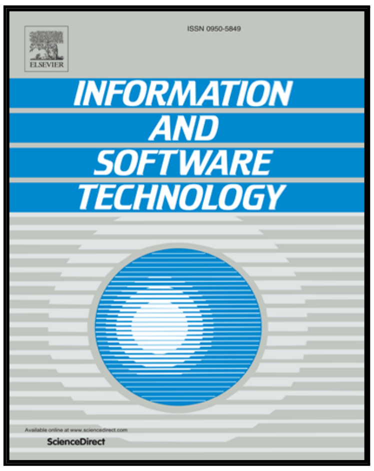

ISSN 0950-5849
ELSEVIER
INFORMATION AND SOFTWARE TECHNOLOGY
Available online at www.sciencedirect.com
ScienceDirect

Dealing with imbalanced data for interpretable defect prediction

Yuxiang Gao , Yi Zhu , Yu Zhao

PII: S0950-5849(22)00141-0
DOI: https://doi.org/10.1016/j.infsof.2022.107016
Reference: INFSOF 107016

To appear in: *Information and Software Technology*

Received date: 7 December 2021
Revised date: 15 July 2022
Accepted date: 18 July 2022

Dealing with imbalanced data for interpretable defect prediction

Yuxiang Gao [a], Yi Zhu [*] [a,b] ,and Yu Zhao [a]
[a] School of Computer Science and Technology, Jiangsu Normal University, China
[b] Key Laboratory of Safety-Critical Software, Nanjing University of Aeronautics and Astronautics, China
[*] Corresponding author.
E-mail addresses: gaoyx@jsnu.edu.cn, zhuy@jsnu.edu.cn, zhaoyu@jsnu.edu.cn

Abstract

Context: Interpretation has been considered as a key factor to apply defect prediction in practice. As interpretation from rule-based interpretable models can provide insights about past defects with high quality, many prior studies attempt to construct interpretable models for both accurate prediction and comprehensible interpretation. However, class imbalance is usually ignored, which may bring huge negative impact on interpretation.
Objective: In this paper, we are going to investigate resampling techniques, a popular solution to deal with imbalanced data, on interpretation for interpretable models. We also investigate the feasibility to construct interpretable defect prediction models directly on original data. Further, we are going to propose a rule-based interpretable model which can deal with imbalanced data directly.
Method: We conduct an empirical study on 47 publicly available datasets to investigate the impact of resampling techniques on rule-based interpretable models and the feasibility to construct such models directly on original data. We also improve gain function and tolerate lower confidence based on rule induction algorithms to deal with imbalanced data.
Results: We find that (1) resampling techniques impact on interpretable models heavily from both feature importance and model complexity, (2) it is not feasible to construct meaningful interpretable models on original but imbalanced data due to low coverage of defects and poor performance, and (3) our proposed approach is effective to deal with imbalanced data compared with other rule-based models.
Conclusion: Imbalanced data heavily impacts on the interpretable defect prediction models. Resampling techniques tend to shift the learned concept, while constructing rule-based interpretable models on original data may also be infeasible. Thus, it is necessary to construct rule-based models which can deal with imbalanced data well in further studies.

Keywords

Software defect prediction, Class imbalance, Interpretable machine learning, Rule-based models

1 Introduction

Software defect prediction is a popular research topic of software engineering. The general idea of software defect prediction is using statistical or machine learning approaches to predict whether a software module (or line, method, etc.) is buggy [1]. Traditionally, the defect prediction aims to predict the defect modules accurately and thus practitioners can arrange limited QA resources to test them. As a result, plenty prior studies try to improve the model to get more accurate prediction [2-8]. Recently, interpretation has also been considered as a key factor for applying defect prediction techniques in practice [9-11]. Since most of the widely-held beliefs (e.g., the relation between code metrics and defect) are only sporadically supported in the data [12], practitioners may not trust the counter-intuitive prediction without proper explanation, and thus they may refuse to adopt defect prediction in practice [9]. Further, using the explanation of defect prediction models can give insight from historical defects and might be helpful to guide QA activities [13, 14]. Thus, interpretation is also concerned and expected to achieve the two goals when interpreting defect prediction models: 1) understanding the most important characteristics that contributed to a prediction of a file, and 2) understanding the characteristics associated with past defects [10].

Most of the defect prediction models are based on machine learning techniques. Fig 1 shows the relationship of XAI (eXplainable AI) approaches for explaining machine learning decisions [15] and the relationship between those XAI approaches and goals of interpreting defect prediction models [10]. To achieve the first goal, recent studies adopt local explanation techniques [14, 49, 75] and model inspection techniques [16, 54] to explain individual predictions of black-box models. To achieve the second goal, model inspection techniques [10, 13], global explanation techniques [17], and transparent models [16, 18-22, 76] are also used by prior studies. Note that the global explanation techniques use transparent models to imitate existing powerful black-box models, and such models are also interpretable. Thus, we assume that both surrogate models (i.e., models used as global explanation) and transparent models are interpretable models in the rest of this paper.

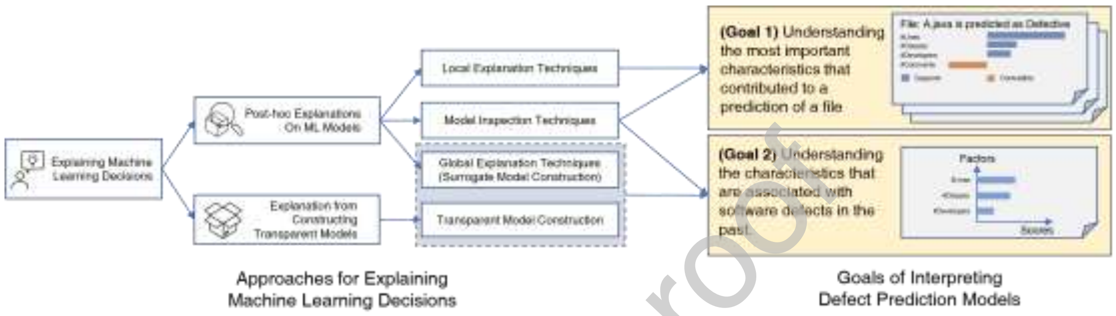

Explaining Machine Learning Decisions
Post-hoc Explanations On ML Models
Explanation from Constructing Transparent Models
Local Explanation Techniques
Model Inspection Techniques
Global Explanation Techniques (Surrogate Model Construction)
Transparent Model Construction
(Goal 1) Understanding the most important characteristics that contributed to a prediction of a file
File: A.java is predicted as Defective
(Goal 2) Understanding the characteristics that are associated with software defects in the past
Factors
Scores
Approaches for Explaining Machine Learning Decisions
Goals of Interpreting Defect Prediction Models

Fig 1. Approaches for explaining machine learning decisions and the relationship between such approaches and goals of interpreting defect prediction models. Both surrogate models and transparent models (in grey dashed box) are considered as interpretable models in this paper. Note that interpretable models are not practical to achieve the first goal mainly due to the poor performance, compared with black-box models.

When we revisit prior studies on building interpretable defect prediction models [16-22, 76], we find that most of them do not consider the problem of imbalanced data. As Tantithamthavorn et al. [11] point out, ignorance of class imbalance is one of the common pitfalls in defect prediction studies and practice. This seems easy to be dealt with, as it is easy to adopt resampling techniques to preprocess the data with public packages such as *imblearn* for python. However, resampling techniques have negative impact on the interpretation, which implies that resampling techniques should be avoided when the model is used for interpretation [25].

Except logistic regression, rest of the classification models investigated by Tantithamthavorn et al. [25] are black-box models. The impact of resampling techniques on rule-based interpretable models[1] is still unclear. Though rule-based interpretable models may not be as accurate as black-box models, they are still valuable since they can provide more information (e.g., the direction of relationship of each feature) than model inspection techniques such as permutation importance. They are also agreed by practitioners to provide insightful explanations about past defects with high quality [10]. Therefore, it is necessary to investigate the impact of resampling techniques on rule-based interpretable models. Further, as Tantithamthavorn et al. [25] suggest that defect prediction models should be trained on original data for interpretation, it is still questionable about the feasibility for constructing those models directly on original data.

Thus, we conduct an empirical study on 47 publicly available datasets with 3 rule-based interpretable models. We observe that 1) resampling techniques change the feature importance, especially on top-ranked feature, 2) oversampling-based resampling techniques (such as random oversampling and SMOTE) tend to increase the complexity of rule-based models, and 3) rule-based interpretable models cannot fit the defective instances well if the data is heavily imbalanced for defective instances. As a result, it is more preferred to construct rule-based interpretable models which can deal with imbalanced data directly. Therefore, we propose an improved rule-based approach, and experiments on 47 publicly available datasets show its advantages on both performance and interpretation.

[1] In this paper we consider tree-based models as rule-based models since tree-based models such as C4.5 trees can also be converted into a set of rules.

1.1 Contributions

Contributions of this paper are:

1) We investigate the impact of resampling techniques on interpretation of rule-based interpretable models from both feature importance and model complexity through an empirical study on 47 publicly available datasets.

2) We investigate the feasibility to construct rule-based interpretable models directly on original data without resampling.

3) We propose an improved rule-based approach to deal with imbalanced data directly and show its advantages on both performance and interpretation.

1.2 Paper Organization

The rest of this paper is organized as follows. We give the motivation for three research questions in Section 2. We give the detailed design of our case study in Section 3. We present the experiment results and gives detailed discussion to address our research questions in Section 4. We introduce and evaluate our proposed rule-based approach in Section 5. We discuss our proposed approach and rule-based models for defect prediction in Section 6. Related work is presented in Section 7. Finally, we give conclusions in Section 8.

2 Motivation

In practice, the defective modules are often less than the non-defective ones, and the defect datasets are often imbalanced [25]. It seems easy to be dealt with by simply applying resampling techniques on training data. The performance will be significantly improved when constructing models on resampled data [25, 27]. However, as Turhan [23] mentioned, *sample selection bias* will be introduced when applying resampling techniques due to changes in data distribution [24]. It is further be confirmed [25], which implies that knowledge derived from the defect prediction models trained on resampled data may be unreliable. Except Logistic Regression, most of the investigated models are not interpretable. It is unclear whether this phenomenon will occur on rule-based interpretable models. Thus, we propose the following research question about the impact on feature importance.

