Automated Classification of Software Bug Reports

Ahmed Fawzi Otoom, Sara Al-jdaeh, Maen Hammad
Department of Software Engineering The Hashemite University
Zarqa, Jordan
aotoom@hu.edu.jo, sara.aljdaeh@yahoo.com, mhammad@hu.edu.jo

ABSTRACT
We target the problem of software bug reports classification. Our main aim is to build a classifier that is capable of classifying newly incoming bug reports into two predefined classes: corrective (defect fixing) report and perfective (major maintenance) report. This helps maintainers to quickly understand these bug reports and hence, allocate resources for each category. For this purpose, we propose a distinctive feature set that is based on the occurrences of certain keywords. The proposed feature set is then fed into a number of classification algorithms for building a classification model. The results of the proposed feature set achieved high accuracy in classification with SVM classification algorithm reporting an average accuracy of (93.1%) on three different open source projects.

CCS Concepts
Software and its engineering→ Maintaining software.

Keywords
Software maintenance; bug reports; automatic classification.

1. INTRODUCTION AND RELATED WORK
Bug tracking systems are used as a centralized medium for communication between users and developers where users or developers submit bug reports for potential help in fixing these bugs. A number of issues arise in dealing with reports including: bug duplicates, bug severity assessment, bug triage, and bug classification into categories. Usually, these issues are dealt with in a manual inspection way which is difficult to be done as of the large number of bug reports to be considered. This has motivated researchers to build automated systems that can tackle the aforementioned issues. For example, the authors in [1] dealt with building an automated system for finding bug duplicates. In the work of [2], the authors suggested a number of techniques for automated severity assessment. Also, there has been an effort spent in targeting the problem of automated bug triage or bug assignment (e.g. [3]).

ICICM 2019, August 23–26, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-7188-9/19/08…$15.00
https://doi.org/10.1145/3357419.3357424

Recently, there has been an extensive research in relation to the area of automated bug reports classification. It aims to distinguishing bug reports into predefined set of categories (classes). The automated way of classification saves great time and effort and helps in reducing uncertainty and improve cost-effectiveness by planning ahead and pre-allocating resources towards source code maintenance [4]. The most common categories of bug reports are: corrective and perfective. Corrective reports deal with fixing defects that are related to errors, flaws and defects in the software. On the other hand, perfective reports are related to new enhancements and features to improve system performance.

This work extends a number of research works in the area of automated classification of software bug reports. For example, the authors in [5] classified issue reports into two classes: bug or none-bug. They proposed an approach that uses the whole document term matrix of three separate projects as a feature set. The feature set is fed into a number classification algorithms: naïve Byes, linear discriminant analysis, k-nearest neighbors, support vector machine, decision trees and random forests. A comparison is carried out between the performance of these classifiers and random forests perform best. Similar to the aforementioned work, the authors in [6], used 1800 issue reports from Mozilla, Eclipse and Jboss projects to build a classification model for discriminating between a bug and a non-bug categories. Manual labeling is used for initial categorizing of reports. Three classification algorithms are implemented: naïve Bayes, ADTree and logistic regression. The average accuracy ranges between 77% and 82%. Recently, the authors in [7] targeted a similar problem of classification of bug reports into: bug or none-bug classes. They proposed a multilayer approach that combines text mining and data mining. The approach is tested on three open source projects and an accuracy ranging between 76.1% - 93.1% has been reported.

Recently, an attention has been paid to modelling data with a probabilistic model for data representation. For example, the authors in [8] applied a topic modeling approach for classifying bug reports into two classes: bugs or other requests. A probabilistic generative model, Latent Dirichlet Allocation, is used for representing documents. A comparison is carried out on three projects and using three classification algorithms and the results are encouraging. A similar approach to that of [8] has been applied by the authors in [9], [10], [11] and [12].

A number of related research works has targeted the area of multiclass classification of software maintenance requests (e.g. [13] and [4]). In the work of [13], maintenance requests are classified into five different classes: corrective, adaptive, perfective, feature addition, and non-functional. A dataset is built based on word distributions and is fed into five different algorithms. Generally, accuracy results are over 50%. The authors in [4] proposed to incorporate word frequencies features

with source code changes for the purpose of classifying maintenance requests into three categories: adaptive, corrective, and perfective. The proposed approach achieved an accuracy of 76% when tested on 11 open source projects.

In this work, we target the problem automated classification of software bug reports. We aim at building a classifier that can accurately classify newly coming bug reports into: corrective or perfective classes. Different from the previous works, we propose a new feature set that is based on the occurrences of certain keywords rather than using the whole document term matrix. This is more efficient in terms of training time. The performance of the proposed feature set is proven by testing it on three projects and across a number of classification algorithms. The research contributions in this paper are:

- An efficient classifier to automatically classify newly reported bug reports.
- A small yet representative feature set that can accurately classify corrective and perfective bug reports.

