Classifying Bug Reports to Bugs and Other Requests Using Topic Modeling

Natthakul Pingclasai
Department of Computer Engineering
Kasetsart University
Bangkok, Thailand
Email: b5310547207@ku.ac.th

Hideaki Hata
Graduate School of Information Science
Nara Institute of Science and Technology
Nara, Japan
Email: hata@is.naist.jp

Ken-ichi Matsumoto
Graduate School of Information Science
Nara Institute of Science and Technology
Nara, Japan
Email: matsumoto@is.naist.jp

***Abstract*—Bug reports are widely used in several research areas such as bug prediction, bug triaging, and etc. The performance of these studies relies on the information from bug reports. Previous study showed that a significant number of bug reports are actually misclassified between bugs and non-bugs. However, classifying bug reports is a time-consuming task. In the previous study, researchers spent 90 days to classify *manually* more than 7,000 bug reports. To tackle this problem, we propose *automatic* bug report classification techniques. We apply topic modeling to the corpora of pre-processed bug reports of three open-source software projects with decision tree, naive Bayes classifier, and logistic regression. The performance in classification, measured in F-measure score, varies between 0.66-0.76, 0.65-0.77, and 0.71-0.82 for HTTPClient, Jackrabbit, and Lucene project respectively.**

***Keywords*-bug classification; topic modeling; bug reports;**

I. Introduction

Poor quality of bug reports is a hidden error that could lead prediction models to produce unreliable results. Bettenburg et al. reported that bug reports often come with incomplete and inadequate information [1]. With these poorly written bug reports, developers have to take much effort on the error inspection process. *Misclassification* of bug reports is one of the serious quality problems that have been addressed recently. This problem simply occurs due to the misunderstanding of reporters and the lack of a better classification support of Bug Tracking System (BTS) [2]. Even though BTS literally refers to the system that acquired for managing the reports related to code-error—that is *bugs*, it is also used for managing the reports of *other requests* that related to software activities, e.g. request for new feature, improvement, documentation and so on. On account of these various usages of BTS, bug reports are more likely to be misclassified.

Distinguishing bugs from other requests is a challenging task since each report requires an individual inspection to identify the incorrectness. A number of studies [2], [3], [4] had to spend a part of their work schedule on manually classifying bug reports. Herzig et al. took 90 days to manually inspect over 7,000 bug reports [3]. They found that about one-third of all examined bug reports are actually misclassified. According to their work, manual inspection is difficult to be done, especially in the study where large number of bug reports is considered. Therefore, automatic classification techniques are desirable.

An automatic classification technique has once been proposed by Antoniol et al. [2]. They raised and tackled the problem of misclassified bug report by building word-based classification models. According to their study, these models yields significant performance.

In this paper, we propose another way of automated technique for classifying bug reports. We adopt topic modeling and apply three classification techniques to compare and indicate the most efficient topic-based classification model. The three classification techniques are decision tree, naive Bayes classifier, and logistic regression.

Our research studies on three open-source software projects (HTTPClient, Jackrabbit, and Lucene) based on the previous study published datasets.

The contributions of this paper can be summarized as follows:

- Propose an automatic classification model on topic-based technique.
- Compare the performance in classification between topic-based and word-based model.

This paper is organized as follows. Section II discusses on the motivation behind our technique. Section III describes in detail about the two phases of our research methodology. In Section IV, we describe our experimental design, including study projects and evaluation metric we used. Section V reports the results after we have conducted all processes. Section VI discusses on our proposed techniques and on issues that threaten to the validity of our study, followed by Section VII describing on our related works. Finally, Section VIII concludes the results and specifies our future work.

II. Bugs and Requests

In this section, we present the motivation behind our technique. We discuss on the issue of misclassification of bug reports before describing the observation on manual inspection process that leads to the idea of applying topic modeling.