***RQ*1. Do top-ranked features of rule-based interpretable models remain consistent if resampling techniques are applied?**

Most of the resampling techniques are either duplicating (synthesizing) instances of minor class or dropping instances of major class. When using oversampling-based techniques, more data points are generated, and more rules may be required to cover the extra data points. Since increasing the model complexity decreases the interpretability [15, 20, 26], we propose the second research question about the impact of resampling techniques on model complexity.

***RQ*2. How will the model complexity change when applying different resampling techniques?**

Since resampling techniques have negative impact on interpretation, resampling techniques are suggested to be avoided if the model is used for interpretation [11, 25]. Most models suffer from imbalanced data, and it is questionable whether rule-based interpretable models trained on original data without resampling can always characterize the past defects and make meaningful explanation about new predictions. Thus, we propose the following research question.

***RQ*3. Is it feasible to construct interpretable defect prediction models with meaningful explanation on imbalanced data?**

3 Case Study Design

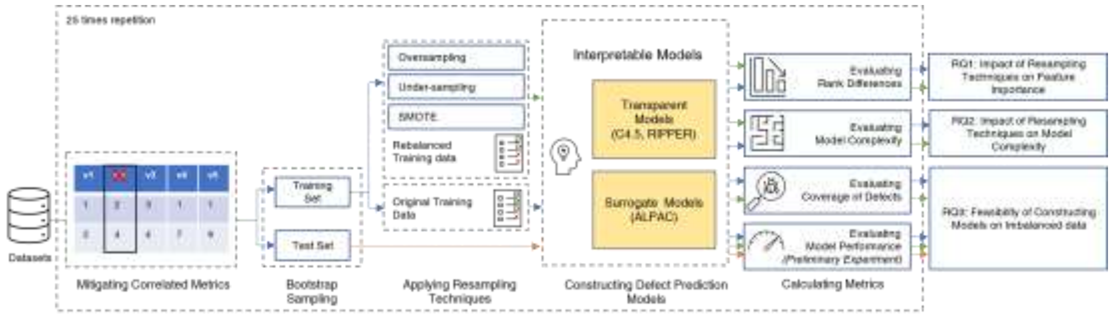

25 times repetition
Datasets
Mitigating Correlated Metrics
Training Set
Test Set
Bootstrap Sampling
Oversampling
Under-sampling
SMOTE
Rebalanced Training data
Original Training Data
Applying Resampling Techniques
Interpretable Models
Transparent Models (C4.5, RIPPER)
Surrogate Models (ALPAC)
Constructing Defect Prediction Models
Evaluating Rank Differences
Evaluating Model Complexity
Evaluating Coverage of Defects
Evaluating Model Performance (Preliminary Experiment)
Calculating Metrics
RQ1: Impact of Resampling Techniques on Feature Importance
RQ2: Impact of Resampling Techniques on Model Complexity
RQ3: Feasibility of Constructing Models on Imbalanced data

Fig 2. An overview of our case study design.

The overview of our case study design is shown in Fig 2. We select publicly available datasets, and we mitigate the correlated metrics and do bootstrap sampling. To find the impact of resampling techniques on interpretable models, we build defect prediction models by feeding different group of training data by applying different resampling techniques. The original imbalanced data is also used to build model for contrast. After building the model, we calculate the metrics of performance and interpretation. The whole process is repeated 25 times. Finally, we analyze the result of all repetitions.

3.1 Datasets

Suggested by Tantithamthavorn et al. [25], we select datasets which are publicly available and from different corpora and domains. Among all the 101 releases, we exclude NASA dataset [31] since Petric et al. [32] argue that problematic data remains in the cleaned version of NASA dataset. Using such problematic data may not make the result trustable. We also exclude some datasets [81], [82] which are not available now (i.e., the websites are not accessible). In total, 47 releases of different projects (3 releases provided by Kim et al. [8], 5 releases provided by D'Ambros et al. [28], 36 releases provided by Jureckzo et al. [29] and 3 releases provided Wu et al. [30]) are selected in this study, which are shown in Table 1.

Table 1 Details of selected datasets

Source	Project	# Instances	% Defective
D'Ambros et al.	Eclipse PDE UI (PDE)	1,497	13.97
	Eclipse JDT Core (JDT)	997	20.66
	Mylyn (ML)	1,862	13.16
	Lucene (LC)	691	9.26
	Equinox (EQ)	324	39.81
Wu et al.	Apache	194	49.5
	Safe	56	39.3
	Zxing	399	29.6
Kim et al.	Columba	1,800	29.4
	Eclipse	659	10.2
	Scarab	1,090	33.6
Jureckzo et al.	Ant-1.3	125	16.00
	Ant-1.4	178	22.47
	Ant-1.5	293	10.92
	Ant-1.6	351	26.21
	Ant-1.7	745	22.28
	Camel-1.0	339	3.83
	Camel-1.2	608	35.53
	Camel-1.4	872	16.63
	Camel-1.6	965	19.48
	Ivy-1.1	111	56.76
	Ivy-1.4	241	6.64
	Ivy-2.0	352	11.36
	Jedit-3.2	272	33.09
	Jedit-4.0	306	24.51
	Jedit-4.1	312	25.32
	Jedit-4.2	367	13.08
	Jedit-4.3	492	2.24
	Log4j-1.0	135	25.19
	Log4j-1.1	109	33.94
	Log4j-1.2	205	92.20

Lucene-2.0	195	46.67
Lucene-2.2	247	58.30
Lucene-2.4	340	59.71
Poi-1.5	237	59.49
Poi-2.0	314	11.78
Poi-2.5	385	64.42
Synapse-1.0	157	10.19
Synapse-1.1	222	27.03
Synapse-1.2	256	33.59
Xalan-2.4	723	15.21
Xalan-2.5	803	48.19
Xalan-2.6	885	46.44
Xalan-2.7	909	98.79
Xerces-1.2	440	16.14
Xerces-1.3	453	15.23
Xerces-1.4	588	74.32

3.2 Mitigating Correlated Features

Most of the feature selection techniques has impact on the feature importance [33]. Since we are exploring the impact of imbalanced data on rule-based interpretable models, we do not expect to involve more factors which potentially impacts on interpretation. Hence, no feature selection techniques are applied. However, multicollinearity heavily impacts on the interpretation [33, 34]. Thus, to reduce such impact, we follow the approach described in literature [35], which are also applied in prior defect prediction studies [25, 34]. We use python package *varclushi* (an approximation to SAS implementation [36] which is used in prior studies) to do the hierarchical clustering for variables and use python package *pandas* to calculate the Spearman correlation among variables. In each cluster, we iteratively remove one attribute with highest $1 - R^2 Ratio$, and evaluate the Spearman correlation of rest attributes in the cluster, until the absolute value of Spearman correlation of rest attributes in the cluster is less than 0.7, i.e., $|\rho| < 0.7$.

3.3 Bootstrap Sampling

To leverage aspects of statistical inference [37, 38] and produce the least bias and variance of performance estimates [39], we follow the practice of [25, 33, 34] to use the out-of-sample bootstrap validation technique to generate training samples. For each dataset with size N, we random choose N instances from the original dataset with replacement as training set. The instances which are not chosen are left as test set. On average, 36.8% instances are left as test instances [38].

3.4 Applying Resampling Techniques

We adopt oversampling, under-sampling and SMOTE [40] as resampling techniques in this study since they are widely used in previous works and have been evaluated to be effective on performance [25, 27, 41-42]. Oversampling randomly duplicates the instances of minor class, while under-sampling randomly removes the instances of major class. SMOTE is a synthetic resampling technique, which generate synthetic data points of minor class from the nearest neighbors. Since both oversampling and SMOTE will increase the number of minor class instances, we call them *oversampling-based techniques* in this paper later. Illustrative examples of the studied resampling techniques are shown in Fig 3. We use the python package *imblearn* as the implementations of such resampling techniques and we use their default parameters

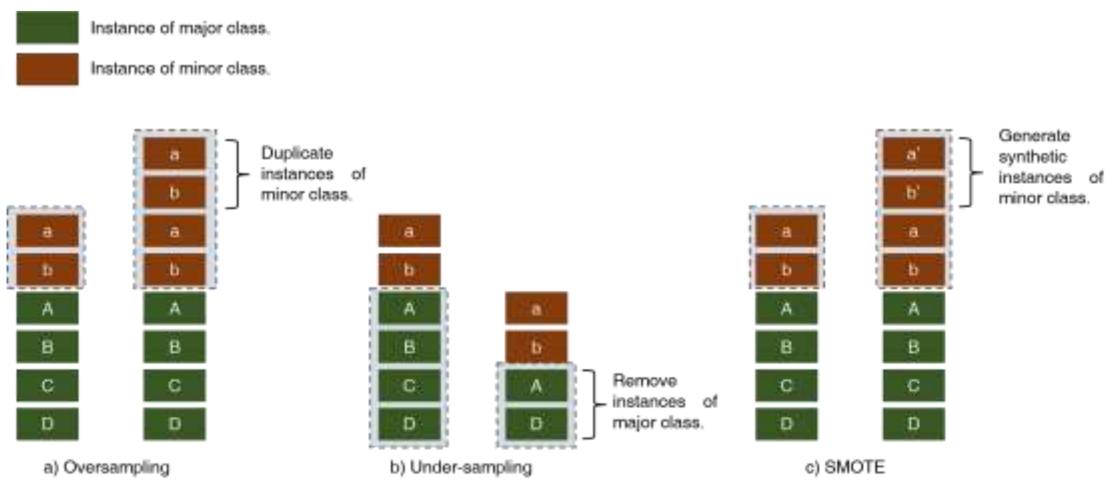

Instance of major class.
Instance of minor class.
Duplicate instances of minor class.
Remove instances of major class.
Generate synthetic instances of minor class.
a) Oversampling
b) Under-sampling
c) SMOTE

Fig 3. Illustrative examples of investigated resampling techniques.

