ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 10, October 2014

Improving Detection Performance of Duplicate Bug Reports Using Extended Centroid Features

Nhan Minh Phuc

Falculty of Information Technology, Tra Vinh University, Viet Nam

Abstract: According to recent work, detection on duplicate bug reports has received much attention. One of the reasons is that duplicate bug reports may consume time of bug triagers and software developers. In previous studies, many schemes have been developed for using text mining techniques or using the information retrieval and natural language processing techniques. In this paper, we propose a method to improve centroid characteristics by adjusting centroids with better initial values than based on Class-Feature-Centroid (CFC) [12]. With the effectiveness of CFC, the centroid-based approach can obtain further improvements for detection performance. The method includes two steps. First, we extract inter-class and inner-class term indices from the corpus. Second, we enhance centroid calculation based on class features. Moreover, for similarity measure we also adapt the calculation of the traditional cosine similarity by denormalized cosine measure which is also used in [12].

Keywords: Bug Reports, Duplication Detection, Feature Weighting, Class-Feature-Centroid

I. INTRODUCTION

Defect reporting is one of important parts of software maintenance process. For many open source software projects such as Bugzilla, Firefox so on, they alway have an issue tracking system which is used to receive bug reports which allow users to describe, track, classify, comment and report dedects. Normally, the bug reports are analyzed by a triager and assigned them to the appropriate programmers who determine the order in which to work on assigned reports based upon such information as a report's priority for fixing bugs. With the bug tracking systems, users can report new bugs, track statuses and comments of bug reports on existing bug reports. This helps to reduce time and cost for bug report triaging. According to [8], on average processing a bug report needs around 30 minutes for the collected data of Sony Ericsson. However, this takes only 30 seconds using automatic detection techniques by natural language processing (NLP) [8]. Although bug tracking systems have signicant benefits, they have challenges in handling duplicate bug reports. Since the system of bug reports process is often uncoordinated, manual inspection is needed to detect whether these bug reports has been reported before.

Duplication Detection Problem

Duplicate bug report problem can be understood as following: two or more bug reports are said to be duplicate if they describe the same bug issue or software defect and thereby have the same solution. In the past, many researchers have noticed the severity of the duplicate bug report problem [1, 2, 6, 8, 10]. For example, in 2006, reported that duplicate reports made up 36% of all reports in the Firefox repository, 17% of Eclipse Platform reports, 14% of Apache 2.0 reports and 13% of Fedora Core reports [6]. These percentages correspond to large numbers of reports: 14159 for Firefox, 9181 for the Eclipse platform, 422 for Apache 2.0, and 4792 for Fedora Core [6]. Moreover, the previous Eclipse dataset collected from October 2001 to August 2005 has 18,165 bug reports, and 20% of the reports are duplication [1]. In addition, 30% of 2,013 reports in the Firefox dataset collected from May 2003 to August 2005 are duplicate. According to previous studies, the problem of duplication detection can be characterized by identifying two or more bug reports that describe the same software defect. Typically, the duplicates can be further classified into two types. Firstly, the duplicate bug reports describe the same failure situation. Secondly, the bug reports describe the different failures with the same source of the software defect. However, since the second duplication type is more complicated and may involve different vocabulary for different bug reports then its detection usually cannot be effective by only exploring the textual information of the bug reports. For solving these problems, the researchers found that specific information in program need to consider such as executable traces. However, these relates to private information, so among research focuses on first type.

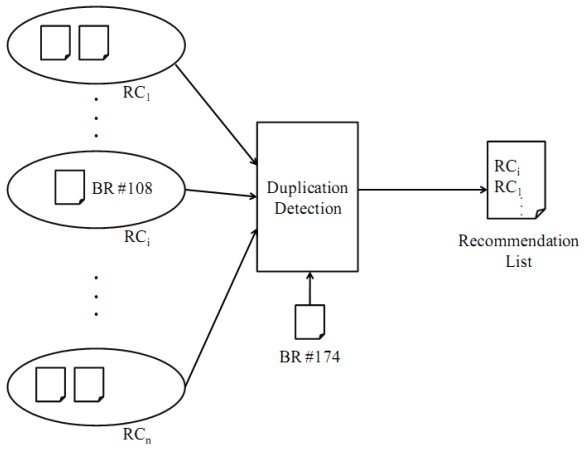

Figure 1.2: The duplication detection is performed to give a recommendation list to predict the likelihood of an incoming bug report being a duplicate of RC_i.

