Scalable Tag Recommendation for Software Information Sites

Pingyi Zhou*, Jin Liu**, Zijiang Yang***, and Guangyou Zhou†

* State Key Lab. of Software Engineering, Computer School, Wuhan University, China
** Department of Computer Science, Western Michigan University, Kalamazoo, Michigan, USA
*** School of Computer, Central China Normal University, Wuhan, China
† Department of Computer Science, Western Michigan University, Kalamazoo, Michigan, USA
{zhou_pinyi, jinliu}@whu.edu.cn, zijiang.yang@wmich.edu, gyzhou@mail.ccnu.edu.cn

Abstract—Software developers can search, share and learn development experience, solutions, bug fixes and open source projects in software information sites such as StackOverflow and Freecode. Many software information sites rely on tags to classify their contents, i.e., software objects, in order to improve the performance and accuracy of various operations on the sites. The quality of tags thus has a significant impact on the usefulness of these sites. High quality tags are expected to be concise and can describe the most important features of the software objects.

Unfortunately tagging is inherently an uncoordinated process. The choice of tags made by individual software developers is dependent not only on a developer's understanding of the software object but also on the developer’s English skills and preferences. As a result, the number of different tags grows rapidly along with continuous addition of software objects. With thousands of different tags, many of which introduce noise, software objects become poorly classified. Such phenomenon affects negatively the speed and accuracy of developers’ queries.

In this paper, we propose a tool called TagMulRec to automatically recommend tags and classify software objects in evolving large-scale software information sites. Given a new software object, TagMulRec locates the software objects that are semantically similar to the new one and exploit their tags. We have evaluated TagMulRec on four software information sites, StackOverflow, AskUbuntu, AskDifferent and Freecode. According to our empirical study, TagMulRec is not only accurate but also scalable that can handle a large-scale software information site with millions of software objects and thousands of tags.

Index Terms—Software Information Site, Tag Recomendation, Software Object, Multi-Classification

I. INTRODUCTION

Software information sites [1, 2, 3] offer indispensable platforms for software developers to search solutions, share experience, offer help and learn new techniques [4], [5], [6], [7], [8]. These sites include online developer Q&A communities [9], such as StackOverflow1, AskUbuntu2, AskDifferent3, and open source software community [10], such as Freecode4, GitHub5. The contents posted on these software information sites, such as a question with answers in a developer Q&A community and a project in an open source software community, are regarded as software objects [1, 2]. As the software information sites evolve, the number of software objects grow significantly, which makes it a very difficult for software developers to locate a particular software object [11], [12]. To address this issue, it is a typical practice for software developers to add tags that are commonly used in social media [13], [14], [15], [16], [17] with each posted content. Since tags normally consists of a few words or abbreviations only, they provide a type of metadata to search, describe, identify, bookmark, classify and organize software objects on software information sites [3]. Most software information sites rely on the tags to classify the software objects in order to improve the performance and accuracy of various operations on the sites [18], [19]. Therefore, the quality of tags are critical for software information sites. High quality tags are expected to be concise and can describe the most important features of the software objects.

Unfortunately tagging is inherently a distributed and uncoordinated process [1]. Each developer is free to choose tags that are deemed most appropriate for a software object. The choice is dependent not only on a developer's understanding of the software object but also on the developer's English skills and preferences. For example, the tags scc, source-code-control, sccs and several other words in StackOverflow are all used to describe version control. In addition, a software object can be labeled with multiple tags. For instance, StackOverflow suggests three to five tags per posting and Freecode allows more than ten tags per posting. As a result, the number of different tags grows rapidly along with continuous addition of software objects. As of today there are more than 20 million questions and 46 thousand tags in StackOverflow. With such large number of different tags, many of which introduce noise, software objects become more and more poorly classified. Such phenomenon affects negatively the speed and accuracy of developers’ queries.

In this paper, we propose a tool called TagMulRec to recommend tags to developers and classify software objects in evolving large-scale software information sites. There are two challenges that have to be addressed: (1) TagMulRec must adapt to dynamic changes. Besides the fact that a large number of software objects are continuously added into a software object 

1http://www.stackoverflow.com
2http://www.askubuntu.com
3http://www.askdifferent.com
4http://www.freecode.com
5http://www.github.com
information site every day, developers can also modify a posted content by attaching new tags or removing existing tags. (2) TagMulRec must be efficient considering the size of the software objects and tags. Taking these challenges into consideration, TagMulRec firstly constructs indices for the description documents of software objects. Then, based on indices, software objects that are semantically similar are retrieved to construct target candidate sets. Next, TagMulRec employs a simple algorithm to rank all tags in the candidate set. The tags with high ranking scores are recommended to developers.

