Labeled Topic Detection of Open Source Software from Mining Mass Textual Project Profiles

Tao Wang, Gang Yin, Xiang Li, Huaimin Wang
National Laboratory for Parallel and Distributed Processing
National University of Defense Technology, Changsha, 410073, China
{taowang.2005, jack.nudt, shockleylee}@gmail.com, whm_w@163.com

ABSTRACT

Nowadays open source software has become an indispensible basis for both individual and industrial software engineering. Various kinds of labeling mechanisms like categories, keywords and tags are used in open source communities to annotate projects and facilitate the discovery of certain software. However, as large amounts of software are attached with no/few labels or the existing labels are from different ontology space, it is still hard to retrieve potentially topic-relevant software. This paper highlights the valuable semantic information of project descriptions and labels, proposes labeled software topic detection (LSTD), a hybrid approach combining topic models and ranking mechanisms to detect and enrich the topics of software by mining the large amount of textual software profiles, which can be employed to do software categorization and tag recommendation. LSTD makes use of labeled LDA to capture the semantic correlations between labels and descriptions and then construct the label-based topic-word matrix. Based on the generated matrix and the generality of labels, LSTD designs a simple yet efficient algorithm to detect the latent topics of software that expressed as relevant and popular labels. Comprehensive evaluations are conducted on the large-scale datasets of representative open source communities and the results validate the effectiveness of LSTD.

Categories and Subject Descriptors

D.2.9 [Software Engineering]: Management-productivity; H.3.3 [Information Storage and Retrieval]: Retrieval models,Clustering,Query formulation

General Terms

Experimentation, Management, Human Factors

Keywords

Open Source Software, Topic Detection, Topic Model, Labeled LDA, Software Profile

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SoftwareMining ’12 August 12, Beijing, China
Copyright 2012 ACM 978-1-4503-1560-9/12/08 ...$10.00.

1. INTRODUCTION

With the evolving and growing of Internet-based open source ecosystems, software developers treat open source software as huge and inexhaustible resources in both individual and industrial software engineering for high quality development tools and reuse source code. Currently, there are nearly one million open source projects hosted in tens of open source communities such as software forges and social directories. Some giant forges such as SourceForge1 and Googlecode2 contain tens of thousands of open source projects, while some social open source directories such as ohloh3 contains even more projects collected from different forges. To facilitate quick discovery and retrieval of desired projects with relevant topics, textual descriptions and various kinds of annotations are attached to open source projects, which are key elements of project profile data. The annotations in project profiles are the most important semantic data generated by labeling systems such as the hierarchical categories in SourceForge, the keywords in Freecode4, the tags in ohloh and in this paper we give them a uniform identifier “label”. Labels identify the functional or technical features of software and do great help for developers to quickly understand unfamiliar software.

However, current labeling systems in open source communities are insufficient for detecting potentially topic-relevant software for two reasons. The first is simply due to label insufficiency. For example, there are a large proportion of projects in the above communities that not marked with any labels or only tagged with very few labels. As shown in Table 1 (according to data in mid-2011), among the 298,402 projects in SourceForge, 39.8% of them have no label and 77% of them have no more than one label. That’s worse for ohloh in which 61.68% have no label and 69.89% have less than two labels. The project profiles in Freecode are considered to be carefully supervised, but 8.61% of them have no labels and 20.6% have less than two labels.

Even for the software annotated with multiple labels, developers often find it hard to retrieval them. There comes the second reason which we call topic heterogeneity in open source communities. It is mainly the result of ontological differences between labeling mechanisms and technical background of labelers in communities.

The software engineering community has conducted plenty of efforts on discovering or retrieval of related software by

1http://sourceforge.net
2http://code.google.com
3http://www.ohloh.net
4http://freecode.com
mining software source code identifiers [8, 11, 20], word frequencies [10], source code comments [19], API calls [14] and hybrid artifacts [1, 3]. These works mainly concentrated on a few project repositories while little attention has been paid to the project profiles in global communities and made no use of software labels. Feature recommendation research [6] uses machine learning methods to mine correlated features of software products from online profiles, which concentrate on domain-specific feature models. Knowledge mining researchers have paid many efforts to use social tags of Web entities to make tag recommendation [16, 17, 18], ontology construction [12] and topic discovery [7, 21].