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 10, October 2014

In open software repositories, bug reports contain pre-defined fields, free-form text, attachments, and dependencies. The pre-defined fields provide a variety of categorical data about the bug reports. The report identification number, creation date, and reporter are fixed when the report is created. For some other values such as the product, component, operating system, version, priority, and severity after submitting to the system they are led by the reporter and sometimes these may also be changed over the lifetime of the report. In general, the reports usually consist of some error messages, meta-information, a short summary and long description.

The duplication detection problem in this research is processed as follows. For a software project, the historical bug reports are first classified into n report clusters (RCs). Each RC has a master bug report (MBR). If an RC has more than one bug report, the bug reports in the RC have the duplicate relationships. For each incoming bug report BR_x, the duplication detection is performed to generate a recommendation list that shows the likelihood order of BR_x being a member of RC_i. The duplication relationship is determined if one of the following conditions is satisfied:

1. For a master report BR_m, a bug report BR_i has been resolved as duplicate with a reference to BR_m in the bug tracking system, and the report status is closed.
2. For two bug reports BR_i and BR_j, if they are marked as the duplicate of BR_m, BR_i is a duplicate of BR_j, and vice versa.

3. If there is another bug report BR_k that is marked as duplicate of BR_i, BR_k is also a duplicate of BR_m. This property is called the transitivity.

Table 1.1: An example of master bug report (BR) and its duplicates

# of Cluster	Master BR and its duplicate	
1	793	689
2	797	792, 703
...	...	...
n	2610	2520, 2407, 525

Generally, when the triagers identify that a new report is a duplicate of an old one, then the new one is marked as duplicate. In this research, given a set of reports on the same defect, the earliest one in the set is marked as the master report as seen in Table 1.1. In this example, last column of each row presents as master bug report and other columns are its duplicates. In Figure 1.1, this bug report #174 is identified as a duplicate in the resolution field. On the bottom of the figure show that this bug report is a duplicate of bug report #108. In this case, these two bug reports are also formed a report cluster. For a software project, the historical bug reports are classified into *n* report clusters. To build the clusters, we use the comments of bug reports to create a mapping file as seen in Table 1.1, which describes a form of the map file which the each row presents as one cluster. Figure 1.2 shows an example for ArgoUML in which a recommendation list is generated for BR # 174. The RC_i containing the MBR # 108 is marked according to the detection judgement.

The rest of the paper is organized into four sections. Section II gives a brief overview of related work. Section III presents the proposed scheme in which extended centroid features are used to improve the performance of duplicate detection. In section IV, the empirical study on different open source projects is elaborated to demonstrate the effectiveness of the proposed scheme. Finally, section V concludes the paper.

II. RELATED WORD

In this chapter, we present related works of the research problem and technique that related to our study. In [6], Hiew developed a centroid-based incremental cluster scheme and get successful for detection of duplicate bug reports around 20% - 50% in recommendation list of top 7 with projects of Eclipse: 20% and Fedora: 31%. Moreover, the paper published in 2007, Runeson, Alexandersson, and Nyholm have proposed a software tool using natural language processsing (NLP) techniques for software projects of Sony Ericsson [8]. However, result of recall rate also achieved in range 31% - 42% with top between 5 and 15 for recommendation list.