3.5 Constructing Defect Prediction Models

In this subsection, we construct the interpretable defect prediction models. We publicly available rule-based models since only regression and rule-based models and their variants are recognized as transparent models [15, 43-45] and the impact of resampling techniques on logistic regression has been fully investigated [25]. We also exclude some prior rule-based work [18, 19, 21, 22] since their implementations are not provided. As a result, we adopt C4.5, RIPPER and ALPAC as interpretable models in this study.

C4.5 [46] and RIPPER [47] are popular transparent models. ALPAC [17] is a surrogate model which uses decision trees to imitate black-box models such as random forest. Both C4.5 and RIPPER are using the internal implementation of Weka 3.8.5, and the implementation of ALPAC are the Java package provided by the authors' prior work [48]. Details settings of those selected models are listed in Table 2.

Table 2 Settings of the models

Type	Classification Model	Settings
Transparent models	C4.5	Default of Weka J48.
	RIPPER	Default of Weka JRip.
Surrogate models	ALPAC	Black-box: Weka Random Forest with 1000 individual trees. White-box: Default of Weka J48.

We train the models on 2 groups of training data, as the overall framework described in Fig 2. The first group training data is original data, while the second group is resampled by each investigated resampling technique.

After constructing the models, we conduct a *preliminary experiment* for all the investigated models on data with different resampling techniques. We use the accuracy on training data to measure how well the models can fit the data, and AUC on test data to evaluate how well they can perform for prediction. We also use confidence and support to evaluate the quality of rules for defective instances. The results are shown in Table 3 below. We find that all the investigated models fit the data well from metric Accuracy on training data, but they perform poorly on test data from metric AUC on test data. Besides, they generate rules with high confidence but low support for defective instances.

Table 3 Results of preliminary experiments. Means are reported with standard deviation in parentheses.

Resampling Technique	Model	Accuracy@Training	AUC@Test	Average Confidence of Rules	Average Support of Rules
None	C4.5	0.941 (0.060)	0.630 (0.108)	0.953 (0.029)	0.069 (0.059)
	RIPPER	0.904 (0.068)	0.630 (0.097)	0.854 (0.078)	0.221 (0.131)
	ALPAC	0.919 (0.062)	0.621 (0.104)	0.850 (0.082)	0.047 (0.039)
Random Oversampling	C4.5	0.960 (0.051)	0.639 (0.095)	0.955 (0.030)	0.043 (0.035)
	RIPPER	0.932 (0.070)	0.646 (0.096)	0.861 (0.079)	0.143 (0.080)
	ALPAC	0.915 (0.055)	0.642	0.835 (0.095)	0.039 (0.028)

			(0.095)		
SMOTE	C4.5	0.945 (0.057)	0.648 (0.103)	0.900 (0.057)	0.046 (0.040)
	RIPPER	0.912 (0.069)	0.656 (0.098)	0.810 (0.110)	0.156 (0.080)
	ALPAC	0.888 (0.060)	0.648 (0.097)	0.745 (0.125)	0.037 (0.030)
Random Under-sampling	C4.5	0.820 (0.096)	0.664 (0.094)	0.800 (0.109)	0.085 (0.074)
	RIPPER	0.772 (0.100)	0.669 (0.097)	0.685 (0.130)	0.264 (0.105)
	ALPAC	0.695 (0.114)	0.604 (0.106)	0.633 (0.112)	0.063 (0.061)

3.6 Evaluating the Impact of Imbalanced Data on Interpretation

3.6.1 Feature Importance

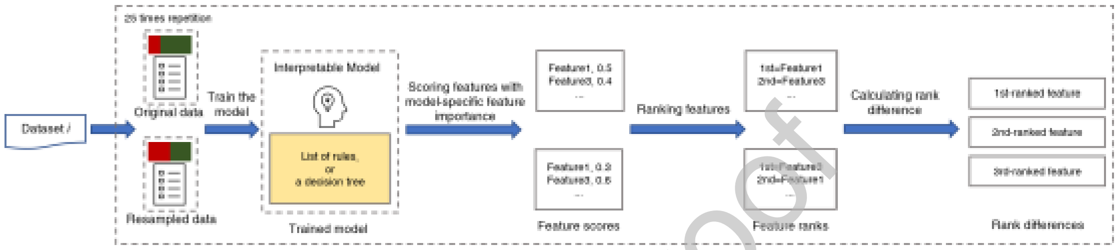


Fig 4. Approach for calculating rank difference of 3 top-ranked features for a given dataset.

As it is mentioned in prior study [25], the *concept drift* problem is detected by the changes of top-ranked features. Several studies focused on factors of interpretation also consider the changes of feature importance [34, 49]. Thus, we are going to investigate the differences of top-ranked features between models trained on original data and resampled data. In detail, we evaluate the *rank differences* of 3 top-ranked features. The approach for calculating such metrics is described in the Fig 4.

Step 1) Scoring the features. We start from scoring the features. Since in this study we only focused on interpretable models, we score the features by the feature importance from model-specific global explanation. We follow the practice of prior study [49] to use *usage* (i.e., the percentage of training instances that satisfy all the terminal nodes after the split which are associated with the metric, i.e., feature) of feature [46] as the model-specific feature importance for models which use decision trees as interpretable representation. Similarly, the feature importance of rule set is defined as the percentage of training instances that satisfy the terms which are associated with the metric.

Step 2) Ranking the features. We use the score calculated in Step 1 to rank the features. If n feature have the same importance score, we assume they share the same rank r, and the next feature's rank is $r+1$. For example, if score of 4 features is [wmc=0.7, loc=0.7, rbo=0.4, rfc=0.3], then the rank is [1st=wmc, 1st=loc, 2nd=rbo, 3rd=rfc].

Step 3) Calculating the rank difference. In step 2, we get the features ranked by the feature importance. We calculate the rank difference for each of the 3 top-ranked features in each repetition. If in one repetition, a feature v appears in rank i from model trained on original data, and appears in rank j from model trained on resampled data, then rank difference (for this repetition) is calculated as $rd(v) = |i - j|$. For example, if the features in model trained on original data are ranked as [1st=loc, 2nd=wmc, 3rd=rfc, 4th=rbo], and features in model trained on resampled data are ranked as [1st=wmc, 2nd=rbo, 3rd=rfc, 4th=loc], then the rank differences of 1st-ranked, 2nd-ranked, and 3rd-ranked feature (for this repetition) are $|1-4| = 3$, $|2-1| = 1$, and $|3-3| = 0$, respectively.

The *rank difference* only measures whether the ranking of features will be changed. If resampling techniques just result in the little ranking changes of features, e.g., ranking changed from first to second, the explanation might not be impacted too much, since both the first and the second feature are all important. To investigate the level of changes for the most important feature, we also investigate *discrepancy* [34]. For a certain dataset, *discrepancy* is the maximum value of rank differences of 1st-ranked feature in all repetitions. Higher *discrepancy* means the most important features could be much less important when resampling techniques are applied.

3.6.2 Model Complexity

Complexity has been considered as one of the key factors for evaluating the interpretability of models in studies of both XAI (eXplainable AI) [15, 26, 50] and interpretable defect prediction [20]. Hence, we investigate the impact of resampling techniques on model complexity. The complexity metrics are usually specific to models. For example, *number of network nodes* are used to evaluate the complexity for Bayesian Network [20]. Accordingly, A decision tree models can be measured from the *number of tree nodes*, and a rule set can be measured by *the number of terms* (in all rules). Since a decision tree can be transformed into set of non-intersected decision rules by traversing all paths from the root to all leaves, number of tree nodes and number of terms are considered the metrics of same type. Thus, we call both as *Number of Nodes* in the rest of this paper.

Table 4 Metrics of model complexity for different type of models.

Model	Number of Nodes	Number of Decisions	Average Length
Decision tree	Number of tree nodes	Number of leaves	Average length of paths
Rule set	Number of rule terms	Number of rules	Average length of rules

Dong et al. [50] decompose the complexity (of a rule) as the product of *number of rules* and *average length of rules*, which are both important for evaluating the complexity of rule-based models. We use *number of leaves* and *average depth* to measure the complexity of a decision tree, which are equivalent to *number of rules* and *average length of rules* (the length of a rule is defined as the number of terms) for rule set in some extent. We rename them as *Number of Decisions* and *Average Length* respectively. Detailed metrics for evaluating model complexity are listed in Table 4. Note that in this paper the default rule for rule-based models is considered as a rule which length is 0.

3.6.3 Coverage of Instances

Rule-based and tree-based models could be used to derive knowledge and lessons from past defects [10], but this requires that the past defects should be well covered by the decision rules. Thus, we use *Coverage of Defects* (CD) to estimate how many defect instances are correctly covered by the rules which consequences are “defective”, among all the defective instances. In detail, for a given rule-based (or tree-based) model, we first find R_d, i.e., a subset of rule set R which only contains rules which consequents are “defective”. Then, we calculate the CD by

$$CD = \frac{\left|\bigcup_{r\in R_d} X_{d,r}\right|}{|X_d|}, \#(1)$$

where X_d and $X_{d,r}$ are the defective instances and defective instances covered by rule r respectively. Larger CD indicates that the rules fit the data better, and the rules have higher quality to describe the past defects.

3.7 Statistical Analysis

We use Wilcoxon signed-rank test to determine the statistical significance of metrics for two group models constructed on original data and resampled data respectively. The null hypotheses are that there is no significant difference between two group of models with significant level α at 0.05. If p-value is smaller than 0.05, we reject the null hypotheses; otherwise, we accept the null hypotheses.

We also use Cliff’s delta (δ) [80] to measure the effect size for data of two groups. Cliff’s delta δ ranges in $[-1,1]$, and the larger $|\delta|$ indicates the larger difference between the data of two groups. We consider there is a “Large (L)”, “Medium (M)”, “Small (S)”, and “Negligible (N)” difference between data of two groups when $|\delta|$ is above 0.474, between 0.33 and 0.474, between 0.147 and 0.33, less than 0.147, respectively.