We believe that there exists a knowledge core hidden in the massive scale of interconnected project labels and descriptions hosted in open source communities, which is fundamental for detecting topics of software. In this paper, we propose labeled software topic detection (LSTD) approach to discover relevant topics for a given project with description by mining the huge amount of software profiles crawled from several large scale open source communities. The main contributions of this paper are as follows:

- The rich semantic information of software description and labels are explored and probabilistic software word-label matrix is constructed from mining large amount of textual project profiles. This matrix preserves the semantic correlations and is leveraged as a base for later software categorization and tag recommendation.
- A simple but efficient algorithm is proposed for topic detection based on the word-label matrix. Different from the time-consuming LDA which runs iteratively on all document sets, it directly computes the potential label set based on the matrix for a given project description with high precision.
- Three large open source communities with different labeling mechanisms are crawled and their project profiles are analyzed. Comprehensive evaluations are conducted and compared with two different previous works. Furthermore, the capabilities of the aggregation of different labeling mechanisms are explored.

The reminder of this paper is organized as follows. Section 2 discusses the related work. Section 3 analyzes the labeling mechanisms and statistical data of profiles in typical open source communities. Section 4 formulates the topic detection problem, introduces the LSTD approach and topic detection algorithm in detail. Section 5 describes the experiment datasets and evaluation measures and section 6 demonstrates case studies. Section 7 summarizes this paper and discusses the future work.

2. RELATED WORK

Though project descriptions and tags contain abundant summarized information about software, little attention has been paid to such data for open source software. Nevertheless, there are many researches closely related or inspiring to our work.

To facilitate discovery or retrieval of related software, Kawaguchi et al. [8] proposes MUDABlue to categorize software systems by using LSA on software source code identifiers. Kuhn [10] uses log-likelihood ratio of word frequencies to extract labels from source code and used them to label software components. Differently, Tian [19] employs the probabilistic topic model LDA [4] to analyze software categories. They treat every software system as a document consisted of a collection of words including code identifiers and comments parsed from source code, cluster topics based on topic similarities and categorize software by software-topic matrices. McMillan et al. [14] use API calls from third-party libraries as attributes for automatic categorization of software applications. They chose decision trees, naive Bayes and support vector machines (SVM) to categorize applications, and find that SVM is most-effective. Cleary et al. [5] use a language-based approach to locate non source code artifacts to a set of predefined topics. Lin and others [13] use LDA to extract concepts from source code and compute the similarity of source file.

Some recent efforts have been conducted on feature recommendation and traceability of software artifacts. Dumitru et al. [6] propose a feature recommendation approach that mines software domain-specific features from publicly available online product profiles, and uses association rules and standard k-nearest neighbor learning strategy to generate domain-specific software feature recommendations. Asuncion et al. [1] propose an approach to automatically records traceability links during the software development process and uses LDA for semantic categorization of artifacts and the topical visualization of the software system. Codebook [3] provides a hybrid graph model of people and artifacts, which can be used to discovery software artifacts through the potential links across teams.

Other research fields such as knowledge management have done plenty of works on tag-based topic models and their applications. Ramage et al. [16] propose Labeled LDA (L-LDA) to predict topic labels for multi-labeled documents by associating each word in a document with the most appropriate tags. Sigurbjörnsson et al. [17] design a generic method to extend Flickr photo annotations based on tag co-occurrence statistics. Griffiths et al. [7] design a LDA-based algorithm to discover the topics from a large set of textual abstracts of a scientific journal. Yin et al. [21] combine geographical clustering and latent topic modeling into one framework and discovered latent topics by using geographical tags. Liu et al. [12] use a subsumption graph to model the tag space and turn this graph into a concept hierarchy, which can be used as domain ontology. Krestel et al. [9] propose a LDA-based tag recommendation technique. It took the tag set attached by users as the document for that project and used LDA to discover every project’s topics which are distributions over tags. Based on these discovered topics, each project can be represented by a set of tags that appear in these topics and additional tags are recommended. These works provide us with valuable inspirations to solve the software topic detection problem by using topic models and mass project profile data.

