PAPER

Cross-Validation-Based Association Rule Prioritization Metric for Software Defect Characterization

Takashi WATANABE[†], *Nonmember*, **Akito MONDEN**[†a)], *Member*, **Zeynep YÜCEL**[†], *Nonmember*, **Yasutaka KAMEI**[††], *and* **Shuji MORISAKI**[†††], *Members*

SUMMARY Association rule mining discovers relationships among variables in a data set, representing them as rules. These are expected to often have predictive abilities, that is, to be able to predict future events, but commonly used rule interestingness measures, such as support and confidence, do not directly assess their predictive power. This paper proposes a cross-validation -based metric that quantifies the predictive power of such rules for characterizing software defects. The results of evaluation this metric experimentally using four open-source data sets (Mylyn, NetBeans, Apache Ant and jEdit) show that it can improve rule prioritization performance over conventional metrics (support, confidence and odds ratio) by 72.8% for Mylyn, 15.0% for NetBeans, 10.5% for Apache Ant and 0 for jEdit in terms of **SumNormPre**(100) precision criterion. This suggests that the proposed metric can provide better rule prioritization performance than conventional metrics and can at least provide similar performance even in the worst case.

key words: *association rule mining, defect prediction, cross-validation, data mining, software quality*

1. Introduction

Association rule mining discovers relationships among variables in a given data set and represents them as association rules. In the software engineering field, it has been applied to discover rules in software defect data sets [1]–[4], such as "(max nest level > 5) and (fan-out > 5) ⇒ faulty". This rule implies that a software module is likely to contain a defect (i.e. a fault) if its maximum nesting level and fan-out complexity are both greater than 5. Such association rules, derived from past software projects, are useful in helping practitioners to not only *understand* the cause of defects but also *identify* defect-prone modules in an ongoing or a future software development project.

Applying association rule mining to real-world data is a challenging task, since an unmanageably large number of rules can often be extracted. To tackle this challenge, this paper focuses on *prioritizing* association rules based on their predictive power. If a rule has enough predictive power (i.e. it can correctly predict many future events), it can be used to plan future actions. In contrast, rules that have little predictive power are fairly useless, and using them to plan actions may even be harmful. In our case, we want to select the most predictive rules to better understand the characteristics of defect-prone modules in previous software product versions so that modules in the next version with similar characteristics can be thoroughly tested or inspected before release [1].

This paper proposes a cross-validation-based metric to quantify these rules' predictive power and hence prioritize them. In the proposed method, a certain number of association rules are derived from a given (training) data set, and their prediction accuracy is evaluated via cross-validation within the data set. Then, we create an empirical metric to estimate the prediction accuracy based on the cross-validation results using log-log regression modeling. In regression model, the objective variable is the cross-validation prediction accuracy. And we choose the rules' confidence and occurrence as predictor variables, since these are regarded as important factors in their predictive power [4].

Various rule interestingness metrics have already been investigated for prioritizing the association rules. Of these, support and confidence are two of the most common. Support is an indicator of rule frequency, i.e. how frequently a given rule's conditions are satisfied in the data set, whereas confidence is an indicator of association strength, i.e. how often the rule is satisfied for a given condition. However, despite these being common metrics, as shown by Le and Lo [5], showed that support and confidence are not very effective to prioritize the rules for rule prioritization. They compared 38 rule interestingness metrics, finding that the odds ratio, which handles rule weakness by considering both correlations and inverse correlations, outperformed all other measures [5]. However, the odds ratio is also not a direct measure of predictive power; and it does not appear to be useful in all situations (as shown in Sect. 3). We therefore aim to devise a direct measure of predictive power based on cross-validation.

To evaluate the effectiveness of the proposed approach, we then conduct a case study using four open-source defect data sets (Mylyn, NetBeans, Apache Ant, and jEdit). This compares the performance of our proposed rule prioritization metric with those of several conventional metrics (odds ratio, confidence, and support) in terms of cross-release defect prediction.

The structure of this paper is as follows. First, Sect. 2

Manuscript received January 15, 2018.
Manuscript revised May 2, 2018.
Manuscript publicized June 13, 2018.

[†]The authors are with Okayama University, Okayama-shi, 700–8530 Japan.
[††]The author is with Kyushu University, Fukuoka-shi, 819–0395 Japan.
[†††]The author is with Nagoya University, Furo-cho, Chikusa-ku, Nagoya, 464–8603 Japan.
a) E-mail: monden@okayama-u.ac.jp

DOI: 10.1587/transinf.2018EDP7020

introduces association rule mining and how it can be used for defect characterization. Then, Sect. 3, introduces several key rule interestingness metrics, including the odds ratio, and gives a motivating example where these metrics are not useful. Next, Sect. 4 proposes our cross-validation approach calculating a rule prioritization metric. Section 5 describes an experiment for evaluating this new approach, and Sect. 6 presents and discusses the experimental results. Finally, Sect. 7 discusses the limitations of the proposed approach, before Sect. 8 summarizes our study and proposes future research directions.

2. Characterizing Defects via Association Rule Mining

2.1 Definitions

Association rule mining was defined by Agrawal et al. as follows [6].

Let $I = I_1, I_2, \ldots, I_m$ be a set of m unique items where each item I_k, $(1 \leq k \leq m)$ takes a binary value. An association rule is denoted by an expression $(A \Rightarrow B)$, where $A \subset I$, $B \in I$, and $B \notin A$. We refer to A and B as the **antecedent** and **consequent** of the rule, respectively. Next, let D be a *database* consisting of n transactions $T_i \subset I$, denoted as $D = T_1, T_2, \ldots, T_n$. We say that T_i satisfies the rule $(A \Rightarrow B)$, if $A \subset T_i \wedge B \in T_i$.