Full text issue listing

Null pointer exception under jdk1.3

Issue #: 174 **Component:** argouml **Version:** 0.7 **Platform:** PC
OS/Version: Windows 98 **Status:** CLOSED **Issue type:** DEFECT **Priority:** P1
Resolution: DUPLICATE **Assigned to:** issues@argouml **Reporter:** a.m.dearden
Subcomponent: Other **Target milestone:**---
URL:
Summary: Null pointer exception under jdk1.3
Status whiteboard:
Description:

```
C:\jdk1.3\Jars>java -jar argouml070.jar
making MultiEditorPane
making Diagram
making Table
making DetailsPane
making ToDoItem
making WizDescription
Exception in thread "main" java.lang.NullPointerException
        at javax.swing.event.SwingPropertyChangeSupport.firePropertyChange(Unkno
wn Source)
        at javax.swing.AbstractAction.firePropertyChange(Unknown Source)
        at javax.swing.AbstractAction.setEnabled(Unknown Source)
        at uci.uml.ui.ToDoItemAction.updateEnabled(Actions.java:1872)
        at uci.uml.ui.TabToDo.updateActionsEnabled(TabToDo.java:168)
        at uci.uml.ui.TabToDo.setTarget(TabToDo.java:155)
        at uci.uml.ui.TabToDo.<init>(TabToDo.java:126)
        at java.lang.Class.newInstance0(Native Method)
        at java.lang.Class.newInstance(Unknown Source)
        at uci.uml.ui.ConfigLoader.loadTabs(ConfigLoader.java:74)
        at uci.uml.ui.DetailsPane.<init>(DetailsPane.java:78)
        at uci.uml.ui.ProjectBrowser.<init>(ProjectBrowser.java:182)
        at uci.uml.Main.main(Main.java:148)

------- Additional comments from Toby.Baier@gmx.net Tue Jul 18 08:37:59 -0700 2000 -------

please look into the bug database before posting one...

*** This bug has been marked as a duplicate of 108 ***
```

Figure 1.1: An example of a duplicate bug report in ArgoUML. In this example BR # 174 is a duplicate of BR # 108

Another method in recent research which is evaluated high by recall rate is better than others is extra textual information and the execution information of Wang et al [6]. With enhancing of software execution information, his approach can identify from 67% to 93% recall rate of duplicate bug reports for Firefox project. However, the improvements for execution information can be very complicated and require advanced instrumentation tools. In addition, another drawback is that this method relate to privacy concern of user behavior information from the

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 10, October 2014

execution information. Sun et al. [9] in 2010 explored the Support Vector Machines (SVM) model with selecting more relevant textual features of bug reports and result for exploring 54 features help their model can get around from 48% to 52% recall rate with top 5 for recommendation list. However, this method used the IDF (inverse document frequency) to improve the discriminative. This leads to large computation when the number of data set increase day by day. Moreover, the SVM model also need to retrain when a new bug report comes, this can cause a great cost in the detection process.

Also in the same year 2010, feature extraction method based n-gram of Ashish Sureka and Pankaj Jalote [10] were proposed and have improved the performance of duplicate detection on bug reports. The method based observation of bug report characteristics which contain many code compound words. However, result recall rate in their technique only achieve at 40% top-10 of recommendation list for the Eclipse project. Main reason is that the n-gram-based approach may cause many noisy features. In Chen's thesis [4], he proposed the basic NLP, N-gram, and the cluster Shrinkage (CS) technique that improve the similarity identification using reweighting the features of bug reports. The method can get around 89% and 76% top 20 of recommendation list for SVN project and ArgoUML, respectively. However, this method used reweighting makes cost for high calculation.

III. DUPLICATION DETECTION DESIGN

This section presents the design of proposed study method to improve duplicate bug reports. In our approach, we definite a cluster as a duplicate bug reports set and when have a new bug report is submitted, we call it is an incoming bug report. Moreover, we also call first bug reports are master bug reports as shown in Table 3.1.

Table 3.1: An example for mapping in duplication bug report

# of Cluster	Incoming bug reports and confirmed duplication	Master bug report
1	793	689
2	797, 792	703
...	...	...
n	2610, 2520, 2407	525

We use the comments for determine bug reports which has been marked as a duplicate. If the bug report is a duplicate bug report, it is marked as a duplicate with a message is added in bug report like form “this issue has been marked as a duplicate of <bug report ID>”. This comment is often marked by developer. According to previous studies, although centroid-based approaches bring many advantages [6], however, they also face much serious with the problem of inductive bias or model misfit [10, 13]. Centroid-based approaches are more susceptible to model misfit because of its assumption that a document should be assigned to a particular class when the similarity of this document and the class is the largest [12]. Therefore, they explored that a centroid based classifier should give more weight to term distributions among the corpus, i.e., inter-class, inner-class and in collection distributions. Centroids considering characteristics of term distribution have shown improved results [7]. From previous studies observation, we propose a model with novel weight representation for centroid based duplicate detection which incorporates both inter-class term distribution and inner-class term distribution to determine term weights in prototype vectors.

In present method, we construct a model for improving the duplicate bug report detection rate. For experiment in preliminary result, we get the recall rate better than others. In particular, the proposed scheme has the following four basic processing steps and subsection elaborates on each of these steps.

