32] analyzed microblogs related to software engineering activities to understand what software engineers do in Twitter. They analyzed the contents of relevant short messages in Twitter, categorized the types of tweets, and found that Twitter is used by the software engineering community for conversation and information sharing. Achananuparp *et al.* created an observatory of software-related microblogs[33]. They created a web-based interface for people to browse many software-related microblogs and visually identify patterns. Prasetyo *et al.* proposed an automated technique to classify software related microblogs into several categories[34]. Pagano and Maalej analyzed the blogging behaviors of software developers in four large communities[35]. Gottipati *et al.* developed a semantic search engine to find relevant posts in software forums[36]. Henβ *et al.* extracted frequently asked questions from mailing lists and internet forums[37].

6.3 Tagging in Software Engineering Field

Treude and Storey analyzed tags in work item systems such as IBM Jazz, and they found that tags help to bridge the gap between social and technical aspects of software development[3-4]. In their study, the impact of tagging is investigated in a large project team with 175 developers over two years. They found that lightweight informal tool support such as tags, plays an important role in helping to improve team-based software development practices[3-4]. Wang *et al.* inferred semantically related software terms and their taxonomy by analyzing 45 470 projects along with their tags in Freecode[17]. They used a term taxonomy construction method which is based on k-medoids clustering algorithm. Thung *et al.* showed that tags are useful to detect similar applications[38]. In their study, they collected tags from SourceForge.Net, and performed weight inference to detect similar applications. A user study and three different metrics (i.e., success rate, confidence, and precision) were used to evaluate their proposed method.

7 Conclusions

In this paper, we proposed TagCombine, to recommend tags in software information sites. We first investigated the tags in software information sites, and considered the benefits of tag recommendation. Next, we proposed a framework named TagCombine, which contains three different components: multi-label ranking component, similarity-based ranking component, and tag-term based ranking component. In the multi-label ranking component, we inferred suitable tags for untagged objects using a multi-label learning algorithm. In the similarity-based ranking component, we recommended tags for untagged objects from the tags of similar objects. In the tag-term based ranking component, we first considered the affinity scores between tags and terms from historical data (i.e., existing tagged objects); for an untagged object, we recommended suitable tags based on the terms in the objects and the affinity scores. Finally, we proposed a sample-based method to combine the three components. We evaluated our method on four popular software information sites, Ask Different, Ask Ubuntu, Freecode, and Stack Overflow. Experimental results showed that for Ask Different, our TagCombine achieves recall@5 and recall@10 scores of 0.674 9 and 0.821 4, respectively; for Ask Ubuntu, our TagCombine achieves recall@5 and recall@10 scores of 0.568 6 and 0.727 3, respectively; for Freecode, it achieves recall@5 and recall@10 scores of 0.639 1 and 0.777 3, respectively; for Stack Overflow, our TagCombine achieves recall@5 and recall@10 scores of 0.596 4 and 0.723 9, respectively. For recommending tags in software information sites, averaging over information sites considered, for recall@5 and recall@10 scores, we improved TagRec proposed by Al-Kofahi *et al.*[6] by 14.56% and 10.55% respectively, and the tag recommendation method proposed by Zangerle *et al.*[7] by 12.08% and 8.16% respectively.

In the future, we plan to investigate more software information sites to evaluate the effectiveness of our method, develop a better technique which could achieve higher recall@5 and recall@10 scores, and consider more tags in tag space. We also plan to experiment with different algorithms to replace the various components of our framework. For our multi-label ranking component, various multi-label learning algorithms can be used to replace our proposed three methods, for example, LEAD[39], class chain method[40], could also be used. In the similarity-based ranking component, we can use different similarity metrics, for example, Euclidean distance, Minkowski distance[15]. We can also use latent semantic indexing (LSI)[41-42] instead of vector space model to cluster tags and terms to reduce tag synonymity in the similarity-based ranking component. In tag-term based ranking component, we can use other methods which consider the relationships between tags and terms to replace our proposed two methods. We used linear combination to tune parameters in the three components in this paper, and we can use other combination ways, e.g., we can perform principal component analysis (PCA)[43] to determine the relative contributions of each component. We plan to investigate these options as future work⑨.

Acknowledgment We would like to thank Wang *et al.* for sharing their Freecode dataset[17].

References

[1] Storey M, Treude C, Deursen A, Cheng L. The impact of social media on software engineering practices and tools. In *Proc. the FSE/SDP Workshop on Future of Software Engineering Research*, November 2010, pp.359-364.

[2] Begel A, DeLine R, Zimmermann T. Social media for software engineering. In *Proc. the FSE/SDP Workshop on Future of Software Engineering Research*, November 2010, pp.33-38.

[3] Treude C, Storey M. How tagging helps bridge the gap between social and technical aspects in software development. In *Proc. the 31st IEEE International Conference on Software Engineering* (*ICSE*), May 2009, pp.12-22.

[4] Treude C, Storey M A. Work item tagging: Communicating concerns in collaborative software development. *IEEE Transactions on Software Engineering*, 2012, 38(1): 19-34.