<table>
<thead>
<tr>
<th>Community</th>
<th>total projects</th>
<th>unique labels</th>
<th>projects (#label=0)</th>
<th>ratio (#label=0)</th>
<th>projects (#label=0.1)</th>
<th>ratio (#label=0.1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freecode</td>
<td>298,402</td>
<td>362</td>
<td>118,750</td>
<td>39.86%</td>
<td>229,752</td>
<td>77.00%</td>
</tr>
<tr>
<td>137,032</td>
<td></td>
<td>102,298</td>
<td>267,429</td>
<td>63.68%</td>
<td>281,012</td>
<td>69.98%</td>
</tr>
<tr>
<td>SourceForge</td>
<td>44,964</td>
<td>6,432</td>
<td>3,777</td>
<td>8.61%</td>
<td>9,035</td>
<td>20.60%</td>
</tr>
<tr>
<td>Total</td>
<td>447,344</td>
<td>137,238</td>
<td>259,000</td>
<td>63.09%</td>
<td>399,797</td>
<td>70.98%</td>
</tr>
</tbody>
</table>

Table 1: Labels in open source communities
### 3. OPEN SOURCE PROJECT PROFILES

Currently, abundant information is available for open source software from various kinds of open source repositories, such as source code, mailing lists and bug reports. These data mainly depict the inner software development process in detail. Compared to project descriptions and labels, such inner repository data has no emphasis on the general features of the projects which are required by the process of software submission. Figure 1 gives an example taken from ohloh. The web page of Mozilla Firefox contains large amount of information about the project, including description text, code analysis, tags, and so on. Among various kinds of information, this paper mainly focuses on project descriptions and labels, which we defined as the profile of a project.

#### 3.1 Labels in Open Source Communities

Open source communities use labels to mark software, mainly used to identify the software function feature, such as “game” and “video”; some others are used to indicate the technical characteristics, such as “Linux” and “XML”. These labels subdivide projects into more concrete domain, and help clustering and understanding these projects. As mentioned in section 1, the three communities adopt different labeling mechanisms to annotate projects. In this section, we analyze the three typical open source labeling mechanisms in detail: categories, keywords and social tags.

**Categories**: SourceForge defines a categorization which has 363 unique categories (in mid-2011) and users can only annotate projects by choosing from these predefined ones. These categories are hierarchical and the relations between them are well defined. In addition, SourceForge has a very strict constraints on labeling authorization that only project managers are allowed to decide/change the categories of a project. Such mechanism ensures the quality of the labels and almost all of the categories are recapitulative and explicit, such as “games” and “multimedia”. However, as new software categories may emerge quickly, it is hard for such rigid mechanism to keep pace.

**Keywords**: Freecode adopts a comparably flexible label creation mechanism that project managers are allowed to add labels(keywords) to a project according to their knowledge. The difference between SourceForge and Freecode is that the latter one has no predefined categories and managers are allowed to label freely. Such mechanism on the one hand enlarges the labels with open vocabulary, and on the other hand ensures the quality of annotations.

**Social tag**: In ohloh, any registered users of the community can delete or add tags to a project with any term based on their experience and understanding. This largely makes it easy to annotate projects and immensely broadens the vocabulary of labels for software. However, as anyone can add tags with any words, the quality of labels is not as good as that in SourceForge and Freecode. Generally speaking, categories in SourceForge mainly reflect the functional feature of projects, and they are well structured but rigid to keep pace with the quick emergence of new software. While keywords and social tags, contain much diverse information, not only functional and technical messages, but also quality and sentimental information. They are flexible but often over-fragmented and sparse [10]. Just as shown in Table 1, though with fewer projects than SourceForge, Freecode has much more unique labels.

![Figure 1: Mozilla Firefox in ohloh](image)

### 3.2 Statistical Features of Project Profiles

Different from academic papers, project profiles usually have short and freely written descriptions and labels. This section gives a statistical study of project profiles in the three open source communities with the data sets presented in Table 1. The number of labels attached usually has significant impact on the chances for the projects of been browsed and discovered by others. Figure 2 presents the distributions of number of labels per project in all three open source communities. The $x$ axis stands for the number of labels of one project and $y$ axis represents the number of projects. From this figure we find that, although with different labeling mechanisms, all of the three communities obey a similar distribution on the number of labels of each project. Most of the projects are attached with a few labels and very few projects are annotated with a large number of labels. Different from power-law, there are summits in both Freecode and ohloh distributions at two or three. Another thing worth noting is that there are considerable differences at the tail of the three distributions, where the maximum size of the project label sets in SourceForge and Freecode is around 20 to 30, but ohloh has a long tail of projects attached by more than 40 labels, which we believe is primarily the result of the social tagging mechanisms.