The rest of this paper is organized as follows: In section 2, we explain the proposed approach part. Section 3 presents the experimental results. Finally, section 4 concludes the paper and suggests future work.

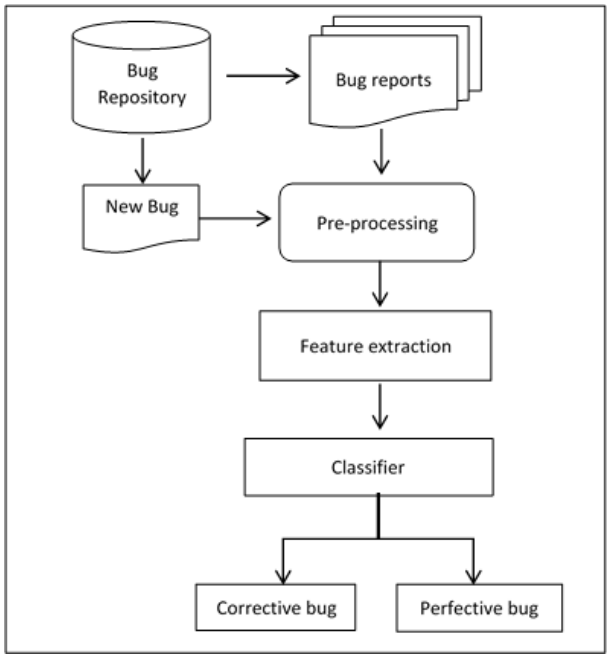

Figure 1. The approach

2. APPROACH

In the following section, we discuss the proposed approach as illustrated in Figure 1.

2.1 Data collection

In this step, we collected bug reports from three open source projects: AspectJ, Tomcat and SWT projects. The numbers of bug reports in each project and time period where these reports have been retrieved are illustrated in table 1.

Table 1. Number of bug reports in each project

Open source project	# of bug reports	Time Period
AspectJ	593	2003-2013
Tomcat	1056	2002-2014
SWT	4151	2002-2014

We manually labeled the reports in to one of the two classes: perfective or corrective based on the summary and description of the report and using a set of keywords that we believe can distinguish between a corrective bug report or a perfective bug report. The manual labeling is necessary step for supervised learning and is inspired by research works like [6], [4] and [7]. For example, words like *crash, error or problem* mean a corrective maintenance request. On the other hand, words like *enhancement, refactoring or performance* mean a perfective maintenance request. The labeling of bug reports is reviewed with software engineering professionals. For example, Table 2 shows three real examples from the three projects and how we manually labeled them using the underlined words.

Table 2. Summary and description of three bug reports

Project	Summary and Description	Class
AspectJ	**[performance]Contribution: Refactoring** to Support LTW World with Reflection Delegates for Bootstrap Types This patch adds an Reflection World interface, changes the various places in code where casts are made to Reflection World to use that interface.	Perfective
Tomcat	Suspicious url pattern **warning** logged to **wrong** webapp"Suspicious url pattern" warnings are logged to the wrong webapp, not the one actually using them.	Corrective
Birt	**Refactoring**: renaming a package suppresses comments at the end of import lines Build Test case is: - create packages p1a and p2; - create class X in p2: package p2; import p1a.*; // comment public class X { } - launch a rename **refactoring** action on p1a.	Perfective

2.2 Pre-processing

In this step, bug reports are transformed into vectors of keywords. Each vector represents a bug report. Two textual contents from the bug report are extracted; summary and description for each bug report. Table 3 shows the summary and description for three bug reports from the open source projects AspectJ, Tomcat, and SWT.

Then, the bug reports are processed in three main steps: In the first step; the text is *tokenized* into keywords (or tokens). Tokenization means large strings are split into tokens (words).

This step is also combined with punctuation removal and the replacement of capitalized letters with small ones.

Table 3. Summary and description of three bug reports

Project	Summary and description
AspectJ	Support additional message insert keys in declare error/warning It would be good to be able to insert the enclosing class name or enclosing member for a joinpoint.
Tomcat	Tomcat Web Application Manager table error html table -> tr wrong formatting when stop (corectivet last) in tomcat web application manager.
SWT	I can't run eclipse I can't run eclipse in redhat linux 7.2. After startup the eclipse, whenever I click the main region in the eclipse. The eclipse will throw an exception and exit.

In the second step; we apply *stemming*. It reduces each term to its basic form. Porter stemming algorithm is used for this reduction Table 4 shows the results after the tokenization and Stemming steps. Each bug report is represented as a set of keywords.