4 Case Study Results

4.1 Impact of Resampling on Feature Importance

Approach. To address RQ1, after constructing the defect prediction model, we calculate the *rank difference* of 3 top-ranked features in each repetition for each dataset and model, and the *discrepancy* for each dataset and model, which are described in Section 3.6.1. We then present the distribution of *rank difference* of 3 top-ranked feature for each model with histograms in Fig 5 and present the *discrepancy* for each model with boxplots in Fig 6.

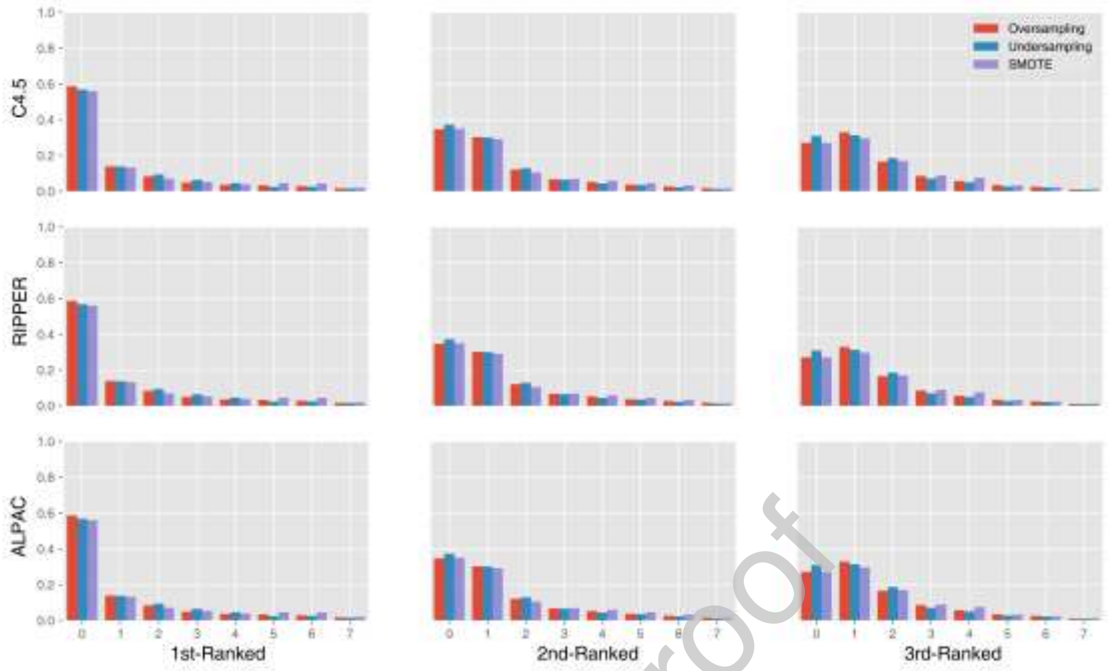


Fig 5. Rank differences of top-ranked features for different models with different resampling techniques. The bar represents the percentage of certain rank difference (e.g., rank difference is 1 for 1st-ranked feature) between the model using original data and model using resampled data for all datasets.

Results. The feature importance is greatly changed if resampling techniques are applied. The optimal situation is that the ranking of features does not change, and the rank difference should all be zeros, which means the rank of features remains consistent. However, we observe that only 55-58, 40-45, 28-42 percentage of 1st-ranked features are still in first rank if resampling techniques are applied for C4.5, RIPPER, and ALPAC respectively. This implies that resampling techniques may change the most important feature of such interpretable models. Further, the 2nd-ranked and 3rd-ranked features may also be changed. This confirms the *data shifting* problem suspected by Turhan [23], and it is identical as Tantithamthavorn et al.'s finding on other black-box models [25].

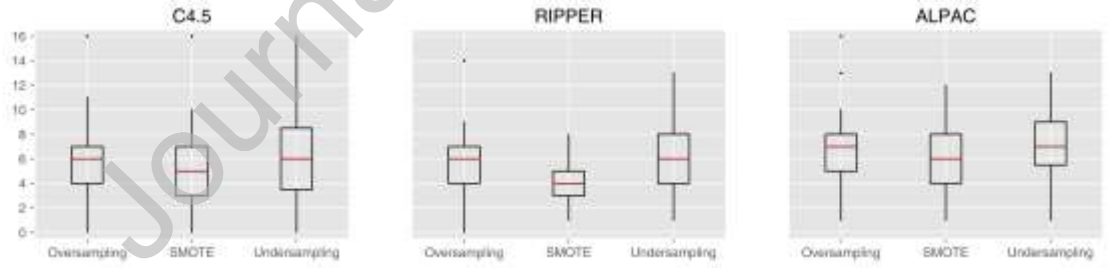


Fig 6. Discrepancy of different resampling techniques on different classification models. Lower discrepancy indicates that the most important features will be less impacted by resampling.

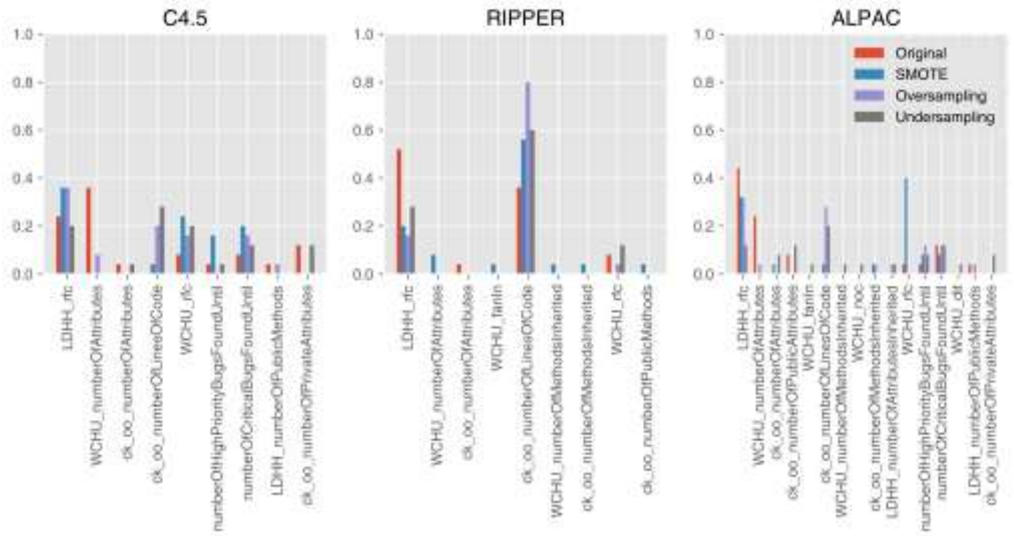


Fig 7. Distribution of top-ranked features for investigated models and resampling techniques in all repetitions for project *PDE*. The bar represents the frequency of the feature. The *x*-axis is the features.

Also, the medians of *discrepancy* are more than 4 as it is shown in Fig 6. This implies that the most important feature in the model trained on original data may become much less important when resampling techniques are applied. An example of changes in top-ranked features for project PDE is shown in Fig 7. When using original but imbalanced data, feature `WCHU_numberOfAttributes` and `LDHH_rfc` are the top-ranked features for C4.5 classifier, and feature `ck_oo_numberOfLinesOfCode` is never top-ranked. However, after rebalancing the data, feature `ck_oo_numberOfLinesOfCode` becomes one of the top-ranked features, while `WCHU_numberOfAttributes` is no longer top-ranked (except few repetitions for data with random over-sampling). Such opposite conclusions may mislead practitioners when they use the drifted explanation to make plans for SQA activities.

Conclusion. Top-ranked features of interpretable models will be changed a lot by all the investigated resampling techniques and interpretable models, and such changes will introduce negative impact on the interpretation. In another word, *concept drift* also occurs on interpretable models.

4.2 Impact of Resampling on Model Complexity

Approach. To address RQ2, we calculate the complexity metrics, i.e., *Number of Nodes*, *Number of Decisions*, and *Average Length*, which are introduced in Section 3.6.2. We calculate *relative difference* of a certain metric by the formula $rd = \frac{(m_{\text{resampled}} - m_{\text{original}})}{m_{\text{original}}}$, where m_{original} and $m_{\text{resampled}}$ represents the value of metric m of defect prediction models constructed on original data and resampled data respectively. We use the boxplots to report the results in Fig 8.