In this paper, the binary items I_i are derived from discrete metrics that evaluate software modules with respect to certain criteria, such as the cyclomatic complexity. Since the cyclomatic complexity is an integer and can potentially take any value in the interval $[0, \infty)$, it does not make sense to use it directly in the evaluation. Such quantitative variables are therefore pre-processed and converted into discrete variables. For example, we could define three cyclomatic complexity categories, namely, low $[0, 10)$, medium $[10, 30)$ and high $[30, \infty)$, and discretize it by defining the following three items.

$I_1 = \{\text{cyclomatic complexity} == \text{low}\}$

$I_2 = \{\text{cyclomatic complexity} == \text{medium}\}$

$I_3 = \{\text{cyclomatic complexity} == \text{high}\}$

For a software module with a cyclomatic complexity of 20, we would obtain $I_1 = \text{false}$, $I_2 = \text{true}$ and $I_3 = \text{false}$.

Transaction T_i describes the state of module i subject to a set of criteria and their respective possible categories. For example, we could define the following three items to evaluate a module with respect to its fan-in complexity based on the same three categories given above.

$I_4 = \{\text{fan-in} == \text{low}\}$

$I_5 = \{\text{fan-in} == \text{medium}\}$

$I_6 = \{\text{fan-in} == \text{high}\}$

Next, let us add another item to describe the fault state of software module i, namely $I_7 = \{\text{faulty} == \text{true}\}$. We can then define a transaction by evaluating a given module with respect to such a set of criteria (cyclomatic and fan-in, complexities, and fault state). For example, a faulty software module i with cyclomatic and fan-in complexities of 20 and 40, respectively, would be described by the following transaction.

$$T_i = \{I_1 = \text{false}, I_2 = \text{true}, I_3 = \text{false}, I_4 = \text{false}, I_5 = \text{false}, I_6 = \text{true}, I_7 = \text{true}\}$$

Given that, we can create a rule $(A \Rightarrow B)$ by taking our antecedent A as $I_2 \wedge I_6$ and our consequent B as I_7 (omitting "true" and "false" for brevity), which means that our rule asserts that $(I_2 \wedge I_6 \Rightarrow I_7)$. According to this rule, the above module (with cyclomatic and fan-in complexities of 20 and 40) would be considered to be fault-prone. Explicitly, we would say that T_i satisfies $(A \Rightarrow B)$.

On this basis a database D defines the state of a set of software modules with respect to a set of K items $\{I_k\}, 1 \leq k \leq K$. We can then use this database and item set to mine for new rules and evaluate their performance, in terms of metrics such as confidence and odds ratio.

2.2 Characterizing Defects Using Association Rules

In this paper, we characterize software defects using association rule mining. After a set of rules has been derived from a set of previous software project module data, we can use it to understand the causes of defects and help in planning ways to avoid adding new defects or detect hidden defects in ongoing or future projects more efficiently. For example, if modules with certain characteristics are found to be defect-prone, similar modules should be thoroughly tested or inspected during software development.

Several various defect prediction models have already been proposed [7]–[11]. However, our focus in this paper is on understanding rather than predicting defects because previous predictive models have been difficult for humans to understand, making it hard for software engineers to recognize and agree why certain modules are (or are not) faulty [1]. Even with simple linear discriminant models, correlations between predictor variables make it difficult to interpret their coefficients clearly. In contrast, association rules are much easier to understand because they are described in a simple and intuitive way, such as (condition $\Rightarrow$ faulty) or (condition $\Rightarrow$ not faulty). Note, however, that even though we are focusing on the understanding rather than prediction, we still test the rules' predictive power, because they need to be able to make predictions if we are to use them for future planning or software process improvements.

To make a prediction, given a rule set and a target module, we need to be aware that more than one rule may match the module, i.e. it may satisfy the antecedents of multiple rules. To handle this, we take the approach described in [1], which keeps the rule set small and understandable by only mining rules whose consequent is "faulty" and ignoring the rest. If at least one of these rules matches a module, then we consider it to be faulty; otherwise, we consider it not to be

faulty; otherwise, we consider it not to be faulty.

3. Rule Metrics and Motivating Example

Several metrics are commonly used in the literature, such as the occurrence, support, and confidence. The *occurrence* **Occ** of a rule is the number of transactions that satisfy the rule,

$$\mathbf{Occ}(A \Rightarrow B) = |T_i| : A \subset T_i \wedge B \in T_i,$$

where the operator |.| denotes the cardinality of a set. We can apply the occurrence metric not only to rules but also to sets of items, such as antecedents or consequents. For instance, the occurrence of the antecedent A is the number of transactions that involve A,

$$\mathbf{Occ}(A) = |T_i| : A \subset T_i.$$

The *support* **Supp** is an indicator of rule frequency, defined as the fraction of transactions in the database that satisfy the rule:

$$\mathbf{Supp}(A \Rightarrow B) = \frac{\mathbf{Occ}(A \Rightarrow B)}{|D|}.$$

Thus, the support is a measure of the rule's statistical significance. In contrast, the *confidence* **Conf** is the probability that the consequent B was preceded by the antecedent A,

$$\mathbf{Conf}(A \Rightarrow B) = \frac{\mathbf{Supp}(A \Rightarrow B)}{\mathbf{Supp}(A)}.$$

The confidence thus helps in determining the rule's strength. In addition to these metrics, we also define the *length*, denoted by ||.||, which describes a rule's complexity in terms of the number of metrics (i.e. items) in its antecedent A:

$$\mathbf{Length}(A \Rightarrow B) = \|A\|.$$

For example, the length of the rule "(cyclomatic complexity = medium)∧(fan-in = high) ⇒ faulty" is 2, since two metrics (cyclomatic complexity and fan-in) are present in the antecedent. Longer lengths indicate more complex rules.

Le and Lo [5] showed, however, that these metrics are not very effective for selecting rules with high predictive power when used alone. In general, they must be used together to select useful rules, in which case lower bounds are typically set for both support and confidence when selecting rules.

That said, our previous study [4] showed that setting constant lower bounds for these metrics is insufficient. For example, we can accept rules with low support if their confidence is very high, but not otherwise. This implies that we need new metrics that intelligently combine these metrics, an idea that we investigate in this paper.