Table 4. Tokenization and stemming steps

Project	Tokenization and stemming
AspectJ	support; addit; messag; insert; key; in; declar; error; warn; it; would; be; good; to; be; abl; to; insert; the; enclos; class; name; or; enclos; member; for; joinpoint;
Tomcat	tomcat; web; applic; manag; tabl; error; html tabl ; tr ; wrong ; format ; when ; stop ; corectivet ; last ; in ; tomcat ; web ; applic ; manag ;
SWT	i ; can ' t ; run ; eclips ; i ; can ' t ; run ; eclips ; in ; redhat ; linux ; after ; startup ; the ; eclips ; whenev ; i ; click ; the ; main ; region ; in ; the ; eclips ; the ; eclips ; will ; throw ; except ; and ; exit ;

The final pre-processing step is *Stop Words Removal* where words that do contribute in differentiating a collection of documents are removed (i.e., “t”, “that”,” the” “they”, “their”, “theirs”, “them”,

“themselves”, “then”, “there”, “after”, “there've”, “there'll”, “these”, and so on). Words of a one letter are also removed. Table 5 shows the three bug reports in Table 4 after applying stop word removal on the tokens.

Table 5. Stop words removal

Project	Tokenization and stemming
AspectJ	support ; addit ; messag ; insert ; key ; declar ; error ; warn ; good ; abl ; insert ; enclos ; class ; name ; enclos ; member ;joinpoint;
Tomcat	tomcat ; web ; applic ; manag ; tabl ; error ; html ; tabl ; tr ; wrong ; format ;stop ; tomcat ; web ; applic ; manag ;
SWT	eclips ; eclips ; redhat ; linux ; startup ;eclips ; click ; main ; region ; eclips ; eclips ; throw ; except ; exit ;

2.3 Feature Extraction

The first step in any classification problem is feature extraction where features are extracted from bug reports to classify a new bug report into the two categories: corrective or perfective. This feature set is chosen based on extensive analysis of the occurrences of certain keywords in the bug reports. We believe that these keywords can be discriminant between the two classes: corrective or perfective. The proposed feature set is illustrated in Table 6.

Table 6. The Feature set

Number	Feature
Feature 1	enhancement
Feature 2	refactoring
Feature 3	improvement
Feature 4	performance
Feature 5	feature
Feature 6	usability
Feature 7	efficiency
Feature 8	contribution
Feature 9	reduction
Feature 10	support
Feature 11	update
Feature 12	modification
Feature 13	add
Feature 14	change
Feature 15	configuration

2.4 Classification

Once features are extracted, they are fed into a number of classification algorithms for the purpose of building the classifier. For this purpose, WEKA tool was used [14] which is a data mining tool written in java and contains different classification algorithms that are used for data analysis.

The classifiers that have been used for the classification experiments are: naive bayes (NB), support vector machines (SVM), and random trees (RT). Naive bayes is a probabilistic classifier based on applying Bayes' theorem with Naive independence assumptions. SVM is used learn a hyperplane that can differentiate between a set of objects with different class memberships. RT is an ensemble classifier that consists of directed graphs build with a random process [14].

The performance of the classifiers is evaluated in terms of classification accuracy. Classification accuracy is calculated as the number of correctly classified samples divided by the total number of samples. For validation, we carried out five-fold cross validation. The cross validation test divides the dataset into five disjoint subsets where four of them are used for training and the fifth one is used for testing. The algorithm is run for five times and the average accuracy across all folds is calculated.

3. EXPERIMENTS AND ANALYSIS

Experiments are conducted to validate the proposed approach. We experiment with the three classification algorithms on the three projects and accuracy is calculated. Moreover, we calculate the average accuracy of each classification algorithm on the three projects.

Table 7 shows the accuracy performance results on AspectJ project. It is clear from this table that the proposed approach is accurate in classification with the highest accuracy reported by Naïve Bayes algorithm of 97.0%. Also, this algorithm provides the highest accuracy on Tomcat data set with an accuracy of 91.4%, which is similar to the SVM classifier performance, as shown in Table 8. Table 9 shows the results on SWT project. Again, the accuracy is high with SVM achieving 91.4%. Across the three projects, the proposed feature set is discriminant and capable of distinguishing between the different classes. Moreover, the feature set seems to have stable performance results across different classifiers with no dramatic changes in performance.