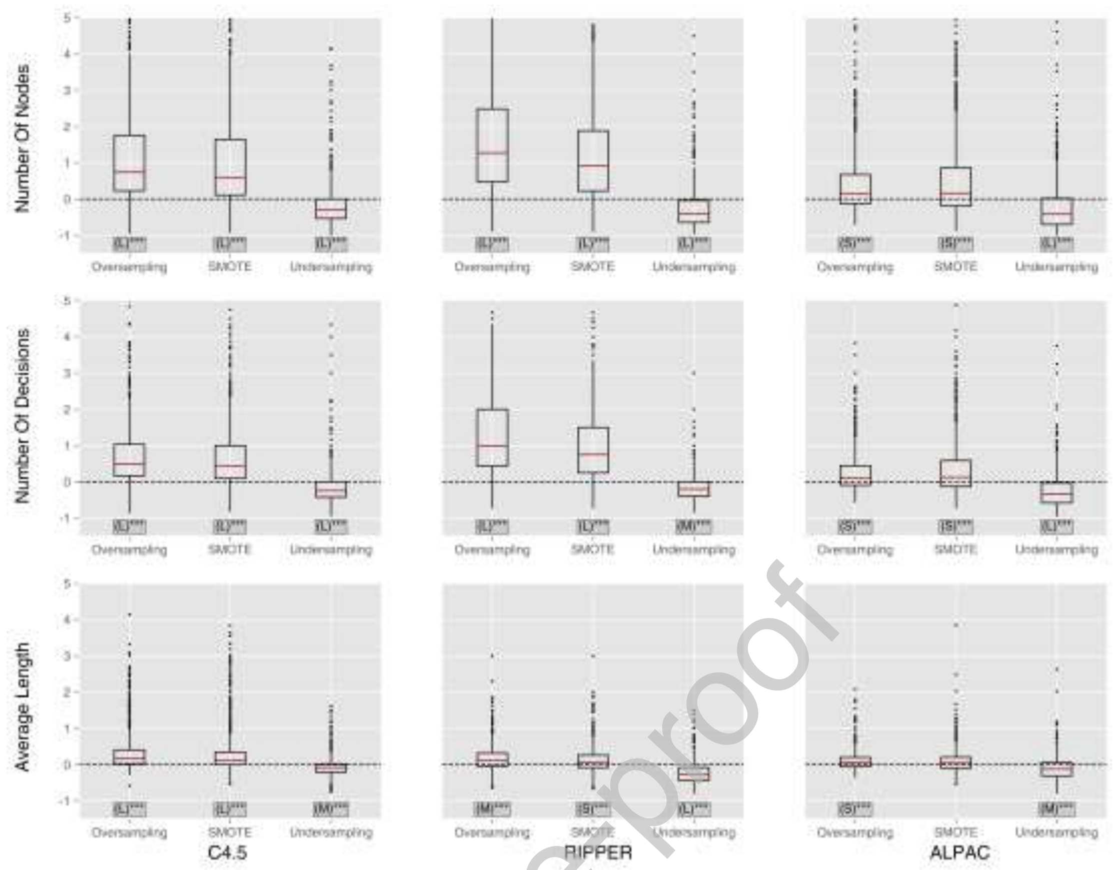


Fig 8. Boxplots for relative difference of complexity metrics when resampling techniques are applied. Black-dashed line indicates no relative difference. The small grey boxes represent the statistical significance and effect size. Note that *, **, *** means $p < 0.001$, $p < 0.01$, and $p < 0.05$ respectively. L/M/S indicates Large/Medium/Small effect size according to Cliff's delta.

Results. The complexity of models depends on the type of resampling techniques. For data resampled by *oversampling-based techniques*, the models tend to be much more complex. Median values of *Number of Nodes* increase up to 73.8, 127.3, and 16.1 percent for C4.5, RIPPER, and ALPAC. Since oversampling-based techniques adding minor class instances, the size of training set increases. The model will become much more complex to cover more resampled instances. Meanwhile, models constructed on under-sampled data tend to be less complex. The median values of *Number of Nodes* decrease about 30.2, 40.5 and 41.7 percent for C4.5, RIPPER and ALPAC. Random Under-Sampling drops the instances of major class, and the size of under-sampled dataset is quite smaller than the original dataset. Thus, when under-sampling is applied, the complexity of models decreases for all investigated models.

Besides, the complexity of C4.5 and RIPPER are more sensitive to oversampling-based techniques compared with ALPAC. For oversampling-based techniques, the *Number of Nodes* for C4.5 and RIPPER increases up to 73.8 and 127.2 percent while it only increases about 16.1 percent for ALPAC. ALPAC attempts to generate more data points close to the assumed decision boundary on resampled data, and one decision rule may cover data points generated from both active learning and resampling. Thus, the complexity of ALPAC may not increase too much.

Conclusion. Resampling techniques greatly impact on the model complexity for investigated tree-based and rule-based models. Oversampling-based techniques (i.e., oversampling and SMOTE) will dramatically increase the complexity for C4.5 and RIPPER. Meanwhile, under-sampling simplifies the model in some extent. Besides, the complexity of active learning-based surrogate models tends to be less impacted by resampling techniques compared with transparent models.

4.3 Feasibility for Constructing Interpretable Models Without Resampling

Approach. To address RQ3, we evaluate the feasibility for constructing interpretable models on original data from two aspects. Firstly, we evaluate how many defective instances are captured by the models with the metric *Coverage of Defects* (*CD*) described in Section

3.6.3. Secondly, we evaluate the operational performance of such models with the metric *AUC*. Since the defective rate impacts on the performance, we divide the projects into 7 groups according to their defective rate[2]. We calculate such metrics for models constructed on both original data and resampled data[3] on all projects and conduct statistical analysis between results of two groups. Finally, we present the results with boxplots.

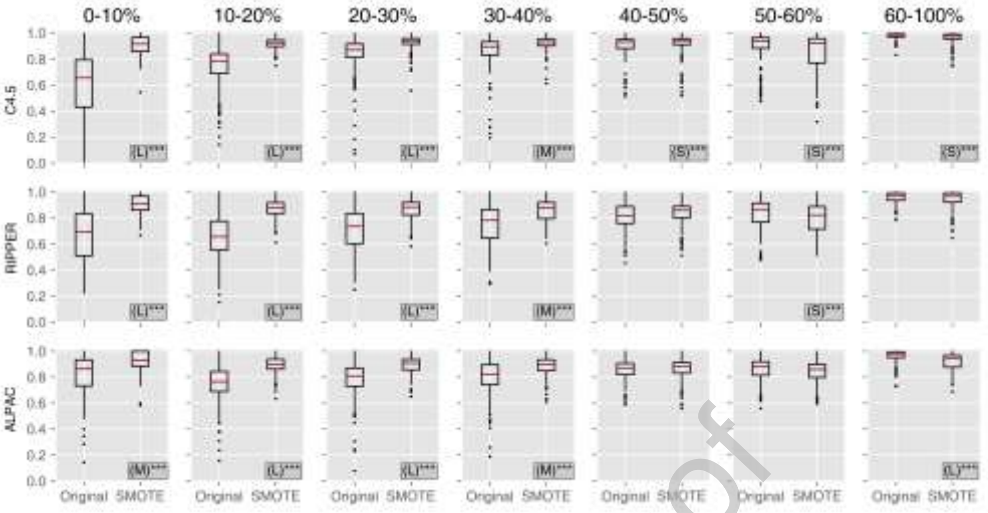


Fig 9. Coverage of Defects for different models constructed on original and resampled data. The columns represent different defective rate from 0% to 100%. The small grey boxes represent the statistical significance and effect size. Note that (1) *, **, *** means $p < 0.001$, $p < 0.01$, and $p < 0.05$ respectively, and (2) L/M/S indicates Large/Medium/Small effect size according to Cliff's delta.

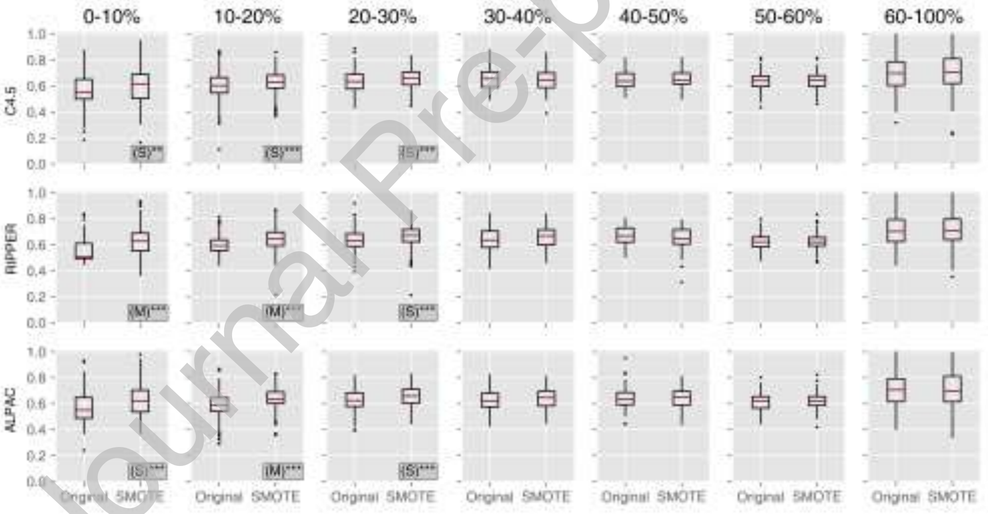


Fig 10. AUC for different models constructed on original and resampled data. The columns represent different defective rate from 0% to 100%. The small grey boxes represent the statistical significance and effect size. Note that (1) *, **, *** means $p < 0.001$, $p < 0.01$, and $p < 0.05$ respectively, and (2) L/M/S indicates Large/Medium/Small effect size according to Cliff's delta.

Result. The coverage of defects (CD) and AUC are shown in Fig 9 and Fig 10. We get the following results based on different defective rate.

Firstly, interpretable models cannot cover the past defects well when the defective rate is below 0.4. As it is shown in Fig 9, when defective rate is smaller than 0.3, both C4.5 and RIPPER models get lower coverage of defects, compared with models constructed on resampled data. Specially, for projects with defective rate less than 0.2, the median values of CDs for C4.5 and RIPPER models are less than 0.8, which means that about 20 percent of defects cannot be captured by the decision rules for more than half of the datasets. ALPAC gets better coverage than C4.5 and RIPPER, but it still cannot cover the defective instances

[2] Since there are few projects with defective rate above 60 percent, we put them in the same group (the last column in Fig 9 and Fig 10).
[3] For simplicity, we use the state-of-the-art method SMOTE to resample the data, and train models based on resampled data as baseline.

well without resampling techniques. In Fig 10, there are also significant differences for metric AUC between models constructed on original data and resampled data. For projects with defective rate between 0.3 and 0.4, there is still significant difference between all the three models constructed on original data and models constructed on resampled data with medium effect size for coverage of defects. Therefore, the interpretable models cannot cover many defective instances as expected when the data is imbalanced for defective instances (i.e., the defective rate is below 40 percent), and thus such models get poor performance for imbalanced data. Though all the models can correctly cover 90 percent of training data on average without resampling (see Table 3 in Section 3.5), they cannot fit the data well due to ignorance of minor class instances. For example, there are 16 defective modules in project *camel-1.0*, while RIPPER gets only one rule "`(ca >= 66) => 1`" for 5 defective modules. In that case, information of the rest defective instances will be ignored. Thus, such rules cannot provide insights about past defects and the model do not have a good operational performance, which indicates that it is not feasible to construct interpretable models on data which is imbalanced for defective instances.

Secondly, when the data is nearly balanced (i.e., defective rate is between 0.4 and 0.6), there are small or negligible differences between models constructed on original data and resampled data for both metrics. In another word, interpretable models do not benefit much from resampling when the data is nearly balanced. However, it is still may not be feasible to construct the models on original data since they cannot achieve good operational performance (i.e., achieve median AUC at 0.7) suggested by Rajbahadur et al. [13].

