A Cost-Effectiveness Metric for Association Rule Mining in Software Defect Prediction

Kinari Nishiura (k-nishiura@okayama-u.ac.jp)
Okayama University
Takeki Kasagi
Okayama University
Akito Monden
Okayama University

Research Article

Keywords: association rule mining, defect prediction, software metrics, software testing

Posted Date: August 25th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1988568/v1

A Cost-Effectiveness Metric for Association Rule Mining in Software Defect Prediction

Kinari Nishiura · Takeki Kasagi · Akito Monden

Received: date / Accepted: date

Abstract This paper proposes a cost-effectiveness metric for association rule mining suitable for software defect prediction where conditions of defective modules are characterized as association rules. Given a certain amount of test effort (or the number of test cases), the proposed metric is the expected number of defects to be discovered in modules that meet an association rule. Since the amount test effort is limited in general and full testing of all modules is ineffective, the proposed metric is useful to focus on the most cost effective set of modules to be tested with limited test effort. The proposed metric is defined based on the exponential Software Reliability Growth Model (SRGM) extended with the module size parameter, assuming that a larger module require more effort to discover defects. To evaluate the effectiveness of the proposed metric, association rules were extracted and prioritized based on the proposed metric using data sets of four open source software projects. The LOC-based cumulative-lift chart, which is often used to evaluate the cost effectiveness of defect prediction, shows that the proposed metric can focus on the rules that can discover more defects than the conventional association rule metrics, confidence and odds ratio.

Keywords association rule mining · defect prediction · software metrics · software testing

Kinari Nishiura
Okayama University, Okayama, Japan
E-mail: k-nishiura@okayama-u.ac.jp

Takeki Kasagi
Okayama University, Okayama, Japan
E-mail: pap466uc@s.okayama-u.ac.jp

Akito Monden
Okayama University, Okayama, Japan
E-mail: monden@okayama-u.ac.jp

Corresponding author: Kinari Nishiura, E-mail: k-nishiura@okayama-u.ac.jp

1 Introduction

Association rule mining is used in a variety of fields including defect prediction in software engineering [5] [11] [12] [17] [19]. For example, the association rule such as "fan-outs > 5 & max nest level > 3 $\Rightarrow$ *buggy*" is extracted to predict and understand defective modules [19]. This example indicates that a software module is likely to contain a defect if its fan-out is greater than five and its maximum nest level is greater than three. By extracting such rules from past software development data and applying them to ongoing projects, we can identify modules that need to assign more effort in software testing [5] [11].

One of the challenges of association rule mining is that so many association rules are generally extracted that it is difficult to know which rules to focus on [11]. It has been common practice to prioritize extracted rules using rule interestingness metrics such as support, confidence, and lift to solve this problem. Moreover, a combined metric of these metrics has also been proposed [19].

However, although conventional interestingness measures can help selecting rules, practical usefulness of the selected rules is still unclear especially in defect prediction. In the defect prediction senario, after selecting a rule, a set of modules that satisfy the rule's condition is obtained; then, these modules are tested and defect discovery is expected. Here, in case that the modules did not contain defects or they are too large to discover defects by testing, then the rule is not useful or even harmful. From the practical point of view, there is a need for a rule interestingness metric that takes into account of the ease of defect discovery with respect to the availabe test effort.

Therefore, we propose a cost-effectiveness metric of association rules. In this proposal, the "cost" is the amount of test effort (or the number of test cases) that can be assgined, and the "effectiveness" is the expected number of defects found when testing all modules that match the antecedent (conditional part) of an association rule. This proposal enables practitioners to select rules that are expected to find many defects, depending on the available test effort.

To evaluate our proposed metric, we conduct an experiment to extract and prioritize association rules using four open source software projects. We employ the LOC (Lines Of Code) -based cumulative lift chart, which is often used to evaluate the cost effectiveness of defect prediction, to compare the usefulness of the proposed metric with conventional interestingness metrics.

