Tag Recommendation in Software Information Sites

Xin Xia*‡, David Lo†, Xinyu Wang*, and Bo Zhou*§
*College of Computer Science and Technology, Zhejiang University
{xxkidd,wangxinyu,bzhou}@zju.edu.cn
†School of Information Systems, Singapore Management University
davidlo@smu.edu.sg

***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*. It is common to see tags in software information sites and many sites allow users to tag various objects with their own words. Users increasingly use tags to describe the most important features of their posted contents or projects.**

In this paper, we propose *TagCombine*, an automatic tag recommendation method which analyzes objects in *software information sites*. *TagCombine* has 3 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 2 software information sites, StackOverflow and Freecode, which contain 47,668 and 39,231 text documents, respectively, and 437 and 243 tags, respectively. Experiment results show that for StackOverflow, our *TagCombine* achieves recall@5 and recall@10 scores of 0.5964 and 0.7239, respectively; For Freecode, it achieves recall@5 and recall@10 scores of 0.6391 and 0.7773, respectively. Moreover, averaging over StackOverflow and Freecode results, we improve TagRec proposed by Al-Kofahi et al. by 22.65% and 14.95%, and the tag recommendation method proposed by Zangerle et al. by 18.5% and 7.35% for recall@5 and recall@10 scores.

***Index Terms*—Software Information Sites, Online Media, Tag Recommendation, TagCombine**

I. 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 online media that are regularly used by software engineers. StackOverflow[1] is a popular Q&A site which focuses on technical questions about software development. SourceForge[2] and Freecode[3] 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 StackOverflow, projects in SourceForge and Freecode. They can be used to search, describe, identify, and bookmark various software objects. For software development, tags also helps to bridge the gap between social and technical aspects [3], [4]. 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 make them easy to use and tagging becomes popular among users. However, we noticed 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. For example, in StackOverflow, we noticed tags "zombie" and "zombies" both describe the zombie process in Unix; Tags "xmlparser", "xml-parser", and "xmlparsing" all describe a parser of an xml file; Tag "xsltproc" is an abbreviation of tag "xsltprocessor". We refer to this phenomenon as tag synonyms.

Some software information sites require users to add tags after they post an object. For example, in StackOverflow, users are requested to add at least 3 tags after they submit a question. 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 the appropriate tags easier, and the tag synonyms problem can also be avoided.

In this paper, we address the following research questions: 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 3 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,

‡The work was done while the author was visiting Singapore Management University.

§Corresponding author.

[1]http://stackoverflow.com/

[2]http://sourceforge.net/

[3]http://freecode.com/

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 2 software information sites, StackOverflow and Freecode, which contain 47,668 and 39,231 text documents, respectively, and 437 and 243 tags, respectively. Experiment results show that for StackOverflow, our *TagCombine* achieves recall@5 and recall@10 scores of 0.5964 and 0.7239, respectively; For Freecode, it achieves recall@5 and recall@10 scores of 0.6391 and 0.7773, respectively. We compare our work with two similar work in the literature: Al-Kofahi et al. propose a tag recommendation system for software work item system such as IBM Jazz, which is based on fuzzy set theory. Zangerle et al. propose a tag recommendation system for Twitter short messages, which recommend tags according to the tags of similar short messages. We apply their tag recommendation systems to our problem. Averaging over StackOverflow and Freecode results, we improve *TagRec* proposed by Al-Kofahi et al. [6] by 22.65% and 14.95%, and the tag recommendation method proposed by Zangerle et al. [7] by 18.5% and 7.35% for recall@5 and recall@10 scores.

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 3 different views, using 3 different components.
3) We evaluate *TagCombine* on 2 popular software information sites, StackOverflow and Freecode. The experiment results show that *TagCombine* achieves the best performance compared with other state-of-the-art methods, i.e., TagRec and Zangerle et al.'s methods.

The remainder of the paper is organized as follows: In Section II, we present the background and elaborate the motivation of our work. In Section III, we propose *TagCombine*, which contains 3 different components, to automatically recommend tags in software information sites. In Section IV, we report the results of our experiment which compares *TagCombine* with the algorithms proposed by Al-Kofahi et al. and Zangerle et al.. In Section V, we present related studies. Finally, in Section VI, we conclude and mention future work.