We have evaluated TagMulRec on four software information sites, StackOverflow, AskUbuntu, AskDifferent and Freecode. StackOverflow is further divided into StackOverflow@small and StackOverflow@large based on their sizes. StackOverflow@small, AskUbuntu, AskDifferent and Freecode are relatively small with tens of thousands of software object and hundreds of tags. We compare TagMulRec against the state-of-the-art method EnTagRec [2] on these four small-scale datasets. The experimental results show that TagMulRec improves EnTagRec by -0.2% and 8.05% in terms of F1-score@5 and F1-score@10 scores. Compared with EnTagRec, TagMulRec achieves three orders of magnitude speed-up. The large-scale software information site StackOverflow@large has more than ten million software objects and forty thousand tags. While EnTagRec method cannot handle such large dataset, TagMulRec achieves F1-score@5 and F1-score@10 scores of 0.449 and 0.294, respectively.

The main contributions of this paper include:

- We automate tag recommendation in large-scale evolving software information sites based on the semantics of software objects. This alleviates the problem of rapid growth of tags by reducing inappropriate tags and different tags referring to the same content.
- We propose an efficient tag-based multi-classification algorithm that is able to handle millions of software objects.
- We evaluate TagMulRec using four software information sites, StackOverflow, AskUbuntu, AskDifferent and Freecode. The experiments show that our approach is as accurate as and more scalable than the existing approach.

The rest of this paper is organized as follows. Section II presents related work. Section III gives an overview of our approach, followed by a detailed explanation in Section IV. Section V evaluates the performance of TagMulRec. Section VI discusses limitations and threats to validity. Finally, Section VII concludes the paper.

II. RELATED WORK

Tag recommendation has been a hot research problem in the fields of social network and data mining [20], [21], [22], [23], [24]. Automatic tag recommendation in software engineering was first proposed by Al-Kofahi et. al. in 2010 [3]. Al-Kofahi et al. proposed a method called TAGREC to automatically recommend tags for work items in IBM Jazz. TAGREC was based on the fuzzy set theory and considered the dynamic evolution of a system. Later a method called TAGCOMBINE [1] was proposed to automatically recommend tags for software objects in software information sites. It consists of three components: a multi-label ranking component, a similarity based ranking component, and a tag-term based ranking component. The multi-label ranking approach adopted by TAGCOMBINE limits its application to relatively small datasets. For a large-scale software information site such as StackOverflow@large, TAGCOMBINE has to train more than forty thousand binary classifier models and the size of each train set is more than ten million. A more recent approach called EnTagRec [2] outperforms TAGCOMBINE in terms of Recall and Precision metrics. EnTagRec consists of two components: Bayesian inference component and Frequentist inference component. However, EnTagRec is not scalable as well, as it also utilizes all information in software information sites to recommend tags for a software object. In contrast, our approach only utilizes a small portion of software information sites that is most relevant to a given software object. In addition, neither TAGCOMBINE nor EnTagRec adapts to the dynamic evolution of software information sites. In contrast, our approach is scalable and is able to handle continuous updates in the software information sites.

In the field of software engineering, tags have become widely used [4], [5], [25], [26], [27]. Storey et. al. proposed a set of pertinent research questions [4], which strives to understand the benefits, risks and limitations of using social media in software development at the team, project and community level, around community involvement, project coordination, project management and individual software development activities. Begel et al. described the potential benefits [5] for social media to both improve communication and coordination in software development teams and support of the creation of new kinds of software development communities. Treude et al. explored how tagging is used to bridge the gap between technical and social aspects of managing work items [25]. They conducted an empirical study on how tagging has been adopted and adapted over the two year of a large project with 175 developers. Their results showed that the tagging mechanism had become a significant part for many informal processes [25]. Thung et al. detected similar software application using software tags [26]. Wang et al. analyzed tags of projects in FREECODE to infer semantic relationships among the tags, and express the relationships as a taxonomy [27].

III. TAGMULREC OVERVIEW

In this section, we present the overall framework of TagMulRec after formally formulate our research question.

A. Problem Formulation

Tags provide a type of metadata to search, describe, identify, bookmark, classify, and organize software objects in software information sites [3]. They are widely used
in the developer Q&A and open source communities. For example, StackOverflow, AskUbuntu and AskDifferent suggest that developers should attach at least three but no more than five tags per posting and Freecode allows developers to create more than ten tags for each posting. Figure 1 shows a question with four tags {C++, standards, C++1z, C++-faq} in StackOverflow. Figure 2 lists a question posted in AskUbuntu with four tags {system-installation, live-usb, ssd, mint}. Figure 3 gives a question posted in AskDifferent with five tags {OSX, yosemite, software-recommendation, pdf, word-processor}. Figure 4 shows a shared project in Freecode with project description and five tags {System Administration, Operating Systems, Monitoring, Software Development, Internet}. Since software developers are free to choose tags, the words used for tags are arbitrary. Even for the words that represent the same meaning, there are differences such as spaces vs. no spaces, upper cases vs. lower cases, acronym vs. full spelling, hyphens vs. no hyphens, etc. Such phenomenon makes it difficult for software developers to search for existing tags, thus become more likely to use their own wording, which leads to more and more synonymous tags with different spelling. Figure 5 gives a small portion of the synonymous tag list in StackOverflow that contains 3429 tags.