1. Feature extraction
2. Feature weighting
3. Similarity calculation
4. Recommendation generation

Figure 3.1 shows the processing flow in our scheme.

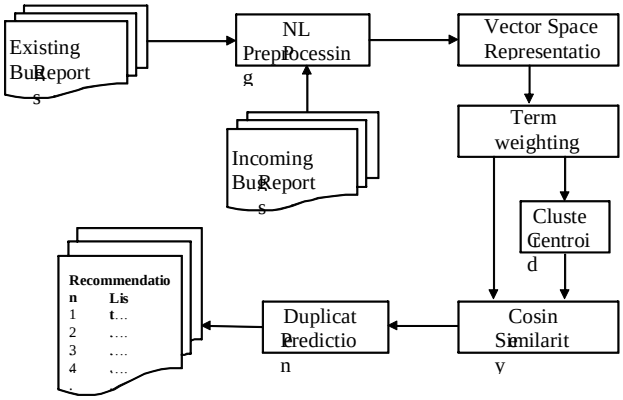

Fig. 3.1: The detection process for duplicate bug reports.

A. Feature extraction

For presentation bug reports, we use the same model with previous studies which is VSM (Vector Space Model). Proposed method performs pre-processing for raw textual data before analysing it, tokenizing the text into words and removing stems from those words. Using the WVTool tool to obtain tokenized which based on letters and non-letters, all non-letter characters are assumed to be separators, thus the resulting tokens contain only letters. Tokenization strips punctuation, capitalization, numbers, and other non-alphabetic constructs. Stemming function is useful to map different grammatical forms of a word to a common term and allows for a more precise comparison between bug reports by creating a more normalized corpus, in our experiments used the common Porter stemming algorithm. We then filter each sequence against a stop list of common words. Stop list remove words such as “a” and “and” that are present in text but contribute little to its comparative meaning. Moreover, to define exactly the dimensions of the vector space, we also use word list creation function with a list of string values in our method.

B. Feature weighting

From observation limitations for the inferior performance of centroid-based classifiers is that centroids do not have

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 10, October 2014

good initial values [12]. In order to overcome this issue, we employ CFC (Class-Feature-Centroid) method, in this method a centroid is built from two important class distributions: inter-class term index and inner-class term index. CFC presents a novel combination of these both and instead of using traditional cosine measure for calculation between two vectors we use denormalized cosine measure because its advantages for calculating the similarity score between a text vector and a centroid [12]. Formula for calculation traditional centroid as follow:

$$\vec{Centroid}_j = \frac{1}{|C_j|} \sum_{\vec{d} \in C_j} \vec{d} \qquad (1)$$

In formula (1) the centroid is the arithmetical average of all document vectors of class C_j. In the CFC method, weight for term t_k of class j is calculated as:

$$W_{ij} = b^{\frac{DF_{ti}^{j}}{C_j}} \times log(\frac{|C|}{CF_{ti}}) \qquad (2)$$

where DF^{j}_{ti} is term ti' s document frequency in class C_j, $|C_j|$ is the number of documents in class C_j , $|C|$ is the total number of document classes, CF_{ti} is the number of classes containing term t, and b is a constant larger than one. In the above formula, the first phrase describes the inner-class term index, and the second phrase describes the inter-class term index.

Function of the inner-class distribution of a term can help to distinguish text classification. For instance, if a term appears many times in documents of category C, and then a test document containing the term is more likely to be of category C. The advantage of this is to limit the inner-class feature weight within range [1,b] and b is a parameter greater than 1. The denominator $|C_j|$ in the formula can smooth the difference of document frequencies across categories. This component can be easily computed by counting the occurrence of terms while traversing the corpus, which only incurs linear-time cost. For the inter-class term index, a good inter-class feature or term should distribute rather differently among classes. In other words, if a term only appears in a few categories, and then the term is a discriminative feature, thus a good feature for classification. Conversely, if a term appears in every category, and then the term is not a good inter-class feature. Extracting inter-class term index help to produce more discriminative features. It is a form is similar to IDF, but the difference is that the CFC is counting the number of categories or classes containing the term. When a term that occurs in all category, we get $|C| = CF_{ti}$) then the value becomes 0. In the case the term only occurs in one category, we get the value is log(|C|). In other situation, the value is got from between them. From observation, we show that such rare terms and bias against popular terms. We can also be calculated the inter-class term index easy in linear time by counting vocabularies while traversing the corpus once.