Table I
EXAMPLES OF TEXTUAL INFORMATION CONTAINED IN BUG REPORTS THAT DEFINES ITS CATEGORY

Bug Report ID	**Textual Information**	**Category**
HTTPCLIENT-1011	... CachingHttpClient.execute() does not catch the IOException ... ... I would actually call this a *bug* ...	BUG
JCR-346	... Fix possible *Null Ptr exception* in ConnectionFactory ... ... Code will *throw npe* if driver string is null...	BUG
LUCENE-1113	... Fix for Document.getBoost() *documentation* ... ... The attached patch fixes the *javadoc* to make clear ...	DOC

A. Misclassification Problems

According to data mining processes, quality of the data is an important factor that determines the efficiency and the reliability of the prediction model. The poor quality of data leads to the negative unexpected detail of predictive results. For instance, bug prediction model that use bug reports labeled as *bugs*, as the predefined data would predict for changes instead of the occurring of defects if such reports are actually *non-bugs*. Furthermore, this poor quality of bug reports results in more time and effort the developers must take to accomplish their works [1].

B. The Idea of Topic Modeling Approach

The idea of applying topic modeling comes from the observation on the classification rules used in the previous manual classification work [3]. Every rule was written based on the textual information of each bug report. For example, there are rules defined that a bug report should categorized to BUG if...

1) it was reported on NullpointerException.
2) its discussion concludes that code has to be changed to perform a corrective maintenance task.
3) it was reported to fix runtime or memory issues that caused by defects.

One example of other-requests report is that a bug report should categorized to DOC if...

1) its discussion unveils that the report was filed due to the issue of documentation.

Table I represents the categorization of bug reports that their textual information follows some of the above rules. For the other rules, they are also defined in the same standard. Thus, we have an idea that within a corpus of bug reports, there are bug-related and other-requests-related topics.

III. METHODOLOGY

A. Overview

Figure 1 shows the overall structure of our model building process. To construct the automatic classification model, we first parse and pre-process the retrieved bug reports. Then, we apply topic modeling to the corpora of pre-processed bug reports. The output of this process is a collection of topic membership vectors. Next, with the discovered vectors and

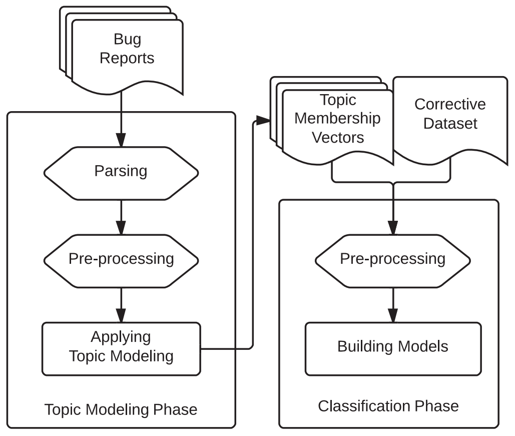


Figure 1. Overview of our automatic classification model building process

the dataset containing the correct type of each bug report, we merge and organize them before using them to train the model. Technically, our methodology is divided into two phases: *topic modeling phase* and *classification phase*. We describe more detail in the following subsections.

B. Topic Modeling Phase

This phase basically consists of three steps, described as follows.

1) Data Parsing: Typically, bug reports are collected as XML documents. Therefore, in the first and foremost step after we have bug reports of the three Apache projects, we parse them. From each bug report, we extract three contents of textual information, i.e. *title*, *description*, and *discussion*. Our pilot study shows that, including discussion provides more detail for describing such bug report and leads to the more specific topics to be discovered in the topic modeling step.

2) Data Pre-processing: This step takes a major role in data noise removal and optimizing the performance of the classification step. We remove out HTML tags and some bad punctuation from each parsed document. Then, we tokenize the document to break a stream of text into a set of terms before passing it to a stemming process, which map each term to its basic form. The algorithm that we use in this

Table II
EXAMPLES OF TOPIC MEMBERSHIP VECTORS OF LUCENE PROJECT

Report ID	Topic ID	Prop.	Topic ID	Prop.	...
Lucene-1097	0	0.709527	5	0.178536	...
Lucene-1098	4	0.425569	5	0.322765	...
Lucene-1099	4	0.531809	8	0.441094	...

Table III
DETAILS OF STUDY SUBJECTS

Project	# of Reports	# of Bugs	# of Other Requests
HTTPClient	745	305	440
Jackrabbit	2,402	938	1,464
Lucene	2,443	697	1,746

research is the Porter Stemming algorithm [5]. Finally, we remove English stopwords like "a", "the", "is" and "that" from the document since they carry less meaningful context.