Other metrics have also been considered. In particular, Le and Lo [5] compared a total of 38 rule interestingness metrics empirically and found that the *odds ratio*

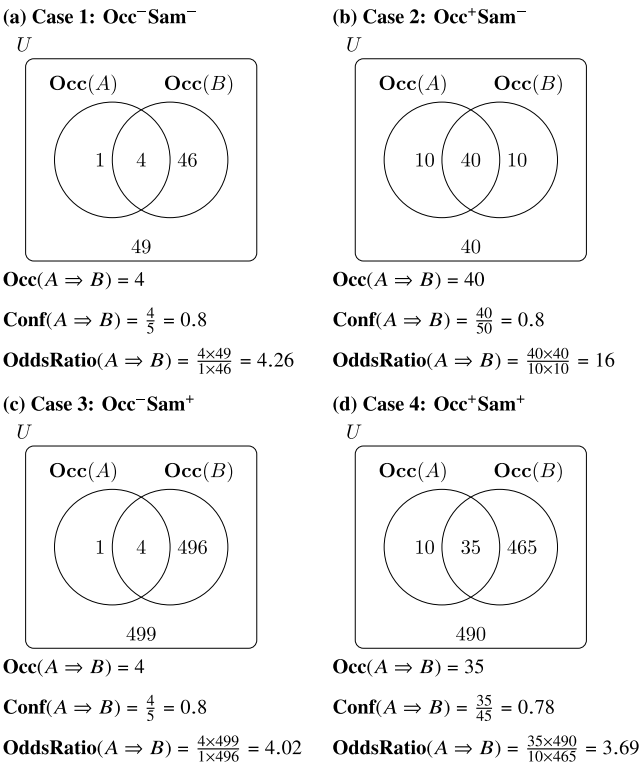

Fig. 1 Motivating examples for the rule ($A \Rightarrow B$).

OddsRatio was the best in quantifying the rules' predictive power. The odds ratio measures how strongly the presence or absence of A is associated with B as follows:

$$\mathbf{OddsRatio}(A \Rightarrow B) = \frac{\mathbf{Occ}(A \wedge B)\,\mathbf{Occ}(\neg A \wedge \neg B)}{\mathbf{Occ}(A \wedge \neg B)\,\mathbf{Occ}(\neg A \wedge B)}. \quad (1)$$

Although the odds ratio is useful in many cases, we also believe that it is not always helpful. To demonstrate this, we now give a motivating example showing how the odds ratio depends on the occurrence and number of samples, comprising the following four cases:

Case 1: Low occurrence, small number of samples ($\mathrm{Occ}^-\mathrm{Sam}^-$)

Case 2: High occurrence, small number of samples ($\mathrm{Occ}^+\mathrm{Sam}^-$)

Case 3: Low occurrence, large number of samples ($\mathrm{Occ}^-\mathrm{Sam}^+$)

Case 4: High occurrence, large number of samples ($\mathrm{Occ}^+\mathrm{Sam}^+$)

Figure 1 shows these four cases for the rule ($A \Rightarrow B$). In Case 1 ($\mathrm{Occ}^-\mathrm{Sam}^-$), five transactions match A, four of which also match B, so the confidence is 4/5 = 0.8, and the occurrence is 4. In this case, the rule's confidence is high, but it only matches four of the transactions in the data set. Since this can easily happen accidentally, we cannot

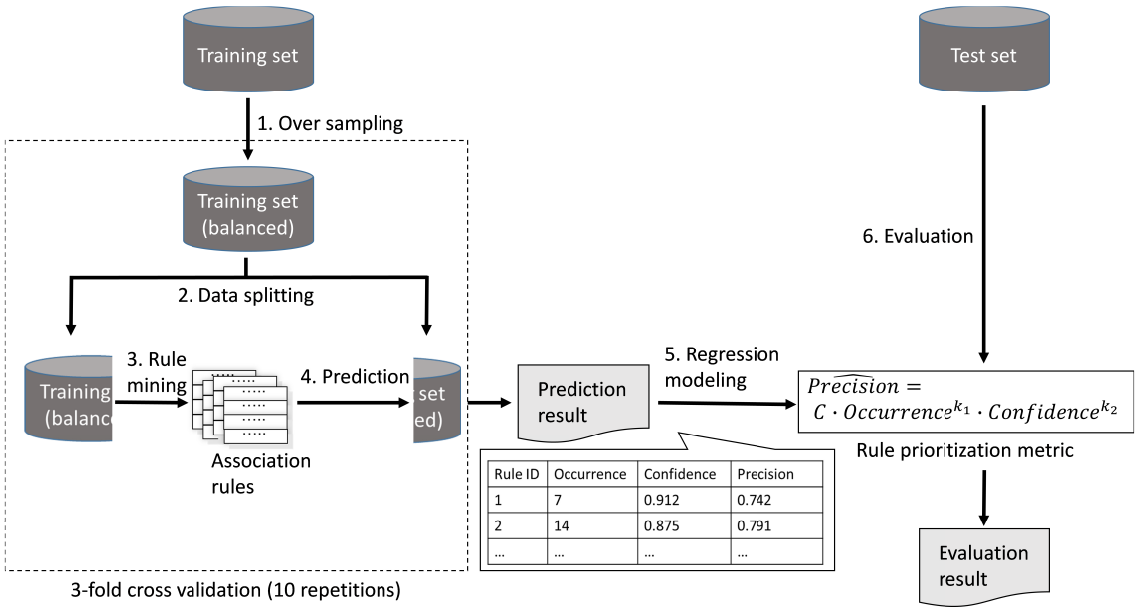

Fig. 2 A procedure of the proposed method.

consider it to have high predictive power. Next, in Case 2 ($\mathrm{Occ}^{+}\mathrm{Sam}^{-}$), the confidence is the same as in Case 1 ($\mathrm{Occ}^{-}\mathrm{Sam}^{-}$), but the occurrence is much higher (40 transactions match the rule, which is unlikely to happen accidentally), so we consider this rule's predictive power to be much higher. The odds ratios are 4.26 in Case 1 ($\mathrm{Occ}^{-}\mathrm{Sam}^{-}$) and 16 in Case 2 ($\mathrm{Occ}^{+}\mathrm{Sam}^{-}$), so they are useful for distinguishing between these two cases (higher odds ratios indicate higher predictive power).