C. Similarity ranking

From previous researches, using cosine for similarity calculation is evaluated as one of effective methods

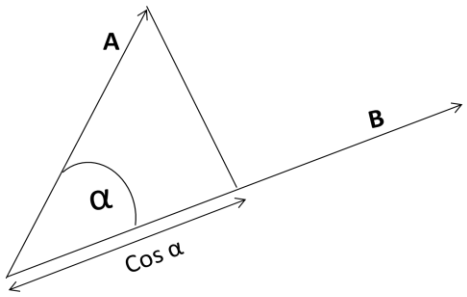

Figure 3.2: An illustration of standard cosine measure

between two vectors as shown in Figure 3.2. In our method, we also follow this method. However, we do not use the same traditional similarity measure between two vectors that replace by adopting this method with another method called denormalized cosine measure which is computed between a document vector and a prototype vector as shown in Figure 3.3.

From previous study [12], the denormalized cosine measure presents the discriminative capability of prototype vectors and improves the accuracy of detection in duplication bug reports. In our model, there are two types of similarity calculations. One of them is called document-based ranking and another type is cluster-based ranking. In document-based ranking, we first perform comparison of incoming bug reports with all bug reports in bug repository, then we sort them by computing similarity values to determine whether bug reports is duplicate or not. Another type is the cluster-based ranking, for this type we re-calculate the cosine values before performing to determine duplicate bug reports. We then average these values of members in cluster. Finally, we compare the incoming bug reports with all bug reports for new cosine values in bug repository and sort them by calculation similarity value in determining duplicate bug reports.

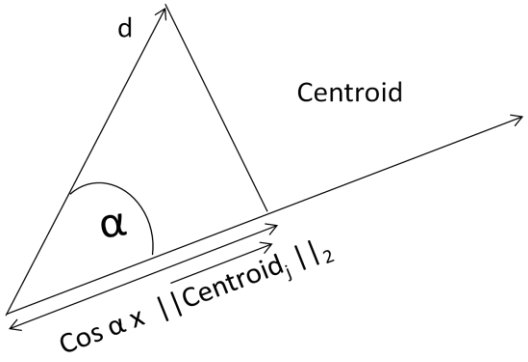

Figure 3.3: An illustration of denormalized cosine measure

D. Top-N Recommendation

Follow-up previous methods, we present the preliminary result by using the top N with recommendation system. We rank for top 20 and observe the result. With the result get from experiment we compare the top-N recommendation list with past researches. In our approach, with preliminary result we get the performance better than others.

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 10, October 2014

IV. PRELIMINARY EXPERIMENTS

In this section, we present our preliminary experimental environment with two projects of open-source datasets are ArgoUML and SVN. Table 4.1 lists the detailed information of these two datasets. We also make a comparison with past research work in performance. The result is described in the following subsections.

Table 4.1: Information of data sets of two open source projects

Description	ArgoUML	SVN
Language	Java	C
Software Type	UML Tool	SCM Tool
SCM	Subversion	Subversion
Repository	Tigris	Tigris
Data period	01/02-07/05	01/03-07/05
# of bug repts	4,613	2,296
#of duplicates	294	305

A. *Experimental Setup*

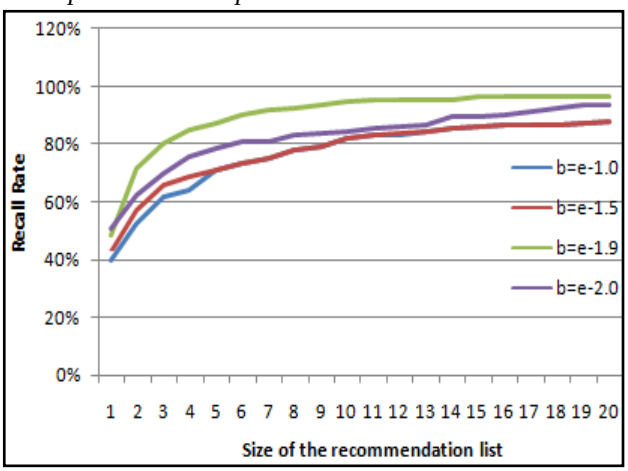

Figure 4.2: The experimental results of various b values for ArgoUML project.