3) Applying Topic Modeling: Topic modeling is designed to automatically extract *topics* from a corpus of text documents, where *topic* is a collection of words that co-occurs frequently in documents of the corpus. Among the well-known techniques of topic modeling, we employ *Latent Dirichlet Allocation* (LDA) in our research. The reason is that LDA is able to discover a set of ideas that well describe the entire corpus [6].

LDA is based on a probabilistic generative model as it makes an assumption that a corpus of documents contains a collection of topics and that each document is represented by the probability distribution over those topics contained within the corpus.

Implementing LDA, generally with topic modeling, there is no single value of number of topics (K) that is appropriate in all situations and all datasets [7]. With small value of K, the model tends to discover more general topics from the input corpus of documents, while applying larger value of K, more specific topics are likely to be produced. Therefore, in this research, we examine on every 10 consecutive number of topics from 150 topics. Note that the larger number of topics requires longer time to complete the training process. The output to be considered after applying LDA to the pre-processed data is a set of *topic membership vectors*. A vector consists of report id and a set of topic id with its proportion as shown in Table II. These vectors capable for indicating the proportion of topics that related to the words contained within each document. In general, the topic with the highest proportion gives the best meaning for labeling document. However, our research aims to classify, not to label the bug report. Our pilot study found that the proportion value doesn't provide much benefit for our classification approach. Therefore, we focus only on the existence of topics in each topic membership vector.

C. Classification Phase

In this phase, we operate on two datasets. One is the output from topic modeling phase; the topic membership vectors, and the other one is the corrective dataset which contains a set of bug report id and its correct type. Since data mining techniques require a specific form of inputs for a proper execution, we again need for a step of data pre-processing to organize the datasets before mining them in order to build classification model. We describe more detail below.

1) Data Pre-processing: To prepare input for mining step, we build a set of K-wide topic vectors for each examined project, using 0 and 1 to indicate the existence of topics distributed in each bug report. In fact, bug reports can be categorized into many types but, in our research, we interest in two major types. We group all the reports labeled as bug to class BUG and group all the other types to class OTHER_REQUESTS. Therefore, lastly, each topic vector is indexed a report type to either BUG or OTHER_REQUESTS. This index is represented as a class attribute in mining process.

2) Building Model: As we aim to compare our topic-based model to the word-based model later in this study. In order to be relevant, we separately build up three topic-based classification models. Each model is constructed from applying one of the three data mining techniques for classification approach used in the previous research [2]. The three techniques are described below.

- *Decision Tree:* A tree-like model providing decision support in which each inner node represents condition, each branch represents the possible values for the condition, and each leaf represents the class labels. In this paper, we employ *Alternating Decision Tree* (ADTree) which combines decision tree and boosting algorithms to generate an effective classifier.
- *Naive Bayes classifier:* A simple probabilistic classifier that allows each attribute to contribute towards the final decision equally and independently.
- *Logistic Regression:* A regression analysis that used for predicting when the dependent variable is a dichotomy.

IV. EXPERIMENTAL DESIGN

A. Study Subjects

Our research makes use of three training data from the previous study[1]. The datasets we gathered are all open-source software projects from Apache, which written in Java and using JIRA bug tracker, i.e. HTTPClient, Jackrabbit and Lucene project. Each dataset contains the rectified results of the misclassified bug reports of each project and has three fields, i.e. report id, original type, and rectified type. We use the report id containing in the datasets to map and retrieve bug-report files from the provider repository[2] and use those

[1]http://www.st.cs.uni-saarland.de/softevo//bugclassify/
[2]https://issues.apache.org/jira/

rectified results as our evaluation baseline. The number of examined bug reports for each project is represented in Table III.

B. Evaluation Method

The performance evaluation metric that we use to evaluate our model is *F-measure* score.

- *F-measure* score is a combined value of *precision* and *recall*:

$$F - measure = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

- The *precision* refers to the ratio of correctly classified BUG/OTHER_REQUESTS reports over all of the bug reports that classified as BUG/OTHER_REQUESTS.
- The *recall* refers to the ratio of correctly classified BUG/OTHER_REQUESTS reports over all of the bug reports that actually are BUG/OTHER_REQUESTS.

