Classifying Software Issue Reports through Association Mining

Mohd Syafiq Zolkeply
School of Computer Science and Informatics
Cardiff University
Cardiff, United Kingdom
zolkeplym@cardiff.ac.uk

Jianhua Shao
School of Computer Science and Informatics
Cardiff University
Cardiff, United Kingdom
shaoj@cardiff.ac.uk

ABSTRACT

Software issue reports classification is a significant task in software maintenance and evolution. Despite the research effort being made over the years, the existing issue reports classification techniques are still inadequate. In this paper, we propose a new approach that is inspired by the Classification Associations Rule Mining (CARM) methodology in data mining, and report the testing of our method on 500 software issue reports extracted from an open source issue tracking system. Our experiments show that our method can achieve a high degree of accuracy in classifying software issue reports.

CCS CONCEPTS

• **Software and its engineering** → **Maintaining software**; • **Applied computing** → *Document management*;

KEYWORDS

Mining software repositories, association rules mining, software maintenance, software issue report

ACM Reference Format:
Mohd Syafiq Zolkeply and Jianhua Shao. 2019. Classifying Software Issue Reports through Association Mining. In *Proceedings of ACM SAC Conference (SAC'19).* ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.1145/3297280.3297608

1 INTRODUCTION

Software maintenance is an important phase in the life cycle of a software application. Activities like bugs fixing, feature enhancement and platform upgrade are usually triggered by issues raised by software users during software maintenance. For example, users of open source software log the issues they encounter in an issue tracking system. Figure 1 shows two examples of issue report and its type;

However, Herzig et al [5] reported, 33.8% of the issue reports submitted by the users were misclassified. Consequently, the software maintenance team often need to re-classify such reports before assigned them to the relevant software developers. To address the issue, some attempts have been made [2, 3, 9, 11, 12] and [10]. Although some progress has been made, the existing approaches typically look for some "dominant" patterns to build a classifier. Such approaches may not work well in practice as issue reports are freely written by users with little constraint on what they can include and what vocabulary they may use. Consider the example given in Figure 1, for instance. `error` appears in both reports, but one is associated with bug reporting and the other feature requesting. As such, `error` is not discriminative enough, thus is likely to be ignored and more dominant terms or combination of terms will be sought to build a classifier by the existing approaches. We argue that non-dominant patterns are also important to building an effective classifier for issue reports.

R1, Type: Bug
Error releasing chunked connections with no response body. http method base release connection does not successfully release the connection if closing the response stream throws an exception.

R2, Type: Feature Request
Allow declare error and declare warning to support type expressions.

Figure 1: Example software issue reports

In this paper, we propose a new approach to classifying issue reports inspired by classification association rule mining methodology [6, 7]. Instead of looking for dominant patterns, our approach looks for any *credible* patterns. That is, when a pattern occurs frequently enough, even if it is not dominant, it is considered useful. We tested our method on 500 software issue reports extracted from an issue tracking system using dataset provided by [5, 13]. Our experiments show that the proposed method can achieve a high degree of accuracy in classifying software issue reports.

2 PROPOSED APPROACH

2.1 Preliminaries

Without loss of generality we assume an issue report R is represented as a vector of terms $R = \langle t_1, t_2, \ldots, t_n, C \rangle$, where each $t_i, 1 \leq i \leq n$ is a distinct term extracted from the issue report text and C is its category. For example, R2 given in

SAC'19, April 8-12, 2019, Limassol, Cyprus

ACM ISBN 978-1-4503-5933-7/19/04…$15.00
https://doi.org/10.1145/3297280.3297608

Example 1 may be represented as

$$R2 = \langle error, warning, type, expressions, feature \rangle$$

Here, R2 is represented, rather arbitrarily, by a vector that contains the nouns appeared in it and a category term `feature`. From a set of vectorised reports $\mathbb{R} = \{V_1, V_2, \ldots, V_m\}$, rules of the form

$$r : a_1, a_2, \ldots, a_k \rightarrow c$$

are generated, where $a_1, a_2, \ldots, a_k$ is called an *association* of terms and c is a category. Our goal is to derive a set of such rules that can be used to classify issue reports accurately. To ensure that the rules that we derive from a corpus of issue reports have some "minimal credibility", we employ two commonly used measures [1]:

- **support.** This is a count of how many times the association of terms $(a_1, a_2, \ldots, a_k)$ of a rule $(r : a_1, a_2, \ldots, a_k \rightarrow c)$ has occurred in a set of reports $(\mathbb{R})$. Support indicates the *strength* of a rule.
- **confidence.** This is the ratio of the number of times a rule $(r : a_1, a_2, \ldots, a_k \rightarrow c)$ has occurred to the number of times the association of terms of the rule $(a_1, a_2, \ldots, a_k)$ has occurred by itself in a set of reports $(\mathbb{R})$. Confidence indicates the *accuracy* of a rule.

