Hierarchical Categorization of Open Source Software by Online Profiles

Tao WANG†‡†, Student Member, Huaimin WANG†, Gang YIN†, Cheng YANG†, Xiang LI†††, and Peng ZOU†††††, Nonmembers

SUMMARY The large amounts of freely available open source software over the Internet are fundamentally changing the traditional paradigms of software development. Efficient categorization of the massive projects for retrieving relevant software is of vital importance for Internet-based software development such as solution searching, best practices learning and so on. Many previous works have been conducted on software categorization by mining source code or byte code, but were verified on only relatively small collections of projects with coarse-grained categories or clusters. However, Internet-based software development requires finer-grained, more scalable and language-independent categorization approaches. In this paper, we propose a novel approach to hierarchically categorize software projects based on their online profiles. We design a SVM-based categorization framework and adopt a weighted combination strategy to aggregate different type of profile attributes from multiple repositories. Different basic classification algorithms and feature selection techniques are employed and compared. Extensive experiments are carried out on more than 21,000 projects across five repositories. The results show that our approach achieves significant improvements by using weighted combination. Compared to the previous work, our approach presents competitive results with more finer-grained and multi-layered category hierarchy with more than 120 categories. Unlike approaches that use source code or byte code, our approach is more effective for large-scale and language-independent software categorization. In addition, experiments suggest that hierarchical categorization combined with general keyword-based searching improves the retrieval efficiency and accuracy.

key words: open source software, software profile, hierarchical categorization, software retrieval

1. Introduction

The Internet-based software online repositories such as SourceForge, Ohloh and RubyForge hold large amounts of software projects, which are fundamentally changing the traditional paradigms of software development. Software development is becoming a kind of global collaborative process, and software developers are increasingly resorting to these publicly-available repositories instead of traditional internal ones for solutions or best practices. They can search the repositories for reusable components [1], [2], discover new technical trends in the domain [3], [4], learn solutions from related systems [5], predict and correct faults [6], and retrieve counterparts for help [7]. Efficient retrieval of the desired projects from among the massive resources available is of vital importance for facilitating the utilization of them.

Hierarchical categorization is considered to be an efficient way for retrieving useful information from large scale data repositories [8]. Firstly, it provides a uniform hierarchical categorization for organizing the huge amounts of software over the Internet. Secondly, by combining keyword-based search with hierarchical categorization users can locate desired resources more efficiently and accurately. Such mechanism has been adopted in the popular software repositories like Sourceforge and RubyForge. However, the software projects in these repositories are categorized manually by software managers, and a large proportion of them are not categorized. In addition, a lot of repositories like Ohloh and Freecode are not categorized at all.

Many studies have examined automatic software categorization, most of which rely on analysing software programs. [9]–[11] focus on analysing identifiers and comment terms in the source code to do categorization. McMillan et.al [12], [13] propose a brand-new approach which leverages the third-party API calls in the program as semantic anchors to categorize software. In these works, most of them only experiment on relatively small collections of projects with flat and coarse-grained categories like “Internet” and “Games/Entertainment”. Such coarse-grained categories are not sufficient enough for retrieving the most related ones among the huge amounts of projects in the repositories. Finer-grained and efficient categorization approach is urgently needed.

In this paper we propose a hierarchical categorization approach which leverages the software online profiles as source information for categorization. A profile mainly consists of several types of attributes, i.e., a piece of description, a set of tags and so on. The profiles cover the functional or technical aspects of the software, which make them effective and alternative source information for categorization. Our approach first constructs a category hierarchy and then designs an SVM-based (Support Vector Machine) categorization approach to classify software hierarchically based on the software online profiles. We present the preliminary results and further analysis on weighted combination for profile aggregation to improve the categorization performance.

This paper is a comprehensive version of our previous
work [14]. The main extensions include: 1) An overview of our approach and a graph are added to illustrate our proposal, which makes the proposal more clear to readers; 2) Two widely used classification algorithms Naive Bayes and kNN are employed for building the hierarchical categorization system, and they are compared with the SVM-based approach, which help readers to choose proper classification algorithms for software categorization; 3) Two additional open source repositories OW2 and RubyForge are added to test our approach. We make use of the hierarchical categorization system built upon the SourceForge, Freecode and Ohloh projects to test the projects in other repositories which only have description but no tags. This experiment proves the effectiveness of the hierarchical categorization system; 4) Two feature selection and weighting approaches including tag-based and topic-based techniques are explored. These two techniques can greatly reduce the feature dimensions while achieve similar performance as our approach proposed in the previous paper; 5) The usability of hierarchical categorization for software retrieving by inviting ten undergraduates to compare the hierarchical categorization based retrieval and general retrieval. The results prove the usability of our approach for software retrieval are studied by human judgements.

Overall, the main contributions of this paper include:

- We propose a software categorization framework which classifies software into 123 multi-grained categories hierarchically. To the best of our knowledge, this is the first work to detail automatic hierarchical software categorization and the number of categories is significantly enlarged compared to previous works.
- We explore the multiple types of attributes in software online profiles for categorization and design an efficient combination strategy to aggregate them from multiple repositories. Such web-based software data is less studied for categorization in the previous works.
- We conduct extensive experiments on more than 21,000 software projects with different categorization algorithms and feature selection techniques. The experiments show promising results for hierarchical categorization and prove the efficiency and usability of our method.