II. Background and Motivation

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

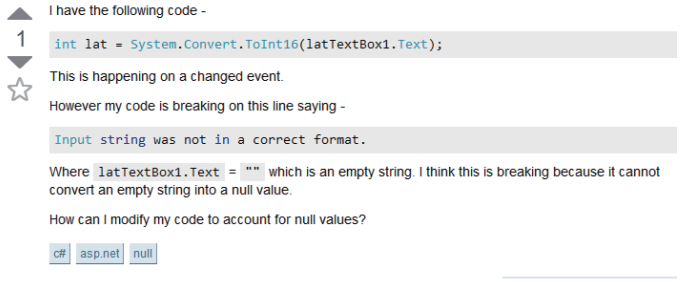

Fig. 1. A question (ID=14,688,802) posted in StackOverflow about string conversion in ASP.Net.

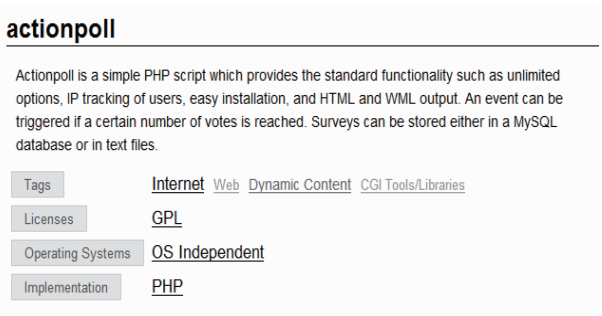

Fig. 2. A project named "actionpoll " in Freecode.

A. 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. Figures 1 and 2 present 2 software objects from 2 software information sites, StackOverflow and Freecode, respectively.

In Figure 1, a user posts a question about string conversion in ASP.Net, which has 3 tags, i.e., "c#", "asp.net", and "null". These 3 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. "c#" and "null" terms both appear in the description of the question, while "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.

In Figure 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 them; "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 4 tags do not appear in the description of the project.

From the above 2 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

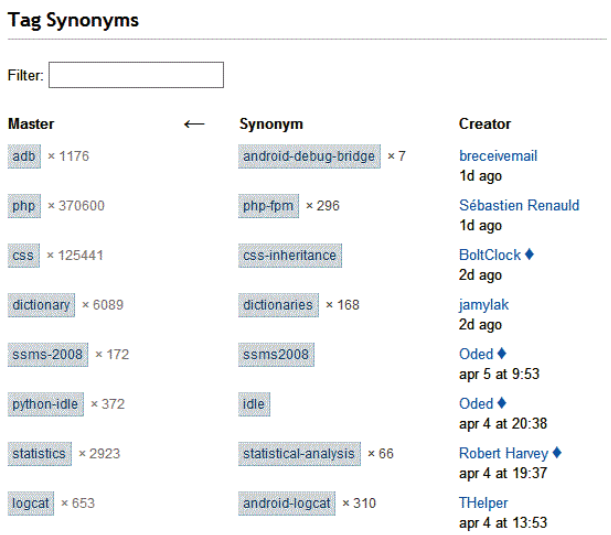

Fig. 3. Tag synonyms in StackOverflow.

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.

B. Motivation

In this section, we present the motivation for automated tag recommendation in software information sites in 3 aspects: tag synonyms, easier posting, and better organization and search.

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). Since there is no pre-defined tag vocabulary, and users can add tags arbitrarily, tag synonyms become an unavoidable phenomenon in software information sites. Figure 3 presents the tag synonym page in StackOverflow. 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 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.

2) Easier Posting: Some software information sites require users to add tags after they post an object. For example, in StackOverflow, users are requested to add at least 3 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 Figures 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 either be some of the terms in the object descriptions or some other latent terms.

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 StackOverflow, 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.

III. *TagCombine*: A Composite Method

In this section, we first present the overall framework of our *TagCombine* method. Then we analyze the 3 components of *TagCombine*, i.e., multi-label ranking component, similarity based ranking component, and tag-term based ranking component. Finally, we describe how these 3 components are combined.

A. Overall Framework

Figure 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", etc), stemming the terms, and filtering terms if their frequencies are less then 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" [8].