A software information site is a set \( S = \{o_1, \ldots, o_n\} \), where \( o_i (1 \leq i \leq n) \) denotes a software object. For a developer Q&A site, such as StackOverflow, the attributes of \( o_i \) include an identifier \( id \), a title \( t_i \), a body \( b_i \), a set of tags \( T_i \), etc. For an open source site, such as Freecode, the attributes of \( o_i \) include a project name \( n_i \), a project description \( b_i \), a set of tags \( T_i \), etc. If we treat the combination of the title \( tt \) and body \( b \) of a software object in a Q&A site as a project description \( d \), we can assume that any software object \( o_i \) contains a description \( o_i, d \) and a set of tags \( o_i, T \). The tags in an information site \( S \) is a set \( T = \{t_1, \ldots, t_m\} \) and the tags associated with an object \( o_i \), i.e. \( o_i, T \), is a subset of \( T \). The research question we try to answer in this paper is the following: given a large set of existing software objects that are labeled with tags, how to multi-classify a new software object \( o_i \) into a set of tags \( o_i, T \).

B. Overall Framework

Figure 6 gives an overview of our method TagMulRec. Let \( S \) be the software information site under consideration. TagMulRec first eliminates software objects without tags or with unreliable tags only. A tag is unreliable if it is rarely used in a software information site. All the remaining software objects are then indexed. Next, given a new software object \( o \), TagMulRec computes a target candidate set \( C = \{(o_1, \delta_1), \ldots, (o_n, \delta_n)\} \) that consists of software objects semantically similar to \( o \). The attribute \( \delta_i \) is a score that quantifies the similarity between \( o_i \) and \( o \). If the context is not clear we use \( \delta(o, o_i) \) to denote the similarity score between \( o \) and \( o_i \). Finally TagMulRec exploits a multi-classification algorithm based on semantics similarity to categorize tags in the target candidate set \( C \). This step produces a ranked tag list \( \langle t_1, \ldots, t_k \rangle \) that is presented to software developers.

IV. TAGMULREC METHOD

In this section, we present each step of TagMulRec method in detail.
A. Preprocessing

Tags are highly recommended but not required. Therefore there may exist software objects without tags in a software information site \( S \). These software objects obviously have to be removed because our tag recommendation exploits existing tags.

If a tag \( t \) appears in \( S \) but very infrequently, there are two possibilities: (1) It was a bad choice and it is not used by others. For example, the spelling of \( t \) is incorrect. In this case, \( t \) should not be recommended. (2) The software object contains a rare topic. It is possible that the new software object \( o \) also contains this rare topic. Even so, \( t \) may not be an appropriate tag to describe the topic as it is not widely agreed upon yet. In this case, it is better for a software developer to create her own tags. Based on the above observation we set a predefined threshold \( \theta \). TagMulRec does not consider those tags with frequency less than \( \theta \). If the frequency of all the tags in a software object \( o_s \) is less than \( \theta \), we remove \( o_s \).

The preprocessing produces \( S' \subseteq S \). As a final step, we remove all the stop words from the description of \( o' \in S' \). These preprocessing rules have also been used in previous research [1], [2].

B. Indexing

TagMulRec assigns each software object in \( S' \) a unique index. TagMulRec also constructs a dictionary \( D \) that contains all the words appearing in the descriptions of software objects in \( S' \). For each entry \( w \in D \), TagMulRec creates a linked list [28], [29] with each node consisting of the index of the software object \( o_{ij} \) such that \( w \in o_{ij}.d \), and the frequency of \( w \) occurring in \( o_{ij}.d \). For an evolving software information site, the dictionary can be incrementally maintained with addition of software objects.
Figure 7 depicts an example with three words software, developer, and object. The linked list associated with object indicates that the description of software objects 35, 77, and 100 contain the word. Due to space limit we omit the information on frequency in the figure.

