Automated Classification of Issue Reports from a Software Issue Tracker

Nitish Pandey, Abir Hudait, Debarshi Kumar Sanyal and Amitava Sen

Abstract Software issue trackers are used by software users and developers to submit bug reports and various other change requests and track them till they are finally closed. However, it is common for submitters to misclassify an improvement request as a bug and vice versa. Hence, it is extremely useful to have an automated classification mechanism for the submitted reports. In this paper we explore how different classifiers might perform this task. We use datasets from the open-source projects HttpClient and Lucene. We apply naïve Bayes (NB), support vector machine (SVM), logistic regression (LR) and linear discriminant analysis (LDA) separately for classification and evaluate their relative performance in terms of precision, recall, *F*-measure and accuracy.

Keywords Bug classification · Naïve Bayes · Support vector machine · Precision · Recall · F-measure

1 Introduction

Software evolves continuously over its lifetime. As it is developed and maintained, bugs are filed, assigned to developers and fixed. Bugs can be filed by developers themselves, testers or customers, or in other words by any user of the software.

N. Pandey (✉) · A. Hudait · D.K. Sanyal
School of Computer Engineering, KIIT University, Bhubaneswar 751024, Odisha, India
e-mail: nitish5808@gmail.com

A. Hudait
e-mail: abirhudait@gmail.com

D.K. Sanyal
e-mail: debarshisanyal@gmail.com

A. Sen
Dr. Sudhir Chandra Sur Degree Engineering College, Kolkata 700074, West Bengal, India
e-mail: amitavasen@yahoo.com

P.K. Sa et al. (eds.), *Progress in Intelligent Computing Techniques: Theory, Practice, and Applications*, Advances in Intelligent Systems and Computing 518,
DOI 10.1007/978-981-10-3373-5_42

For open-source projects, defect tracking tools like GNATS [1], JIRA [2] or Bugzilla [3] are commonly used for storing bug reports and tracking them till closure. Proprietary software also uses similar tools. However, along with bugs (for corrective maintenance), it is common to file change requests that ask for adaptation of the software to new platforms (adaptive maintenance) or to incorporate new features (perfective maintenance). Similarly, there may be requests to update the documentation, which may not be clubbed as a bug due to its usually less serious impact. Requests for code refactoring, discussions and request for help are other categories for which users may file an issue. However, the person filing the reports may not always make a fine-grained distinction between these different kinds of reports and instead record them as bugs only. The consequence could be costly: developers must spend their precious time to look into the reports and reclassify them correctly. Hence, it is worthwhile to explore whether this classification could be performed automatically.

Machine learning, especially techniques in text classification and data mining, provides invaluable tools to classify bug reports correctly. We call a report in a software defect tracking system as an *issue report* irrespective of whether it refers to a valid bug or it is related to some other issue (as discussed above). In this paper, we study how machine learning techniques can be used to classify an issue report as a *bug* or a *non-bug* automatically. One simple way to distinguish between these two kinds of requests is to look for the text patterns in them. Certain words that describe errors or failures in the software are more common in descriptions that truly report a bug. This suggests that supervised learning techniques like classification can be used. A classifier is initially trained using the data of an issue tracker and subsequently used to label a new issue report.

Contribution: We use a collection of issue reports from the open-source projects HttpClient [4] and Lucene [5] as present in the issue tracker JIRA [2]. Only summary part of each report is parsed and used. We study the performance of various classifiers on these issue summaries. More specifically, we apply naïve Bayes (NB) classifier, support vector machine (SVM), logistic regression (LR) and linear discriminant analysis (LDA) separately for classification and evaluate their relative performance in terms of precision, recall, *F*-measure and accuracy. In an attempt to find the best classifier, we observe that, in terms of *F*-measure, SVM followed by NB performs significantly better than other classifiers for both HttpClient and Lucene projects. The classification accuracies obtained by NB and SVM are also better than those of other classifiers for each project. For each of *F*-measure and accuracy, the values for NB and SVM are close to each other. Hence, NB or SVM appears to be a better choice compared to other classifiers for automatic issue report classification.