Figure 3 shows the distribution of project description length in three communities, where $x$ axis represents the description length and $y$ axis denotes the percent of projects with the corresponding description length. From this figure we can find that for all three communities, most of the description lengths are about 40 words. In SourceForge, most of the description lengths are between 30 and 50 words. Freecode is mainly composed of the projects with description length at about 30 to 100, while for ohloh is 30 to 150. Furthermore, for SourceForge and Freecode, the length of about 50 words witnesses a peak in the curve, while for ohloh, project with about 10 words seizes the highest percentage.

From Figure 2 and Figure 3 we can observe that: (1) The number of labels attached to each project for different communities are quite different which mainly induced by the different labeling mechanisms; (2) Most of the project descriptions are short and the content are not as strict as academic papers. These characteristics bring the difficulties such as how to merge different labeling mechanisms to a u-
4. Labeled Topic Detection

This section formally defines the software topic detection problem and then proposes a new hybrid approach to solve this problem which mainly contains two steps.

4.1 Approach Overview

Suppose \( P = \{p_1, p_2, ..., p_N\} \) is a set of software profiles in which \( p_i = (W_i, L_i) \) is the description of the \( i \)th project. We suppose that, for most projects, the descriptions at a point of view, reflect the possibility the word \( w \) has been assigned with label \( l \) has been as-

\[ P = \{p_1, p_2, ..., p_N\} \]

with each element \( l_i \in L_i \) being a set of unique words and \( L_r \subseteq L \) is the detected label set for the given project.

\[ P = \{p_1, p_2, ..., p_N\} \]

where \( P \) and \( W_r \) are the inputs of \( F \), and \( L_r \subseteq L \) is the detected label set for the given project.

We propose a Labeled Software Topic Detection (LSTD) approach which consists of two steps: the first step is to mapping every word in the project description to a label by which it is generated and construct the word-label matrix; the second step is to detect the topics for the given project based on the word-label matrix.

4.2 Word-Label Matrix Construction

We propose to use L-LDA [16], an extension of LDA, to compute the semantic relations between labels and words. LDA is a probabilistic topic model which views document as a mixture of topics and every topic is presented as a distribution of words. Similar to LDA, L-LDA views a document as a bag of words and assumes that every word in the document is generated by a topic. The different is that, L-LDA builds correspondences between latent topics and labels by constraining LDA topics to user labels. By this L-LDA captures the hidden word-label correlations and gives every latent topic a name with label.

At this step, we implemented L-LDA based on Gibbs sampling-based LDA in the MALLET toolkit [13]. Then a set of project profiles with descriptions and tags are used as the training dataset and the outcome of this step is the word-label matrix. In this paper we mark this matrix as \( \mathbf{W} \times |L| \) matrix \( A \), where \( |W| \) is the total number of is unique words and \( |L| \) is the number of unique labels. Item \( a_{i,j} \) in \( A \) represents the number of times label \( l_j \) assigned to word \( w_i \). At the second step, we design an effective topic detection algorithm that uses the matrix \( A \) to capture latent topics for a given project.

4.3 Labeled Topic Detection Algorithm

For an unlabeled project, two factors should be taken into consideration for detecting its labels: one is the appropriateness of labels for the project, and the other is the popularity of labels. This section designs a topic detection algorithm to discover a suitable label set for a given open source project based on the word-label matrix. The detected labels are ranked by their appropriateness and popularity.

We suppose that, for most projects, the descriptions attached to them reflect their topics. Based on this assumption we can specify the topics for a given project by synthesis the topics of the words in the description. For a project profile \( p_r \) with description \( W_r = \{w_{r,1}, w_{r,2}, ..., w_{r,N}\} \), we calculate the project labels by synthesizing the labels of all the words in \( W_r \). For every appearance of word \( w_{r,i} \), we calculate its possibility of being associated with label \( l_j \) by using following formula:

\[ P(l_j|w_{r,i}) = \frac{a_{i,j}}{\sum_{k=1}^{W_r} a_{i,k}} \times \frac{a_{i,j}}{\sum_{k=1}^{W_r} a_{i,k}} \]  

\[ P(l_j|w_{r,i}) = \frac{a_{i,j}}{\sum_{k=1}^{W_r} a_{i,k}} \times \frac{a_{i,j}}{\sum_{k=1}^{W_r} a_{i,k}} \]  