Thirdly, when defective rate is larger than 0.6, the defective instances become major class instances, and thus the models constructed on original data can cover the past defects well and achieve a good operational performance with median AUC larger than 0.7 for all the three models without resampling the data. Thus, we assume that it may be feasible to construct interpretable models for data which defective rate is larger than 0.6.

Conclusion. The feasibility of constructing interpretable models on original data depends on the defective rate. Rule-based and tree-based interpretable models cannot cover the defect instances well if the defective rate is quite small (i.e., below 0.4), and they cannot achieve good operational performance for data which is nearly balanced. Thus, it is not feasible to construct interpretable models for data which defective rate is below 0.6. On the other hand, for data which defective rate is larger than 0.6, such models can cover the defects well and achieve good operational performance, which implies the feasibility to construct interpretable models on data with high defective rate.

5 Improved Rule Induction Approach

5.1 Motivation

If practitioners are going to adopt interpretable models for defect prediction, they may want to focus more on the interpretation behind the prediction. However, if we use resampling techniques to rebalance the data, *concept drift* occurs. Meanwhile, if we follow the suggestion of Tantithamthavorn et al. [25] to use original data directly, such interpretable models may not be capable to provide reliable knowledge and insights due to low coverage of past defects and poor performance. Thus, we propose an improved rule-based approach to deal with imbalanced data without applying resampling techniques.

5.2 Approach

Algorithm 1 Rule Set Generation

```
Input: Training Data $X$.
Output: A rule set $R$.
1  Let $R$ be an empty set
2  Divide $X$ to $X_{\text{major}}$ and $X_{\text{minor}}$
3  Set $t(x) = 0$ for all $x \in X_{\text{minor}}$
4  do
5      Initialize an empty rule $r$ with label of minor class.
6      while true
7          Let $X_{\text{minor},r}$ as the set of minor class instances covered by $r$
```

```
8          for each candidate term $T_i$ based on $X_{\text{minor},r}$ [4] do
9              Calculate its gain $g_i$ by Equation (4)
10         if no candidate available then
11             $t(x) = t(x) + 1$ for all $x \in X_{\text{minor},r}$
12             break
13         Add $T_i$ to $r.\text{terms}$, where $i = \text{argmax} g_i$
14         Calculate the confidence of rule $r$ as $cr$
15         if $cr \geq MinimumConfidence$ then
16             $t(x) = t(x) + 1$ for all $x \in X_{\text{minor},r}$
17             Accept rule $r$ and add it to the rule set $R$ with its confidence
18             break
19         else
20             if $|r.terms| \geq MaximumTerms$ then
21                 $t(x) = t(x) + 1$ for all $x \in X_{\text{minor},r}$
22                 break
23   until $t(x) > 0$ for all $x \in X_{\text{minor}}$
24   Add a default rule $r_d$ to $R$ with label of major class with the confidence
25   return $R$
```

Algorithm 1 shows our proposed rule induction approach. We follow the common framework of rule induction algorithm. We first divided the training set X into X_{major} and X_{minor} which contains major and minor class instances. Then we iteratively build rules for minor class instances (Line 4-23 in Algorithm 1). Finally, a default rule r_d is added for the rest instances (Line 24 in Algorithm 1). To better deal with imbalanced data, we make the following improvements on the traditional rule induction algorithm.

Soft-Removal for Covered Instances. In traditional rule induction algorithms, the covered instances will be removed to avoid overlapping (i.e., different rules of traditional algorithms will cover the same instances in training data). However, for most defect datasets, the number of minor class instances is quite small. If we simply remove covered instances, the rest instances will be much more imbalanced, and the distribution of minor class instances may be "broken up". Thus, we propose the *soft-removal mechanism* to reuse the information of covered instances. Suppose x_i is the i-th instance of minor class. We denote $t(x_i)$ to represent the frequency of instance x_i being covered or marked as noise, and we calculate the weight of instance x_i by

$$w(x_i) = \exp\left(-\eta \cdot t(x_i)\right), \#(2)$$

where η is the hyper-parameter to control the speed of weight decrement. Weight of x_i will decrease when they are covered by a new rule or marked as noise. If some instances have been covered by many rules, their weights will be approximate to 0, which indicates that using such weights to calculate the gain is almost equivalent to remove those instances directly. On the other hand, since the weights will never equal to 0, such instances can still be covered again even if it has already been covered by other rule(s). The covered minor class instances will still provide information and can be used for calculating the confidence to determine whether a rule stops to grow.

Improvement on FOIL Gain. When building rules, we need to select the best term (i.e., cover the most minor class instances with highest confidence) among several candidates. Thus, gains of each candidate term need to be calculated. An example of gain function for imbalanced data is Quinlan's FOIL gain [33]. It is defined as

$$G_{FOIL} = n_{np} \cdot \left(\log \frac{n_{np}}{n_{np} + n_{nn}} - \log \frac{n_p}{n_p + n_n}\right) \#(3)$$

where n_{np}, n_{nn} are the number of major class and minor class instances covered by the rule after adding term T to the rule, and n_p, n_n are the number of major class and minor class instances covered by the rule without term T respectively. It is effective on imbalanced data, as the better confidence (estimated by $\frac{n_{np}}{n_{np}+n_{nn}}$) and coverage of minor class instances (coefficient n_{np}) will lead to larger gain. Since we introduce soft-removal mechanism for minor class instances, we use sum of weights for instances of minor class $\sum_{x_i \in X_{\text{minor}}} w(x_i)$ instead of using

[4] In our implementation, we use similar discretization approach of C4.5. We first enumerate possible cutoffs for the values in a feature. Then, for each *cutoff* of the *feature*, there will be two candidate terms, which are "*feature <= cutoff*" and "*feature > cutoff*".

n_{np} as coefficient, to pay more attention on minor class instances. Further, the gain is also multiplied with $1/(n_{nn}+1)$ to decrease the importance of candidate terms with more major class instances covered. Finally, the improved gain function for candidate terms is defined as

$$G = \frac{1}{n_{nn}+1} \cdot \left(\sum_{x_i \in X_{\text{minor}}} w(x_i) \right) \cdot \left(\log \frac{n_{np}}{n_{np}+n_{nn}} - \log \frac{n_p}{n_p+n_n} \right). \#(4)$$

Tolerance of Lower Coverage. Traditional rule induction algorithms construct rules which maximize the accuracy for instances of both major and minor class. If the data is heavily imbalanced, minor class instances tend to be ignored. However, defect prediction models should be able to characterize past defects, and most defect instances (usually minor class instances) should be covered. Thus, inspired by Shehzad [67], we tolerate rules with lower confidence (of minor class instances). A rule will be accepted if its confidence is larger than *MinCoverage* (Line 15 in Algorithm 1), which also takes imbalance rate into consideration[5].

An illustrative example of soft-removal mechanism and tolerance of lower confidence in Fig 11. The third line is example of our proposed approach. All the three approaches start with a rule covering several defective instances. After removing the covered instances, traditional algorithms cannot cover more defective instances since there are too few defective instances. On the contrary, our proposed approach can cover the rest instances by covering two instances which has already been covered and tolerating two clean instances which will be misclassified as defective ones. Thus, our proposed approach can cover more defective instances compared with traditional rule-based models constructed on original data. Meanwhile, it can get simpler decision boundary compared with traditional models constructed on re-balanced data, which result in better generalization performance and smaller model complexity.

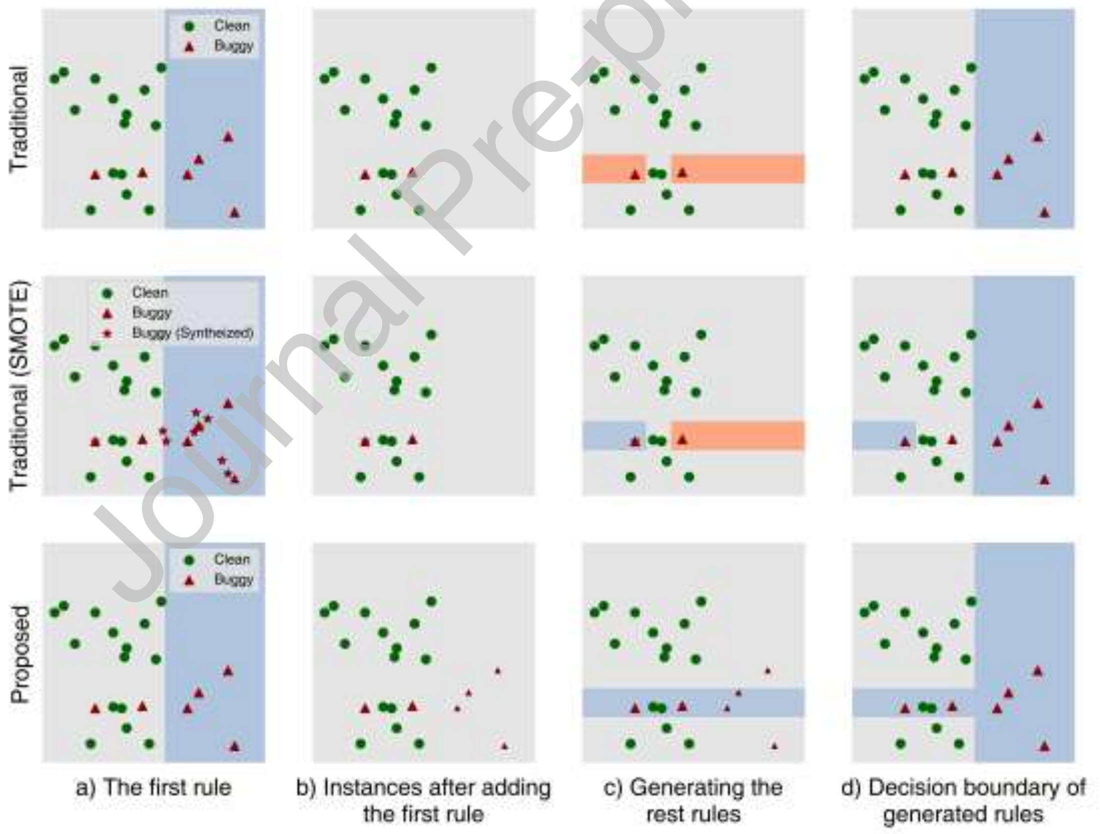


Fig 11. An illustrative example of improvements of our proposed approach. The blue region is the space covered by generated rule(s), while the red region is the ignored rules which only cover one defective instance.