Roadmap: A brief background of the current research is provided in Sect. 2. Related work is reported in Sect. 3. Our proposed approach is outlined in Sect. 4, while Sect. 5 describes the experiments, results and threats to validity of the results. The conclusion appears in Sect. 6.

2 Background

Software issue reports capture crucial information about the problem faced by the user who filed the report. A host of issue tracking tools is available, each with varying degrees of sophistication in recording the issue filed. The variations occur in the number of fields that the user needs to fill into the number of stages that the issue goes through before it is declared closed. Note that closing could refer to either fixing the issue or declaring it as void (i.e., invalid). We used issue reports from the issue tracking tool JIRA [2]. An issue in JIRA could report a host of different things like bug, maintenance, improvement, document update, code refactoring. In our discussion we will categorize reports into two classes: *bug* and *non-bug* (note: we use *non-bugs* to refer to all reports that are not categorized as *bug* in JIRA). We use supervised learning tools [6] to automatically segregate the reports into these two categories. Supervised learning involves two steps: (1) training a classifier using labeled samples and (2) classifying an unknown test case after it is trained. We use four kinds of supervised learning algorithms: (1) naïve Bayes (NB) classifier, (2) support vector machine (SVM), (3) logistic regression (LR) and (4) linear discriminant analysis (LDA).

3 Related Work

Analysis of software issue reports submitted to issue tracking tools is a common research area due to its applications in triaging issue reports [7], grouping bugs into different types, estimating issue resolution time and providing feedback on the quality of reports. The extent and cost of misclassification are studied in [8]. Researchers have suggested various methods to automatically classify the reports so that even if the original issue type reported by the user is incorrect, the right type can be inferred and used for further analysis by application engineers. Antoniol et al. [9] manually classified 1800 issues collected from issue trackers of Mozilla, Eclipse and JBoss projects into two classes: bug and non-bug. They investigated the use of various information contained in the issue reports for the classification. They also performed automatic classification of (a smaller subset of) issue reports using naïve Bayes, ADTree and linear logistic regression classifiers. Recently, Ohira et al. [10] manually reviewed 4000 issue reports in JIRA from four open-source projects and manually classified them based on their impact (e.g., security bug, performance bug) on the project. Pingclasai, Hata and Matsumoto [11] reported results of automated bug report classification done with topic modeling (using latent Dirichlet allocation) followed by application of one of the three techniques—ADTree, naïve Bayes classifier and logistic regression—on the issue repositories of three open-source projects. Chawla and Singh [12] proposed a fuzzy logic-based technique to classify issue reports automatically. They have reported higher values of F-measure compared to [11] for each of the same three projects. However, [12]

used a smaller dataset; so one might wonder whether the results would hold when the repository is much larger. Wu et al. [13] developed the BugMiner tool that uses data mining on historical bug databases to derive valuable information that may be used to improve the quality of the reports as well as detect duplicate reports. Zhou et al. [14] employed text mining followed by data mining techniques to classify bug reports. Like the preceding works, we too study automatic classification of issue reports but use a partially different collection of classifiers (e.g., SVM and LDA are added). However, we do not use topic modeling but a simple term–frequency matrix as an input to the classifiers. We use the highly reliable R [15] environment for experiments. We attempt to identify the classifiers that can be used with satisfactory performance.

4 Our Approach

We now describe the approach to classify the unseen reports into their belongingness. The approach is shown schematically in Fig. 1. The issue reports are first parsed, and only the summary from each report is taken. The body of the report as well as other details like heading, ID, category, description are ignored since it is time-consuming to process them and is usually not found to be of added value to classification [9]. The summary is then preprocessed: common words and stop words are removed from each summary, each word is stemmed and tokenized into terms, and finally the frequency of each term is computed to create the term–frequency matrix (*tfm*) for each report. As argued in [9], *tfm* is probably better suited compared to *tf-idf* (term frequency–inverse document frequency) indexing for software issue classification. Note that we remove terms with frequency lower than a threshold. The preprocessed reports (or alternatively, the *tfm*) are divided into training and testing sets. The training set is used to train the classifier, while the testing set is used to study how well—with respect to chosen metrics—the classifier performs the task of classification.