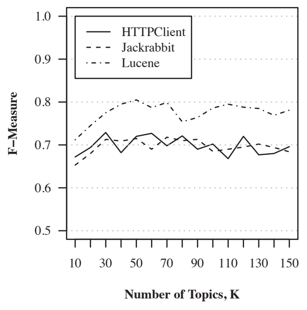
In this paper, we employ 10-fold cross-validation to validate the model. The idea of 10-fold cross-validation is that the dataset is randomly partitioned into ten parts, one part is retained as a testing data and the other nine parts are used as a training data. We report the average value after 10 runs of cross-validation.

V. RESULTS

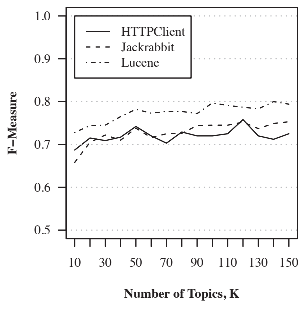
A. Performance

The evaluation results are shown in Table IV where the first column determines the number of topic that was used to train the model. For the other columns, three consecutive columns are grouped into a set of study subjects. Each set reports the performance of a classification model that employed one of the three classifier algorithms. As we see in Table IV, the three models yield the classifying performance in a quite similar range for each study subject where F-measure score varies between 0.66-0.76, 0.65-0.77, and 0.71-0.82 for HTTPClient, Jackrabbit, and Lucene respectively. Note that the models yield significantly better results when perform with the bug reports of Lucene project. However, the trends resulting from the increasing number of topics are different in each model.

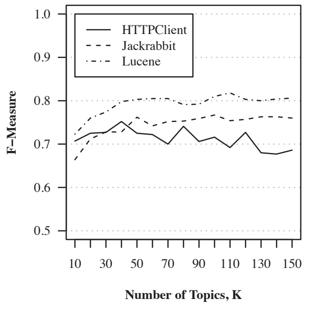
Figure 2 shows the performance curves of the three study cases for each classification model. The x-axis denotes the number of topics we applied on such model. The y-axis indicates the F-measure score that the model can achieve. We can see that increasing the number of topics doesn't tend to help improving or downgrade the performance of ADTree model. While the curves in ADTree model are oscillating, there are quite explicit trends in the other two models. For naive Bayes and logistic regression model, the classification performance is slightly rising up in Jackrabbit and Lucene case as the larger number of topics is applied. For HTTPClient case, the curve oscillates in naive Bayes model while gradually falls in logistic regression model.



(a) ADTree model



(b) Naive Bayes model



(c) Logistic Regression model

Figure 2. Performance curves of HTTPClient, Jackrabbit, and Lucene bug report classification

According to the results, we conclude that topic-based classification model is able to classify bug reports with significant performance and when consider on trends of the performance, naive Bayes model seems to have a higher opportunity to be further developed.

B. Topic-based vs Word-based Model

In this subsection, we compare the classifying performance between topic-based and word-based model. To build word-based classification model, we first extract word vectors from each corpus of pre-processed bug reports. a corpus represents one of the software project. Based on these word vectors, we follow the methodology conducted in the previous study [2] to construct the model. We adopt a feature selection algorithm to select subsets of the features (words) with a number of 20, 50, and 100. Finally, we train the classification models with the three classifiers; ADTree, naive Bayes classifier, and logistic regression before evaluating the performance of each model.

Table V shows the comparison among the performance of word-based and topic-based classification models. For topic-based models, we collect the results from the models trained from 50 topics since they are likely to produce significantly high performance for all cases with less time required. We marked an asterisk (*) on the higher F-measure score in each comparison where we see that topic-based model slightly outperforms word-based model in all cases, except one HTTPClient case.