The reminder of this paper is organized as follows. Section 2 discusses the related works on software categorization and profile mining. Section 3 describes the hierarchical categorization approach in detail. Section 4 presents the experiment questions and settings and Sect. 5 evaluates our approach. We discuss the efficiency, usability and validity of our work in Sect. 6. Sect. 8 summarizes the paper and discuss the future work.

2. Review of Precious Work

In this section, we review the previous works on software categorizations and software profiles mining.

2.1 Software Categorization

Many works have been conducted on software categorization. According to the source information used for analysis, these works can be mainly classified into two groups.

The first is about categorization based on source code identifiers and comments. [9]–[11] are the typical works that leverage source code information to do categorization. In most of these works, software projects are viewed as documents consisting of source code identifiers and comments. Textual classification approaches are employed to categorize. [11] extracts source code identifiers and comments as documents, then makes use of SVM approach to categorize the software into predefined topic and language categories. MUDABlue [9] and LACT [10] first generate categories from source code and then use LSA and LDA approaches separately to classify software. The second group is software categorization based on other information like API calls [12], [15]. As the source code of many commercial software are not available, McMillan et al. [15] proposed a new categorization approach based on software API calls. The basic idea is that: external APIs and methods in software are grouped by their functions, thus they can be indicators of categories for the software that use these APIs.

Most of these works categorize software based on the source code or byte code information. Considering the scale of the repositories and the complexity of software source code or byte code, such approaches are not efficient for repository-scale categorization. In this paper we explore the capability of software online profiles for scalable and efficient categorization.

2.2 Software Online Profile Mining

As more and more software projects are published with profiles, many researchers pay attention to such online data for different aims and get valuable results.

Dumitru et. al [3] study the software online feature descriptions in Softpedia to assist domain analysis. They propose an incremental diffusive clustering algorithm to discover domain-specific features in the massive amounts of software descriptions, and then recommend features for domain analysts after an initial input is provided. McMillan et. al [16] go a step further. By locating chains of function invocations in the source code, they correlate the mined features with corresponding implementation modules. In addition, tags are widely used to describe software features at repositories. David Lo et. al study the value of software collaborative tags for different aims. In [17] they make use of co-occurrence of tags in Freecode to measure the similarities between pairs of tags and construct taxonomy of tags. Besides, they explore the software categories, license, programming languages and other related tags in SourceForge to find similar applications [18].

These works study the new types of software information, but they analyse these information independently and
focus on one single repository. Differently, we aggregate the software descriptions and collaborative tags across multiple repositories for hierarchical categorization.

3. Methodology

In this section, we first design a hierarchical categorization framework and then discuss the software online profiles for the categorization.

3.1 Overview of Our Approach

Our approach makes use of software different types of online profiles across multiple repositories to do categorization. The overview of our approach mainly consists of data collecting, category system construction, weighted profiles combination, hierarchical categorization and deployment.

The overview of our approach is illustrated in Fig. 1, which mainly consists of the following components.

1) Data crawling and preprocessing. This component first crawls the software homepage, then extracts the online attributes including software descriptions, categories and tags. Then stemming and stop word removing will be done to preprocessing.

2) Hierarchical categories construction. Hierarchical categorization is essential for organizing the large amounts of resources. In this paper, we adapt the widely used categories in SourceForge to construct the hierarchical categories.

3) Weighted profiles combination. The same software’s profiles in different repositories are not always the same. In this paper, we design a weighted approach to combine the different types of profiles to enrich the software information for categorization.

4) Hierarchical categorization system. After constructing the hierarchical categories and the preparing the training examples, we train and deploy the hierarchical categorization system.

3.2 Online Profiles for Software Categorization

Different from the previous works which make use of source code or byte-code information, we leverage the software online profiles for categorization in this paper. As an alternative source information, they are less studied before.

3.2.1 Software Online Profiles in Repositories

We mainly focus on two types of software profile attributes: software descriptions and collaborative tags. The software resources online are often published with brief descriptions which give high-level summaries. For example, MySQL is hosted in several large software repositories and the descriptions of MySQL in different communities are different, each highlights some features of the software. The combination of these descriptions will give a more comprehensive summary of it. Its description in SourceForge is “MySQL is a well-known relational database manager used in a wide variety of systems, including ..., MySQL is a good choice for any situation requiring a database”. The terms like “SQL”, “database manager”, “Oracle” which often appear more frequently in database related software, suggest the category of this software.

In addition to the high-level descriptions, collaborative tagging is widely used in software repositories to annotate resources. For example, in Ohloh MySQL is tagged with “mysql, jdbc, ...”. These annotations aggregate the crowds understanding and reflect the features of the resources, which provide useful information for software categorization [18].

The software profiles present high-level and important features of the resource, which is complementary to the detailed source code or API calls. Source code identifiers or API names often reflect detailed features of the software at a granularity of method, class or package. While the software descriptions and tags provide high-level functional or technical features about the whole software. This is the key motivation for us to explore the capability of these two types of software profile attributes for categorization.