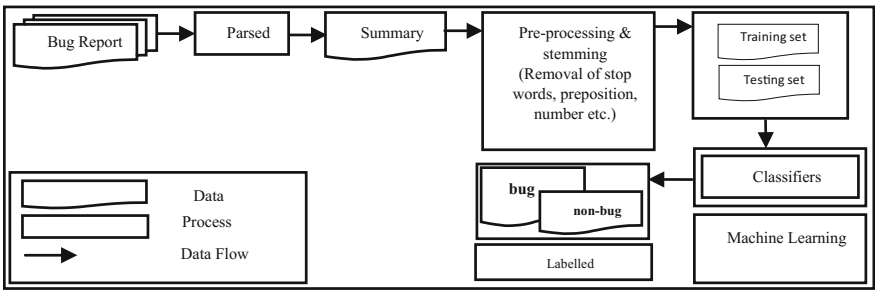

Fig. 1 Proposed approach for issue classification

5 Experiments, Results and Discussion

We use a subset of manually classified issue reports from HttpClient [4] and Lucene [5] projects as provided by [8]. The issue reports were, in turn, extracted from JIRA by researchers [8]. Only the summary part of the reports is taken into consideration for classification. The count of reports used is shown in Table 1.

For each project, we randomly partition the dataset into two subsets for training and testing, respectively, in the ratio 80:20. Only report summary is used for processing. The classifiers are trained first. In the testing phase, each issue report is first preprocessed just like we do in the training phase so that we have only unique terms in the datasets now. These preprocessed datasets are then given to the classifiers to assign labels. The classifier uses past knowledge (i.e., of training phase) to find the belongingness of the reports. The experiments for each project are conducted ten times, each time with a random 80:20 partition to compute the parameters of interest (as explained in the next subsection), and their average values are reported.

We use the R [15] language and environment to perform the experiments. We used R version 3.2.1 which contains implementations of the classifiers NB, SVM, LR and LDA.

5.1 Performance Measures

To measure the performance of the classifiers, we use the metrics: precision, recall, *F*-measure and accuracy. Before we define them, we look at four important quantities that measure how the classifier classified the test inputs as belonging to or not belonging to the (positive) class *bug*.

1. True positive (TP): number of reports correctly labeled as belonging to the class.
2. True negative (TN): number of reports correctly rejected from the class.
3. False positive (FP): number of reports incorrectly labeled as belonging to the class.
4. False negative (FN): number of reports incorrectly rejected from the class.

The entries of the confusion matrix, in terms of the above vocabulary, are indicated in Table 2.

Using the measurements and the following formulae, we calculate precision, recall, *F*-measure and accuracy.

Table 1 Projects and their issue reports

Project name	#Total reports	#Bug	#Non-bug
HttpClient	500	311	189
Lucene	253	110	143

Table 2 Confusion matrix of classifier

		Results of classifier	
		Bug	Non-bug
True classification	Bug	TP	FN
	Non-bug	FP	TN

(a) **Precision**: It is the ratio of the number of true positives to the total number of reports labeled by the classifier as belonging to the positive class.

$$precision = \frac{TP}{TP + FP} \tag{1}$$

(b) **Recall**: It is the ratio of the number of true positives to the total number of reports that actually belong to the positive class.

$$recall = \frac{TP}{TP + FN} \tag{2}$$

(c) ***F*-measure**: It is the harmonic mean of precision and recall.

$$F = 2 \times \frac{precision \times recall}{precision + recall} \tag{3}$$

(d) **Accuracy**: It measures how correctly the classifier labeled the records.

$$accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{4}$$

5.2 Results and Discussion

We classify the issue reports on HttpClient and Lucene in JIRA as bug or non-bug, i.e., we perform binary classification. The performance of each classifier in terms of the above metrics for each project is given in Tables 3 and 4. The highest and

Table 3 Precision, recall, *F*-measure and accuracy for NB, SVM, LR and LDA on HttpClient project

	NB	SVM	LR	LDA
Precision	**0.752658**	0.736468	0.676794	0.709587
Recall	0.802054	**0.843185**	0.599136	0.692016
***F*-measure**	0.775167	**0.784175**	0.633574	0.699153
Accuracy	**0.714**	0.711788	0.564	0.621396