In contrast, Case 3 ($\mathrm{Occ}^{-}\mathrm{Sam}^{+}$) and Case 4 ($\mathrm{Occ}^{+}\mathrm{Sam}^{+}$) show situations where the odds ratio is not useful. Here, the confidences are almost the same (0.8 and 0.78), but the occurrence is very low in Case 3 ($\mathrm{Occ}^{-}\mathrm{Sam}^{+}$) and very high in Case 4 ($\mathrm{Occ}^{+}\mathrm{Sam}^{+}$). Despite its high confidence, the rule in Case 3 ($\mathrm{Occ}^{-}\mathrm{Sam}^{+}$) is useless because of its very low occurrence, unlike the rule in Case 4 ($\mathrm{Occ}^{+}\mathrm{Sam}^{+}$), which has much higher predictive power. Despite that, the odds ratio is lower in Case 4 ($\mathrm{Occ}^{+}\mathrm{Sam}^{+}$) than in Case 3 ($\mathrm{Occ}^{-}\mathrm{Sam}^{+}$). These examples therefore show that the odds ratio cannot properly account for the effect of very low occurrences.

4. Proposed Method

This section describes the proposed method for calculating a rule prioritization metric, that can quantify the predictive power of association rules, illustrated in Fig. 2. This proceeds according to the following six steps.

Step 1 involves oversampling the data set. This step is essential, particularly for imbalanced defect data where the majority of the instances are not faulty [12], [13]. This is because using such imbalanced data sets to train prediction models leads to high error rates when predicting the faulty modules [14], because of the rules relating to such modules being discarded on the grounds of low occurrence. Oversampling, a technique commonly used to mitigate this type of problem, artificially increases the number of minority instances in the training set, yielding a more balanced distribution of faulty and non-faulty instances. This increases the occurrences of certain rules, allowing us to derive more suitable association rules. Here, we employ random oversampling, which is the simplest technique and is effective for a variety of data sets and prediction models [8].

Steps 2–4 carry out cross-validation. In Step 2, we randomly split the training data set into two subsets; one is used to extract the association rules in Step 3, and the other is used to evaluate the rules' prediction accuracy (precision) in Step 4. Here, we employ employ three-fold cross-validation, so Steps 3 and 4 are repeated three times to obtain the final prediction results. In addition, to obtain stable results, we repeat the data division process (Step 2) 10 times, each followed by three repetitions of Steps 3–4, and use all of the results as input to Step 5.

Next, in Step 5, we calculate an empirical prediction accuracy metric via log-log regression modeling based on cross-validation results. The regression process's objective variable, i.e. the *cross-validation precision* **Precision**, is defined in the standard way as the ratio of the number of correct faulty module predictions (i.e. true positives or TP) to the total number of faulty module predictions (including the false positives or FP). For the rule ($A \Rightarrow$ faulty), this is

$$\textbf{Precision}(A \Rightarrow B) = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}.$$

Meanwhile, the predictor variables are the rule's confidence and the occurrence. We choose to employ these two metrics because, in our previous study [4], we found that they were key indicators of rule prediction power. A low occurrence is acceptable, if the confidence is very high, but not

Table 1 Summary of the defect data sets used in our experiments.

Project	No. of metrics		Version	Total No. of modules	No. of faulty modules	% of faulty modules
	Product	Process				
Mylyn	8	2	2.0	1230	848	68.94
			3.0	1502	606	40.34
NetBeans	8	2	4.0	4660	367	7.87
			5.0	9332	319	3.41
Apache Ant	11	0	1.6	351	92	26.21
			1.7	745	166	22.28
jEdit	11	0	4.0	306	75	24.50
			4.1	312	79	25.32

if the confidence is lower. In addition, we chose to employ log-log regression rather than the simpler linear regression because the variable distributions are usually skewed, i.e. do not follow Gaussian distributions, and log transformation is a common way to improve the fitting of linear regression models in such cases [15].

On the basis of our two predictor variables confidence and occurrence, the log-log regression model to estimate the prediction precision is as follows[†]:

$$\log\left(\widehat{precision} + 0.5\right) = k_1 \log(\mathbf{Occ}) + k_2 \log(\mathbf{Conf}) + C$$

where k_1 and k_2 are partial regression coefficients and, C is a constant. We have added 0.5 to the precision on the left-hand side of the equation because the actual precision values are sometimes zero, and adding 0.5 (before model construction) is the recommended way to avoid computing log(0) and hence to stabilize the data set's variance [16]. We can then transform this equation by exponentiating both sides, yielding the following:

$$\widehat{precision} = C' \cdot \mathbf{Occ}^{k_1} \cdot \mathbf{Conf}^{k_2} - 0.5, \quad (2)$$

where $C' = \exp C$. Using this, we can then obtain values for the regression parameters using least squares approximation.

Finally, in Step 6, we evaluate the effectiveness of Eq. (2) using the test data set. Here, we carry out cross-release prediction, i.e. we mine the rules based on data from a previous version of the software, and then apply the extracted rules to the subsequent version. In the experimental evaluation below, we compare the performance of our proposed rule prioritization metric with those of conventional metrics (namely, the odds ratio, confidence, and support).

5. Experimental Setup

5.1 Data Sets

In these experiments, we employed defect data sets from the following four open-source software projects: (1) Mylyn (versions 2.0 and 3.0), an Eclipse task management plug-in; (2) NetBeans (versions 4.0 and 5.0), a Java software development platform; (3) Apache Ant (versions 1.6 and 1.7), a software build tool; and (4) jEdit (versions 4.0 and 4.1), a text editor. Table 1 gives an overview of these data sets. The Mylyn and NetBeans data sets are described in more detail in [4]. The Apache Ant and jEdit data sets were obtained from the tera-PROMISE data repository [17], having been donated by Jureczko et al. [18], [19]. They are described in more detail in [19].