In Equation 2, \( a_{i,j} \) is the number of times that word \( w_{r,i} \) has been assigned with label \( l_j \), \( \sum_{k=1}^{W_r} a_{i,k} \) is the sum of times that word \( w_{r,i} \) has been assigned with all labels, \( \sum_{k=1}^{W_r} a_{i,j} \) is the total number of words assigned with label \( l_j \). This e-

\[ P(l_j|w_{r,i}) = \frac{a_{i,j}}{\sum_{k=1}^{W_r} a_{i,k}} \times \frac{a_{i,j}}{\sum_{k=1}^{W_r} a_{i,k}} \]  

quation consists of two parts: the first part, from the word’s point of view, reflects the possibility the word \( w_{r,i} \) has been assigned with label \( l_j \); and the second part, from the label’s point of view, reflects the possibility label \( l_j \) has been assigned to word \( w_{r,i} \). The combination of the two parts can
reflect the probability the word being assigned with a label based on the large amount of project profiles.

To get the overall possibility for a project of being assigned to a label, we synthesize the possibility of every word for been assigned to that label by using Equation 3:

$$P(l_j|p_r) \propto \varphi(l_j|p_r) \times \sum_{i=1}^{\mid W_r \mid} P(l_j|w_{r,i})$$  \hspace{1cm} (3)

In Equation 3, the second part is simply the sum of possibility for all words been assigned label $l_j$. The first part $\varphi(l_j|p_r)$ is a coefficient which reflects the popularity of label $l_j$ which we call Popularity Coefficient. In this paper, to simplify the computation, we set this coefficient as:

$$\varphi(l_j|p_r) = \sum_{i=1}^{\mid W_r \mid} f(l_j|w_{r,i})$$  \hspace{1cm} (4)

where

$$f(l_j|w_{r,i}) = \begin{cases} 0 & \text{if } p(l_j|w_{r,i}) > \delta \\ 1 & \text{elsewhere} \end{cases}$$  \hspace{1cm} (5)

As can be seen from Equation 4 and 5, we set the popularity coefficient as the number of words in the description of this project that has been assigned with label $l_j$ with possibility higher than the threshold $\delta$, which reflects the effect caused by topic assignment of other words in the document. That is, the more a label been assigned with different words, the higher the possibility for this label to be assigned to this project. The whole topic detection algorithm is shown in Algorithm 1.

Algorithm 1 Labeled Software Topic Detection Algorithm

Input: word-label matrix $A$, project description $W_r$, number of topics to detect $K$  
Output: detected label set $L_r$

1: for $(l_j : l_1 \rightarrow l_{\mid L_j \mid})$ do  
2: for $(w_{r,i} : w_{r,0} \rightarrow w_{r,\mid W_r \mid})$ do  
3: for $(a_{i,k} : a_{i,1} \rightarrow a_{i,\mid L_j \mid})$ do  
4: $A_j^{(t)} += a_{i,k}$  
5: end for;  
6: for $(a_{x,j} : a_{1,j} \rightarrow a_{\mid W_r \mid,j})$ do  
7: $A_j^{(w)} += a_{x,j}$  
8: end for;  
9: $P(l_j|w_{r,i}) = \frac{a_{x,j}}{A_j^{(w)}} \times \frac{a_{i,k}}{A_j^{(t)}}$  
10: $P(l_j|p_r) = P(l_j|w_{r,i})$  
11: end for;  
12: $P(l_j|p_r) = \varphi(l_j) \times P(l_j|p_r)$  
13: end for;  
14: descending sort $L_r$ according to $P(l_j|p_r)$  
15: return top $K$ labels in $L_r$

Some words in project description $W_r$ may not exist in the constructed word-label matrix, which makes it impossible to compute their possibility to be assigned to any label. However, as huge amount of project profiles are used at the word-label matrix construction process, this kind of words is usually of small proportion and have little influence on the accuracy of our topic detection algorithm. So we just ignore these kinds of words while computing.

5. EXPERIMENT SETUP

The goal of our approach is to detect the latent topics of software by using the labels(i.e. tags, keywords and categories) from open source communities. To accomplish this, we first construct the word-label matrix and then use it to detect the hidden topics of software by the topic detection algorithm proposed in subsection 4.3.