The rest of this paper is organized as follows: In Section 2, we explain association rule mining and their application to defect prediction. In Section 3, we propose a cost-effectieness metric of association rules. Section 4 describes the experimental evaluation of the proposed metric. Section 5 shows the results of the experiment. Section 6 discusses the threats to validity of the experiment. Finnaly, Section 7 shows the summary of this paper and future issues.

2 Related Works

2.1 Association Rule Mining

Association rule mining is a data analysis method proposed by Agrawal et al. [1]. This method can exhaustively extract co-occurring events from data. In the defect prediction scenario, let $M = \{M_1, M_2, \ldots, M_n\}$ be a set of software modules, let $C = \{C_1, C_2, \ldots, C_m\}$ be a set of module metrics, and let the (categorial) value of C_k of a module m_i be v_{ki}. Then, an association rule to find defects is expressed in the form of $(A \Rightarrow B)$, where A (antecedent) is one or more concatenation of $(C_k = v_{ki})$, and B (consequent) is "*buggy*" (*i.e.* a module contains a defect). The concatenation in the antecedent is denoted like $(C_x = v_{xi})\&(C_y = v_{yj})$ $(x \neq y)$ using the notation "&".

The *support* and the *confidence* are well-known rule interestingness metrics for selecting or prioritizing association rules as follows:

$$\text{support(A, B)} = \frac{\text{the number of modules that satisfy both A and B}}{\text{the number of all modules}}$$

$$\text{confidence(A, B)} = \frac{\text{the number of modules that satisfy both A and B}}{\text{the number of modules that satisfy A}}$$

The higher support means that A and B more often co-occur, which means that the rule is very often satisfied. The confidence indicates the strength of the association between A and B. Since association rule mining generally extracts many rules, it is common practice to set lower bounds on both support and confidence to reduce the obtained rule set.

2.2 Association Rule Mining in Defect Prediction

There have been attempts to use association rules for defect prediction and characterization in the past. Song et al. [17] applied association rule mining to bug-related data measured during software development (e.g., causes of bugs, effort for fixing bugs, etc.) to identify the types of bugs that are likely to co-occur and the conditions where bug fixing effort increases. Morisaki et al. [12] extended the association rule to allow a quantitative variable in the consequent of rules, and identified the conditions where the average (or the standard deviation) of bug fixing effort becomes large. Kamei et al. [5] combined logistic regression analysis and association rule mining to predict defect-prone modules. In contrast to these studies, this paper does not focus on prediction accuracy of the rules, but rather on the cost-effectiveness of the rules.

On the other hand, Watanabe et al. [19] proposed a method for prioritizing rules based on the estimated accuracy of defect prediction by cross-validation. Monden et al. [11] proposed a rule reduction technique that can eliminate complex (long) and/or similar rules without sacrificing the prediction performance.

Although these studies help eliminate unnecessary rules and obtain rules with high prediction accuracy, they cannot select rules in terms of cost-effectiveness.

3 Proposed Cost-Effectiveness Metric

In this section, we propose a new metric to prioritize association rules considering their cost-effectiveness in defect prediction.

We assume that the number of test cases t is given, and an association rule r is extracted in the form of "$A \Rightarrow buggy$". Then, the proposed metric $Cp(r, t)$ denotes the expected number of discoverable defects when t is assigned to a set of modules M that matches the antecedent A of the given rule r.

To compute $Cp(r, t)$, we follow an apporach to assess the cost-effectiveness of defect prediction proposed in [9], which employs the exponential Software Reliability Growth Model (SRGM) extended with a module size parameter. Assuming that test effort t_i is assigned to a (single) module m_i, the expected number of defects found $\hat{H}(t_i)$ is defined as follows:

$$\hat{H}(t_i) = a[1 - exp(-bt_i)] \tag{1}$$
$$b = \frac{b_0}{S}$$

b : Probability of finding a defect per test case
a : The number of defects in module m_i, before testing
S : Size of a module m_i
t_i : The number of test cases
b_0 : Constant