C. Similarity Score Computation

Given a new software object $o$ and an existing one $o_i \in S'$, TagMulRec computes a similarity score $\delta(o, o_i)$ based on their descriptions. Equation 1 gives the formula that is used for this computation.

$$\delta(o, o_i) = \varphi(o, o_i) \cdot \phi(o.d) \cdot \Sigma_{w \in o.d} (#o_i.d.w \cdot #O_w \cdot \psi(w) \cdot \rho(o_i, d))$$  \hspace{1cm} (1)

The term $\varphi(o, o_i)$ is a score factor that treats each word in $o.d$ is regarded as a query term. Its value depends on the frequency of the words of $o.d$ appearing in $o_i.d$, as shown in Equation 2.

$$\varphi(o, o_i) = \Sigma_{w \in o_i.d | o.d} #o_i.d.w / |o_i.d|$$  \hspace{1cm} (2)

For example, if $o.d = \text{software engineering}$ and $o_i.d = \text{software developer}$ look for help from StackOverflow, the number of common words is 1. Since the number of words in the query text $o.d$ is 2, we have $\varphi(o, o_i) = 1/2$.

The term $\phi(o.d)$ is a query normalization factor that is computed by Equation 3.

$$\phi(o.d) = 1/\sqrt{\psi(o.d)^2 \cdot \Sigma_{w \in o.d} (#S'_w \cdot \psi(w))^2}$$  \hspace{1cm} (3)

In both Equations 1 and 3, $\psi(w)$ denotes the weight of the word $w$ and $\psi(o.d)$ the weight of the query text $o.d$. We can set the weight to make a word or a query text more important than others. Although in our experiment we treat all words and queries the same weight, software developers can adjust the values.

The term $#o_i.d.w$ denotes the frequency of $w$ occurring in $o_i.d$, which can be easily obtained by searching the dictionary constructed by TagMulRec. The term $#S'_w$ is the number of the software objects whose description includes $w$. This can be obtained by retrieving the length of the linked list associated with $w$ in the dictionary $D$.

The term $\rho(o_i, d)$, computed by Equation 4, is a standardized parameter of $\delta(o, o_i)$.

$$\rho(o_i, d) = \psi(o_i, d) / \sqrt{|o_i.d|}$$  \hspace{1cm} (4)

In Equation 4, $\psi(o_i, d)$ is the weight of the software object description and $|o_i.d|$ is the size of the description.

Based on Equation 1, TagMulRec is able to compute the similarity score between the new software object $o$ and $o_i \in S'$. This produces a target candidate set $C^k_o$ that contains $k$ software objects that has the highest similarity scores with $o$. The size of the target candidate set, $k$, can be adjusted. We use Lucene\(^9\) to implement the Indexing and Similarity Score Computation.

D. Multi-Classification of Software Objects

Given a target candidate set $C^k_o$ of the software object $o$, let $c_{max}$ and $c_{min}$ be the maximum and minimum scores in $C^k_o$. TagMulRec normalizes the similarity score $\delta(o, o_i)$ of $o_i \in C^k_o$ by using Equation 5, which results in a normalized score value within the range of [0,1].

$$\delta_{norm}(o, o_i) = (\delta(o, o_i) - c_{min}) / (c_{max} - c_{min})$$  \hspace{1cm} (5)

Let $T_i$ be the set of tags of $o_i \in C^k_o$. The tags of all the software objects in the target candidate set is thus $T_o = \cup_{i=1}^{c_{max}} T_i$. For each tag $t_j \in T_o$, we compute the score of $t_j$ using Equation 6:

$$\delta(t_j) = \Sigma_{i=1}^{c_{max}} (#o_i.t_j) \cdot \delta_{norm}(o, o_i),$$  \hspace{1cm} (6)

where $#o_i.t_j$ denotes the frequency of tag $t_j$ in $o_i$. We rank all tags in $T_o$ by their scores and obtain a ranked list $TL$. The list $T L_{\text{top}K}$ with $K$ highest score tags in $TL$ are recommended to software developers for the software object $o$.

V. EXPERIMENTS

In this section, we first shows the experimental settings, then evaluation metrics are presented. Last, we answer some research questions based on experimental results.

A. Experiment Setup

We evaluate TagMulRec on four software information sites, StackOverflow, AskUbuntu, AskDifferent, and Freecode. We further divide StackOverflow into two dataset with different sizes: StackOverflow@small contains the software objects posted form July 1st, 2008 to December 10th, 2008; StackOverflow@large contains all the software objects posted before July 1st, 2012. For AskUbuntu and AskDifferent, we consider all the software objects posted before April 30th, 2012. Finally for Freecode, all the posted software objects are considered. All the selected data have been published for relatively long time to ensure that their tags are stable. Such experimental setup leads to four small-scale datasets StackOverflow@small, AskUbuntu, AskDifferent, Freecode and one large-scale dataset StackOverflow@large. StackOverflow@small, AskUbuntu, AskDifferent and StackOverflow@large can be collected by

\(^9\)http://www.lucene.apache.org/
StackExchange Data Explorer\textsuperscript{7}. Freecode can be acquired by crawling web data of the Freecode website\textsuperscript{8}.