These two measure are specified by the users of our method and tuned for particular datasets, and we search for all the rules from a set of vectorised reports that have the minimum support and confidence. The rules are known as *credible* rules.

2.2 Rules Generation

To derive a set of all credible rules, we use the *rules generation* algorithm shown in Algorithm 1 which is inspired by the CARM methodology [6, 7]. The algorithm works as follows. Each distinct category c_j is considered in turn (step 2). For each c_j, we select the subset of vectors T_{c_j} from $\mathbb{R}$ that contain c_j as a category (step 3). We then extract single terms from T_{c_j} that have sufficient support (step 4), and we denote this set as L_1 and call it *large single terms*. Note that while we calculate support for each term in T_{c_j} only, the support calculation itself is based on the entire dataset $\mathbb{R}$, not T_{c_j}.

Next, it goes into iteration (step 5). Each association of i terms in L_i $(t_1, t_2, \ldots, t_i)$ is paired with category c_j to form a rule $(t_1, t_2, \ldots, t_i \rightarrow c_j)$ and we check if the rule has sufficient confidence (steps 6-7). Note that this confidence calculation needs no further scan of the dataset $\mathbb{R}$, as the support for $t_1, t_2, \ldots, t_i$ is already available, first from step 4 then from step 9 (see below), and the occurrence of $t_1, t_2, \ldots, t_i \rightarrow c_j$ can be obtained by a scan of T_{c_j}. Rules with sufficient confidence are retained and others are discarded (step 8). Once rule generation is done for associations of i terms, the Generate function attempts to generate associations of $i+1$ terms from the i terms, following the well-known apriori principle [1] (step 9). After this, the algorithm goes into the second round, attempting to find associations having two terms to form rules. The generated rule is retained in $\mathcal{R}$. No more associations of terms may be generated, and computation involving the `bug` category is complete.

Algorithm 1 *Rules Generation*

input: report vectors $\mathbb{R}$ and class values C
the user defined thresholds $minSupp$, $minConf$
output: a set of credible rules $\mathcal{R}$

1. $\mathcal{R} \leftarrow \emptyset$;
2. **for** each c_j in C **do**
3. $T_{c_j} \leftarrow$ Select$(\mathbb{R}, c_j)$
4. $L_1 \leftarrow \{\ v \mid minSupp \leq |v|, v$ in the domain of $T_{c_j}\}$
5. **for** $(i = 1,\ L_i \neq \emptyset,\ i{+}{+})$ **do**
6. **for** each $t_1, t_2, \ldots, t_i$ in L_i **do**
7. **if** conf$(t_1, t_2, \ldots, t_i \rightarrow c_j) \geq minConf$
8. $\mathcal{R} \leftarrow \mathcal{R} \cup t_1, t_2, \ldots, t_i \rightarrow c_j$
9. $L_{i+1} \leftarrow$ Generate(L_i)
10. **return** $\mathcal{R}$

2.3 Report Classification

Multiple rules generated by our association mining may be fired during the classification process. To deal with this situation, we resort to *majority vote*. This is shown in Algorithm 2.

Algorithm 2 *Report Classification*

input: a set of credible rules $\mathcal{R}$ and
a vectorised issue report $V = \langle t_1, t_2, \ldots, t_k \rangle$
output: the category of V

1. **for** each rule $r : t_1, t_2, \ldots, t_k \rightarrow c_j$ in $\mathcal{R}$ **do**
2. **if** V *covers* r
3. $Count_{c_j}{+}{+}$
4. $S \leftarrow \max_{\forall c_j}(Count_{c_j})$
5. **if** $|S| > 1$
6. **return** "unable to classify"
7. **else**
8. **return** c_j associated with S

We take each derived rule $r : t_1, t_2, \ldots, t_k \rightarrow c_j$ in turn (step 1). If the vector to be classified (V) covers r, i.e. every term in the antecedent of r appears in V (step 2), then r is fired and we increase the counter for category c_j by 1 (step 3). Once all the rules are checked, we consider the categories that have the largest counts (step 4). When there is a clear winner (i.e. there is a single largest count), this category will be returned as the category for the report. If there is a tie, then our method will report that it is unable to classify the vector.

3 EMPIRICAL EVALUATION

3.1 Experiment Setup

We used datasets from [5, 13] in our experiments which were extracted from two Open Source Software (OSS) projects. The distribution of data used in training and testing are shown in Table 1.