In this model, the probability of detecting a defect is inversely proportional to the module size. Given a certain number of test cases t_i, the ease of finding a defect becomes the same if the module size is the same. Similarly, a double size module requires double test cases to find the same number of defects [9]. Regarding the constant b_0, this paper uses $b_0 = 6.932$. This value is obtained by $b_0 = -(\frac{S}{t_i}) \cdot log(1 - \frac{\hat{H}(t_i)}{a})$ from equation (1), assuming unit testing and assigning 100 test cases per 1,000 lines of code (i.e., $\frac{S}{t_i} = 10$), which can detect 50% of all bugs (i.e., $\frac{\hat{H}(t_i)}{a} = 0.5$). The reason for assuming that 100 test cases are assigned per 1,000 lines is based on the following fact; (1) a large japanese software development company uses "100 cases per 1,000 lines" as the company standard for unit testing [14], (2) a medium-sized japanese software company assigned 99.31 test cases per 1,000 lines on average [10]. Therefore, we believe "100 test cases per 1,000 lines of code" is not the unrealistic value. In addition, the assumption that 50% of all bugs can be found under this condition is based

on the case that an average of 50% of bugs in the unit test cases is detected in some japanese software company [10]. Although b_0 can vary among different projects, this paper uses $b_0 = 6.932$ as one of the realistic values.

Since the set of modules M can contain two or more modules, we need to define how to assign test cases t to each modules in M. We assume that the number of test cases for each of the modules $m_1, \ldots, m_n$ is assigned in proportion to the size (lines of code) of the module. That is, let the size of module m_i is s_i, then the number of test cases t_i to be assigned to module m_i is $t_i = t \cdot s_i / \sum_{j=1}^{n} s_j$. Then, $Cp(r,t)$ is defined as follows

$$Cp(r,t) = \sum_{1}^{n} \hat{H}(t_i)$$

Our metric enables us to prioritize association rules based on their cost-effectiveness by calculating $Cp(r,t)$ for all extracted rules.

4 Evaluation

4.1 Dataset

As shown in Table 1, this experiment employed defect data sets from the four open source software projects: (1)Mylyn, (2) NetBeans, (3) Apache Ant and (4) jEdit. These data sets are also used in past defect prediction/characterization studies [11] [19]. Details of Mylyn and NetBeans data sets are described in [18]. Details of Apache Ant and jEdit data sets are described in [3] [4]. One module in this experiment refers to one source file.

In additon, Table 2 and 3 summarizes the metrics used in this experiment. These are reduced metrics set from original data sets to avoid extracting too many association rules. We removed one of the metrics that have high correlation coefficient with other metric. Also, metrics with very low correlation coefficients with the number of defects were removed. The resultant metrics sets are the same as those used in a previous work on defect prediction using association rule mining [11].

Table 1 Summary of defect data sets.

Project	Version	No. of metrics		No. of modules	% of defective modules
		Product	Process		
Mylyn	3.0	8	2	1,502	40.34
NetBeans	5.0	8	2	9,332	3.41
Apache Ant	1.7	11	0	745	22.28
jEdit	4.1	11	0	312	25.32

Table 2 Used metrics in Mylyn and NetBeans data sets

Name	Explaination
SLOC	Source lines of code
NBD	Nested block depth
PAR	The number of parameters
VG	Cyclomatic complexity
CBO	Coupling between object classes
NOF	The number of fields
NOM	The number of methods
RFC	Response for a class
CHURN	Added lines + deleted lines
BFC	The number of times a file was involved in a bug-fix transaction

Table 3 Used metrics in Apache Ant and jEdit data sets

Name	Explaination
WMC	Weighted methods per class
CBO	Coupling between classes
RFC	Response for a class
LCOM	Lack of cohesion in methods
LCOM3	Lack of cohesion in methods
NPM	The number of public methods
DAM	Data access metric
MOA	Measure of aggregation
CAM	Cohesion among methods of class
CBM	Coupling between methods
MaxCC	Maximum value of cyclomatic complexity of methods in a class

4.2 Extraction of Association Rules

Before extracting association rules, all metrics in data sets are discretized into three categories, high, medium and low, using equal frequency binning. Then, we extracted a set of association rules of the form "$A \Rightarrow buggy$" from each dataset using the association rule mining tool NEEDLE [13]. For extracting rules, the following conditions were used: minimum transactiosn = 5, minimum confidence = 0.6, and maximum rule length = 3. These conditions follow the past study that uses association rule mining in defect prediction [19]. Table 4 shows overview of the rules extracted from four data sets. As shown in Table, more than 800 rules were extracted from each project.