Then we build the 3 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), then we modify the classifier to output ranking scores for the tags given an unlabeled software object[4] (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 [8] (Step 4). We calculate the similarity between 2 software objects by computing cosine

[4]More description is available in Section III-B

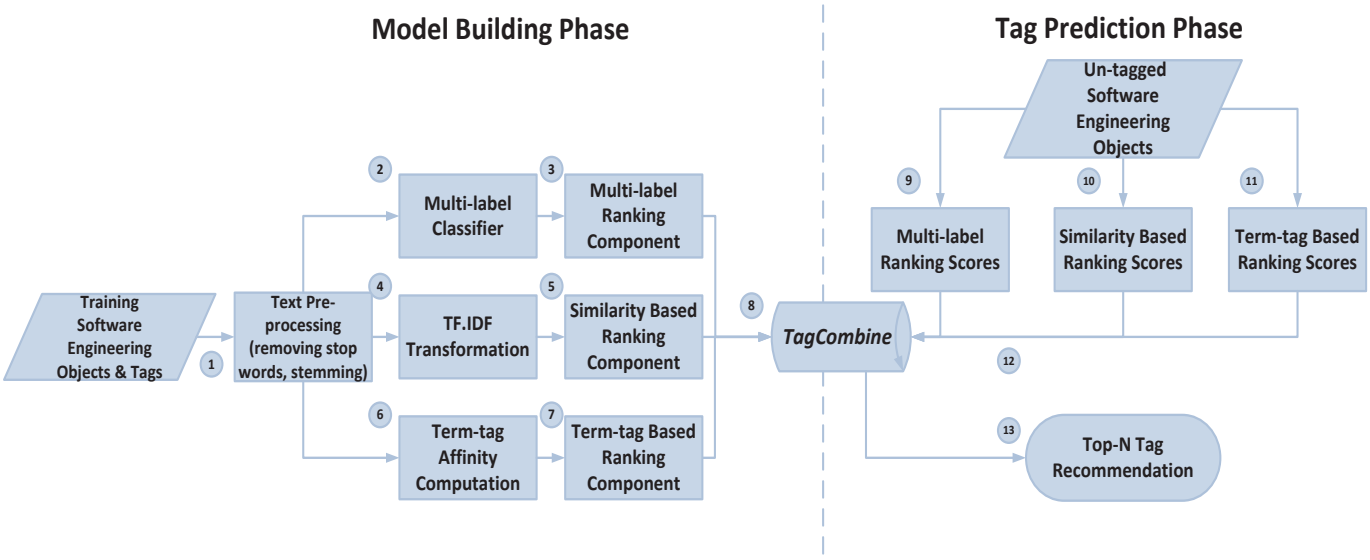

Fig. 4. Overall Framework of *TagCombine*

similarity of their TF.IDF vector representations[5](Step 5). Next, we compute the tag-term affinity scores using historical tagged objects (Step 6). We then use these affinity scores to rank tags for a given unlabeled software object[6] (Step 7). Finally, *TagCombine* uses these 3 components (Step 8). It ranks tags based on the scores outputted by the 3 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, 10, and 11). We compute these scores using the 3 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 highest scores are recommended for the object (Step 13).

B. Multi-label Ranking Component

Formally, multi-label learning [5], [9] 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 otherwise), the goal of multi-label learning is to learn a hypothesis $h : \chi \rightarrow 2^{|L|}$ which is used to predict the label set for a new instance.

There are various multi-label learning algorithms, which can be divided into 2 categories: problem transformation methods and algorithm adaptation methods [5], [9]. 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.

Due to the large tag and term spaces, we use Binary Relevance (BR) for the multi-label ranking component. Binary Relevance (BR) method creates $|L|$ binary datasets from the input dataset. Each of the $|L|$ binary datasets represents one label from L [5], [9]. 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 to other classification algorithms, c.f., [10]. We modify the implementation of Binary Relevance (BR) method in Mulan[7] [11] to construct the multi-label ranking component.

Definition 1: (**Multi-label Ranking Scores.**) Consider a historical software object collection SE, and its corresponding

[5]More description is available in Section III-C
[6]More description is available in Section III-D

[7]http://mulan.sourceforge.net/datasets.html

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 $tag \in TAGS$.

C. 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 otherwise. Following vector space modeling [8], we represent the text in the i-th software object as a vector of term weights denoted by $se_i = \langle w_{i,1}, w_{i,2}...w_{i,v}\rangle$. 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:

$$w_{i,j} = \frac{tf_{i,j}}{Num\ of\ Terms\ in\ Obj_i} \times log(\frac{Num\ of\ Objects}{df_j}) \quad (1)$$