3.2.2 Weighted Combination of Software Profiles

Although software descriptions and tags are both high-level summaries of the resources, software descriptions are often given by software managers to exhibit and popularise the resource. There are many words that are not related to soft-
ware functions or techniques, which are often noise for categorization. Differently, collaborative tags are often labelled by software users, maintainers or other software stackholders to annotate the key features of the resource. These tags covers the functional or technical aspects of the resource. They are often of better quality except some idiosyncratic or misspelling ones. These two types of profile attributes are complementary to each other and the combination of them will give more comprehensive summary. However, because most of the projects are annotated with much less tags compared to the number of words in the descriptions, direct combination of them will dilute the weights of tags.

To balance the impacts of tags and descriptions on categorization, we distinguish the tags from the the common words in descriptions and design a weighted combination strategy to strengthen the weights of tags. We duplicate the tags several times before combining them with software description. The duplicate time is decided according to the ratio between the length of software descriptions and its tags. The intuition behind this is as follow. As annotated by crowds, collaborative tags present the key features of the software, and they should be equivalent to software description at summarizing the software. Thus, the overall normalized term frequency of all the tags for a software should be proximity to that of the description. So we repeat the tags many times to make the total length of them be proximate to that of software description. For tags from different repositories, as we will take steps to only retain the commonly used ones as discussed in the experiments, currently the sources of these tags are not taken into consideration for deciding their weights. The duplication times \( \delta \) is decided according to Eq. (1).

\[
\delta = \alpha \times \sqrt{\frac{\sum_k t_{kj}}{\sum_m t_{mj}}}
\]  

In Eq. (1) \( t_{kj} \) represents the appearance number of term \( k \) in project \( j \) and \( t_{mj} \) is that of tag appears in project \( j \). In practice, some software are only annotated with very few tags which fail to cover all aspects of what description presented. To reduce the influence of these tags in such software, we take the square root over the ratio. In addition, we multiply it by another parameter \( \alpha \) to control the overall duplication times. When \( \alpha \) is set to 0, it becomes the simple combination without duplication which views tags as common words in descriptions.

In this paper, we use the traditional TF-IDF to represent the importance of a term for distinguishing a project at categorization. As the duplication of tags will increase total number of words and the number of tags, such operation will increase normalized term frequency of the tags and decrease the term frequency of the words in description. Because it will not change the the inverse document frequency, the duplication of tags will increase the TF-IDF weights of tags and decrease that of description words.

3.3 Hierarchical Software Categorization

3.3.1 Category Hierarchy Definition

As there are more than one million projects over the Internet, coarse-grained categorization is not efficient. We propose a hierarchical categorization system which includes multi-grained categories that are organized into a hierarchical structure. The category hierarchy is built according to the topics the category covers and the relations among them.

In this paper we model the software category hierarchy as a tree and the relation between a pair of child and parent means “IS-A”. It has the constrains of Asymmetric, Anti-reflexive and Transitive. Figure 2 presents a simple demonstration of the category hierarchy under the category “Multimedia”. In this tree, each subcategory can have zero, one or several subcategories, but can only have one direct parent as constrained above. For a software, the most specific categories can be internal or leaf categories.

In this hierarchy, a category is allowed to have one single subcategory, like the node “Players” in Fig. 2. Because it is not mandatory for projects to be assigned with leaf categories, this setting makes sense. This is motivated by the requirement to category software as specific as possible. For projects under “Players”, they will be further tested to see if can be assigned with the more specific “MP3”.

3.3.2 Hierarchy Categorization Learning

There are many ways to build the hierarchical categorization model, which can be mainly classified as big-bang approach and top-down approach[19]. Big-bang approach treats all the categories at once and learns a single classifier for the entire hierarchy. Such approach will reduce the total size of classification model considerably. However, as the total number of categories is often large, it is difficult to build a single accurate classifier. Top-down approach adopts a different strategy. It builds multiple local classifiers and predicts each subcategory separately. In [20] it has proved that the top-down approach is superior to the big-bang approach. Thus, in this paper we adopt a top-down approach for training the model. We build local binary classifier per node in the tree except the virtual “Root", which is the mostly used approach in literature to construct the hierarchical categor-
To build the local classifier for each node in category hierarchy, there are two key issues to consider. The first is what classification approach to use and the second is how to select the positive and negative examples for training. For the first issue, there are quite many classification approaches to build the classifier like SVM, kNN, Decision Tree and so on. In this paper we choose different approaches of SVM, kNN and Naïve Bayes as the basic classification algorithm to build the hierarchical categorization system. We choose SVM as our final classifier because it has been proved superior to others for software categorization.

For the second issue, we adopt a “sibling” strategy in our framework. For a given category node in the hierarchy, the examples of both the node and its descendants are viewed as positive examples, and those categorized with its siblings as well as the descendants of the siblings are viewed as negative ones. For those categories that have no siblings, the negative examples consists of those that categorized with the siblings of its nearest ancestor. For example, in Fig. 2 the category “MP3” has no sibling, so its negative examples are those labelled with “CD Audio” which is the sibling category of its parent. Such strategy is adopted to avoid the serious imbalance problem [20].