4.3 Rule Prioritization and Its Evaluation

We used three values of t for the number of test cases assuming unit testing; 100, 500, and 1,000. For each t, we calculated the proposed cost-effectiveness metric $Cp(r,t)$ for each rule based on Section 3. Rules were prioritized in the decreasing order of $Cp(r,t)$. For comparison with the proposed metric, the

Table 4 Summary of the extracted association rules.

Project	No. of rules	Average confidence	Average support
Mylyn	1,712	0.795	0.0579
NetBeans	882	0.731	0.0554
Apache Ant	1,534	0.801	0.0575
jEdit	971	0.770	0.0705

prioritization of rules by it confidence and *odds ratio* is also employed. The confidence is one of the most commonly used rule interestingness measure, while Le and Lo [7] showed that the odds ratio outperformed many other rule interestingness measures.

Based on the prioritization, we use LOC-based cumulative-lift charts for evaluating the usefulness of the prioritization. The LOC-based cumulative lift chart is a commonly-used graph to evaluate the cost-effectiveness of defect prediction results [2] [8] [15]. In this chart, the x-axis considered as the requred test effort and the y-axis is the maximum number of discoverable defects by the assigned test effort [9] The x-axis denotes the cumulative lines of code (LOC) of selected modules, and the y-axis is the cumulative number of defects in the selected modules. In this experiment, the rules from rank 1st to n-th are used to select modules. For each selected rule, a set of modules that satisfy the rule's antecedent are selected. Here, each module is selected only once, that is, if a module is already selected by a previously-selected rule, then that module is not selected anymore. Then, to draw a LOC-based cumulative lift chart, we calculate total lines of code of the selected modules as well as the maximum number of bugs that can be found in the selected modules. The prioritization is considered good if the cumulative LOC of the selected modules is small and the maximum number of bugs found is large.

5 Result and Discussion

5.1 Rule prioritization result

Figure 5, 10 and 15 shows the LOC-based cumulative lift chart for $t = 100$, $t = 1,000$ and $t = 10,000$ respectively. Each Figure includes prioritization results using three rule interestingness metrics: proposed metric, confidence and odds ratio. In Figure 5 (a) Mylyn, the curve of the proposed metric comes above that of the confidence and odds ratio in low sum of SLOC area. This indicates that the proposed metric can effectively identify the reduced set of buggy modules that can be tested in low cost. Similarly, in Figure 5 (b) NetBeans and (c) Apache Ant, the curve of the proposed metric is above that of other two metrics in both low and high LOC areas. In Figure 5 (c) jEdit, the curve of the proposed metric is similar to that of other metrics, but in middle and high LOC areas, the curve of the proposed metric is above that of other metrics.