For the four small-scale datasets, we remove those tags that occur 50 times or less. In order to maintain similar ratios between the the threshold value and the size of software information site, we set the threshold to 10000 for the much larger dataset StackOverflow@large. A software object is removed if the frequencies of all its tags are below the threshold. Table I summarizes the statistics of the five datasets. Columns 2 and 3 give the number of software objects and tags. Columns 4 and 5 list the software objects and tags after removing the low-frequency software objects and tags.

For each of the five datasets, we randomly select 10000 software objects and treat them as a validation set. For each software object $o \in V$, we compute ten tags and use $\text{Recall}_{@k}$, $\text{Precision}_{@k}$ and $F1$-score@$k$ as evaluation metrics. We compare TagMulRec against EnTagRec, a state-of-the-art tag recommendation method [2], on the four small-scale datasets. EnTagRec cannot handle StackOverflow@large so we conduct experiments on the large dataset with TagMulRec only. All the experiments were conducted on an Ubuntu 16.04 computer with Intel Core i7 3.6G and 8G RAM.

### B. Evaluation Metrics

To evaluate TagMulRec, we use the metrics $\text{Recall}_{@k}$, $\text{Precision}_{@k}$, and $F1$-score@$k$, all of which are frequently used to evaluate recommendation systems or classification tasks in software engineering literature [30], [31], [32], [33]. In particular, $\text{Precision}_{@k}$ and $\text{Recall}_{@k}$ were used to evaluate EnTagRec.

- $\text{Recall}_{@k}$: $\text{Recall}_{@k_i}$ is the percentage of tags selected out of the recommended lists $TL_{topK}$ in the software object’s true tags. For a software object $o_i$, $\text{Recall}_{@k_i}$ is computed by Equation 7. $\text{Recall}_{@k}$, computed by Equation 8, is the mean $\text{Recall}_{@k_i}$ values of the software objects in the validation set $V$.

$$\text{Recall}_{@k_i} = \begin{cases} |T_i| > K, & \text{Recall}_{@k} = \frac{|TL_{topK} \cap T_i|}{K} \\ |T_i| \leq K, & \text{Recall}_{@k} = \frac{|TL_{topK} \cap T_i|}{|T_i|} \end{cases}$$

$$\text{Recall}_{@k} = \frac{\sum_{i=1}^{|V|} \text{Recall}_{@k_i}}{|V|}$$  

- $\text{Precision}_{@k}$: $\text{Precision}_{@k_i}$ is the percentage of software object’s truth tags in the recommended lists $TL_{topK}$. For a software object $o_i$, $\text{Precision}_{@k_i}$ is defined by Equation 9. $\text{Precision}_{@k}$ is the mean $\text{Precision}_{@k_i}$ values of the software objects in the validation set $V$, as defined by Equation 10.

$$\text{Precision}_{@k_i} = \frac{|TL_{topK} \cap T_i|}{K}$$

$$\text{Precision}_{@k} = \frac{\sum_{i=1}^{|V|} \text{Precision}_{@k_i}}{|V|}$$

- $F1$-score@$k$: this metric combines $\text{Precision}_{@k_i}$ and $\text{Recall}_{@k_i}$ for a software object $o_i$. $F1$-score@$k_i$ is defined by Equation 11. $F1$-score@$k$ is the mean of $F1$-score@$k_i$, as defined by Equation 12.

$$F1 - \text{score}_{@k_i} = 2 \cdot \frac{\text{Precision}_{@k_i} \cdot \text{Recall}_{@k_i}}{\text{Precision}_{@k_i} + \text{Recall}_{@k_i}}$$

$$F1 - \text{score}_{@k} = \frac{\sum_{i=1}^{|V|} F1 - \text{score}_{@k_i}}{|V|}$$

We also compare the efficiency between the two tools. EnTagRec runtime includes the training time and the predicting time. We use the average predicting time to evaluate the efficiency of EnTagRec. TagMulRec runtime includes the time to construct the candidate set and the time to recommend tags. Because TagMulRec constructs candidate set dynamically, we use the average running time of tag recommendation to evaluate the efficiency of TagMulRec.

### C. Research Questions

We are interested in the following four research questions.

**RQ1: how effective and efficient is TagMulRec in recommending tags for small-scale software information sites?** To answer RQ1, we compare TagMulRec against EnTagRec [2] on four software information sites, StackOverflow@small, AskUbuntu, AskDifferent and Freecode. We evaluate them using the metrics $\text{Recall}_{@k}$, $\text{Precision}_{@k}$, $F1$-score@$k$ with two $k$ values, 5 and 10. Time usage is also recorded.