As shown in Table 1, the NetBeans, Apache Ant and jEdit data sets were imbalanced, i.e. only small percentages of the modules were faulty. We therefore applied oversampling (Step 1 in Fig. 2) before carrying out cross-validation (Steps 2–4). The Mylyn data set, on the other hand, was relatively balanced, so we skipped Step 1 in this case.

5.2 Association Rule Extraction

Before mining the rules, we discretized all of the process and product metrics in Table 1 into three categories, namely H (high), M (medium), and L (low), using equal-frequency binning so that each category contained approximately the same number of modules. We then used NEEDLE [2], [20] to mine the association rules, using the following thresholds: minimum transactions = 5 and minimum **Conf** = 0.6. We also set a maximum rule length threshold of 3 because this gave us sufficient numbers of rules and longer rules are less easily understood by humans.

Table 2 summarizes the rules extracted from the four data sets and shows the impact of oversampling. The NetBeans data set stands out as particularly a clear example, illustrating the necessity of oversampling, as this data set has a very low percentage of faulty modules (Table 1). Without oversampling, it would not have been possible to extract any rules with minimum occurrences of 5, the criterion used in this paper. We therefore generated additional 3882 modules using oversampling, which increased the number of rules extracted to 26247. The NetBeans data set's average occurrence was particularly high (317.0, about 10 times higher than any of the other projects) because of having a much higher number of modules than the other projects.

More than 10,000 rules were extracted for each project, which we consider to be sufficient to build stable regression-based metrics for each project.

5.3 Evaluation Criterion

Before defining the criterion used to evaluate the rule prioritization metrics, we first define the *normalized precision* **NPrec**($A \Rightarrow$ faulty) for a given association rule ($A \Rightarrow$

[†]Here we drop the rule terms for brevity, e.g. we use **Conf** instead of **Conf**($A \Rightarrow B$).

Faulty) as follows:

$$\textbf{NPrec}\,(A \Rightarrow \text{faulty}) = \frac{\textbf{Precision}\,(A \Rightarrow \text{faulty})}{\text{ratio of faulty modules to all modules}} - 1. \quad (3)$$

The first term on the right-hand side of this equation, normalizes the precision by its baseline value, namely, the fraction of modules that are faulty. The additional negative one term simply adjusts this metric's baseline to be zero.

Given an association rule $(A \Rightarrow \text{faulty})$ and a test data set to be predicted by the rule, if $\textbf{NPrec}\,(A \Rightarrow \text{faulty}) > 0$, then the rule has at least some predictive power, with higher values indicating higher predictive power.

On the basis of this definition, we evaluate the performance of the rule prioritization metrics by defining a metric called *sum of normalized precision* (**SumNormPre**) as follows:

$$\textbf{SumNormPre}(n) = \sum_{i=1}^{n} \textbf{NPrec}(r_i), \quad (4)$$

where r_i is ith rank rule, when they are sorted in descending order according to the given rule prioritization metric.

Intuitively, **SumNormPre** is the sum of prediction performance of the top n ranked rules identified by the prioritization metric, so higher values indicate better prioritization metrics.

6. Results and Discussion

6.1 Results

The execution cost of our method is low, where Steps 1 to 6 take about 15 minutes for each dataset.

Figure 3 shows the total prediction performance of the top 100 rules for the Mylyn, NetBeans, Apache Ant and jEdit, respectively, using four different rule metrics, namely, the support, confidence, odds ratio and proposed metrics. The x-axis indicates the rule rank, assuming that the rules are arranged in decreasing order according to the given rule metric (e.g. for the support metric, the first rule is the one with the highest support value). The y-axis indicates the corresponding **SumNormPre** values.

We should note that, although moving toward higher rule ranks along the x-axis should lead to higher

Table 2 Summary of the extracted association rules and oversampling impact (i.e. number of modules added).

Project	No. of rules	Average confidence	Average occurrence	Average support	No. of modules added
Mylyn	49081	0.799	49.8	0.0404	0
NetBeans	26274	0.734	317.0	0.037	3882
Apache Ant	12421	0.797	27.1	0.054	151
jEdit	26241	0.778	22.7	0.0510	139