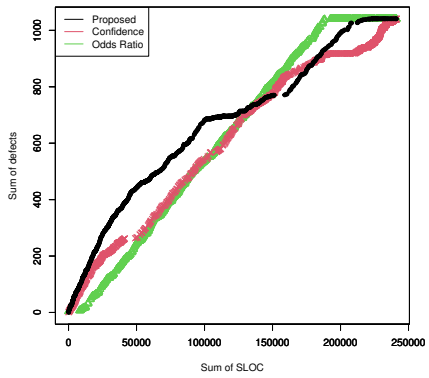

Fig. 1 (a) Mylyn

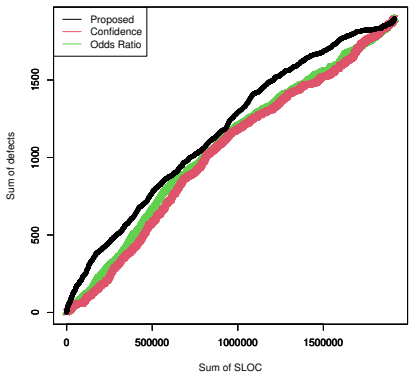

Fig. 2 (b) NetBeans

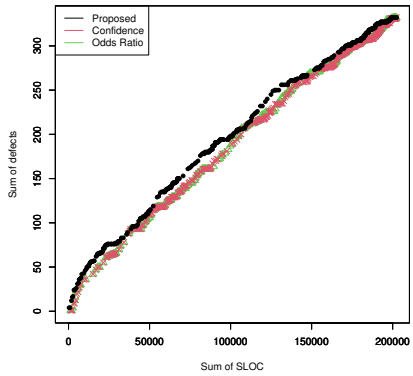

Fig. 3 (c) Apache Ant

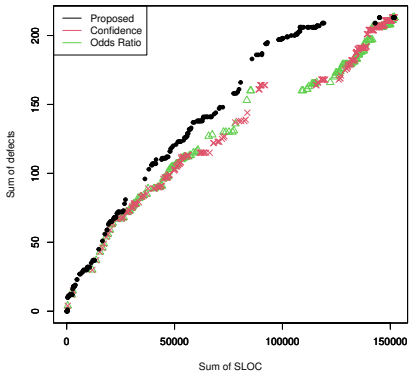

Fig. 4 (d) jEdit

Fig. 5 Cummulative lift chart for t=100

For cases of larger test cases ($t = 1,000$ and $t = 10,000$), as shown in Figure 10, graphs of $t = 1,000$ is very similar to that of $t = 100$. On the other hand, as shown in Figure 10, graphs of $t = 10,000$ is quite different from that of $t = 100$ and $t = 1,000$. Especially, for (a) Mylyn, (c) Apache Ant and (d) jEdit, the proposed metric is not effective to identify buggy modules. These results indicate that setting the appropriate number of test cases t is important in using the proposed metric $Cp(r,t)$, as it is dependent of t. For example, if we wants to test the top 10,000 lines of code that are the most cost-effective ones to conduct unit testing under the unit testing standard of 100 test cases per 1 KSLOC, then we can calculate the proposal metric with $t = 1,000$.

Overall, the result shows the effectiveness of the proposed metric when an appropriate t value is given.