3.3.3 Hierarchical Categorization Prediction

The hierarchical software categorization problem is a non-mandatory leaf node prediction problem [21]. It means that the most specific predicted category for a testing software can be any node (i.e. internal or leaf node) in the category hierarchy except the “Root”.

For a given software, we still take a top-down approach to predict its categories. We first test the given software over all the first-level categories with corresponding local classifiers. Then we only go down to test those subcategories whose parents have been predicted positive. One thing we need to note is that one software may belong to two or more categories of the same level in practice, which is a multi-label classification problem [22]. As we adopt binary SVM as our basic local classifier which predicts the categories of the same level separately, the multi-label problem can be solved naturally.

4. Experiments Design

In this section, we describe the experiment questions, experiment dataset and settings as well as the corresponding evaluation metrics.

4.1 Research Questions

To explore the effectiveness of different software profile attributes for categorization and comprehensively evaluate the proposed hierarchical categorization framework, we focus on the following four experiment questions.

- **RQ1**: Which type of profile attributes is more effective for categorization, software descriptions or collaborative tags?
- **RQ2**: Will combination of different attributes improve the performance?
- **RQ3**: Will the constructed categorization system be effective on external repositories?
- **RQ4**: Will different feature selection and weighting strategies be effective?
- **RQ5**: Are software online profiles as effective as API calls extracted from source code or byte code?

For experiment question Q1, we explore the effectiveness of the two types of profile attributes for categorization: software descriptions and collaborative tags. In Q2, we aim to see if the designed combination strategy work or not. For Q3, we employ the constructed categorization system on other repositories to see if it is also effective. For Q4, we do further experiments and try different feature selection and weighting strategies to see if they will work. For Q5 we compare the capability of software profiles with API calls for categorization.

4.2 Dataset and Experimental Settings

To address the above research questions and validate our approach, we firstly focus on three large and popular open source repositories: SourceForge, Ohloh and Freecode. SourceForge is one of the largest and most popular open source community. It has predefined a hierarchical category system with 363 categories. Ohloh and Freecode are both large open source repositories which collect more than 400,000 and 45,000 software project respectively. In Ohloh and Freecode, they adopt a collaborative tagging mechanism to annotate and organise these collected resources. We crawl project profiles from Ohloh and Freecode to enrich the profile of software in SourceForge.

In addition, we testing our approach and the constructed hierarchical categorization system on software projects in another two open source repositories RubyForge† and OW2‡. These two repositories employ categorization mechanism to organize the projects in them as that in SourceForge. In these two repositories, some of the categories used are the same as that in SourceForge. So we choose the projects which are assigned with such categories as the ground-truth to test the hierarchical categorization system build upon SourceForge, Freecode and Ohloh. This make it possible to automatic the evaluation process.

To get the software online profiles of projects in the first three repositories, we crawl the software homepage, parse to extract the profile attributes including descriptions, categories and tags, and then select those projects that exist in more than two repositories which not only have categories in SourceForge but also have tags in Ohloh or Freecode. We do stop words removing and stemming over the descriptions

†http://rubyforge.org/
‡http://www.ow2.org/
and tags, and then retain those which have combined descriptions of more than 10 words. Besides, we only retain such tags that appears more than 50 projects. After these steps, we get a number of 18,032 unique software and a total of 5,429 unique tags.

For Rubyforge and OW2, we use the dataset published by FLOSSmole in Mar. 2014 [23]. We adopt similar steps to preprocess these projects’ descriptions as we do on projects in the first three repositories. And we only retain these projects whose descriptions are of more than 10 words. The detailed information of the dataset is shown in Table 1.

The projects from SourceForge has at least one category for each and has descriptions of about 20 words and 3 categories in average. There are 9,813 and 10,357 projects in Ohloh and Freecode which also exist in the 18,032 SourceForge projects. These software in Ohloh and Freecode have similar description length and average tags. The retained projects only account for a small proportion of the total software in these repositories because we take a strict matching strategy on full-name to search projects and a large proportion of projects in Ohloh and Freecode do not have any tags. While for projects in OW2 and RubyForge, after filtering these projects which have no categories appear in the SourceForge categories and these projects which have short descriptions of less than 10 words, we get a number of 57 projects in OW2 dataset and 3,077 ones in RubyForge dataset. And there are 27 categories which appear in SourceForge categories, and that for RubyForge is 96.

To build the category hierarchy we reform the predefined one in SourceForge. We first transform the original DAG structure hierarchy to tree structure, then prune the hierarchy by deleting categories with only a few examples and combining similar ones. Finally we construct a single category hierarchy with a virtual “Root” category. For category pruning, the thresholds for deletion are set to 500, 100, 50, 50 for the four levels categories. In the end we constructed a uniform hierarchy consists of 123 categories with four levels. Figure 2 presents a part of the constructed category hierarchy under “Multimedia”.

In the constructed category hierarchy, there are 12 top categories at level one including “Multimedia, Games/Entertainment” and so on. The rest three levels have 61, 41 and 9 categories respectively. The average number of positive samples for the first level categories is 2215.33. And it decreases dramatically to 382, 196 and 120 for the next three level. This will affect the performance of our approach and more samples will be included in the future.