Table 1: Dataset

Project	Training Data	Testing Data
Http-Client	200	50
AspectJ	200	50
Total	400	100

3.2 Data Preparation

To vectorise issue reports, we used NLTK toolkit [8]. Similar to previous studies [2, 3, 9, 11, 12] and [10], we performed standard text processing and manually specify the keywords used as features to classify an issue report. That is, we randomly selected 10 issue reports used in previous studies [5, 13], and extracted a maximum of 60 keywords from them. We represent our reports as a binary table shown below, i.e report 1 contains keyword `error`, not `bug`, etc:

ID	error	bug	···	issue	Category
1	1	0	···	1	bug
:	:	:	:	:	:
500	1	1	···	0	non-bug

3.3 Experimental Results

We observe the classification accuracy by varying *minSupp*, *minSupp* and *number of keywords* used in our method, and measure the accuracy as follows:

$$Accuracy = \frac{T_{correct}}{T_{total} - T_{unknown}}$$

where $T_{correct}$ is the number of correctly classified reports, T_{total} the total number of reports used in testing, and $T_{unknown}$ the number of reports that our method is unable to classify as a result of our majority vote. We excluded these "unable to classify" reports from our accuracy measure because they are not classified by our method as `bug` or `non-bug`, so we are unable to establish whether our method has correctly or wrongly classified them.

It is worth noting that allowing unclassified cases is an important feature of our new method: when there is no clear evidence (majority) to suggest that a report is `bug` or `non-bug`, we argue that it is better to leave such cases to the human expert rather than forcing a classification. Once these reports are classified by human experts, we can incorporate this knowledge into our classifier through a form of machine learning, thereby gradually reducing the number of "unable to classify reports" in future and allowing our model to evolve over time. Further discussion on this topic is beyond the scope of the current paper. In all experiments, we compare our method to several popular machine learning algorithms such as Multinomial Naive Bayes (NB), Decision Tree (DT - C4.5), Support Vector Machine (SVM) and Ripper (JRip) available from the WEKA toolkit [4].

3.3.1 Varying minConf on Accuracy. This experiment assessed the impact of varying *minConf* on classification accuracy, while fixing *minSupp* at 5 and keywords at 60. Figure 2 shows the accuracy of using our method to classify the testing reports and plotted the accuracy of four standard machine learning algorithms used in our study for comparison and reference.

The number of rules derived from our association mining decreases as *minConf* increases, since more accurate rules are demanded, naturally less such rules will be there to find. However, the accuracy of our classification suggests that highly confident rules will not help classify reports accurately. As we can see, when *minConf* was set at 80% or less, our method outperformed the four popular machine learning methods. While this may be counter-intuitive, we argue, this validates our hypothesis that is searching for dominant patterns (rules) only may not help classify issues reports accurately, and it is better to look for all credible patterns.

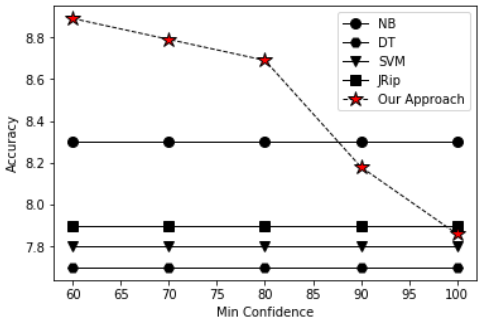

Figure 2: Accuracy vs *minConf*

3.3.2 Varying minSupp on Accuracy. This experiment assessed the impact of varying *minSupp* on classification accuracy, while fixing *minConf* at 60% and keywords at 60. Figure 3 shows the accuracy of using our method to classify the testing reports. The accuracy of other four methods used in our study are also plotted in Figure 3 for comparison. Again, a similar pattern is observed here: as we increase *minSupp* (requiring stronger rules), less rules are obtained, but accuracy of classification increases, and when *minSupp* was set low, our method performed better than the four standard techniques. This suggests, as in the case of varying *minConf*, it is useful to look for many credible rules, not just a few strong rules in issue reports classification.

3.3.3 Varying Number of Keywords on Accuracy. This experiment assessed the impact of varying the number of keywords used in report vectorisation on classification accuracy, while fixing *minConf* at 60% and *minSupp* at 10. To avoid bias, we randomly chose the keywords from the set of 60. Figure 4 shows the accuracy of using our method to classify the testing reports compared with four other methods used in our study.

The number of rules generated increased as more keywords were used in representing the issue reports. This is expected as the more keywords are used, the more associations among the keywords will form, since with the classification association mining methodology, each association of terms can be

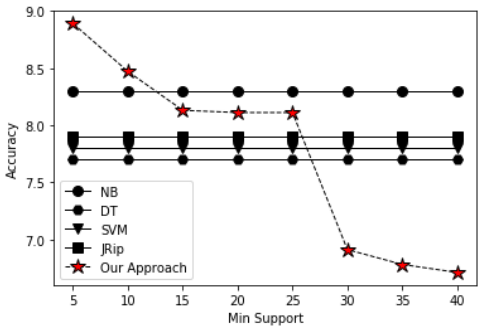

Figure 3: Accuracy vs *minSupp*

obtained independently from other associations. It is clear from Figure 4 that the accuracy is better when more keywords used to represent issue reports, and when both *minConf* and *minSupp* were set low, our method performed best in comparison to the four popular methods we used in our study. This is consistent with our observations from experiments involving varying *minConf* and *minSupp*: the more rules are generated, the greater the classification accuracy is.