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 the j-th term appears in. We measure the similarity between two objects by computing the cosine similarity [8] of their vector representations se_m and se_n as follows:

$$SimScore(se_m, se_n) = \frac{se_m \cdot se_n}{|se_m||se_n|} \quad (2)$$

More concretely, let $se_m = \langle w_{m,1}, w_{m,2}, ...w_{m,v}\rangle$, and $se_n = \langle w_{n,1}, w_{n,2}, ...w_{n,v}\rangle$. The numerator of Equation 2 which is the dot product of the two vectors is computed as follows:

$$se_m \cdot se_n = w_{m,1} \times w_{n,1} + w_{m,2} \times w_{n,2} + ... + w_{m,v} \times w_{n,v} \quad (3)$$

The $|se_n|$ and $|se_m|$ in the denominator of Equation 2, denote the sizes of the two vectors. The size of a vector se_m is computed as follows:

$$|se_m| = \sqrt{w_{m,1}^2 + w_{m,2}^2 + ... + w_{m,v}^2} \quad (4)$$

The tag recommendation steps in 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 Equation 2 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. Extract the tags that appear in the top-K objects. For each of such tags, compute the number of objects in the top-K list that are tagged by it; let's denote this count for tag t as $vote_t$.

The likelihood of tag t, in tag space $TAGS$, to belong to se is then computed as follows:

$$\frac{vote_t}{\sum_{t' \in TAGS}(vote_{t'})} \quad (5)$$

Definition 2: (**Similarity Based Ranking Scores.**) Consider 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 $tag \in TAGS$.

D. 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.

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

$$Aff(tag, t) = \frac{n_{t,tag}}{n_{tag}} \quad (6)$$

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: (**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:

$$TagTerm_{se}(tag) = 1 - \prod_{t \in se}(1 - Aff(tag, t)) \quad (7)$$

E. TagCombine

As shown in previous sections, we can get the multi-label ranking scores, similarity based ranking scores, and tag-term based ranking scores for a new software object se. In this section, we propose *TagCombine*, which is a method that combines all of the 3 components. A linear combination of the scores of 3 components is used to compute the final *TagCombine* scores.

Definition 5: (***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:

$$TagCombine_{se}(tag) = \alpha \times MultiLabel_{se}(tag) + \beta \times SimRank_{se}(tag) + \gamma \times TagTerm_{se}(tag) \quad (8)$$

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. Figure 5 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 the sample size as 10% of SE.

The algorithm described in Figure 5 accepts input criterion EC as an input. We can set this input criterion EC as example-based Recall@k [5], [9] defined in Definition 6.

Definition 6: (**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:

$$Recall@k = \frac{1}{m}\sum_{i=1}^{m}\frac{|Rank_i \cap Tag_i|}{|Tag_i|} \quad (9)$$

For example, suppose there are 2 software objects, and 3 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:

$$Recall@2 = \frac{1}{2}(\frac{|\{1,2\} \cap \{1,2,3\}|}{|\{1,2,3\}|} + \frac{|\{1,3\} \cap \{1\}|}{|\{1\}|}) = \frac{1}{2}(\frac{2}{3} + \frac{1}{1}) = \frac{5}{6}$$

IV. Experiments and Results

We evaluate our *TagCombine* method on the StackOverflow and Freecode. We compare our method with *TagRec* proposed by Al-Kofahi et al. [6], and the tag recommendation method proposed by Zangerle et al. [7]. The experimental environment is a Windows 7 64-bit, Intel(R) Xeon(R) 2.53GHz server with 24GB RAM.

A. Experiment Setup

Table I presents the statistics of the two datasets. For the StackOverflow dataset, we parse the challenge data published in MSR 2013 mining challenge site[8] [12]. MSR challenge data contains StackOverflow data from 2008 to 2012, and it is 12GB 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 have stabilized (i.e., no new tags are likely to be added). For the Freecode dataset, we use the same dataset used by Wang et al. [13].

We use WVTool[9] [14] 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 that use rare tags and thus recommending these tags do not help much in mitigating the synonym problem that is addressed in this paper. After we filter terms and tags, we get 47,668 objects, 437 tags, and 4,007 terms for StackOverflow, and 39,231 objects, 243 tags and 2,995 terms for Freecode.

Stratified 10-fold cross validation is used to evaluate *TagCombine*, i.e., we randomly divide the dataset into 10 folds, and 9 folds are use to train the model, while the remaining 1 fold is used to evaluate the performance. We repeat the process ten times and compute the mean and standard deviation. The distribution of tags in the training and test folds are the same as the original dataset. For the evaluation metric, we use recall@k described in Definition 6.

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 their 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*.

We are interested to answer the following research questions:

> **RQ1** How recall@5 and recall@10 of *TagCombine* compare to those of *TagRec* and Zangerle et al.'s method ?
> **RQ2** What is the effect of varying the number K in similarity based ranking component to the performance of *TagCombine*?