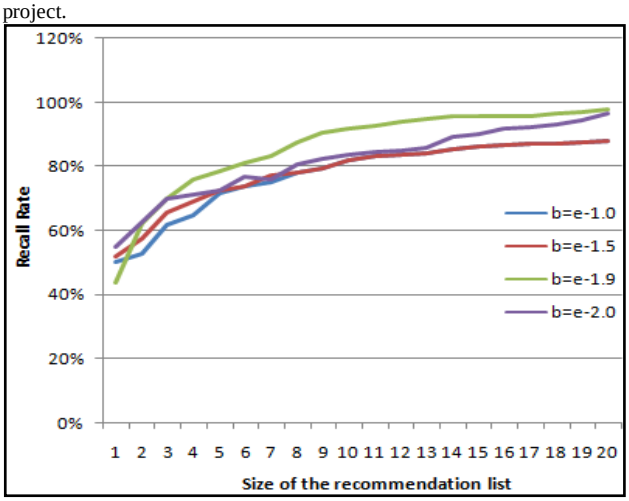

Figure 4.3: The experimental results of various b values for SVN project.

To evaluate the CFC scheme and related duplication detection schemes, the recall rate metrics in Equation (4.1) was used in the experiments. This represents that how many bug reports can be correctly detected in the top-*k* items of the recommendation rank list. The recall rate used in the experiments is defined as follows:

$$\text{Recall rate} = \frac{\#\text{of correct prediction}}{\text{Total \#of duplicate bug reports}} \qquad (4.1)$$

In our model, we use parameter b to calculate weighting feature. In our experiment, we study CFC's performance with varying values of b. Then, we observe that parameter b get the best performance result when b gets value e - 1.9 with (e -1 ≤ b ≤ e - 2.0) in our experiments. The results are shown in figures 4.2 to 4.3.

B. *Comparison Evaluation*

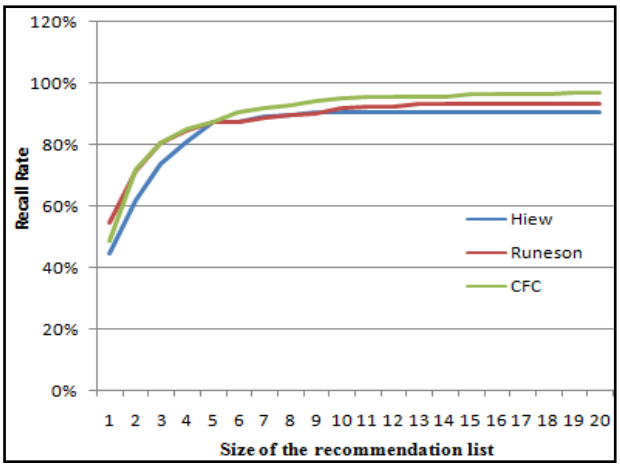

Figure 4.4: Detection performance for the ArgoUML dataset

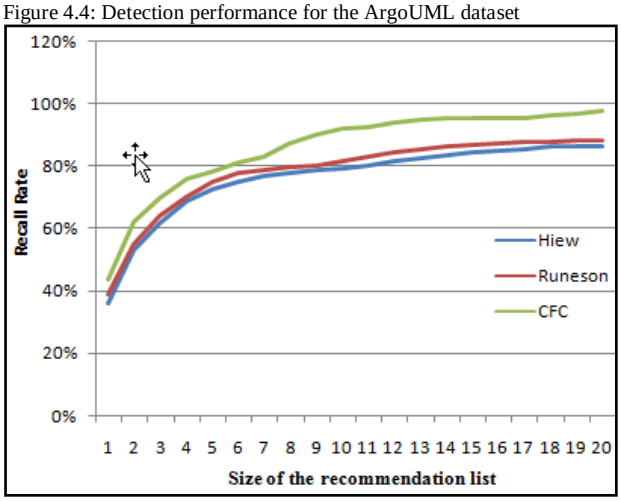

Figure 4.5: Detection performance for the SVN dataset

To explore the effectiveness of CFC in comparison with related detection schemes, experiments were conducted to study the work of Hiew and the work of Runeson. The results are shown in Figures 4.4 to 4.5.