Table II gives the experimental results. It can be observed that TagMulRec outperforms EnTagRec in terms of $\text{Precision}_{@k}$ and $F1$-score@$k$. Since $\text{Recall}_{@k}$, $\text{Precision}_{@k}$ and $F1$-score@$k$ are defined as mean values, we compared their distributions using test procedure $T$ [2], [34] to gain more insight. Across the four small-scale datasets, the corresponding $p$-value is very small (<0.01),

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#software object</th>
<th>#tags</th>
<th>#final software object</th>
<th>#final tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>StackOverflow@small</td>
<td>50000</td>
<td>9233</td>
<td>47814</td>
<td>438</td>
</tr>
<tr>
<td>StackOverflow@large</td>
<td>11203032</td>
<td>44265</td>
<td>10421906</td>
<td>427</td>
</tr>
<tr>
<td>AskUbuntu</td>
<td>36868</td>
<td>1784</td>
<td>24941</td>
<td>344</td>
</tr>
<tr>
<td>AskDifferent</td>
<td>14836</td>
<td>824</td>
<td>11459</td>
<td>178</td>
</tr>
<tr>
<td>Freecode</td>
<td>47978</td>
<td>9018</td>
<td>43644</td>
<td>274</td>
</tr>
</tbody>
</table>

\textsuperscript{7}http://www.data.stackexchange.com/

\textsuperscript{8}http://www.freecode.com
TABLE II: TagMulRec vs. EnTagRec using metrics Recall@k, Precision@k, and F1-score@k

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Recall@5</th>
<th>Precision@5</th>
<th>F1-score@5</th>
<th>Time(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>StackOverflow@small</td>
<td>0.680</td>
<td>0.805</td>
<td>0.284</td>
<td>0.346</td>
</tr>
<tr>
<td>AskUbuntu</td>
<td>0.700</td>
<td>0.815</td>
<td>0.383</td>
<td>0.358</td>
</tr>
<tr>
<td>AskDifferent</td>
<td>0.715</td>
<td>0.880</td>
<td>0.421</td>
<td>0.369</td>
</tr>
<tr>
<td>Freecode</td>
<td>0.659</td>
<td>0.640</td>
<td>0.383</td>
<td>0.382</td>
</tr>
<tr>
<td>Average</td>
<td>0.687</td>
<td>0.785</td>
<td>0.368</td>
<td>0.363</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Recall@10</th>
<th>Precision@10</th>
<th>F1-score@10</th>
<th>Time(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>StackOverflow@small</td>
<td>0.777</td>
<td>0.868</td>
<td>0.229</td>
<td>0.195</td>
</tr>
<tr>
<td>AskUbuntu</td>
<td>0.821</td>
<td>0.876</td>
<td>0.257</td>
<td>0.200</td>
</tr>
<tr>
<td>AskDifferent</td>
<td>0.845</td>
<td>0.944</td>
<td>0.245</td>
<td>0.240</td>
</tr>
<tr>
<td>Freecode</td>
<td>0.758</td>
<td>0.755</td>
<td>0.224</td>
<td>0.205</td>
</tr>
<tr>
<td>Average</td>
<td>0.800</td>
<td>0.860</td>
<td>0.232</td>
<td>0.185</td>
</tr>
</tbody>
</table>

which shows the significance of the difference in the Recall@k, Precision@k and F1-score@k values between TagMulRec and EnTagRec.

The total time to compute 10 tags by EnTagRec and TagMulRec for the 10000 software objects is about 21400 seconds and 32 seconds, respectively. Table II gives the average time to compute a single tag. It can be observed that TagMulRec achieves a three orders of magnitude speedup.

RQ2: how effective and efficient is TagMulRec in recommending tags for large-scale software information sites?

In order to investigate the performance of TagMulRec on large-scale software information sites, we attempted to compare TagMulRec against EnTagRec on the dataset StackOverflow@large. However, after spending more than three months of training time, EnTagRec does not return any results. This is due to the fact that EnTagRec utilizes all information in software information site to train model. When the scale of a software information site is large, EnTagRec may not be able to obtain a trained model within an acceptable time period.

As a result, we can measure the performance of TagMulRec only. Table III shows the Recall@k, Precision@k, and F1-score@k values (k = 5 and 10) on both StackOverflow@large and StackOverflow@small. In particular, TagMulRec achieves a F1-score@5 score of 0.449 and a F1-score@10 score of 0.294 on the large-scale dataset StackOverflow@large. Because StackOverflow suggests three to five tags per software object and developer may likely pay attention to the top few recommended tags only, the metric F1-score@5 is more important than F1-score@10. It can be observed that the time consumption per tag is significantly higher for StackOverflow@large. However, the total time to compute ten tags each for all the 10000 software objects is less still than 16000 seconds. Therefore, we claim that TagMulRec can be both effective and efficient for large-scale software information sites.