The first research question is the most important one. The answer would shed light on the effectiveness of our approach as compared to existing state-of-the-art solutions. In the second research question, we would like to investigate the effect of varying a parameter of *TagCombine* namely the parameter K of the similarity based ranking component.

B. RQ1: Recall@k of TagCombine

Table II and III present the experiment results comparing *TagCombine* with *TagRec* and Zangerle et al.'s method. For StackOverflow and Freecode, recall@5 of *TagCombine* are 0.5946, and 0.6391, respectively. Recall@10 are 0.7239 and 0.7773, respectively.

From Table II, the improvement of *TagCombine* over *TagRec* is significant. Averaging over information sites considered, *TagCombine* outperforms *TagRec* by 22.65% and 14.95% for recall@5 and recall@10 values, respectively. For Freecode, *TagCombine* achieves the highest improvements of

[8]http://2013.msrconf.org/challenge.php
[9]http://sourceforge.net/projects/wvtool/

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 4;
  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 5 ;
      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

Fig. 5. $EstimateWeights$: Estimation of α, β, and γ in *TagCombine*

TABLE I
STATISTICS OF COLLECTED SOFTWARE OBJECTS IN 2 INFORMATION SITES, STACKOVERFLOW AND FREECODE: THE NUMBER OF OBJECTS COLLECTED (# OBJECTS), 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, WE GET THE FINAL NUMBER OF OBJECT (# FINAL OBJECTS), 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.

Community	# Objects	# Tags	# Terms	# Final Objects	# Final Tags	# Final Terms
StackOverflow	50,000	9,616	28,456	47,668	437	4,007
Freecode	45,470	7,163	23,146	39,231	243	2,995

TABLE II
EXAMPLE-BASED RECALL@5 AND RECALL@10 FOR *TagCombine*, *TagRec*, AND THE IMPROVEMENT OF *TagCombine* OVER *TagRec*. THE RESULT IS RECORDED WITH FORMAT: MEAN±STANDARD DEVIATION. THESE ARE THE MEANS AND STANDARD DEVIATIONS OF THE 10 ITERATION RESULTS OF 10-FOLD CROSS-VALIDATION.

	StackOverflow		**Freecode**	
Algorithms	Recall@5	Recall@10	Recall@5	Recall@10
TagCombine	0.5946± 0.0034	0.7239± 0.0033	0.6391± 0.0060	0.7773± 0.0050
TagRec	0.5251± 0.0039	0.6453± 0.0050	0.4840± 0.0042	0.6603± 0.0032
Improvement Over *TagRec*	**13.24%**	**12.18%**	**32.05%**	**17.72%**

TABLE III
EXAMPLE-BASED RECALL@5 AND RECALL@10 FOR *TagCombine*, ZANGERLE ET AL.'S METHOD, AND THE IMPROVEMENT OF *TagCombine* OVER ZANGERLE ET AL.'S METHOD. THE RESULT IS RECORDED WITH FORMAT: MEAN±STANDARD DEVIATION. THESE ARE THE MEANS AND STANDARD DEVIATIONS OF THE 10 ITERATION RESULTS OF 10-FOLD CROSS-VALIDATION.

	StackOverflow		**Freecode**	
Algorithms	Recall@5	Recall@10	Recall@5	Recall@10
TagCombine	0.5946± 0.0034	0.7239± 0.0033	0.6391± 0.0060	0.7773± 0.0050
Zangerle et al.	0.4560± 0.0260	0.6654± 0.0060	0.5995± 0.0048	0.7339± 0.0056
Improvement Over *Zangerle*	**30.39%**	**8.79%**	**6.61%**	**5.91%**

32.05% and 17.72% over *TagRec* for recall@5 and recall@10, respectively.

From Table III, the improvement of *TagCombine* over Zangerle et al.'s method is significant. Averaging over information sites considered, *TagCombine* outperforms Zangerle et al.'s method by 18.5% and 7.35% for recall@5 and recall@10, respectively. For StackOverflow, *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.

Another interesting conclusion is *TagCombine* improves recall@5 more than recall@10. Averaging over techniques compared, the improvement on recall@5 are 21.82% and 19.33% for StackOverflow and Freecode, respectively, while the improvement on recall@10 are 10.49% and 11.82%, respectively.