Table 7. Accuracy results on AspectJ project

Classification algorithm	Accuracy (%)
Naïve Bayes	97.0
SVM	96.6
Random Trees	96.6

Table 8. Accuracy results on Tomcat project

Classification algorithm	Accuracy (%)
Naïve Bayes	91.4
SVM	91.4
Random Trees	85.8

Table 9. Accuracy results on SWT project

Classification algorithm	Accuracy (%)
Naïve Bayes	90.2
SVM	91.4
Random Trees	86.4

Table 10. Average Accuracy results across different projects

Classification algorithm	AspectJ	Tomcat	SWT	Average Accuracy (%)
Naïve Bayes	97.0	91.4	90.2	92.9%
SVM	96.6	91.4	91.4	93.1%
Random Trees	96.6	85.8	86.4	89.6%

Table 10 shows the average accuracy of each learning algorithm on the three datasets. We can conclude that SVM provides the highest accuracy performance on the three projects with a reported accuracy of 93.1%.

4. CONCLUSIONS AND FUTURE WORK

Automatic classification of newly incoming bug reports into predefined categories reduces maintenance cost. It helps maintainers to understand and process these change requests. In this paper, an efficient approach has been proposed to classify bug reports into corrective and perfective classes. As a result; this helps maintainers to quickly understand these bug reports and hence, allocate resources for each category.

The approach is based on utilizing machine learning algorithms to classify bug reports. The used dataset is a set of small number of representative features that have been proposed to enhance the accuracy and the training time which is different from previous approaches that depend on the whole data term matrix. These proposed features achieved high accuracy in classification with SVM classification algorithm reporting the highest average accuracy of (93.1%) on three different open source projects, which makes these features valid to be applied on other projects.

Our future work aims to achieve the following topics:

- Using machine learning techniques to predict fault locations.
- Add more categories (classes) as adaptive or preventive for change requests.
- Integrate the proposed classifier within boosting algorithms for anenhanced performance

5. REFERENCES

[1] Bettenburg, N., Premraj, R., Zimmermann, T. and Kim, S., 2008, September. Duplicate bug reports considered harmful… really?. In 2008 IEEE International Conference on Software Maintenance (pp. 337-345). IEEE.

[2] Tian, Y., Lo, D. and Sun, C., 2012, October. Information retrieval based nearest neighbor classification for fine-grained bug severity prediction. In *2012 19th Working Conference on Reverse Engineering* (pp. 215-224). IEEE.

[3] Anvik, J., Hiew, L. and Murphy, G.C., 2006, May. Who should fix this bug?. In *Proceedings of the 28th international conference on Software engineering* (pp. 361-370).

[4] Levin, S. and Yehudai, A., 2017, November. Boosting automatic commit classification into maintenance activities by utilizing source code changes. In *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering* (pp. 97-106). ACM.

[5] Pandey, N., Sanyal, D.K., Hudait, A. and Sen, A., 2017. Automated classification of software issue reports using machine learning techniques: an empirical study. *Innovations in Systems and Software Engineering*, *13*(4), pp.279-297.

[6] Antoniol, G., Ayari, K., Di Penta, M., Khomh, F. and Guéhéneuc, Y.G., 2008, October. Is it a bug or an enhancement?: a text-based approach to classify change requests. In *CASCON* (Vol. 8, pp. 304-318).

[7] Zhou, Y., Tong, Y., Gu, R. and Gall, H., 2016. Combining text mining and data mining for bug report classification. *Journal of Software: Evolution and Process*, *28*(3), pp.150-176.

[8] Pingclasai, N., Hata, H. and Matsumoto, K.I., 2013, December. Classifying bug reports to bugs and other requests using topic modeling. In *2013 20th Asia-Pacific Software Engineering Conference (APSEC)* (Vol. 2, pp. 13-18). IEEE

[9] Fu, Y., Yan, M., Zhang, X., Xu, L., Yang, D. and Kymer, J.D., 2015. Automated classification of software change messages by semi-supervised Latent Dirichlet Allocation. *Information and Software Technology*, *57*, pp.369-377.

[10] Limsettho, N., Hata, H., Monden, A. and Matsumoto, K., 2014, November. Automatic unsupervised bug report categorization. In *2014 6th International Workshop on Empirical Software Engineering in Practice* (pp. 7-12). IEEE.

[11] Tian, K., Revelle, M. and Poshyvanyk, D., 2009, May. Using latent dirichlet allocation for automatic categorization of software. In *2009 6th IEEE International Working Conference on Mining Software Repositories* (pp. 163-166). IEEE.

[12] Zibran, M.F., 2016, May. On the effectiveness of labeled latent dirichlet allocation in automatic bug-report categorization. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)* (pp. 713-715). IEEE.

[13] Hindle, A., German, D.M., Godfrey, M.W. and Holt, R.C., 2009, May. Automatic classication of large changes into maintenance categories. In *2009 IEEE 17th International Conference on Program Comprehension* (pp. 30-39). IEEE.

[14] Witten, I.H. and Frank, E., Data Mining: Practical machine learning tools with Java implementations. 2000.