Table 4 Precision, recall, *F*-measure and accuracy for NB, SVM, LR and LDA on Lucene project

	NB	SVM	LR	LDA
Precision	0.771069	**0.811171**	0.476134	0.504317
Recall	0.51246	0.538948	**0.541953**	0.510138
***F*-measure**	0.612841	**0.640172**	0.500015	0.503319
Accuracy	**0.721569**	0.719608	0.560785	0.568627

second highest values of each metric are highlighted for each project. In case of HttpClient, SVM performs best while NB is the second best in terms of precision, recall and *F*-measure. The performance of the other classifiers is far worse. Highest accuracy is provided by NB in classification of HttpClient reports with SVM at the second position. For Lucene, in terms of precision and *F*-measure, SVM again performs best while NB is second best. LR, however, gives the highest recall value in case of Lucene. The highest accuracy is again achieved by the NB classifier, while SVM is behind by a small margin. Overall, both SVM and NB perform very well for each project.

5.3 Threats to Validity

The results obtained in this paper are sensitive to the choice of the datasets. In particular, if the datasets are changed, the outcomes, i.e., precision, recall, *F*-measure and accuracy, may change. The datasets we used are small, and hence, the results might not be reflective of the outcome for a larger sample. We used data from only two open-source projects. The results may be different for other projects.

6 Conclusion

We used four classifiers, namely NB, SVM, LR and LDA, to classify issue reports into bug and non-bug categories. The experiments were conducted using issue reports of the projects HttpClient and Lucene from the JIRA issue tracker. In terms of *F*-measure, SVM followed by NB performs significantly better than other classifiers for HttpClient and Lucene. The classification accuracies obtained using NB and SVM are comparable. They are far better than those of other classifiers for each project. Hence, NB or SVM appears to be a good choice for automatic issue report classification. Since the datasets are not quite large and belong to only two projects, we refrain from making general comments on the exact numerical results. Implementations of NB and SVM are widely available. Our results suggest it might

be profitable to classify the reports using these classifiers before further analysis by developers or managers. In future, we plan to use other classifiers including ensemble classifiers and expand the dataset to larger number of reports and projects.

References

1. https://www.gnu.org/software/gnats/.
2. https://www.atlassian.com/software/jira.
3. https://www.bugzilla.org/.
4. https://hc.apache.org/httpclient-3.x/.
5. https://lucene.apache.org/.
6. F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys (CSUR)* 34.1:1–47, 2002.
7. D. Čubranić. Automatic bug triage using text categorization. In *Proceedings of the 16th International Conference on Software Engineering & Knowledge Engineering (SEKE'2004)*, 2004.
8. K. Herzig, S. Just, and A. Zeller. It's not a bug, it's a feature: How Misclassification Impacts Bug Prediction. In *Proceedings of the 35th IEEE/ACM International Conference on Software Engineering*, 2013.
9. G. Antoniol, et al. Is it a bug or an enhancement? A text-based approach to classify change requests. In *Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds (CASCON'2008)*, ACM, 2008.
10. M. Ohira, et al. A dataset of high impact bugs: manually-classified issue reports. In *Proceedings of the IEEE/ACM 12th Working Conference on Mining Software Repositories (MSR'2015)*, 2015.
11. N. Pingclasai, H. Hata, K. Matsumoto. Classifying bug reports to bugs and other requests using topic modeling. In *Proceedings of 20th Asia-Pacific Software Engineering Conference (APSEC'2013)*, IEEE, 2013.
12. I. Chawla, S. K. Singh. An automated approach for bug classification using fuzzy logic. In *Proceedings of the 8th ACM India Software Engineering Conference (ISEC'2015)*, 2015.
13. L. L. Wu, B. Xie, G. E. Kaiser, R. Passonneau. BugMiner: software reliability analysis via Data Mining of Bug Reports. In *Proceedings of the 23rd International Conference on Software Engineering & Knowledge Engineering (SEKE'2011)*, 2011.
14. Y. Zhou, Y. Tong, R. Gu, H. Gall. Combining text mining and data Mining for bug report classification. In *Proceedings of 30th IEEE International Conference on Software Maintenance and Evolution (ICSME'2014)*, 2014.
15. https://www.r-project.org/about.html.