C. RQ2: Effect of Varying of K

The similarity based ranking component of *TagCombine* chooses K most similar objects. We would like to investigate how the performance of *TagCombine* varies for various K values. In this section, we choose $K \in \{5, 25, 50, 75\}$ and compute recall@5 and recall@10 of *TagCombine* for StackOverflow and Freecode datasets. Table IV presents the experiment results of varying K in the similarity based ranking component. For StackOverflow, recall@5 and recall@10 vary from 0.5697-0.5978, and 0.6952-0.7285, respectively. For Freecode, recall@5 and recall@10 vary from 0.5781-0.6416, and 0.7353-0.7773, respectively.

We notice that the performance of *TagCombine* with K=5 achieves the worst performance. Since the collections 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) [15], if we choose K values which are too small (e.g., K=5), then the selected K most similar objects can not represent the true distribution of tags in the information site. For other K values (e.g., $K \in \{25, 50, 75\}$), *TagCombine* exhibits stable performance – the differences among different K values are small. For this reason, we should choose a K value which is big enough.

D. Threats to Validity

There are several threats that may potentially affect the validity of our study. Threats to internal validity relates to errors in our experiments. We have double checked our experiments and datasets, still there could be errors that we did not notice.

Threats to external validity relates to the generalizability of our results. We have analyzed 2 popular software information sites and more than 80,000 software 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 refers to the suitability of our evaluation measures. We use recall@5 and recall@10 as our evaluation measures. These are also used by past studies to evaluate the effectiveness of tag recommendation [6], [7]. Thus, we believe there is little threat to construct validity.

V. RELATED WORK

In this section, we briefly review related studies. We first review *TagRec* and Zangerle et al.'s work 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.

A. 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 showed that we could outperform it.

There are many tag recommendation studies in the social network and data mining fields [7], [16], [17]. They analyze social media sites such as Flickr, Delicious, and Twitter. The work by Zangerle et al. is one of the latest studies that recommends 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 we can outperform it.

B. 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] write two position

TABLE IV
EXAMPLE-BASED RECALL@5 AND RECALL@10 FOR *TagCombine* WITH DIFFERENT K VALUES ($K \in \{5, 25, 50, 75\}$) IN SIMILARITY BASED RANKING COMPONENT.THE DATA IS RECORDED WITH FORMAT OF MEAN±STANDARD DEVIATION OF THE RESULTS OF THE 10 ITERATIONS USING 10-FOLD CROSS-VALIDATION. THE BEST RESULTS IS IN BOLD.

	StackOverflow		**Freecode**	
TagCombine K	Recall@5	Recall@10	Recall@5	Recall@10
5	0.5697±0.0025	0.7053±0.0037	0.5781±0.0215	0.7353±0.0098
25	0.5801±0.0035	0.6952±0.0073	**0.6416±0.0066**	0.7752±0.0044
50	0.5946±0.0034	0.7239±0.0033	0.6391±0.0060	**0.7773±0.0050**
75	**0.5978±0.0033**	**0.7285±0.0030**	0.6275±0.0058	0.7703±0.0046

papers to describe the outlook of research in social media for software engineering. They propose a set of research questions around the impact of social media for software engineering at team, project and community levels. Hong et al. compare developer social networks and general social networks and examine how developer social networks evolve over time [18]. Surian et al. employ graph mining and graph matching to find collaboration patterns in SourceForge.Net [19]. Surian et al. collect information in SourceForge.Net, and build a large-scale developer collaboration network to recommend suitable developers, using random walk with restart (RWR) method [20].

Bougie et al., and Tian et al. analyze microblogs related to software engineering activities to understand what software engineers do in Twitter [21], [22]. They analyze the contents of relevant short messages in Twitter, categorize the types of tweets, and find that Twitter is used by the software engineering community for conversation and information sharing. Palakorn et al. create an observatory of software-related microblogs [23]. They create a web-based interface for people to browse many software-related microblogs and visually identify patterns. Prasetyo et al. propose an automated technique to classify software related microblogs into several categories [24]. Barua et al. use LDA to automatically detect the main topics in StackOverflow [25]. Pagano and Maalej analyze the blogging behaviors of software developers in four large communities [26]. Gottipati et al. develop a semantic search engine to find relevant posts in software forums [27]. Henβ et al. extract frequently asked questions from mailing lists and internet forums [28].