<table>
<thead>
<tr>
<th>Repository</th>
<th>Num. of software</th>
<th>Avg. description length</th>
<th>Num. unique categories</th>
<th>Avg. categories</th>
<th>Num. unique tags</th>
<th>Avg. tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>SourceForge</td>
<td>18,032</td>
<td>19.69</td>
<td>307</td>
<td>2.98</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Ohloh</td>
<td>9,813</td>
<td>20.84</td>
<td>-</td>
<td>-</td>
<td>5,373</td>
<td>5.73</td>
</tr>
<tr>
<td>Freecode</td>
<td>10,357</td>
<td>25.17</td>
<td>-</td>
<td>-</td>
<td>940</td>
<td>4.85</td>
</tr>
<tr>
<td>OW2</td>
<td>57</td>
<td>18.01</td>
<td>27</td>
<td>2.54</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RubyForge</td>
<td>3077</td>
<td>15.90</td>
<td>96</td>
<td>2.34</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

4.3 Evaluation Metrics

In previous works like [12], [15], the precision, recall and F-Measure are widely used to evaluate the performance of flat categorization system. In this paper we adopt hierarchical metrics including hierarchical precision (hP), hierarchical recall (hR) and hierarchical f-measure (hF) over each category as defined in Eq. (2).

\[
hP = \frac{\hat{P}_i \cap \hat{T}_i}{\hat{P}_i}, \quad hR = \frac{\hat{P}_i \cap \hat{T}_i}{\hat{T}_i}, \quad hF = \frac{2 \cdot hP \cdot hR}{hP + hR} \tag{2}
\]

In Eq. (2), for each category \(i\), \(\hat{P}_i\) is the predicted sample set for category \(i\). It consists of software whose categories are predicted as category \(i\) or the descendants of \(i\). \(\hat{T}_i\) is the true sample set that consists of all projects labelled with category \(i\) and the descendants of \(i\). The three metrics \(hP\), \(hR\) and \(hF\) represent the average precision, recall and F-Measure for each category over all the testing examples where the hierarchical structure is concerned.

To measure the average performance over all the categories, we make use of Micro Average on hierarchical Precision (Micro-hP), Recall (Micro-hR) and F-Measure (Micro-hF) as Eq. (3) which are similar to these used in [19], [24].

\[
Micro-hP = \frac{\sum_{i} (\hat{P}_i \cap \hat{T}_i)}{\sum_{i} \hat{P}_i}, \quad Micro-hR = \frac{\sum_{i} (\hat{P}_i \cap \hat{T}_i)}{\sum_{i} \hat{T}_i}, \quad Micro-hF = \frac{2 \cdot Micro-hP \cdot Micro-hR}{Micro-hP + Micro-hR} \tag{3}
\]

5. Experiment Evaluations

In the experiments, we first build the hierarchical categorization model and then test on large amounts of projects. Different classification algorithms are used as the basic classifier and the parameters are set as default except the parameter \(c\) for SVM. We set \(c\) to 0.5 as it achieves the best performance after a simple test. We first use 5-fold cross validation and hierarchical metrics to measure our approach and then compare our approach with a previous work which categories software based on API calls.

5.1 RQ1: Software Descriptions and Collaborative Tags for Categorization

Two different types of profile attributes are explored in this work: software descriptions and collaborative tags. These
attributes have not been extensively studied for software categorization before. In this section we conduct experiments with Naïve Bayes, kNN and SVM on four sets of different attributes. Firstly we do experiment based on SourceForge software descriptions. Secondly we aggregate the software descriptions from the three repositories to test. Then we experiment on combination of software tags from Ohloh and Freecode. Finally, we simply combine software descriptions with tags from all three repositories to do categorization.

Figure 3 shows the comparison of the four sets of experiment results. Comparing the different algorithms, from the third one in Fig. 3 we can see that the SVM achieves best results over all different attribute datasets, and the Naïve Bayes get the poorest results. While for different online attributes, synthesizing the precision and recall we can see that experiments on dataset of tags alone and the combination of tags and descriptions achieve similar F-Measure and perform better than that on software descriptions.

Table 2 shows the detailed experiment results based on SVM algorithm. Based on SourceForge descriptions, we get an overall micro precision of about 58.68% and recall of 48.37%. By aggregating the descriptions from other repositories, the precision is only improved slightly while the recall is almost the same. This is because that a large proportion of homonymous software’s descriptions in different repositories overlap a lot, which fails to provide much additional information. For collaborative tags, though each software project has much less tags than description words, the experiment based on combined tags achieves better results. The micro precision is similar to that of Combined Des., but the micro recall shows a significant improvement of about 13.68% and F-Measure gets an improvement of 8.50%. This is due to the better quality of collaborative tags. Different from software descriptions which often contains many words that reflect no technical or functional features, most of the tags have specific meaning and reflect some aspects of features for the resource. Thus, categorization based on collaborative tags achieves better results.