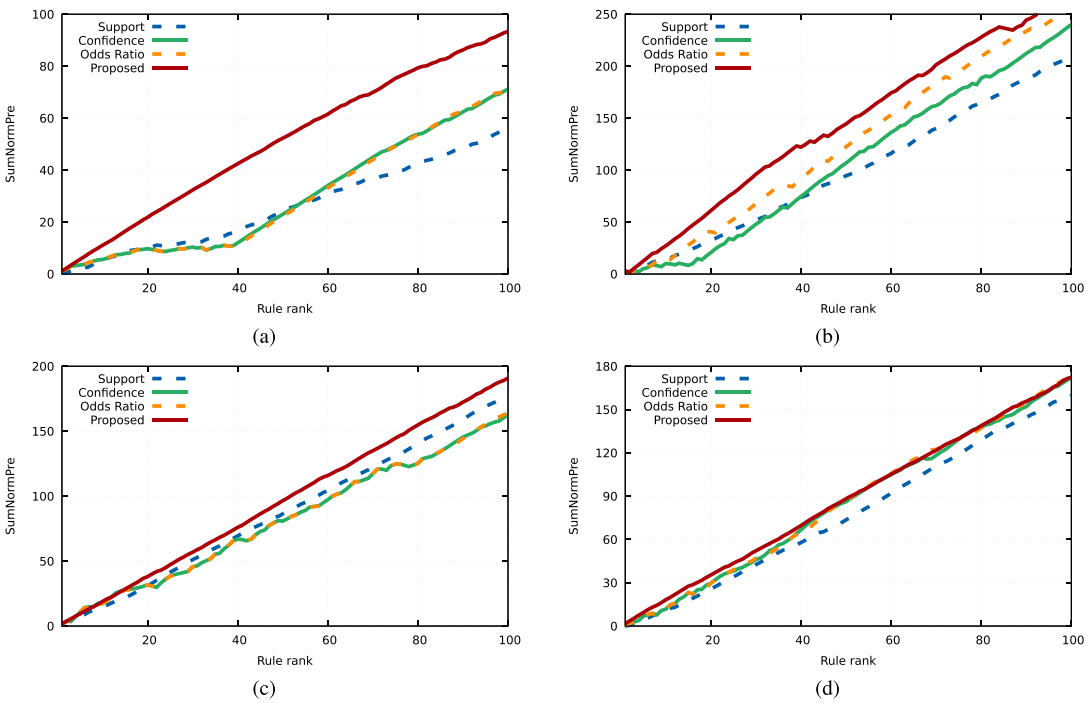

Fig. 3 Results for the (a) Mylyn, (b) NetBeans, (c) Apache Ant, and (d) jEdit data sets.

Table 3 AUCs for **SumNormPre**(100) and **SumNormPre**(all rules).

AUC type	Data set	Support	Confidence	Odds Ratio	Proposed
SumNormPre(100)	Mylyn	2569	2943	2925	**5086**
	NetBeans	9918	10847	12547	**14432**
	Apache Ant	8769	8168	8177	**9691**
	jEdit	7739	8518	**8575**	8573
SumNormPre(all rules)	Mylyn	419279	518219	521436	**534056**
	NetBeans	557608	586857	594963	**599601**
	Apache Ant	1605529	1681531	1698634	**1696396**
	jEdit	526226	580137	583261	**583368**

Table 4 Numbers of the rules meeting $Occ^{-}Sam^{+}$ criteria, out of the top 100 of each metric.

Data set	All rules	**OddsRatio**(100)	**SumNormPre**(100)
Mylyn	22	18	0
NetBeans	0	0	0
Apache Ant	69	62	0
jEdit	17	13	0

Table 5 Numbers of the rules meeting $Occ^{+}Sam^{+}$ criteria, out of the top 100 of each metric.

Data set	All rules	**OddsRatio**(100)	**SumNormPre**(100)
Mylyn	80	43	69
NetBeans	1	1	1
Apache Ant	1	0	1
jEdit	0	0	0

SumNormPre values, there is a small decline at around a rank of 90 in Fig. 3-b. This is due to the data sets used for rule extraction and prediction being taken from different software versions, with an older version being used for rule extraction and a newer one being used to evaluate the prediction performance (see Table 1). Thus, the predictions made by the extracted rules are not always correct and can actually cause the **SumNormPre** value to decline, as shown in Fig. 3-b

These results show that, for the Mylyn, NetBeans and Apache Ant data sets, the proposed metric demonstrated better performance than the conventional metrics (support, confidence, and odds ratio), although its performance was similar to that of the best conventional metric for the jEdit data set. Overall, however, the proposed metric performed well on all four data sets.

Table 3 shows area under the curve (AUC) of **SumNormPre**(n) for $n = 100$ rules and all rules. This indicates that the proposed metric produced improvements in the **SumNormPre**(100) value of 72.8% for Mylyn (from 2943 to 5086), 15.0% for NetBeans (from 12547 to 14432) and 10.5% for Apache Ant (from 8769 to 9691). For jEdit, its **SumNormPre**(100) value was very similar to that of the odds ratio metric. The AUCs for **SumNormPre**(all rules) show smaller improvements by the proposed metric, but we believe that this case is less important because practitioners are not usually interested in the lower-ranked rules.

6.2 Discussion

Here, using the results for the four data sets presented in Sect. 5.1, we focus on the observation frequency for the two cases ($Occ^{-}Sam^{+}$ and $Occ^{+}Sam^{+}$), where the odds ratio is expected to fail but the proposed approach is expected to perform well.

First, we consider the rules, for the $Occ^{-}Sam^{+}$ case, looking for occurrences of less than 10 and confidences of more than 0.9. Table 4 shows the number of rules for each data set that satisfied these conditions, taken from the top 100 rules sorted by odds ratio (i.e. **OddsRatio**(100)) and sum of normalized precision (i.e. **SumNormPre**(100)).

Table 5 shows the results of a similar analysis for the $Occ^{+}Sam^{+}$ case, this time looking for occurrences of more than 100 and confidences of more than 0.9. As is clear from Tables 4 and 5, that these cases were not unusual in these real-world data sets. In particular, $Occ^{-}Sam^{+}$ case arose in all of the data sets except the one for NetBeans. On the other hand, although the $Occ^{+}Sam^{+}$ case cropped up quite frequently in the Mylyn data set, it was very rare in the others.

In addition, the $Occ^{-}Sam^{+}$ rules were often mistakenly selected by the odds ratio metric as being useful, whereas they were successfully discarded by **SumNormPre**. Similarly, the odds ratio metric often failed to identify the predictive power of $Occ^{-}Sam^{+}$ rules, whereas the proposed metric was more successful in giving them higher priorities.

Next, we investigate why the performance of the proposed metric was better than that of the odds ratio metric, despite it being empirically shown to be the best of the 38 rule metrics considered in a past study [5]. Tables 6 and 7 show the top 10 rules for the Mylyn project, ranked by the proposed and odds ratio metrics, respectively. These show each rule's rank, support, confidence, and prediction precision. For further details of the metric abbreviations used (such as CBO, CHURN, and BFC), please see our previous study [4]. Table 7 shows that all of the rules with **Conf** = 1.000 were ranked first because a confidence of 1 means that the denominator term in Eq. (1) is 0, leading to the Odds Ratio becoming infinite. This means it cannot effectively prioritize the rules further and ranks several rules as joint first.