5.1 Data Collection

We carry out the experiments on datasets from three typical open source communities: SourceForge, Ohloh and Freecode. SourceForge hosts more than 300,000 projects and it is still growing every day. SourceForge predefined a hierarchical category system with 363 categories currently. Ohloh is a large, mature open source community which has collected more than 400,000 projects and Freecode listed about 45,000 projects currently.

To ensure the data quality, we first make use of Porter Stemmer [15] to stem the labels and descriptions, then use p-core [2] algorithm at level 3 for SourceForge, Freecode and level 10 for ohloh to eliminate the projects that have blank labels and those idiosyncratic or misspelling labels that not reflect the common perspective. Such difference in p-core level is due to the different labeling mechanisms used in these communities. Table 2 presents the three groups of pre-processed datasets. Oh, Sf and Fc denote the dataset from ohloh, SourceForge and Freecode respectively. The columns represent number of projects, labels, unique labels, words and unique words in these profiles respectively.

Table 2: Preprocessed experiment data sets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#projects</th>
<th>#labels</th>
<th>#unique labels</th>
<th>#words</th>
<th>#unique words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oh</td>
<td>20,266</td>
<td>98,508</td>
<td>363</td>
<td>1,061,154</td>
<td>772,175</td>
</tr>
<tr>
<td>Sf</td>
<td>363,302</td>
<td>98,319</td>
<td>31,436</td>
<td>28,963</td>
<td>87,587</td>
</tr>
<tr>
<td>Fc</td>
<td>1,061,154</td>
<td>98,960</td>
<td>15,236</td>
<td>30790</td>
<td>1,061,154</td>
</tr>
</tbody>
</table>

5.2 Evaluation Metrics

Precision and recall are usually used in Information Retrieval for evaluation. In our context, precision is the fraction of labels correctly detected and recall is the fraction of correctly detected labels over all the true ones. However, for topic detection, it is always difficult to calculate the recall metric as is hard to define the complete ground-truth labels which largely rely on human judgment. So as to comprehensively evaluate the proposed approach, we adopt two different metrics for different evaluations:

- Objective evaluation: In objective evaluation, we account original user attached labels as the ground-truth and only those exactly the same labels among the detected ones are viewed as correct. And the standard precision and recall metrics are adopt.

- Subjective evaluation: We invite two reviewers, who both have more than fifteen years of experiences on open source development, to make judgements on if a tag correct or not for a given project. We used two metrics: Mean Success at rank $k$($MS@k$) which reflects the ability to return a correct label in the top $k$ recommendations and Mean Precision at rank $k$($MP@k$) that means the fraction of labels correctly detected within top $k$ detected ones.

The main different between objective and subjective evaluation is the definition of the $\alpha$trueas label for a given project.
In this paper, we adopt one or both of these evaluations for different case studies according to the characteristics of the datasets.

6. CASE STUDIES

To evaluate the ability of LSTD on data with different labeling mechanisms, we conducted three experiments and compared with previous related works. In subsection 6.1, we compare LSTD with LACT [19] for software categorization on SourceForge. In subsection 6.2, we compare our approach with LDA-TR [9] for tag recommendation on ohloh dataset; And in subsection 6.2.1, we evaluate the ability of LSTD for aggregating different labeling mechanisms.

6.1 Open Source Software Categorization

In SourceForge, the community predefined a hierarchical category and a great of projects are attached with one or more categories. LACT categorizes software systems by analyzing the source code with LDA. We compare LSTD with LACT on the same 43 software systems used in the original work.

We construct the word-label matrix based on the 28,963 projects excluding the testing ones, and then run the topic detection algorithm on these 43 testing files whose categories are removed. Different from LACT, our method do not need the parameter of number of topics but iteration times for constructing the word-label matrix. We record the results for different iteration. Besides, as the categories in SourceForge are hierarchical and some of the projects may be cross-category, one project may belong to more than one category. In fact, a great many of SourceForge projects are attached with two or three categories. So in this case study we use subjective evaluation which is more rational and focuses on MP@K (K = 1, 2, 3, 5). Because of lack of space, we just presents part of the detailed results in Table 3.