In the fourth row, we simply combine the software descriptions and tags. Based on this data we get better precision but worse recall than that of tags. Compared to collaborative tags, such combination treats each tag as common word in description. On the one hand, such combination aggregates more information of the software; On the other hand, it introduces more noisy words and dilutes the weights of tags. Thus, it fails to improve the categorization performance over that of Combined Tags.

5.2 RQ2: Weighted Combination of Different Attributes for Categorization

To make full use of the descriptions and tags for categorization, we propose a weighted combination strategy which aims assign more weights to more important attributes by duplication. In Eq. (1) we design a parameter $\alpha$ which controls the overall duplication time for tags. Figure 4 presents the performances for different values of $\alpha$.

It shows that as the $\alpha$ increases from 0 to 1, both precision and recall have a great improvement, and the F-Measure improves from about 62.41% to about 68.38%. This verifies the effectiveness of our weighted combination strategy. While the parameter $\alpha$ keeps on increasing from 1 to 3 and larger, the precision decreases slowly, and the recall and F-Measure keep almost the same. This is reasonable. As $\alpha$ increases, the weights of tags will become much larger than that of words in descriptions. This leads to the result that the descriptions have only slight effect on distinguishing software categories.

Overall, from the above results we can find that collaborative tags are more effective attributes than descriptions for categorization. Nevertheless, those two types of attributes are mutually complementary, and the weighted combination of them which assigns greater weights to tags properly will improve the overall performance. In addition,
from Table 2 we can see that, even without the additional tags and descriptions, our categorization framework is still applicable by only using the descriptions from SourceForge and achieves an overall F-measure of about 53.02%.

5.3 RQ3: Experiments on External Repositories

In this subsection, we evaluate the constructed categorization system in RQ2 on two external software repositories OW2 and RubyForge. We experiment with projects in OW2 and RubyForge to see the performance of the category system on external repositories. Different from the projects in the SourceForge dataset which are annotated with tags in Freecode and Ohloh, the datasets in OW2 and RubyForge have no tags, and only software descriptions are available. In their descriptions, there are words which are the same as the tags in Ohloh and Freecode. In this experiment, we view such words as their tags and improve their weights as we do in RQ2. The detailed results are shown in Table 3.

As shown in the table, the average precision over all projects in OW2 is about 58.03%, the Micro-hR is 84.29%, and the F-Measure reaches 60.31%. The results on RubyForge projects is a little higher than that for OW2, and the F-Measure reaches 63.75%. Compared with the best performance in RQ2, the precision is a little lower while the recall is higher. Such differences mainly due to two reasons. The first is that the these projects are of no tag, which affects the precision. The second is that most of the testing projects are assigned with high-level categories (mainly the top 2 levels of categories in the constructed four level category hierarchy). This will improve the recall of the categorization and meanwhile decrease the precision metric.

5.4 RQ4: Feature Selection for Categorization

5.4.1 Tag-Based Feature Selection

As shown in RQ1 and RQ2, tags performs better than descriptions in software categorization, and the weighted combination of tags and descriptions can improve the categorization performance. In this section, we use these tags as filter and only these description words that appear in tags will be retained, and then we combine the tags and retained descriptions to do categorization.

The statistics of tags and common words in the dataset used in RQ2 are illustrated in Table 4. The number of distinguish tags is 5,328, which is much smaller than that for description words of 17,381. The total number of tags in the dataset takes more than 80 percent of the total terms in descriptions, which implies that the majority of the description words are also appear in tags. By filtering the descriptions with tags will greatly reduce the feature dimension.

After filtering the descriptions, all the retained words used to be used as tags. However, we still view these words in descriptions different from the real tags because they are not so carefully chosen as the real tags, and we adopt the weighted combination strategy as we did in RQ2. The detailed experiment results are shown as Table 5.

The Micro-hF based on filtered features improves from 63% to 68% when $\alpha$ increases from 0 to 1. If $\alpha$ keeps on increasing to 2 and 3, the Micro-hF keeps almost the same. This trend is similar to that in the weighted combination in RQ2. For the category performance, the best performances of experiments based on data before and after filtering are similar. While from Table 4 we can see that the dimension of feature space will dramatically drop from 17,381 to 5,328 after filtering, which will delete those irrelevant words, re-

Table 3  Categorization performance on OW2 and RubyForge projects.

<table>
<thead>
<tr>
<th>Repository</th>
<th>Micro-hP</th>
<th>Micro-hR</th>
<th>Micro-hF</th>
</tr>
</thead>
<tbody>
<tr>
<td>OW2</td>
<td>0.5803</td>
<td>0.8429</td>
<td>0.6031</td>
</tr>
<tr>
<td>RubyForge</td>
<td>0.6130</td>
<td>0.8225</td>
<td>0.6375</td>
</tr>
</tbody>
</table>

Table 4  Statistics of tags and common words in the experiment dataset.

<table>
<thead>
<tr>
<th></th>
<th>Num. distinguish terms</th>
<th>Num. total terms</th>
<th>Num. software</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>5,328</td>
<td>17,381</td>
<td>898,501</td>
</tr>
<tr>
<td>des. ratio (tag/des.)</td>
<td>31.23%</td>
<td>898,501</td>
<td>80.03%</td>
</tr>
<tr>
<td>software contains tag in des.</td>
<td>18,032</td>
<td>18,032</td>
<td>100%</td>
</tr>
</tbody>
</table>

Fig. 4  Categorization performance with different values of $\alpha$ based on weighted combination of profiles.