In addition, note that all of these rules' support values are low, so their predictive power may not necessarily be very high, as shown by the precision column (which ranges from 0.467 to 0.882). In contrast, Table 6 shows that the proposed metric did not rank all of the **Conf** = 1.000 rules in the top 10. Instead, rules with both relatively high confidence and relatively high support appear in the top 10, and

Table 6 Top 10 rules for the Mylyn data set, ranked by the proposed metric.

Rank	Support	Confidence	Precision	Rule
1	0.223	0.979	0.856	$(CBO = H) \wedge (CHURN = H) \Rightarrow$ faulty
2	0.223	0.979	0.856	$(CBO = H) \wedge (CHURN = H) \wedge (BFC = H) \Rightarrow$ faulty
3	0.170	0.991	0.870	$(MLOC = H) \wedge (CBO = H) \wedge (CHURN = H) \Rightarrow$ faulty
4	0.148	0.995	0.854	$(NOF = H) \wedge (CBO = H) \wedge (CHURN = H) \Rightarrow$ faulty
5	0.186	0.983	0.843	$(MLOC = H) \wedge (CHURN = H) \Rightarrow$ faulty
6	0.186	0.983	0.843	$(MLOC = H) \wedge (CHURN = H) \wedge (BFC = H) \Rightarrow$ faulty
7	0.146	0.989	0.876	$(PAR = H) \wedge (CBO = H) \wedge (CHURN = H) \Rightarrow$ faulty
8	0.145	0.989	0.843	$(MLOC = H) \wedge (PAR = H) \wedge (CHURN = H) \Rightarrow$ faulty
9	0.140	0.989	0.816	$(MLOC = H) \wedge (NOF = H) \wedge (CHURN = H) \Rightarrow$ faulty
10	0.121	0.993	0.840	$(NBD = H) \wedge (NOF = H) \wedge (CHURN = H) \Rightarrow$ faulty

Table 7 Top 10 rules for Mylyn data set, ranked by odds ratio.

Rank	Support	Confidence	Precision	Rule
1	0.033	1.000	0.882	$(MLOC = H) \wedge (VG = M) \wedge (CHURN = H) \Rightarrow$ faulty
1	0.033	1.000	0.667	$(PAR = H) \wedge (NOM = M) \wedge (CBO = H) \Rightarrow$ faulty
1	0.033	1.000	0.871	$(PAR = H) \wedge (NOM = M) \wedge (CHURN = H) \Rightarrow$ faulty
1	0.024	1.000	0.500	$(NBD = M) \wedge (PAR = H) \wedge (VG = M) \Rightarrow$ faulty
1	0.023	1.000	0.500	$(MLOC = M) \wedge (PAR = L) \wedge (CHURN = H) \Rightarrow$ faulty
1	0.021	1.000	0.500	$(MLOC = M) \wedge (VG = H) \wedge (NOM = M) \Rightarrow$ faulty
1	0.021	1.000	0.688	$(NBD = L) \wedge (CBO = H) \wedge (CHURN = H) \Rightarrow$ faulty
1	0.019	1.000	0.625	$(PAR = H) \wedge (VG = M) \wedge (CBO = H) \Rightarrow$ faulty
1	0.016	1.000	0.583	$(PAR = L) \wedge (CBO = H) \wedge (CHURN = H) \Rightarrow$ faulty
1	0.015	1.000	0.467	$(PAR = H) \wedge (VG = M) \wedge (RFC = H) \Rightarrow$ faulty

Table 8 Regression coefficients of the rule prioritization equations obtained by the proposed approach.

Data set	Regression coefficients		
	C'	k_1	k_2
Mylyn	exp(0.270)	0.445	0.002
NetBeans	exp(0.308)	0.531	0.001
Apache Ant	exp(−0.019)	0.190	0.007
jEdit	exp(0.037)	0.310	0.008

these all show relatively good prediction performance (precisions of 0.816 − 0.876). This suggests that the proposed metric is appropriately considering both the support and precision of rules.

Finally, we consider the equations of the rule prioritization metrics obtained by the proposed approach, which are shown in Table 8 for the four data sets. Here, we can observe that all of the equations have much higher multipliers for the confidence than the occurrence, but the multiplier values are not necessarily very similar to each other. This highlights the need to calculate separate rule prioritization metrics for each data set.

7. Potential Limitations

This section discusses the potential limitations of our work. First, the data sets used in this study were limited to four open-source projects, which could be a source of bias in the results. To increase their generality, it will therefore be important to conduct experiments using data sets taken from other software projects in our future work.

Second, the numbers of source file defects were always obtained by analyzing commit comments in relevant version control system. Although this practice is common in defect prediction studies [21], it has the limitation that any defects not recorded in commit comments cannot be identified. Further study will therefore be required to improve the accuracy of defect collection from version control systems.

Finally, to extract the association rules (Step 3 of the proposed method) before predicting defects (Step 4), we set certain threshold values, namely, minimum transactions = 5, minimum Conf = 0.6 and maximum rule length = 3. Since these value choices may have affected the results, we are planning to conduct experiments using different values in a future work.

8. Conclusion

This paper has proposed a cross-validation-based metric to quantify the prediction power of the association rules used to characterize software defects to prioritize them effectively. The results of evaluating this metric experimentally using four open-source data sets showed that it was able to improve the **SumNormPre**(100) values by 72.8% for the Mylyn data set, 15.0% for NetBeans, and 10.5% for Apache Ant. For jEdit, it produced very similar results to the odds ratio metric. These results suggest that the proposed metric can provide better rule prioritization performance than conventional metrics and can at least provide similar performance even in the worst case.

In a future work, we are planning to conduct experiments with a broader range of data sets to evaluate the generality of our approach. In addition, we are planning to combine our rule prioritization approach with rule reduction [1] to better identify useful association rules.

Acknowledgments

This work was supported in part by JSPS KAKENHI Grant

number 17K00102.