RQ3: does the size of target candidate set C affect the performance of TagMulRec?

The size of target candidate set C affects experiment results. In order to answer RQ3, we first investigate the tag coverage rate of the target candidate set on the five datasets, then we evaluate how Recall@k and Precision@k changes with the size of target candidate sets.

- TagCov_i is the percentage of tags selected out of candidate set in the software object's truth tags. Equation 13 shows how to compute the tag coverage rate TagCov_i of target candidate set for a software object o_i.

\[
\text{TagCov}_i = \frac{|T_{o_i} \cap T_i|}{|T_i|}
\]  

(13)

- TagCov is the average tag coverage rate of the validation set V can be computed by Equation 14.

\[
\text{TagCov} = \sum_{i=1}^{|V|} \text{TagCov}_i \frac{|V|}{|T_i|}
\]  

(14)

Figure 8 depicts the change of tag coverage rate along with the size of the target candidate set. The x-axis and y-axis denote the size of the candidate set and the tag coverage rate. The data for the five data sets Freecode, StackOverflow@small, AskUbuntu, AskDifferent and StackOverflow@large are given by blue, red, green, cyan and black lines, respectively. It can be observed that (1) with the size of candidate set increasing, the tag coverage rate increases as well and tends to be stable finally, and (2) with the size of candidate set increasing, the growth rate of tags coverage rate decreases and tends to be 0 finally. The reason that the tag coverage rate tends to be stable is that some noise data are introduced with the enlargement of the target candidate set.

For the four small-scale datasets StackOverflow@small, AskUbuntu, AskDifferent and Freecode, Figures 9, 10, 11, 12, 13, and 14 depict the values of Recall@5, Recall@10, Precision@5, Precision@10, F1-score@5 and F1-score@10 respectively, along the size of the candidate set. In addition to Freecode, higher tag coverage rates correspond to higher values of Recall@5, Recall@10, Precision@5, Precision@10, F1-score@5 and F1-score@10 in the small-scale datasets. For StackOverflow@small and StackOverflow@large, Figures 15 shows Recall@5 and Recall@10 values along with the size of the candidate set, Figures 16 shows
### TABLE III: Recall@k, Precision@k, and F1-score@k values of TagMulRec on large and small scale StackOverflow

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Recall@5</th>
<th>Precision@5</th>
<th>F1-score@5</th>
<th>Recall@10</th>
<th>Precision@10</th>
<th>F1-score@10</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>StackOverflow@small</td>
<td>0.680</td>
<td>0.284</td>
<td>0.400</td>
<td>0.777</td>
<td>0.165</td>
<td>0.272</td>
<td>0.045</td>
</tr>
<tr>
<td>StackOverflow@large</td>
<td>0.809</td>
<td>0.310</td>
<td>0.449</td>
<td>0.892</td>
<td>0.176</td>
<td>0.294</td>
<td>160</td>
</tr>
</tbody>
</table>

Fig. 8: Effect of candidate set size on five datasets Freecode (blue), StackOverflow@small (red), AskUbuntu (green), AskDifferent (cyan) and StackOverflow@large (black).

Fig. 9: Recall@5 values: Freecode (blue), StackOverflow@small (red), AskUbuntu (green), and AskDifferent (cyan).

Fig. 10: Recall@10 values: Freecode (blue), StackOverflow@small (red), AskUbuntu (green), and AskDifferent (cyan).

Fig. 11: Precision@5 values: Freecode (blue), StackOverflow@small (red), AskUbuntu (green), and AskDifferent (cyan).

Precision@5 and Precision@10 values and Figures 17 shows F1-score@5 and F1-score@10 values. It can be observed that: (1) Recall@5, Recall@10, Precision@5, Precision@10, F1-score@5 and F1-score@10 values tend to be more stable as the size of candidate set increases, and (2) Recall@5, Recall@10, Precision@5, Precision@10, F1-score@5 and F1-score@10 values are related to tag coverage rate in most cases.

**RQ4: does the tag threshold value affect the performance of TagMulRec?**

In our approach we filter tags whose frequency is less than the tag threshold value. For the small-scale software information sites, we set the threshold value to 50, which has also been used in previous works [1], [2]. Considering the size of software information sites, we set the threshold value to 10000 for the large-scale software information site. In this group of experiments, we investigate whether the tag threshold value affect the performance of TagMulRec on the large-scale software information site. To this end, we evaluate the change of Recall@k, Precision@k and F1-score@k values by setting the tag threshold values to both 50 and 10000.