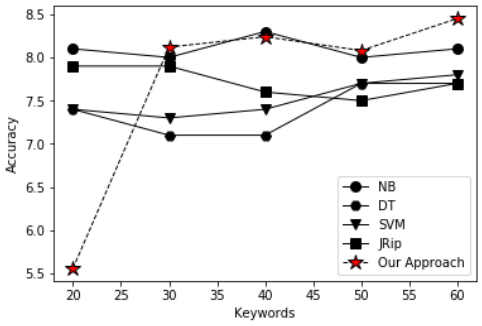

Figure 4: Accuracy vs Number of Keywords

4 THREATS TO VALIDITY

The main threat to the validity of our study would be in term of the dataset size and number of projects used in testing our proposed method. Unlike other related studies [2, 3, 9, 11, 12] and [10] where experiments have been conducted on relatively large scale datasets spanning across many different projects, our work has been tested on a relatively small dataset of 500 reports from two open source projects. However, the experiments reported in this paper have clearly shown the merit and promise of our new approach. We will continue to evaluate the performance of our method using larger datasets extracted from more Open-Source Software (OSS) projects.

5 CONCLUSIONS

In this paper, we proposed a new approach to classifying issue reports for software maintenance. Our method is inspired by the Classification Association Rule Mining methodology. Instead of looking for some dominant patterns from issue reports to build a classifier, we search for all credible patterns to build not so strong on their own, but collectively powerful rules to classify issue reports. This is useful and important when dealing with cases where multiple semantics may be associated with certain keywords or an issue report may actually contain multiple issues. Our preliminary experiments show that our method is able to achieve high accuracy in classifying issue reports extracted from a mixture of projects.

REFERENCES

[1] Rakesh Agrawal and Ramakrishnan Srikant. 1994. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 487–499.

[2] Giuliano Antoniol, Kamel Ayari, Massimiliano Di Penta, Foutse Khomh, and Yann-Gaël Guéhéneuc. 2008. 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 '08)*. ACM, New York, NY, USA, Article 23, 15 pages.

[3] Indu Chawla and Sandeep K. Singh. 2015. An Automated Approach for Bug Categorization Using Fuzzy Logic. In *Proceedings of the 8th India Software Engineering Conference (ISEC '15)*. ACM, New York, NY, USA, 90–99.

[4] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.* 11, 1 (Nov. 2009), 10–18.

[5] Kim Herzig, Sascha Just, and Andreas Zeller. 2013. It's Not a Bug, It's a Feature: How Misclassification Impacts Bug Prediction. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*. IEEE Press, Piscataway, NJ, USA, 392–401.

[6] Wenmin Li, Jiawei Han, and Jian Pei. 2001. CMAR: accurate and efficient classification based on multiple class-association rules. In *Proceedings 2001 IEEE International Conference on Data Mining*. 369–376.

[7] Bing Liu, Wynne Hsu, and Yiming Ma. 1998. Integrating Classification and Association Rule Mining. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98)*. AAAI Press, 80–86.

[8] Edward Loper and Steven Bird. 2002. NLTK: The Natural Language Toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1 (ETMTNLP '02)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 63–70.

[9] Nitish Pandey, Debarshi Kumar Sanyal, Abir Hudait, and Amitava Sen. 2017. Automated classification of software issue reports using machine learning techniques: an empirical study. *Innovations in Systems and Software Engineering* (24 Jul 2017).

[10] N. Pingclasai, H. Hata, and K. Matsumoto. 2013. Classifying Bug Reports to Bugs and Other Requests Using Topic Modeling. In *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, Vol. 2. 13–18. https://doi.org/10.1109/APSEC.2013.105

[11] Hanmin Qin and Xin Sun. 2018. Classifying Bug Reports into Bugs and Non-bugs Using LSTM. In *Proceedings of the Tenth Asia-Pacific Symposium on Internetware (Internetware '18)*. ACM, New York, NY, USA, Article 20, 4 pages.

[12] P. Terdchanakul, H. Hata, P. Phannachitta, and K. Matsumoto. 2017. Bug or Not? Bug Report Classification Using N-Gram IDF. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 534–538.

[13] Xin Ye, Razvan Bunescu, and Chang Liu. 2014. Learning to Rank Relevant Files for Bug Reports Using Domain Knowledge. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*. ACM, New York, NY, USA, 689–699.