[5] Tsoumakas G, Katakis I. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining* (*IJDWM*), 2007, 3(3): 1-13.

[6] Al-Kofahi J, Tamrawi A, Nguyen T T, Nguyen H A, Nguyen T N. Fuzzy set approach for automatic tagging in evolving software. In *Proc. the 26th IEEE International Conference on Software Maintenance* (*ICSM*), September 2010.

[7] Zangerle E, Gassler W, Specht G. Using tag recommendations to homogenize folksonomies in microblogging environments. In *Proc. the 3rd Int. Conf. Social Informatics*, Oct. 2011, pp.113-126.

[8] Xia X, Lo D, Wang X, Zhou B. Tag recommendation in software information sites. In *Proc. the 10th Working Conference on Mining Software Repositories*, May 2013, pp.287-296.

[9] Baeza-Yates R A, Ribeiro-Neto B A. Modern Information Retrieval — The Concepts and Technology Behind Search (2nd edition). Boston, MA, USA: Addison-Wesley Publishing Company, 2011.

[10] Tsoumakas G, Katakis I, Vlahavas I. Mining multi-label data. In *Data Mining and Knowledge Discovery Handbook*, Maimon O, Rokach L (eds.), Springer US, 2010, pp.667-685.

[11] Tsoumakas G, Katakis I, Vlahavas I. Random k-labelsets for multilabel classification. *IEEE Transactions on Knowledge and Data Engineering*, 2011, 23(7): 1079-1089.

[12] Zhang M, Zhou Z. Multilabel neural networks with applications to functional genomics and text categorization. *IEEE Transactions on Knowledge and Data Engineering*, 2006, 18(10): 1338-1351.

[13] McCallum A, Nigam K. A comparison of event models for naive Bayes text classification. In *Proc. AAAI-98 Workshop on Learning for Text Categorization*, July 1998.

[14] Tsoumakas G, Spyromitros-Xioufis L, Vilcek J, Vlahavas I. MULAN: A Java library for multi-label learning. *Journal of Machine Learning Research*, 2011, 12: 2411-2414.

[15] Han J, Kamber M. Data Mining: Concepts and Techniques (2nd edition). San Francisco, CA, USA: Morgan Kaufmann, 2006.

[16] Zhang M, Zhou Z. ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognition*, 2007, 40(7): 2038-2048.

[17] Wang S, Lo D, Jiang L. Inferring semantically related software terms and their taxonomy by leveraging collaborative tagging. In *Proc. the 28th IEEE International Conference on Software Maintenance* (*ICSM*), September 2012, pp.604-607.

[18] Bacchelli A. Mining challenge 2013: Stack Overflow. In *Proc. the 10th Working Conference on Mining Software Repositories*, June 2013.

[19] Wurst M. The word vector tool: User guide. December 2007. http://wvtool.sf.net, Mar. 2015.

[20] Yang L, Qiu M, Gottipati S, Zhu F, Jiang J, Sun H, Chen Z. CQArank: Jointly model topics and expertise in community question answering. In *Proc. the 22nd ACM International Conference on Conference on Information & Knowledge Management*, October 27-November 1, 2013, pp.99-108.

[21] Surian D, Liu N, Lo D, Tong H, Lim E, Faloutsos C. Recommending people in developers' collaboration network. In *Proc. the 18th Working Conference on Reverse Engineering* (*WCRE*), Oct. 2011, pp.379-388.

⑨The source code and datasets of TagCombine can be downloaded from https://github.com/xinxia1986/Tag2, July 2015.

[22] Xia X, Lo D, Qiu W, Wang X, Zhou B. Automated configuration bug report prediction using text mining. In *Proc. the 38th IEEE Annual Computer Software and Applications Conference* (*COMPSAC*), July 2014, pp.107-116.

[23] Shihab E, Ihara A, Kamei Y, Ibrahim W M, Ohira M, Adams B, Hassan A E, Matsumoto K I. Predicting reopened bugs: A case study on the Eclipse project. In *Proc. the 17th Working Conference on Reverse Engineering* (*WCRE*), Oct. 2010, pp.249-258.

[24] Osman M H, Chaudron M R, van der Putten P. An analysis of machine learning algorithms for condensing reverse engineered class diagrams. In *Proc. the 29th IEEE International Conference on Software Maintenance* (*ICSM*), September 2013, pp.140-149.

[25] Xia X, Feng Y, Lo D, Chen Z, Wang X. Towards more accurate multi-label software behavior learning. In *Proc. the 2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering* (*CSMR-WCRE*), Feb. 2014, pp.134-143.

[26] Xia X, Lo D, Wang X, Zhou B. Accurate developer recommendation for bug resolution. In *Proc. the 20th Working Conference on Reverse Engineering* (*WCRE*), Oct. 2013, pp.72-81.

[27] Marlow C, Naaman M, Boyd D, Davis M. HT06, tagging paper, taxonomy, Flickr, academic article, to read. In *Proc. the 17th Conference on Hypertext and Hypermedia*, August 2006, pp.31-40.