<table>
<thead>
<tr>
<th>Iteration time</th>
<th>MP@1</th>
<th>MP@2</th>
<th>MP@3</th>
<th>MP@5</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>0.85</td>
<td>0.67</td>
<td>0.60</td>
<td>0.48</td>
</tr>
<tr>
<td>800</td>
<td>0.85</td>
<td>0.71</td>
<td>0.64</td>
<td>0.49</td>
</tr>
<tr>
<td>1000</td>
<td>0.90</td>
<td>0.75</td>
<td>0.68</td>
<td>0.59</td>
</tr>
</tbody>
</table>

For LSTD the highest precision comes with iteration 1000 for categorizing every project to one category (MP@1) at about 0.9 which means that most of the tested projects are correctly classified to one category. Even classifying every project into two or three categories, the average precision reaches 0.75 and 0.68 respectively. While for LACT, as presented in the original work, the highest precision 0.74 comes from 40 topics with distribution threshold of 0.02. The results indicate that our approach works well for classifying software with descriptions. Different from LACT for which source code are needed to be analyzed, LSTD only relies on software description, which is much more efficient and is eligible to make real-time categorization. On the other hand, LACT may generate new categories that not exist in the predefined hierarchical categories, while LSTD can only detect the existing categories. However, all the generated categories by LACT are presented as a collection of words and manual naming is needed.

6.2 Open Source Software Tag Recommendation

In this section we conducted a comparison with LDA-TR [9] for tag recommendation. The experiment files are from ohloh and consist of 9,927 project profiles as the training set and 1,000 profiles as the testing set each of which has more than 10 tags. For each of the 1000 test files 1-5 tags are randomly retained for test. The differences are that: for the training profiles, LSTD uses both descriptions and tags while LDA-TR only uses the tag information; for test profiles, LSTD uses the descriptions and the retained tags while LDA-TR uses the retained tag information. As mentioned in [9], the best results are reached for recommending 10 tags, so we compare the precision, recall for recommending 10 tags. To comprehensively evaluate the approaches, we first make an objective evaluation as adopted in the original work, and then do a subjective evaluation in addition.

6.2.1 Objective evaluation

For objective evaluation, only these detected tags that appeared in the original test profiles are viewed as correct. Figure 4 and 5 present the objective evaluation results on precision and recall.

As shown in these two figures, our approach shows a clear edge over LDA-TR on precision, recall, with more than 39% and 40% overall improvements respectively, which suggests the usefulness of these description information for topic detection and tag recommendation.

Another thing need to note is that the number of retained tags has much greater influence for LDA-TR than that for LSTD. This is different from the results reported in [9] which may due to the different characteristics of the data. Taking precision as an example, for LDA-TR when the retained tags changes from 1 to 5 the precision rises from 24% to 48% which increases about two times while for LSTD the precision changes from 45% to 58% which only increases about 1.3 times. Such difference may be mainly due to the fact that LDA-TR greatly depends on tags while our method mainly relies on description.

In addition, from these figures we find that the best result for absolute precision and recall are less than 60%. However,
after an in-depth analysis we found that lots of tags detected by our method that not appear in user attached tags are semantically correct. So we conduct a further subjective evaluation on our method.

6.2.2 Subjective evaluation

Although adopted in the original work, objective evaluation is not so rationale, so we conduct a subjective evaluation in addition that based on experts judgments to compare LSTD with LDA-TR on $MS@k$ and $MP@k$ with the results generated in objective evaluation. In ohloh, the projects usually have more tags than that in SourceForge, and for the test projects used in this case study all have more than ten tags. So in this evaluation we calculate $MS@K(K = 1, 5)$ and $MP@K(K = 3, 5, 8, 10, 15)$. Table 4 shows the subjective evaluation of LDA-TR and LSTD.

Based on the metrics mean success at rank 1 and 5($MS@1$ and $MS@5$), we observe that LSTD achieves higher value at finding good label at rank 1 while both methods can return a good label at rank 5 for almost all projects. While for precision we can see that our approach still works better than LDA-TR on all precision ranks and gets remarkable high precision. From this table we can see that both LDA-TR and LSTD get higher mean precision than that in objective evaluation, which indicates that lots of tags, though not appear in user attaches ones, are considered correct by experts.