Table IV compares the Recall@k, Precision@k and F1-score@k values after using tag threshold values 50 and 10000. It can be observed that a threshold value of 10000 achieves
TABLE IV: Recall@k, Precision@k, and F1-score@k value and time consumption of TagMulRec on StackOverflow@large

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Recall@5</th>
<th>Precision@5</th>
<th>F1-score@5</th>
<th>Recall@10</th>
<th>Precision@10</th>
<th>F1-score@10</th>
<th>Time(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.640</td>
<td>0.343</td>
<td>0.444</td>
<td>0.749</td>
<td>0.205</td>
<td>0.310</td>
<td>162</td>
</tr>
<tr>
<td>10000</td>
<td>0.809</td>
<td>0.310</td>
<td>0.449</td>
<td>0.892</td>
<td>0.176</td>
<td>0.294</td>
<td>160</td>
</tr>
</tbody>
</table>

![Graph](image1.png)

**Fig. 12:** Precision@10 values: Freecode (blue), StackOverflow@small (red), AskUbuntu (green), and AskDifferent (cyan).

![Graph](image2.png)

**Fig. 14:** F1-score@10 values: Freecode (blue), StackOverflow@small (red), AskUbuntu (green), and AskDifferent (cyan).

![Graph](image3.png)

**Fig. 13:** F1-score@5 values: Freecode (blue), StackOverflow@small (red), AskUbuntu (green), and AskDifferent (cyan).

![Graph](image4.png)

**Fig. 15:** Recall@5 values: StackOverflow@small (red solid) and StackOverflow@large (black solid); Recall@5 values: StackOverflow@small (red dotted) and StackOverflow@large (black dotted).

A slight higher Recall@k values, while a threshold value of 50 achieves a slight higher Precision@k values. As for F1-score@k and time consumption, there is almost no difference. Based on the experiments, we conclude that the performance of TagMulRec does not depend on the preset tag threshold values.

VI. THREATS TO VALIDITY

There are several threats that can potentially affect the validity of our research results.

1) Potentially Biased Results. Our tag recommendation assumes that existing tags in a software information site are correct. However, human errors are inevitable. We do apply some filtering rules, such as time interval of dataset, to alleviate the problem. These filtering rules have also been used in past research [1], [2]. However, this issue, such as how to deal with large number of synonymous tags, cannot be completely solved. We reimplemented the EnTagRec method, which may not be the same as the original EnTagRec [2].

2) Generalizability of Algorithms. In this paper, we evaluate TagMulRec on five software information sites. There are more than 120000 software objects in the four small-scale
datasets and more than 11 million software objects in the large-scale dataset. Even so, more case studies are needed to generalize our findings. In the future, more software information sites will be used to evaluate TagMulRec.

3) Suitability of Evaluation Metrics. In this paper, Recall@k, Precision@k and F1-score@k are used as our evaluation metrics. Recall@k and Precision@k have been used in the past to evaluate the performance of tag recommendation for software information sites [1], [2], [3] and for social media and network [35], [36], [37], [38]. Time usage is also used in this paper. Because of differences in operating systems, hardware, the development environment and others, time usage may be not suitability in repeated our experiments. It is possible that more suitable metrics can be adopted. For example, since our tag recommendation is a multi-classification process [39], the evaluation metrics of multi-label classification [40], [41] will be used in our future work.

VII. CONCLUSION AND FUTURE WORK

In this paper, we proposed a new software object multi-classification method. To the best of our knowledge, TagMulRec is the first tool to automatically recommend tags for large-scale evolving software information sites. TagMulRec achieves effectiveness and efficiency by (1) creating index for software object descriptions, (2) constructing target candidate sets that include software objects semantically similar to the given software object, and (3) utilizing multi-classification algorithms to rank tags in the target candidate set. We evaluated the performance of TagMulRec on four software information sites with large number of software objects and tags. Our empirical study confirmed that our method is promising.

Our current work is based on text only. In the future, we plan to consider code snippets to make our tag recommendation more accurate. We will also conduct experiments on more software information sites with more evaluation metrics.

ACKNOWLEDGMENT

Jin Liu and Zijiang Yang are the corresponding authors. We would like to thank Xin Xia for his constructive comments, which helped us to improve the manuscript. This work is partly supported by National Natural Science Foundation of China (NSFC) (grant No. 61572374, U16362068, U1135005, 71161015, 61462056, 61472318, 61632015), the Academic Team Building Plan from Wuhan University and National Science Foundation (NSF) (grant No. DGE-1522883).

REFERENCES