C. Tagging in Software Engineering Field

Treude et al. analyze tags in work item systems such as IBM Jazz, and they find 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 2 years. They find 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. infer semantically related software terms and their taxonomy by analyzing 45,470 projects along with their tags in Freecode [13]. They use a term taxonomy construction method which is based on k-medoids clustering algorithm. Thung et al. show that tags are useful to detect similar applications [29]. In their study, they collect tags from SourceForge.Net, and perform weight inference to detect similar applications. A user study and three different metrics (i.e., success rate, confidence, and precision) are used to evaluate their proposed method.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we propose *TagCombine*, to recommend tags in software information sites. We first investigate the tags in software information sites, and consider the benefits of tag recommendation. Next, we propose a framework named *TagCombine*, which contains 3 different components: multi-label ranking component, similarity based ranking component, and tag-terms based ranking component. In the multi-label ranking component, we infer suitable tags for untagged objects using a multi-label learning algorithm. In the similarity based ranking component, we recommend tags for untagged objects from the tags of similar objects. In the tag-term based ranking component, we first consider the affinity scores between tags and terms from historical data (i.e., existing tagged objects); for an untagged object, we recommend suitable tags based on the terms in the objects and the affinity scores. Finally, we propose a sample-based method to combine the 3 components. We evaluate our method on 2 popular software information sites, StackOverflow and Freecode. Experiment results show that for StackOverflow, our *TagCombine* achieves recall@5 and recall@10 scores of 0.5964 and 0.7239, respectively; For Freecode, it achieves recall@5 and recall@10 scores of 0.6391 and 0.7773, respectively. For recommending tags in software information sites, averaging over information sites considered, we improve TagRec proposed by Al-Kofahi et al. by 22.65% and 14.95%, and the tag recommendation method proposed by Zangerle et al. by 18.5% and 7.35% for recall@5 and recall@10 scores, 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 a 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 BR+MultiNaiveBayes method, for example, ML.kNN [30], LEAD [31], Random K-labelsets [32], class chain method [33], etc, could also be

used. In the similarity based ranking component, we can use different similarity metrics, for example, we can use Euclidean distance, Minkowski distance [34], etc. We can also use Latent Semantic Indexing (LSI) [35], [36] 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 method. We use linear combination to tune parameters in the 3 components in this paper, we can use other combination ways, e.g., we can perform a Principal Component Analysis (PCA) [37] to determine the relative contributions of each component. We plan to investigate these options as future work.

ACKNOWLEDGMENT

This research is sponsored in part by NSFC Program (No.61103032) and National Key Technology R&D Program of the Ministry of Science and Technology of China (No2013BAH01B01). We would also like to thank Wang et al. for sharing their Freecode dataset [13].

REFERENCES

[1] M. Storey, C. Treude, A. van Deursen, and L. Cheng, "The impact of social media on software engineering practices and tools," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*. ACM, 2010, pp. 359–364.

[2] A. Begel, R. DeLine, and T. Zimmermann, "Social media for software engineering," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*. ACM, 2010, pp. 33–38.

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

[4] C. Treude and M.-A. Storey, "Work item tagging: Communicating concerns in collaborative software development," *Software Engineering, IEEE Transactions on*, vol. 38, no. 1, pp. 19–34, 2012.

[5] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *International Journal of Data Warehousing and Mining (IJDWM)*, vol. 3, no. 3, pp. 1–13, 2007.

[6] J. Al-Kofahi, A. Tamrawi, T. Nguyen, H. Nguyen, and T. Nguyen, "Fuzzy set approach for automatic tagging in evolving software," in *Software Maintenance (ICSM), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–10.

[7] E. Zangerle, W. Gassler, and G. Specht, "Using tag recommendations to homogenize folksonomies in microblogging environments," *Social Informatics*, pp. 113–126, 2011.

[8] R. A. Baeza-Yates and B. A. Ribeiro-Neto, *Modern Information Retrieval - the concepts and technology behind search, Second edition*. Pearson Education Ltd., Harlow, England, 2011.

[9] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Mining multi-label data," *Data mining and knowledge discovery handbook*, pp. 667–685, 2010.

[10] A. McCallum, K. Nigam *et al.*, "A comparison of event models for naive bayes text classification," in *AAAI-98 workshop on learning for text categorization*, vol. 752. Citeseer, 1998, pp. 41–48.

[11] G. Tsoumakas, J. Vilcek, L. Spyromitros, and I. Vlahavas, "Mulan: A java library for multi-label learning," *Journal of Machine Learning Research*, vol. 1, pp. 1–48, 2010.

[12] A. Bacchelli, "Mining challenge 2013: Stack overflow," in *The 10th Working Conference on Mining Software Repositories*, 2013.