Since we tolerate the overlapping and low confidence for decision rules, it may not be

[5] In our experiment the function *MinConfidence* is defined as $MinConfidence(x) = \exp\left\{-\frac{(x-0.5)^2}{2\sigma^2}\right\}$ empirically to calculate the minimum confidence for each rule, where $x \in [0,1]$ is the defective rate of the data.

practical to use the rules for prediction directly (i.e., the instance will be defective if it fulfills any of the rules which target label is "*defect*"). Instead, we compute average confidence of rules that the test instance fulfills (including the "*default rule*") as the probability for being defective, and then predict the label (defective or clean) with the optimal threshold[6]. The algorithm of predicting an instance is shown in Algorithm 2.

Algorithm 2 Predict an Instance

```
Input: Instance $x$, Ruleset $R$, and the Optimal Threshold $t_{optimal}$
Output: The label of instance $x$.
1  Let $SR$ be an empty set.
2  for each rule $r \in R$ do
3      if $x$ satisfies rule $r$ then
4          $SR = SR \cup \{r\}$
5  Compute score of the instance $x$ by $s(x) = \frac{1}{|SR|}\sum_{r \in SR} r.\mathrm{confidence}$
6  if $s(x) \geq t_{\mathrm{optimal}}$ then
7      return "defective"
8  else
9      return "clean"
```

5.3 Evaluation

5.3.1 Experiment Setup

We use the same settings described in Section 3. We use all the 47 datasets, and we mitigate the correlated metrics after bootstrap sampling. We adopt C4.5, RIPPER and ALPAC for comparison. All the interpretable models are using 4 group of training data (original data, oversampled data, under-sampled data, and data rebalanced by SMOTE), while our proposed rule-based model is only trained on original data.

Table 6 Confusion Matrix.

	Predicted Positive	**Predicted Negative**
Actual Positive	TP	FN
Actual Negative	FP	TN

Evaluation on Performance. We adopt *AUC*, *F1-Score and False Positive Rate* (*FPR*) for performance evaluation. As one of *threshold-independent* metrics, *AUC* is often used to measure the discriminatory power of different models [51, 52], and it is less impacted by imbalanced data [25, 27]. The larger AUC indicates the better performance. For *threshold-dependent* metrics, we adopt

$$F1\text{-}Score = 2 \cdot P \cdot R/(P+R), \#(5)$$

and

$$FPR = FP/(FP+TN), \#(6)$$

based on confusion matrix described in Table 6. *F1-Score* is a harmonic mean of precision $P = TP/(TP+FP)$ and recall $R = TP/(TP+FN)$. Larger *F1-Score* indicates better classification performance of defect prediction models. *FPR* is the proportion of clean instances which are misclassified as defective, and smaller *FPR* is expected by practitioners [9].

Evaluation on Interpretation. We evaluate the interpretation of our proposed approach based on two aspects concerned by practitioners. Firstly, the interpretable model should cover the defect instances well to provide the insight about past defects. Thus, we adopt *Coverage of Defects* (CD) to evaluate how well the models fit the past defects. Secondly, for rule-based and tree-based model, simpler rules will be better for interpretation. Thus, we also adopt *Number of Rules*, and *Average Length* for evaluation of interpretation, which are defined in Section 3.6.2.

Statistical Analysis. We use Scott-Knott Effect Size Difference (Scott-Knott ESD) test [39], an improved version based on Scott-Knott test [53], to statistically evaluate models with different resampling techniques applied. Scott-Knott test is a hierarchical cluster analysis which partition set of treatment means into statistically distinct groups at certain confidence level, and it has been popularly adopted in prior studies to rank the different models and resampling techniques [16, 42]. Scott-Knott ESD test improves Scott-Knott test by correcting the non-normal distribution and merging negligible effect size groups into one group. Two models have significant differences with nonnegligible effect size if they are in different Scott-Knott

[6] The optimal threshold is defined as $t_{optimal} = \mathrm{argmax}_t F(t)$, where t is all possible thresholds based on training data, and $F(t)$ is the target metric, e.g., *F1-Score* in our experiments.

ESD groups.

5.3.2 Results

The results are shown in Fig 12-14. Different groups are in different color, and model in upper groups are significantly better than those in lower groups. The adopted resampling technique (ROS, RUS, SMOTE for Oversampling, Under-sampling and SMOTE respectively) for compared rule-based models is in the parentheses (e.g., C4.5(ROS) means the C4.5 model constructed on resampled data with Random Over-Sampling). For different metrics, "↑" indicates "the larger the better" while "↓" indicates "the smaller the better". Medians for each model are marked in red line, and the dashed black line in each plot is the median value of our proposed method.

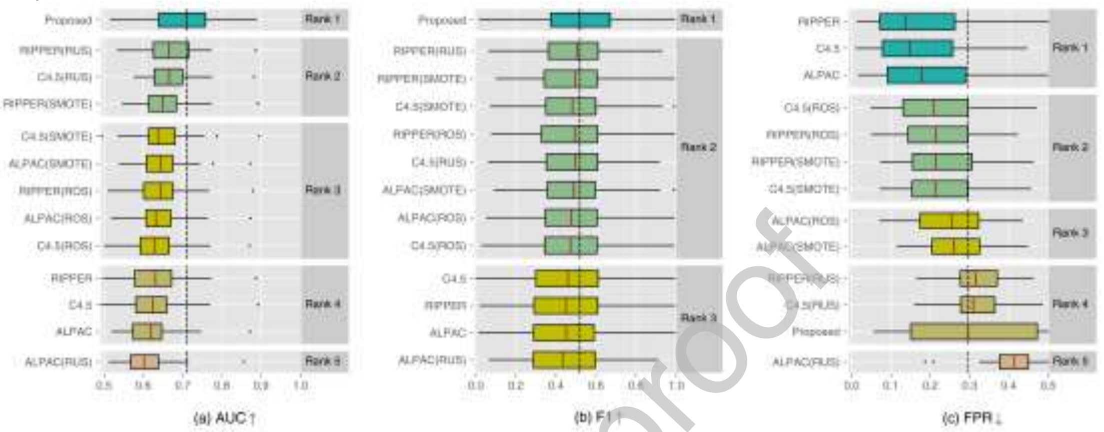


Fig 12. Results of performance metrics.

Performance. Fig 12 shows the performance of investigated models constructed with different resampling techniques. Overall, our proposed rule-based model significantly outperforms all interpretable models (constructed on original data and resampled data) on metric *AUC* and *F1-Score*. The median *AUC* and *F1-Score* of our proposed method are 0.704 and 0.523. Thus, we assume that our proposed rule-based model can get better performance without using any resampling techniques to preprocess the data, which will make the interpretation about past defects more trustable.

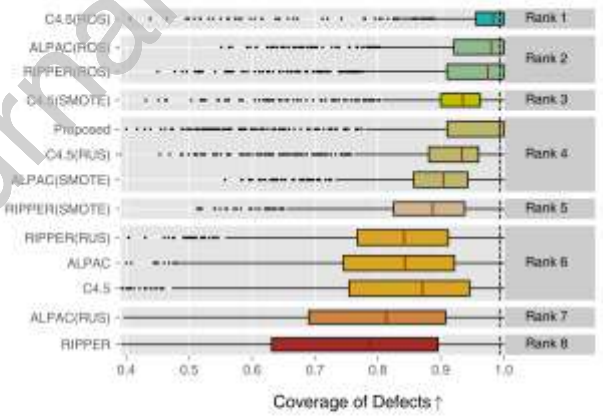


Fig 13. Results of coverage of defects.

Coverage of Defects. Fig 13 shows the results of coverage of defects for all the investigated models constructed on original and resampled data. For most datasets, our proposed model can cover more than 90 percent of defective instances, which is significantly better than all investigated models constructed on original (imbalanced) data. Further, the coverage of defects of our proposed model is comparable to rule-based models with popular resampling techniques. This indicates that our proposed model can capture and provide more knowledge and insights about past defects, compared with other rule-based models when they are constructed on original data.

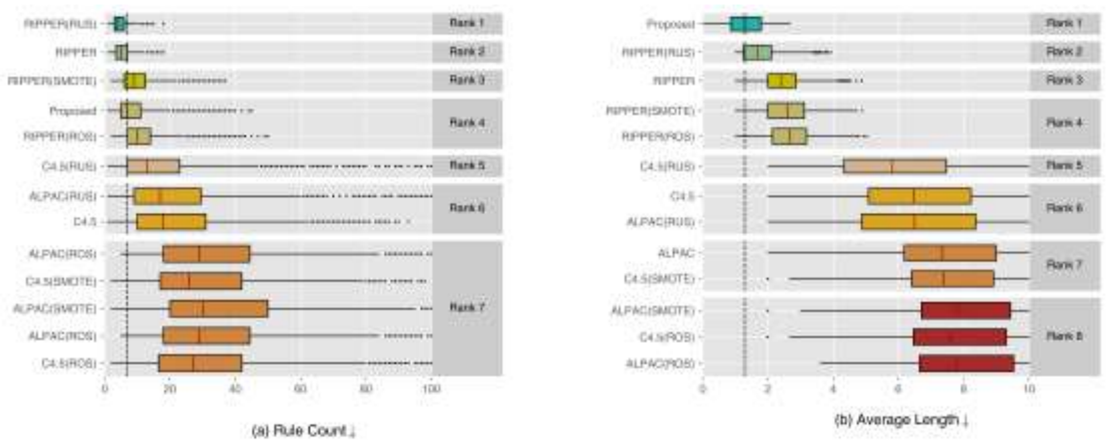


Fig 14. Results of model complexity.