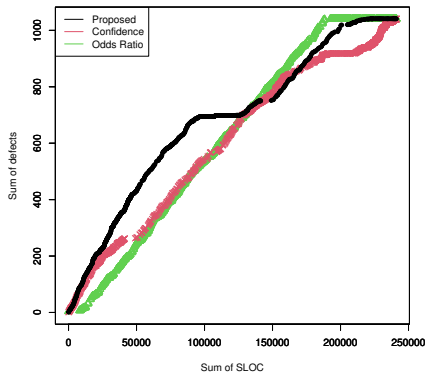

Fig. 6 (a) Mylyn

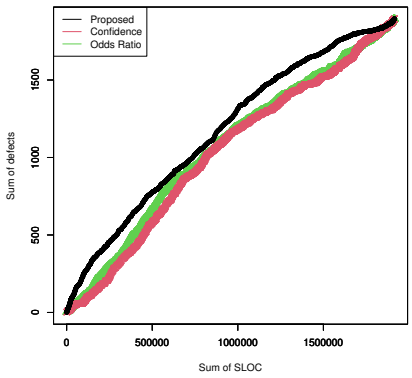

Fig. 7 (b) NetBeans

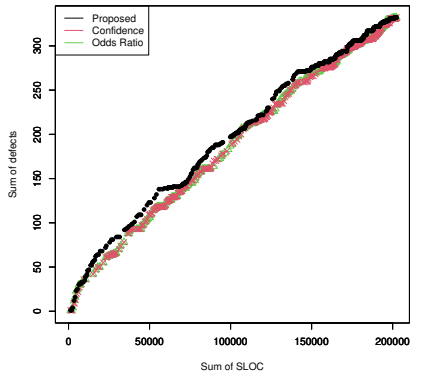

Fig. 8 (c) Apache Ant

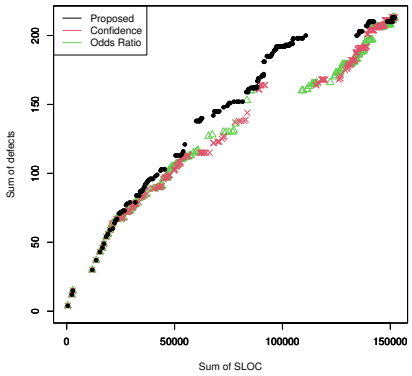

Fig. 9 (d) jEdit

Fig. 10 Cummulative lift chart for $t = 1,000$

5.2 Analysis of prioritized rules

In this subsection, we target Mylyn data set, and analyze in detail the top five prioritized rules for each metric. The top five rules prioritized by the proposed metric ($t = 1000$), confidence and odds ratio are summarized in Table 5, Table 6 and Table 7. In the tables, NOM, CHURN, BFC, etc. represent metrics of modules described in Table 2.

In Table 5 (proposed metric), "NOM=Middle" (which means that the number of methods is between 3 and 7) appeared in all five rules, and "CHURN=High" (which means that added lines + deleted lines ≥ 63) appeared in four rules. It is natural that the rules include the condition CHURN=High because modules with large changes are buggier than that with small changes in general. Among such buggy modules, rules target that are easy to test (that is, not too

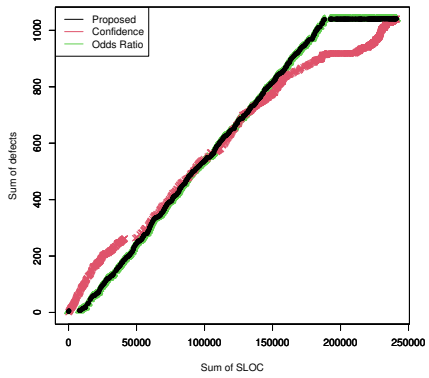

Fig. 11 (a) Mylyn

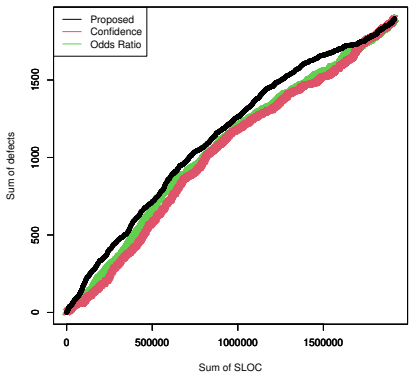

Fig. 12 (b) NetBeans

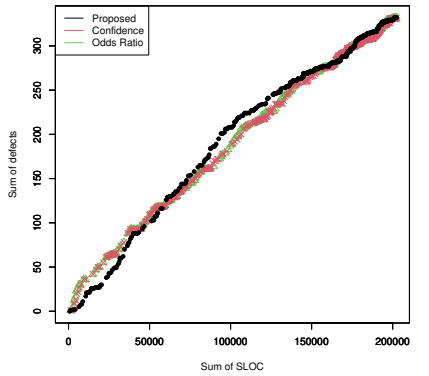

Fig. 13 (c) Apache Ant

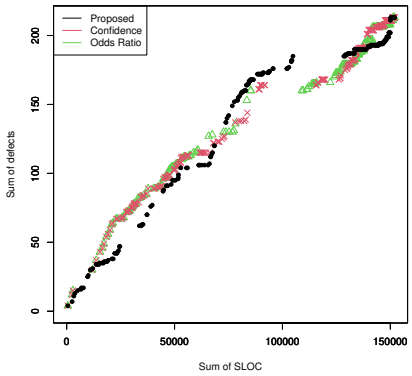

Fig. 14 (d) jEdit

Fig. 15 Cummulative lift chart for $t = 10,000$

large) by the condition NOM=Middle. On the other hand, as shown in Table 6, when using confidence to prioritize rules, the rank 1 rule has the condition "SLOC=large" (which means that lines of code $\geq$ 74). Since large modules require more effort to test, it can be said that using the confidence is not always cost-effective to select modules in software testing. Similarly, in Table 7, when using odds ratio to prioritize rules, the rank 2 rule also has the condition SLOC=large. These results suggest that the proposed metric is able to identify cost-effective modules for detecting bugs in testing, while conventional metrics often identify not cost-effective modules.