[13] S. Wang, D. Lo, and L. Jiang, "Inferring semantically related software terms and their taxonomy by leveraging collaborative tagging," in *Software Maintenance (ICSM), 2012 IEEE International Conference on*. IEEE.

[14] M. Wurst, "The word vector tool: User guide," *Operator Reference, Developer Tutorial, URL:http://wvtool. sf. net,[Access date: December 2007]*, 2007.

[15] H. He and E. Garcia, "Learning from imbalanced data," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 21, no. 9, pp. 1263–1284, 2009.

[16] C. Marlow, M. Naaman, D. Boyd, and M. Davis, "Ht06, tagging paper, taxonomy, flickr, academic article, to read," in *Proceedings of the seventeenth conference on Hypertext and hypermedia*. ACM, 2006, pp. 31–40.

[17] B. Sigurbjörnsson and R. Van Zwol, "Flickr tag recommendation based on collective knowledge," in *Proceedings of the 17th international conference on World Wide Web*. ACM, 2008, pp. 327–336.

[18] Q. Hong, S. Kim, S. Cheung, and C. Bird, "Understanding a developer social network and its evolution," in *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*. IEEE, 2011, pp. 323–332.

[19] D. Surian, D. Lo, and E.-P. Lim, "Mining collaboration patterns from a large developer network," in *Reverse Engineering (WCRE), 2010 17th Working Conference on*. IEEE, 2010, pp. 269–273.

[20] D. Surian, N. Liu, D. Lo, H. Tong, E. Lim, and C. Faloutsos, "Recommending people in developers' collaboration network," in *Reverse Engineering (WCRE), 2011 18th Working Conference on*. IEEE, 2011, pp. 379–388.

[21] G. Bougie, J. Starke, M.-A. Storey, and D. M. German, "Towards understanding twitter use in software engineering: preliminary findings, ongoing challenges and future questions," in *Proceedings of the 2nd international workshop on Web 2.0 for software engineering*. ACM, 2011, pp. 31–36.

[22] Y. Tian, P. Achananuparp, I. Lubis, D. Lo, and E. Lim, "What does software engineering community microblog about?" in *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*. IEEE, 2012, pp. 247–250.

[23] A. Palakorn, N. L. Ibrahim, Y. Tian, D. LO, and E. P. LIM, "Observatory of trends in software related microblogs." 27th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2012.

[24] P. K. Prasetyo, D. Lo, P. Achananuparp, Y. Tian, and E.-P. Lim, "Automatic classification of software related microblogs," in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 2012, pp. 596–599.

[25] A. Barua, S. Thomas, and A. Hassan, "What are developers talking about? an analysis of topics and trends in stack overflow," *Empirical Software Engineering*, pp. 1–36, 2012.

[26] D. Pagano and W. Maalej, "How do developers blog?: an exploratory study," in *Proceedings of the 8th working conference on Mining software repositories*. ACM, 2011, pp. 123–132.

[27] S. Gottipati, D. Lo, and J. Jiang, "Finding relevant answers in software forums," in *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 2011, pp. 323–332.

[28] S. Henβ, M. Monperrus, and M. Mezini, "Semi-automatically extracting faqs to improve accessibility of software development knowledge," in *Proceedings of the 2012 International Conference on Software Engineering*. IEEE Press, 2012, pp. 793–803.

[29] F. Thung, D. Lo, and L. Jiang, "Detecting similar applications with collaborative tagging," in *Software Maintenance (ICSM), 2012 IEEE International Conference on*. IEEE.

[30] M. Zhang and Z. Zhou, "Ml-knn: A lazy learning approach to multi-label learning," *Pattern Recognition*, vol. 40, no. 7, pp. 2038–2048, 2007.

[31] M. Zhang and K. Zhang, "Multi-label learning by exploiting label dependency," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2010, pp. 999–1008.

[32] G. Tsoumakas and I. Vlahavas, "Random k-labelsets: An ensemble method for multilabel classification," in *Machine Learning: ECML 2007*. Springer, 2007, pp. 406–417.

[33] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," *Machine Learning*, pp. 1–27, 2011.

[34] J. Han and M. Kamber, *Data mining: concepts and techniques*. Morgan Kaufmann, 2006.

[35] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American society for information science*, vol. 41, no. 6, pp. 391–407, 1990.

[36] T. Hofmann, "Probabilistic latent semantic indexing," in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 1999, pp. 50–57.

[37] I. T. Jolliffe, *Principal component analysis*. Springer verlag, 2002.