Table IV
WEIGHTED F-MEASURE OF THE TOPIC-BASED CLASSIFICATION MODELS SEPARATED BY CLASSIFIER ALGORITHM

# of Topics	ADTree			Naive Bayes classifier			Logistic Regression		
	HTTPClient	Jackrabbit	Lucene	HTTPClient	Jackrabbit	Lucene	HTTPClient	Jackrabbit	Lucene
10	0.672	0.653	0.712	0.687	0.658	0.728	0.707	0.664	0.723
20	0.694	0.68	0.745	0.715	0.705	0.744	0.725	0.712	0.76
30	0.729	0.713	0.775	0.709	0.722	0.745	0.727	0.728	0.774
40	0.682	0.709	0.795	0.717	0.71	0.765	0.752	0.728	0.798
50	0.72	0.715	0.805	0.742	0.738	0.782	0.725	0.762	0.803
60	0.727	0.69	0.787	0.72	0.716	0.773	0.722	0.742	0.805
70	0.698	0.718	0.799	0.703	0.725	0.777	0.7	0.752	0.805
80	0.721	0.71	0.754	0.729	0.726	0.777	0.741	0.753	0.791
90	0.69	0.713	0.764	0.72	0.744	0.772	0.706	0.759	0.792
100	0.702	0.685	0.786	0.72	0.745	0.797	0.716	0.767	0.81
110	0.668	0.69	0.795	0.725	0.745	0.791	0.692	0.754	0.818
120	0.72	0.695	0.788	0.758	0.752	0.787	0.727	0.757	0.803
130	0.677	0.702	0.785	0.72	0.737	0.783	0.68	0.763	0.8
140	0.68	0.694	0.769	0.712	0.749	0.8	0.677	0.763	0.804
150	0.696	0.684	0.781	0.725	0.753	0.794	0.686	0.76	0.806

Table V
COMPARISON BETWEEN WEIGHTED F-MEASURE OF THE WORD-BASED AND TOPIC-BASED CLASSIFICATION MODELS

	Selected Features	ADTree			Naive Bayes classifier			Logistic Regression		
		HTTPClient	Jackrabbit	Lucene	HTTPClient	Jackrabbit	Lucene	HTTPClient	Jackrabbit	Lucene
Word-based	20	0.675	0.687	0.764	0.623	0.6	0.474	0.698	0.678	0.741
	50	0.702	0.69	0.761	0.666	0.619	0.475	0.753	0.711	0.774
	100	0.705	0.688	0.752	0.673	0.646	0.482	0.76*	0.759	0.775
Topic-based	(Topic=50)	0.72*	0.715*	0.805*	0.742*	0.738*	0.782*	0.725	0.762*	0.803*

VI. DISCUSSION

A. Topic-based Technique

Since topic is a collection of words that frequently occur together, topic modeling is capable for linking the words that appear in similar contexts. Thus, the uses of words with multiple meanings can be distinguished by the difference in topics. Compared to topic modeling, feature selection technique may has less efficient in capturing the latent meaning of each word in the corpus. With this strong point of topic modeling, our topic-based classification technique can be further developed for applying in software engineering tasks as a quality assuring process of bug reports.

Technically, the time that our proposed technique takes to train a classification model depends on the number of topics being applied. However, in case of all required tools and datasets are provided with all variables defined, we can complete the whole process within a few hours. Therefore, this technique can help developers reducing their time in bug assigning and fixing processes as they do not have to waste much effort on the manual classifying poor bug reports.

B. Threats to Validity

Research results are related to the previous study. This threatens to construct validity. By using the published datasets from the previous study, even though they have written a fixed set of rules to classify bug reports to each category, we cannot ensure that there is no error in the manual inspection process. Furthermore, the categorization of bug reports depends on an individual perspective. If the different set of rules were employed in the their study, our classification model would definitely produce the different results.

Cross validation does not preserve chronology. There is also a threat to construct validity in our evaluation process. We employ 10-fold cross validation in our study. This technique ignores the issue of chronological change that may have an influence on the trend of defect occurrence in practice.

Study subjects are limited to open-source software projects. All the three projects are written in Java and using JIRA bug tracker. These limitations threaten to external validity. Although we studied on more than 5,000 (2,443 for the largest dataset) bug reports, the results might not be generalized to the projects that are written in other language or employing other Bug Tracker System.

Configuration for Latent Dirichlet Allocation is not simple. There is no general solution for choosing the optimal parameter settings for LDA [7]. Internal validity is threatened due to this issue. Even though we studied on several numbers of topics and the models can yield acceptable results, we cannot ensure that our setting is optimal for the methodology we conducted.