Table 5 Top 5 Rules of Mylyn (proposed metric)

Rank	Rule	Confidence	Support	$Cp(r,t)$
1	NOM=Middle & CHURN=High ⇒ *buggy*	0.942	0.118	56.7
2	NOM=Middle & CHURN=High & BFC=High ⇒ *buggy*	0.942	0.117	56.7
3	NOM=Middle & CHURN=High & SLOC=High ⇒ *buggy*	0.981	0.043	56.4
4	NOM=Middle & CBO=High & BFC=High ⇒ *buggy*	0.939	0.100	56.2
5	NOM=Middle & CHURN=High & CBO=High ⇒ *buggy*	0.989	0.070	15.0

Table 6 Top 5 Rules for Mylyn (confidence)

Rank	Rule	Confidence	Support	$Cp(r,t)$
1	SLOC=High & VG=Middle & CHURN=High ⇒ *buggy*	1.000	0.033	48.1
2	PAR=High & NOM=Middle & CBO=High ⇒ *buggy*	1.000	0.033	35.4
3	PAR=High & NOM=Middle & CHURN=High ⇒ *buggy*	1.000	0.033	39.0
4	NBD=Middle & PAR=High & VG=Middle ⇒ *buggy*	1.000	0.024	23.6
5	SLOC=Middle & PAR=Low & CHURN=High ⇒ *buggy*	1.000	0.023	13.0

Table 7 Top 5 Rules for Mylyn (odds ratio)

Rank	Rule	Confidence	Support	$Cp(r,t)$
1	BFC=High ⇒ *buggy*	0.729	0.689	36.8
2	SLOC=High & VG=Middle & CHURN=High ⇒ *buggy*	1.000	0.033	48.1
3	PAR=High & NOM=Middle & CBO=High ⇒ *buggy*	1.000	0.033	35.4
4	PAR=High & NOM=Middle & CHURN=High ⇒ *buggy*	1.000	0.033	39.0
5	NBD=Middle & PAR=High & VG=Middle ⇒ *buggy*	1.000	0.024	23.6

6 Threats to Validity

We discuss threats to validity in the experiments in this paper. First, this paper only includes experiments on four open source projects. In order to improve the generality of the results, it will be a challenge to conduct experiments with industry projects in the future.

In addition, in this paper, the bugs found in the open source projects are assumed to be latent bugs in the experiments, but there are likely to be more undiscovered latent bugs in the projects. Therefore, even modules that are determined to be bug-free may contain undiscovered latent bugs, which is a common issue in bug prediction studies.

In this paper we employ the exponential SRGM to compute the probability of detecting a defect. Since there are many other SRGMs proposed in literature [6] [16], it is our future work to consider employing other SRGMs.

7 Conclusion

This paper proposes a metric to select cost-effective association rules for defect prediction. In the experiment, we used data sets from four open source projects to extract rules and prioritize them based on the proposed metric. Futhermore, we compared the proposed metric with the conventional metrics, confidence and odds ratio. The results showed that the proposed metric is more effective in selecting cost-effective rules for bug detection than the conventional metrics. Based on the experimental result, we believe that the proposed metric is helpful for selecting rules that are likely to detect more bugs when limited test effort is available.

Future issues include conducting experiments using more data and conducting evaluations at actual testing situation.

8 Competing Interests

The authors declare that they have no competing interests.

9 Funding

This work was supported in part by JSPS KAKENHI Grant number JP20K11749 and JP20H05706.

10 Availability of Data and Materials

The datasets generated and/or analyzed during the current study are available from the corresponding author on reasonable request.

11 Authors' Contributions

Kinari Nishiura wrote the main manuscript text, Takeki Kasagi did most of the experiments and collected the data, and Akito Monden conceived the fundamental idea and oversaw the entire study. All authors reviewed the manuscript.

References

1. R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD'93), pp.207-216, 1993.
2. S. Feng, J. Keung, X. Yu, Y. Xiao, K. E. Bennin, M. A. Kabir, M. Zhang, "COSTE: Complexity-based OverSampling TEchnique to alleviate the class imbalance problem in software defect prediction," Information and Software Technology, Vol. 129, No. 106432, 2021.