6.3 Aggregated Tag Recommendation

Different communities employ different labeling mechanisms and each has its own superiorities and drawbacks. How to integrate these different labels into a uniform one and provide a general tag recommendation is a problem. LSTD, by aggregating different communities’ project profiles to construct a uniform word-label matrix, provides a way out. In this section we construct four word-label matrices: the first three are constructed from the three open source communities and the last one is the aggregation of all these communities (i.e. the union of the first three data sets). Then we employ them to detect the latent topics of 200 projects randomly selected from these communities. Figure 6 presents the changes of mean precision with different precision $k$ values for all matrices. There are four curves with different colors in this figure, representing the precision trends for $M_{oh}$, $M_{fc}$, $M_{sf}$ and $M_{ag}$, which represents the matrix constructed from ohloh, Freecode, SourceForge and aggregated data set respectively.

This figure shows a general trend that the precision declines along with the increase of value $k$. This complies with practical experiences. It also clearly depicts the comparison of semantic power of the first three word-label matrices, which can be ranked descending as $M_{oh}$, $M_{fc}$, and $M_{sf}$. While for $M_{oh}$ and $M_{ag}$, the result is different: $M_{oh}$ outperforms $M_{ag}$ when $k \leq 8$, while $M_{ag}$ is better when $k>8$. The above results comply with the analysis of the open source labeling mechanisms in section 3. In SourceForge, it predefines well-structured categories and lots of them may never appear in any project description, while Freecode and ohloh adopt a kind of folksonomy with much more labels coming from free users who hold more knowledge for projects.

Therefore the word-label matrix from SourceForge will contain less semantic connections among labels and description words. For the difference between $M_{oh}$ and $M_{ag}$, because $M_{ag}$ merges all the project profile sets from three communities and contains a more unified label vocabulary. Therefore more appropriate labels may be available for a specific project. Furthermore, for the curve of $M_{ag}$, its downward tendency is slower than other three. This suggests the complementarity of different profile sets will make LSTD more stable, and reveals the possibility to construct a unified ontology of software projects based on the vocabulary from multiple communities.

Table 5 presents the detailed results of this experiment. Based on the metric mean success at rank 1 and 5($MS@1$ and $MS@5$), we observe that all four matrices perform well in detecting a good label for software projects within the top-$k$ labels detected by our method. The $MS@5$ in the table shows that for almost all projects by all four matrices, the topic detection method can return a good label within top 5. For the columns of mean precisions($MP@k$, $k = 3, 5, 8, 10, 15$) in this table, we can see that the precisions at rank 3 and 5 are fairly high which validate our method in detecting latent topics for software projects.

7. CONCLUSION AND FUTURE WORK

Massive open source software require efficient labeling mechanism to enrich their topic-based labels. This paper firstly analyzed the differences and characteristics of the three typical labeling systems in open source communities and then proposed LSTD approach to detect latent topics for given software: constructs the word-label matrix from mass open source project profiles and then uses the matrix to directly compute the latent label set. After the matrix been built, it can quickly compute the label set with comparatively high precision. The evaluation results verify the effectiveness of LSTD for different community projects and it can be improved by choosing and combining appropriate sets of training profiles from different communities.

Besides, we are developing an oss searching and ranking
system named Influx⁵, which has crawled OSS data from several influential forges and communities (such as SourceForge, ohloh, Freecode and OW2). Currently, Influx can provide some interesting mechanisms for mining oss data, such as cross-forge profiles, synergy analysis, and project ranking in some forges (such as OW2). The topic detection algorithm proposed in this paper will be further tested and integrated to Influx.

Future research will be conducted to improve the performance of LSTD, such as the popularity function in section 4 which can be extended to capture more semantic and social factors of open source projects, like the feature information in software development repositories. Besides, current LSTD simply ignore the words that not appear in word-label matrix, which may reduce the precision of topic detection, and an evolving approach which can add new word-label correspondence incrementally may help.

8. ACKNOWLEDGEMENT

This research is supported by the National Science Foundation of China (Grant No.60903043) and the National High Technology Research and Development Program of China (Grant No. 2012AA010101). Many thanks to Yanxu Zhu, Xue OuYang, Biaocheng Zhao and Yue Yu for discussion and their help on data collecting. We also gratefully thank Jiangning Liu and Sanwei Huang who are CEO and CTO of CVICSE(http://www.cvicse.com), for their hard work and valuable reviews on our test results.

9. REFERENCES


⁵http://influx.trustie.net