Table 4  Statistics of tags and common words in the experiment dataset.
reduce the cost and accelerate the process of model training and prediction.

5.4.2 Topic-Based Feature Selection

In this subsection we use latent topic of documents as features to do categorization. Different from the tag-based feature selection approach, topic model LDA will view documents at the topic level instead of term level. Using LDA, each document will be represented as a distribution of topics, each of which is assigned with a weight of probability for representing this document. Then we use the topics as the features and the probabilities as the corresponding weights to categorize the documents.

We firstly to specify the number of topics to generate and the number of iterations for generating these topic distributions. In this subsection we set the number of iterations as 1,000 which will converge and experiment on different topic numbers to see the effect.

Table 6 presents the detailed results for categorization based on topic model. As the number of topics increases, the Micro-hF will be improved. But when the number of topics reaches a specific threshold, the performance will not improve too much. This is due to the fact that the number of topics reflect the granularity of the generated topics, and too raw-grained topics will not be able to distinct the documents. We set the max number of topics to 4,500 which is similar to the number of distinguished tags at which the Micro-hF is about 57.50%. This performance is similar to the previous ones, but the topic model training takes quite a longer time.

Table 6 Categorization performance based on topic distributions.

<table>
<thead>
<tr>
<th>Number of topics</th>
<th>Micro-hP</th>
<th>Micro-hR</th>
<th>Micro-hF</th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
<td>0.6536</td>
<td>0.4388</td>
<td>0.5250</td>
</tr>
<tr>
<td>500</td>
<td>0.6944</td>
<td>0.4691</td>
<td>0.5600</td>
</tr>
<tr>
<td>1000</td>
<td>0.7119</td>
<td>0.4679</td>
<td>0.5647</td>
</tr>
<tr>
<td>2000</td>
<td>0.7054</td>
<td>0.4704</td>
<td>0.5644</td>
</tr>
<tr>
<td>4500</td>
<td>0.6912</td>
<td>0.4923</td>
<td>0.5750</td>
</tr>
</tbody>
</table>

Compare the tab and topic-based feature selection approaches, we can see that the tag-based feature selection can greatly reduce the feature dimension without losing much performance, while for topic-based feature selection, although it can reduce the feature dimension to desired level, but it is hard to decide the best granularity and the topic model training is time-consuming. Thus, tag-based feature selection is better at balancing performance and efficiency.

5.5 RQ5: Weighted Combination of Software Profiles and API Calls for Categorization

APIs are grouped into packages and libraries according to their functions and thus good indicators of the software category. McMillan et. al [15] explore API calls and their experiments suggest that API calls are as effective as source code identifiers and comment terms for categorization. In this experiment we compare our approach by using profile combination with \(\alpha = 1\) with their best setting (using API packages).

In the experiment, we test our approach on the dataset used in their experiments. Among their SourceForge dataset, we search for the projects which are tagged in Ohloh or Freecode and get a total of 849 ones which are used as the testing set. Then we train the categorization model based on the dataset in which the testing set are rejected. The final results for the 22 categories are shown in Table 7.

As can be seen in Table 7 our approach performs better on some categories while worse at some others. Overall, the two approaches achieve similar F-Measure over all the 22 categories as shown in the last row of the table. The name of API packages and classes are implementation-related attributes which mainly reflect the functionality of the package or the class, which is fine-grained features. While the software profiles emphasis on high-level summaries of the resource. These two types of attributes reflect the features of the software from different perspectives with different granularity. The results suggest that the software profiles are effective alternative to software APIs for categorization.

It should be noted that we do categorization hierarchi-
cally with more finer-grained categories. Our approach organizes these categories hierarchically and achieves high accuracy at these specific categories as well. For example, in our constructed category hierarchy, the category “Compilers, Testing” in the 22 categories are organized under the category “Software Development”. The “Software Development” category are divided into more specific ones like “Build Tools, Object Oriented, Algorithms, Quality Assurance” and so on. Table 8 presents the results for the categories under “Software Development”.

### 6. Hierarchical Categorization for Retrieval

Internet-based software development greatly relies on efficient categorization of the massive software in repositories. In this section we analyse the efficiency of our approach for categorizing the Internet-scale repositories and how hierarchical categorization will help software retrieval.

#### 6.1 Efficiency Study

The cost of software categorization mainly consists of training data obtaining and preprocessing, categorization model training and prediction. To get the profiles of all the 417,344 software in Ohloh, we design and deploy a crawler and profile extractor in a server (8×2.13G CPUs, 16GB RAM and 2TB storage) which is connected to the Internet with network bandwidth of 100M. It takes less than 3 days to get all the software profiles in Ohloh, and the total size of the extracted profile attributes is about 100 MB. The time cost for model training and category prediction mainly depends on the classification algorithm used. We do our experiments using SVM algorithm on a computer of Intel(R) Core(TM) i5-3320M CPU @ 2.6M with 4GB RAM. The total time for five-cross validation over 18,032 projects is about 620 seconds, in which the training time is about 540 seconds and testing time is about 80 seconds. Based on this we can estimate that the time for categorizing the total 417,344 software in Ohloh will take about 31 minutes once the categorization model is built.