VII. Related Work

So far, bug reports pay an important role in several software engineering tasks such as predicting future bugs [8], [9], triaging reported bugs [10], [11], and categorized bugs into some specific types [4]. With these significant tasks, quality of bug reports is important and being concerned periodically. Among a number of studies, there are a few that addressed the issue of misclassification of bug reports [2], [3].

Antoniol et al. [2] initially addressed the misclassification of bug reports. They found that less than half of bug reports are actually related to bugs. This reflects to the complex usage of Bug Tracking System which is a cause of misclassification issue. They propose an automated classification approach. They claimed that textual information contained in each bug report is sufficient to distinguish between bugs and other software activities. Herzig et al. [3] extended this research by show the amount of data noise is introduced by misclassified bug reports and indicate the possible impacts. They reported that bug report classifications are unreliable and lead to bias in prediction. However, high effort was taken in manual inspection process. Our paper then addresses the issue and proposes another alternative automated classification technique.

VIII. Conclusion and Future Work

In this paper, we conducted an automatic classification model for classifying bug reports based on textual information contained in each report. Topic modeling was adopted for clustering and structuring large corpora of bug reports. Our technique aims to reduce time and effort required for manual inspection, as well as extend opportunities for a research study to obtain more effective results. Studied on three open-source software projects (HTTPClient, Jackrabbit, and Lucene), we show the possibility of topic-based classification models. We built the models separately from three classifier algorithms, i.e. ADTree, naive Bayes, and logistic regression. After evaluating the results, we concluded that:

- A collection of topics extracted from a corpus of bug reports is able to distinguish bugs from other software activities. The classification models yield the F-measure score between 0.65 and 0.82.
- Among three models, the model building from naive Bayes classifier seems be the most efficient. The performance trends of naive Bayes model are rising up for two cases and oscillating for one case as the larger number of topics is used to train the model.

Furthermore, we also compare the performance between topic-based and the word-based classification model. The results reported that topic-based model outperforms in almost of the evaluated cases.

Our future work includes the studying on cross-project training to find out that are there general topics that can define the properties of bugs and non-bugs. Some other machine learning techniques are considered in order to optimize the classifying performance. Finally, we plan to study on more projects that employ other Bug Tracking Systems to achieve more general results.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number 25880015.

References

[1] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, “What makes a good bug report?” in *Proc.* SIGSOFT ’08/FSE-16, 2008, pp. 308–318.

[2] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, “Is it a bug or an enhancement?: a text-based approach to classify change requests,” in *Proc.* CASCON ’08, 2008, pp. 23:304–23:318.

[3] K. Herzig, S. Just, and A. Zeller, “It’s not a bug, it’s a feature: how misclassification impacts bug prediction,” in *Proc.* ICSE ’13, 2013, pp. 392–401.

[4] F. Thung, S. Wang, D. Lo, and L. Jiang, “An empirical study of bugs in machine learning systems,” in *Proc.* ISSRE, 2012, pp. 271–280.

[5] M. F. Porter, “Readings in information retrieval,”ch. An algorithm for suffix stripping, 1997, pp. 313–316.

[6] D. M. Blei and J. D. Lafferty, ”In Text Mining: Classification, Clustering, and Applications”, Topic Models, 2009, pages 71-94.

[7] S. Grant and J. Cordy, “Estimating the optimal number of latent concepts in source code analysis,” in *Proc.* SCAM ’10, 2010, pp. 65–74.

[8] E. Shihab, A. Mockus, Y. Kamei, B. Adams, and A. E. Hassan, “High-impact defects: a study of breakage and surprise defects,” in *Proc.* ESEC/FSE ’11, 2011, pp. 300–310.

[9] H. Hata, O. Mizuno, and T. Kikuno, “Bug prediction based on fine-grained module histories,” in *Proc.* ICSE ’12, 2012, pp. 200–210.

[10] A. Tamrawi, T. T. Nguyen, J. Al-Kofahi, and T. Nguyen, “Fuzzy set-based automatic bug triaging: Nier track,” in *Proc.* ICSE’ 11, 2011, pp. 884–887.

[11] D. Cubranic and G. C. Murphy, “Automatic bug triage using text categorization,” in *Proc.* SEKE, 2004, pp. 92–97.