References

[1] A. Monden, J. Keung, S. Morisaki, Y. Kamei, and K. Matsumoto, "A heuristic rule reduction approach to software fault-proneness prediction," Software Engineering Conference (APSEC), 2012 19th Asia–Pacific, pp.838–847, IEEE, Dec. 2012.
[2] S. Morisaki, A. Monden, T. Matsumura, H. Tamada, and K. Matsumoto, "Defect data analysis based on extended association rule mining," Fourth International Workshop on Mining Software Repositories (MSR'07:ICSE Workshops 2007), pp.3–3, May 2007.
[3] Q. Song, M. Shepperd, M. Cartwright, and C. Mair, "Software defect association mining and defect correction effort prediction," IEEE Trans. Softw. Eng., vol.32, no.2, pp.69–82, Feb. 2006.
[4] T. Watanabe, A. Monden, Y. Kamei, and S. Morisaki, "Identifying recurring association rules in software defect prediction," 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS), pp.1–6, June 2016.
[5] T.-D.B. Le and D. Lo, "Beyond support and confidence: Exploring interestingness measures for rule-based specification mining," 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), pp.331–340, March 2015.
[6] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," SIGMOD Rec., vol.22, no.2, pp.207–216, June 1993.
[7] Y. Kamei, A. Monden, and K. Matsumoto, "Empirical evaluation of svm-based software reliability model," Proc. 5th ACM-IEEE Int'l Symposium on Empirical Software Engineering (ISESE2006), pp.39–41, 2006.
[8] Y. Kamei, A. Monden, S. Matsumoto, T. Kakimoto, and K. Matsumoto, "The effects of over and under sampling on fault-prone module detection," First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007), pp.196–204, Sept. 2007.
[9] Y. Kamei, A. Monden, S. Morisaki, and K. Matsumoto, "A hybrid faulty module prediction using association rule mining and logistic regression analysis," Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '08, New York, NY, USA, pp.279–281, ACM, 2008.
[10] T.M. Khoshgoftaar and E.B. Allen, "Modeling software quality with classification trees," Recent Advances in Reliability and Quality Engineering, vol.2, pp.247–270, 2001.
[11] 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 Trans. Softw. Eng., vol.39, no.10, pp.1345–1357, Oct. 2013.
[12] K.E. Bennin, J. Keung, A. Monden, Y. Kamei, and N. Ubayashi, "Investigating the effects of balanced training and testing datasets on effort-aware fault prediction models," Computer Software and Applications Conference (COMPSAC), pp.154–163, IEEE, 2016.
[13] Z. Sun, Q. Song, and X. Zhu, "Using coding-based ensemble learning to improve software defect prediction," IEEE Trans. Syst., Man, Cybern., Part C (Applications and Reviews), vol.42, no.6, pp.1806–1817, Nov. 2012.
[14] L. Pelayo and S. Dick, "Applying novel resampling strategies to software defect prediction," NAFIPS 2007 - 2007 Annual Meeting of the North American Fuzzy Information Processing Society, pp.69–72, June 2007.
[15] B. Kitchenham and E. Mendes, "Why comparative effort prediction studies may be invalid," Proceedings of the 5th International Conference on Predictor Models in Software Engineering, PROMISE '09, New York, NY, USA, pp.4:1–4:5, ACM, 2009.
[16] K. Yamamura, "Transformation using (x + 0.5) to stabilize the variance of populations," Researches on Population Ecology, vol.41, no.3, pp.229–234, Dec. 1999.
[17] J.S. Shirabad and T. Menzies, "The PROMISE Repository of Software Engineering Databases," School of Information Technology and Engineering, University of Ottawa, Canada, 2005.
[18] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," Proceedings of the 6th International Conference on Predictive Models in Software Engineering, PROMISE '10, New York, NY, USA, pp.9:1–9:10, ACM, 2010.
[19] M. Jureczko and D. Spinellis, "Using object-oriented design metrics to predict software defects," In Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej, pp.69–81, 2010.
[20] S. Morisaki, A. Monden, H. Tamada, T. Matsumura, and K. Matsumoto, "Mining quantitative rules in a software project data set," Information and Media Technologies, vol.2, no.4, pp.999–1008, 2007.
[21] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," Proceedings of the 30th international conference on Software engineering, pp.181–190, ACM, 2008.

Takashi Watanabe is a Master's student in the Division of Electronic and Information Systems Engineering at the Graduate School of Natural Science and Technology, Okayama University. He received a B.E. degree in Information Technology from Okayama University in 2016. His research interests include software measurement and analytics.

Akito Monden is a professor in the Graduate School of Natural Science and Technology at Okayama University, Japan. He received the B.E. degree (1994) in electrical engineering from Nagoya University, and the M.E. and D.E. degrees in information science from Nara Institute of Science and Technology (NAIST) in 1996 and 1998, respectively. His research interests include software measurement and analytics, and software security and protection. He is a member of the IEEE, ACM, IEICE, IPSJ and JSSST.

Zeynep Yücel is an assistant professor at Okayama University, Japan. She obtained her B.S. degree from Bogazici University, Istanbul, Turkey, and her M.S. and Ph.D. degrees from Bilkent University, Ankara, Turkey in 2005 and 2010, all in electrical engineering. She was a postdoctoral researcher at ATR labs in Kyoto, Japan for 5 years, before being awarded a JSPS fellowship in 2016. Her research interests include robotics, signal processing, computer vision, and pattern recognition.

Yasutaka Kamei is an associate professor at Kyushu University in Japan. He has been a research fellow of the JSPS (PD) from July 2009 to March 2010. From April 2010 to March 2011, he was a postdoctoral fellow at Queen's University in Canada. He received his B.E. degree in informatics from Kansai University, and M.E. and Ph.D. degrees in information science from Nara Institute of Science and Technology. His research interests include empirical software engineering and mining software repositories (MSR).

Shuji Morisaki is an associate professor at Nagoya University, Japan. Previously, he has been a software engineer in the Japanese software industry. He received a D.E. in information science from Nara Institute of Science and Technology, Japan in 2001. His research interests include empirical software engineering, software reviews and inspections, and mining software repositories.