In contrast, categorizing the repository software based on source code will be more complex. Firstly, considering the huge amounts of projects in the repositories, to get the source code of all the software is time-consuming. Secondly, the analysis of the source code is quite complex. For example, in June 2011 MySQL has about 1,333,855 lines of code and 298,918 lines of comments. To parse the source code of it for identifiers and comments is quite a time-consuming process, let alone the huge amounts of software which are implemented in various programming languages. Although the current hardware can greatly accelerate this process, the cost on CPU or memory is much bigger than that for categorization based on online profiles to achieve similar efficiency.

#### 6.2 Usability Study

Hierarchical Category can be used in different ways for retrieving software. Users can simply search with a keyword, or browse the hierarchical categories for desired software directly, and they can also combine the hierarchical category with general keyword-based searching to narrow down the searching scope.

We invited ten undergraduate students to use the search engine in three ways (suppose the desired software is under the category C and often be queried with keyword Q, the general keyword searching is based on the lucene indexer and searcher): (A) just use keyword Q as query to search; (B) use keyword plus category (Q + C) as query to search; (C) use keyword Q to search and then use category C to filter. Each of these students is asked to search 20 times respectively, and they are required to rank the performance of the three ways for searching as Best, Normal or Worst based on the top 10 returned results. Table 9 shows the results.

The evaluation results shows that general keyword-based searching plus category filtering is preferred by the majority of these respondents as it is voted as best for 61.11% of the queries. These correspondents suggest that the hierarchical categorization can be useful for software retrieval. Firstly, the hierarchical categorization can greatly narrow down the searching scope; Secondly, the returned software are more relevant than keyword-based searching alone.

### 7. Threats to Validity

There are some threats to validity which may affect the experiment results of our approach. One is that the software may be incorrectly categorized in SourceForge or incorrectly tagged in Ohloh and Freecode. In our paper, we minimize such threat by including as many samples as possible to reduce the ratio of such incorrectly categorized software. Another threat is that the same name software we crawled from multiple repositories may not be the same software. To minimise such threat, in this paper we adopt a strict full-name matching strategy to filter the false ones.

The software tags have a great impact on the performance of our categorization approach and the tag shortage
may affect the generality of our approach. However, for those that have no tag we can classify them based on their descriptions. As shown in Table 2, the categorization precision based on descriptions achieves 58.68% as well. In addition, as social tagging is becoming widely accepted, more and more software will be annotated with tags.

8. Conclusion and Future Works

Internet-scale software repositories require more scalable and finer-grained categorization. In this paper we propose a hierarchical categorization approach by mining software online profiles across multiple repositories. The proposal categorizes software hierarchically with 123 categories based on online profiles, and is sufficiently scalable that it can handle the huge amounts of software in repositories efficiently. Extensive experiments have been conducted with different basic classification algorithms and feature selection techniques. The results show that our approach achieves promising performance over the whole category hierarchy, and such categorization greatly improve the efficiency for retrieving software. We have developed an open source software searching system named Influx1, and a demo of the hierarchical categorization system proposed in this paper has been integrated into Influx and can be tried now.

There are several possible ways to improve the categorization performance of the proposal. Firstly, more software can be crawled from websites over the Internet to enrich the training set. Secondly, in this paper we treat the tags from different repositories equally. Further analysis about the characteristics of these tags will be done to explore their potential for categorization. In addition, software attributes of API calls and software profiles reflect software features at different granularity, and we will study how to make full use of the strengths of these different attributes. Besides, current category hierarchy contains 123 categories, which do not be able to cover all the categories of the open source software. In recently years, new software techniques like cloud computing, big data and so on are emerging. In the future work, we will build a continuous updating system which can include new emerging categories into the system.

Acknowledgments

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. 2012AA011201). We would like to thank Xiao Li, Yue Yu and Li Fang for their collaboration. We would also thank the anonymous reviewers for providing us constructive comments and suggestions.

References


1http://influx.trustie.net

[18] F. Thung, D. Lo, and L. Jiang, “Detecting similar applications with

Tao Wang born in 1984, Ph.D. candidate in Computer Science, National University of Defense Technology (NUDT). His work interests include open source software engineering, machine learning and knowledge discovering in open source software.

Huaimin Wang born in 1962, professor in National University of Defense Technology (NUDT). His current research interests include middleware, software Agent, trustworthy computing.

Gang Yin born in 1975, associate professor in National University of Defense Technology (NUDT). His current research interests include distributed computing, information security and software engineering and machine learning.

Cheng Yang born in 1991, Ph.D. candidate in Computer Science, National University of Defense Technology (NUDT). His research interests include machine learning and big data analytics.

Xiang Li born in 1988, Ph.D. candidate in Computer Science at The University of Western Ontario and NUDT. His research interests include data mining, machine learning and related real-world applications.

Peng Zou born in 1957, professor in National University of Defense Technology (NUDT) and Academy of Equipment. His research interests include network and information security, distributed computing and software engineering.