[28] Sigurbjörnsson B, Van Zwol R. Flickr tag recommendation based on collective knowledge. In *Proc. the 17th International Conference on World Wide Web*, April 2008, pp.327-336.

[29] Hong Q, Kim S, Cheung S, Bird C. Understanding a developer social network and its evolution. In *Proc. the 27th IEEE International Conference on Software Maintenance* (*ICSM*), September 2011, pp.323-332.

[30] Surian D, Lo D, Lim E P. Mining collaboration patterns from a large developer network. In *Proc. the 17th Working Conference on Reverse Engineering* (*WCRE*), Oct. 2010, pp.269-273.

[31] Bougie G, Starke J, Storey M A, German D M. Towards understanding Twitter use in software engineering: Preliminary findings, ongoing challenges and future questions. In *Proc. the 2nd International Workshop on Web 2.0 for Software Engineering*, May 2011, pp.31-36.

[32] Tian Y, Achananuparp P, Lubis I, Lo D, Lim E. What does software engineering community microblog about? In *Proc. the 9th IEEE Working Conference on Mining Software Repositories* (*MSR*), June 2012, pp.247-250.

[33] Achananuparp P, Lubis I N, Tian Y, Lo D, Lim E P. Observatory of trends in software related microblogs. In *Proc. the 27th IEEE/ACM International Conference on Automated Software Engineering*, September 2012, pp.334-337.

[34] Prasetyo P K, Lo D, Achananuparp P, Tian Y, Lim E P. Automatic classification of software related microblogs. In *Proc. the 28th IEEE International Conference on Software Maintenance* (*ICSM*), September 2012, pp.596-599.

[35] Pagano D, Maalej W. How do developers blog?: An exploratory study. In *Proc. the 8th Working Conference on Mining Software Repositories*, May 2011, pp.123-132.

[36] Gottipati S, Lo D, Jiang J. Finding relevant answers in software forums. In *Proc. the 26th IEEE/ACM International Conference on Automated Software Engineering*, November 2011, pp.323-332.

[37] Henβ S, Monperrus M, Mezini M. Semi-automatically extracting FAQs to improve accessibility of software development knowledge. In *Proc. the 34th International Conference on Software Engineering*, June 2012, pp.793-803.

[38] Thung F, Lo D, Jiang L. Detecting similar applications with collaborative tagging. In *Proc. the 28th IEEE International Conference on Software Maintenance* (*ICSM*), September 2012, pp.600-603.

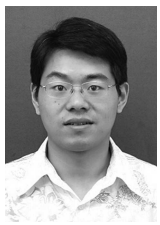
[39] Zhang M, Zhang K. Multi-label learning by exploiting label dependency. In *Proc. the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, July 2010, pp.999-1008.

[40] Read J, Pfahringer B, Holmes G, Frank E. Classifier chains for multi-label classification. *Machine Learning*, 2011, 85(3): 333-359.

[41] Deerwester S, Dumais S T, Furnas G W, Landauer T K, Harshman R. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 1990, 41(6): 391-407.

[42] Hofmann T. Probabilistic latent semantic indexing. In *Proc. the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, August 1999, pp.50-57.

[43] Jolliffe I. Principal Component Analysis. Springer-Verlag New York, 2002.



Xin-Yu Wang received his Bachelor's and Ph.D. degrees in computer science from Zhejiang University, Hangzhou, in 2002 and 2007 respectively. He was a research assistant in Zhejiang University during 2002∼2007. He is currently an associate professor in the College of Computer Science and Technology, Zhejiang University. His research interests include software engineering, formal methods, and very large information systems.

Xin Xia received his Ph.D. degree in computer science from the College of Computer Science and Technology, Zhejiang University, Hangzhou, in 2014. He is currently a research assistant professor in the College of Computer Science and Technology at Zhejiang University. His research interests include software analytic, empirical study, and mining software repository. He is a member of CCF, ACM, and IEEE.

David Lo received his Ph.D. degree in computer science from the School of Computing, National University of Singapore, Singapore, in 2008. He is currently an assistant professor in the School of Information Systems, Singapore Management University. He has close to 10 years of experience in software engineering and data mining research and has more than 130 publications in these areas. He received the Lee Foundation Fellow for Research Excellence from the Singapore Management University in 2009. He has won a number of research awards including an ACM Distinguished Paper Award for his work on bug report management. He has published in many top international conferences in software engineering, programming languages, data mining and databases, including ICSE, FSE, ASE, PLDI, KDD, WSDM, TKDE, ICDE, and VLDB. He has also served on the program committees of ICSE, ASE, KDD, VLDB, and many others. He is a steering committee member of the IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER) which is a merger of the two major conferences in software engineering, namely CSMR and WCRE. He will also serve as the general chair of ASE 2016. He is a leading researcher in the emerging field of software analytics and has been invited to give keynote speeches and lectures on the topic in many venues, such as the 2010 Workshop on Mining Unstructured Data, the 2013 Génie Logiciel Empirique Workshop, the 2014 International Summer School on Leading Edge Software Engineering, and the 2014 Estonian Summer School in Computer and Systems Science.