As shown in the experimental results, the proposed CFC scheme has significant improvements in most cases of all two projects. Figure 4.5 shows that the CFC scheme can improve the TF-IDF scheme in both Runeson-based and Hiew-based methods up to 10% in the SVN dataset. As the SVN dataset, the CFC-based scheme performed well in the ArgoUML dataset. Figure 4.4 shows that the CFC-based scheme can also improve the TF-IDF scheme in both Runeson-based and Hiew-based methods up to 5% in the ArgoUML dataset.

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 10, October 2014

V. CONCLUSION

In this study, a CFC-based term weighting scheme is proposed to automatically detect duplicate bug reports. The experimental results show that using CFC scheme can achieve significant improvements. For SVN and ArgoUML projects, the recall rate can be further improved about 10% with a list size of 20 predictions. The reason is that in CFC, a centroid is built from two important class distributions: inter-class term index and inner-class term index. Moreover, CFC scheme also adopts a denormalized cosine measure, instead of a normalized prototype vector.

REFERENCES

[1] John Anvik, Lyndon Hiew, and Gail C. Murphy, "Coping with an Open Bug Repository", in Proceedings of the 2005 OOPSLA workshop on Eclipse technology eX-change (eclipse '05), 2005, pp. 35-39.

[2] John Anvik, Lyndon Hiew, and Gail C. Murphy, "Who Should Fix this Bug?" in Proceedings of the 28th International Conference on Software Engineering (ICSE '06). New York, NY, USA: ACM, 2006, pp. 361-370..

[3] Nicolas Bettenburg, Rahul Premraj, Thomas Zimmermann, and Sunghun Kim, "Duplicate Bug Reports Considered Harmful... Really?" in Proceedings of the 24th IEEE International Conference on Software Maintenance (ICSM 2008), 2008, pp. 337-345.

[4] Zhi-Hao Chen, "Duplicate Detection on Bug Reports using N-Gram Features and Cluster Shrinkage," Master Thesis, Yuan Ze University, Jul. 2011.

[5] Yguarat a Cerqueira Cavalcanti, Eduardo Santana de Almeida, Carlos Eduardo Albuquerque da Cunha, Daniel Lucr edio, and Silvio Romero de Lemos Meira, "An Initial Study on the Bug Report Duplication Problem", in Proceedings of the 14th European Conference on Software Maintenance and Reengineering, 2010, pp. 264-267.

[6] Lyndon Hiew, "Assisted Detection of Duplicate Bug Reports, Master Thesis", the University of British Columbia, May 2006.

[7] Bradley Kjell, W. Addison Woods, and Ophir Frieder, "Discrimination of Authorship using Visualization, Information Processing and Management", vol. 30, no. 1,pp. 141-150, Jan. 1994.

[8] Per Runeson, Magnus Alexandersson, and Oskar Nyholm, "Detection of Duplicate Defect Reports Using Natural Language Processing", in Proceedings of the 29th International Conference on Software Engineering (ICSE 2007), 2007, pp. 499-510.

[9] Chengnian Sun, David Lo, Xiaoyin Wang, Jing Jiang, and Siau-Cheng Khoo, "A Discriminative Model Approach for Accurate Duplicate Bug Report Retrieval", in Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE 2010), vol. 1. ACM, 2010, pp. 45-54.

[10] Ashish Sureka and Pankaj Jalote, "Detecting Duplicate Bug Report Using Character N-Gram-based Features", in Proceedings of the 17th Asia Pacific Software Engineering Conference, 2010, pp. 366-374.

[11] Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik, and Jiasu Sun, "An Approach to Detecting Duplicate Bug Reports using Natural Language and Execution Information", in Proceedings of the 30th International Conference on Software Engineering(ICSE '08). New York, NY, USA: ACM, 2008, pp. 461-470.

[12] Guan, H., Zhou, J., Guo, M.: "A class-feature-centroid classifier for text categorization". In: WWW 2009: Proceedings of the 18th international conference on World Wide Web, pp. 201-210. ACM, New York (2009).

[13] S. Tan. "An improved centroid classifier for text categorization. Expert Systems with Applications", pp.35 (1-2):279285, 2008.

BIOGRAPHY

Nhan Minh Phuc - received the B.Sc in Information Technology from the Ho Chi Minh City University of Natural Sciences-Vietnam National University, and M.Sc in Computer Science and Engineering from Yuan Ze University, Taiwan. He is now working in Tra Vinh University, Viet Nam.