3. M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," Proc. 6th International Conference on Predictive Models in Software Engineering (PROMISE'10), pp.9:1–9:10, 2010.
4. M. Jureczko and D. Spinellis, "Using object-oriented design metrics to predict software defects," In Models and Methods of System Dependability, Oficyna Wydawnicza Politechniki WrocÅCawskiej, pp.69–81, 2010.
5. Y. Kamei, A. Monden, S. Morisaki, and K. Matsumoto, "A hybrid faulty module prediction using association rule mining and logistic regression analysis," Proc. 2nd ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '08), pp.279-281, 2008.
6. P. K. Kapur, M. Basirzadeh, S. Inoue, and S. Yamada, "Stochastic differential equation based SRGM for errors of different severity with testing-effort," International Journal of Reliability, Quality and Safety Engineering, Vol. 17, No. 3, pp. 179-197, 2010.
7. T. D. B. Le and D. Lo, "Beyond support and confidence: Exploring interestingness measures for rule-based specification mining," Proc. 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), pp. 331-340, 2015.
8. T. Mende and R. Koschke, "Revisiting the evaluation of defect prediction models," Proc. 5th International Conference on Predictor Models in Software Engineering (PROMISE), No. 7, pp. 1-10, 2009.
9. A.Monden, T.Hayashi, S.Shinoda, K.Shirai, J.Yoshida, M.Barker, and K. Matsumoto, "Assessing the cost effectiveness of fault prediction in acceptance testing," IEEE Transactions on Software Engineering, Vol. 39, No. 10, pp. 1345-1357, 2013.
10. A. Monden, M. Tsunoda, M. Barker, and K. Matsumoto, "Examining software engineering beliefs about system testing defects," IT Professional, no. 2, pp. 58-64, 2017.
11. A. Monden, J. Keung, S. Morisaki, Y. Kamei, and K. Matsumoto, "A heuristic rule reduction approach to software fault-proneness prediction," Proc. 19th IEEE Asia-Pacific Software Engineering Conference (APSEC2012), pp.838-847, 2012.
12. S. Morisaki, A. Monden, T. Matsumura, H. Tamada, and K. Matsumoto, "Defect data analysis based on extended association rule mining," Proc. 4th InternationalWorkshop on Mining Software Repositories (MSR'07), pp.17-24, 2007.
13. S. Morisaki, A. Monden, H. Tamada, T. Matsumura, and K. Matsumoto, "Mining quantitative rules in a software project data set," IPSJ Journal, Vol.48, No.8, pp.2725-2734, 2007.
14. Takamasa Nara, "Software quality assurance and evaluation techniques," Japan Symposium on Software Testing Kansai (JaSST'09 Kansai), 2009. (in Japanese)
15. Y. Qu, Q. Zheng, J. Chi, Y. Jin, A. He, D. Cui, H. Zhang, T. Liu, "Using K-core Decomposition on Class Dependency Networks to Improve Bug Prediction Model's Practical Performance," IEEE Transactions on Software Engineering, Vol. 47, No. 2, pp. 348-366, 2021.
16. O. Singh, D. Aggrawal, A. Anand, and P. K. Kapur, "Fault severity based multi-release SRGM with testing resources," International Journal of System Assurance Engineering and Management, Vol. 6, pp. 36–43, 2015.
17. Q. Song, M. Shepperd, M. Cartwright, and C. Mair, "Software defect association mining and defect correction effort prediction," IEEE Transactions on Software Engineering, Vol.32, No.2, pp.69-82, 2006.
18. T. Watanabe and A. Monden and Y. Kamei and S. Morisaki, "Identifying recurring association rules in software defect prediction," Proc. 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS), pp.1-6, 2016.
19. T. Watanabe, A. Monden, Z. Yücel, Y. Kamei, and S. Morisaki, "Cross-validation-based association rule prioritization metric for software defect characterization," IEICE Transactions on Information and Systems, Vol.E101-D, No.9, pp.2269-2278, 2018