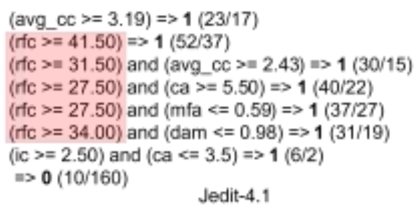
Model Complexity. Fig 14 shows the results of model complexity for all the investigated models constructed on original and resampled data. In general, our proposed model can generate small rule set with short rules. In Fig 14 (a), we find that the rule count of our proposed model is significantly better than all the tree-based models (C4.5 and ALPAC). In Fig 14 (b), we find that the rules generated by our proposed model is shorter than all the investigated models. The median of average length for our model is below 2, which is significantly better than other models. Besides, we notice that C4.5 and ALPAC generate trees which have more than 15 rules and achieve median depths more than 6, which implies that the tree-based models may be "*difficult to read especially when there are multiple parameters, and the trees get un-manageable*" [10]. Thus, we assume the generated rules of our proposed model is less complex than tree-based models and can be easy for human to understand like the state-of-the-art rule-based model RIPPER.

6 Discussion

6.1 Analysis about proposed approach

We first discuss *why our improvements are effective for imbalanced data*. When building single rules, the gain for candidate terms is calculated by (4). A candidate term will get higher gain if it can cover more minor class instances and less major class instances. Thus, more positive instances can be picked for each single rule. Additionally, more minor class instances could be correctly classified by tolerating few major class instances to be misclassified. This may reduce the risk of overfitting (on major class instances) which also results in shorter rules. With such improvements, our approach generates rules which can cover 80 percentage of defective modules on average and achieves median AUC larger than 0.7, which implies that our approach performs well for the explanations to be trusted. Besides, the model complexity of our proposed model is much smaller than decision trees. As it is shown in Fig 14 (b), the average length of rules is less than 2 on average, which means the explanations is simple enough for human to understand.

We show an example of rules generated by our proposed approach from project JEdit-4.1 in Fig 15. Such rules can provide insights and characteristics about the past defects. For example, in Fig 15 there are five rules related to attribute `rfc`, which indicates that there a strong relationship between the past defects and larger `rfc`.



```
(avg_cc >= 3.19) => 1 (23/17)
(rfc >= 41.50) => 1 (52/37)
(rfc >= 31.50) and (avg_cc >= 2.43) => 1 (30/15)
(rfc >= 27.50) and (ca >= 5.50) => 1 (40/22)
(rfc >= 27.50) and (mfa <= 0.59) => 1 (37/27)
(rfc >= 34.00) and (dam <= 0.98) => 1 (31/19)
(ic >= 2.50) and (ca <= 3.5) => 1 (6/2)
 => 0 (10/160)
                    Jedit-4.1
```

Fig 15. An example of rules generated by our proposed approach from Java project JEdit-4.1. The number of defective instances and clean instances covered by the rule is in the parenthesis.

We also discuss several *shortcomings* of our proposed model. In Fig 12 (c), our model gets higher *FPR* compared with traditional methods. Since we accept rules with lower confidence, it is obviously that some clean instances which are close to the decision boundary will be misclassified as defective ones. This may increase the effort when using the proposed model to allocate the QA resources for testing and inspection, but we assume it is acceptable since the main goal of constructing interpretable models is to deriving knowledge and insights from past defects. Besides, the computational cost of our model is higher than other methods due to soft-removal mechanism. Improvements on such shortcomings may be our future work.

6.2 Rethinking rule-based interpretable models in explainable software defect prediction

Rule-based (and tree-based) models play an important role in explainable software defect prediction. They are completely transparent models [15, 43-45], and they are also preferred by practitioners with high agreement on insightfulness and quality of explanations to understand the characteristics about past defects [10]. However, such models perform poorly since they cannot fit the data well, especially on the minor class instances (usually defective instances in defect prediction). In that case, information about defective modules may not be fully captured, and thus meaningful and useful knowledge about past defects may not be derived. Further, a model could not be trusted if they cannot achieve good operational performance [13, 26, 76]. Thus, we suggest that *rule-based models in future work should be able to fit the past defects well and achieve a good operational performance while maintaining the interpretability, to explain the past defects from a global perspective*.

Through the case study, we find that imbalanced data have great negative impact on interpretable models, which can be summarized in two aspects. Firstly, from the results of RQ3, we find that it is not feasible to construct interpretable models directly on original data, since the models poorly fit the data (especially on defective instances) and cannot achieve good operational performance. Secondly, from the results of RQ1 and RQ2, we also find that such negative impact of class imbalance cannot be easily mitigated by resampling the training data, as the interpretation will be harmed from both feature importance and model complexity. Thus, *it is necessary to reconsider the problem of imbalanced data and address the problem without using resampling techniques when constructing rule-based models for defect prediction.*

Further, the rule-based models should be simple and easy to understand. From practitioners' perspective, decision trees may be difficult to read when the trees are too large [10]. However, we find that trees generated by C4.5 are much more complex than expected with median length more than 5, while there is no significant difference on performance, compared with rule-based model RIPPER. Moreover, complex trees achieve worse performance compared with simpler rules generated by our proposed approach. Besides, Chen et al. [76] find that the simple tree-based model FFT outperforms several state-of-the-art defect prediction models. The reason may be that the "lumpy" software data can be divided into several separate regions, each with different properties [76]. Therefore, *models with simple rules may match such characteristics of SE data*, and we suggest that *interpretable models with simpler rules would be more preferred to improve both performance and interpretation for explainable software defect prediction*.

6.3 Threats to validity

We give discussion about threats which may impact on the results of our empirical study.

Scenario. In this study, we only examine the scenario of within-project defect prediction. There are many other scenarios (e.g., cross-project defect prediction [55] and just-in-time defect prediction [56]), and the impact of resampling techniques has not been investigated for these scenarios. It may not be the same as our findings in within-project defect prediction.

Datasets. In this study, we investigate 47 releases of open-source projects. Though they are widely used in prior defect prediction studies, our findings based on those datasets may not be able to be generalized to all software systems, especially for proprietary or commercial systems.

Resampling Techniques. In this study, we only investigate 3 popular resampling techniques, i.e., oversampling, under-sampling, and SMOTE. There are plenty of resampling techniques adopted in prior studies [57-59]. Though resampling could result in sample select bias [23, 24], there is no evidence about whether those novel resampling techniques will have great impact on interpretation. We think it is worth to be explored in future.

Models. For simplicity, we only choose C4.5, RIPPER and ALPAC as rule-based interpretable models in this study. There are more transparent and surrogate models proposed by prior machine learning studies, which could be potential applied to defect prediction. Results based on limited models may not be general for all interpretable defect prediction models.

Implementation. The implementation of resampling techniques is python package *imblearn*, and the implementation of models is *Weka* except ALPAC which is implemented by the author [48]. All the hyper-parameters are not optimized. This may introduce biases to results of our empirical study.

Evaluation. We adopt *AUC*, *F1-Score*, and *False Positive Rate* (*FPR*) to evaluate the performance of defect prediction models. Those metrics were widely used in prior defect prediction studies [1]. Still, such metrics are not able to fully evaluate the interpretable models. Recently, effort have been suggested to be considered as one of prediction performance [57]. Further, to evaluate the interpretation, a carefully-designed user study suggested by prior studies [10, 14] would be better.

7 Related Work

7.1 Defect Prediction on Imbalanced Data

Class imbalance is a nature for software defect data, and it has great impact on both performance and interpretation [25]. To improve the performance, resampling techniques are effective and widely adopted. For example, Kamei et al. [41] improve the performance of defective modules by employing oversampling and under-sampling techniques. Agrawal et al. [42] investigate the benefit of tuning the hyperparameters SMOTE from multi-performance criteria and suggest that pre-processing especially those handling the imbalanced data is necessary for defect prediction tasks. Bennin et al. [27] investigate the impact of resampling techniques on performance under different imbalance ratio through a large-scale experiment. Novel resampling techniques have also been proposed to improve the performance [58-60]. Besides, learning approaches which can deal with imbalanced data are also feasible [5, 7, 61]. Besides the problem itself, class imbalance problem has also been considered as key problem(s) to be tackled in other research topics of defect prediction. For example, since deep learning-based models can automatically extract features from source code and achieve better performance [3, 4], rebalancing source code-based data is also necessary [62]. Handling imbalanced data in other scenarios such as just-in-time (JIT) defect prediction [63] and cross-project defect prediction (CPDP) [64] are also discussed in prior studies.

So far, only Tantithamthavorn et al. [25] have investigated the impact of resampling techniques on interpretation. They find that resampling techniques have negative impact on interpretation, and they suggest that resampling should be avoided when the model is built for interpretation. Most of their investigated models are well performed but not interpretable. As a supplement for their work, we investigate the impact of resampling on interpretable models, and find the feasibility to construct interpretable models on original but imbalanced data directly with our proposed approach.

7.2 Interpretation of Software Defect Prediction

Table 7 Prior studies towards explainable software defect prediction. Note that one paper may adopt several interpretation techniques.

Category	Studies	Interpretation Technique
Transparent model construction	Mori et al. [16]	Improved Naïve Bayes
	Yadav et al. [65]	Fuzzy Rules
	Diamantopoulos et al. [18]	Rule Set
	Monden et al. [19]	Association Rules
	Dejaeger et al. [20]	Bayesian Network
	Singh et al. [21]	Fuzzy Rules
	Chen et al. [76]	Fast-and-Frugal Tree
	Singh et al. [22]	Rule Set
Global explanation	Moeyersoms et al. [17]	Decision Tree
Local explanation	Rajapaksha et al. [14]	Association Rules
	Jiarpakdee et al. [49]	LIME and BreakDown
	Pornprasit et al. [75]	RuleFit
Model inspection	Mori et al. [16]	PDP
	Esteves et al. [54]	SHAP
	Rajbahadur et al. [13]	VarImp, SHAP

Jiarpakdee et al. [10]	ANOVA, PDP, VarImp

Interpretation of software defect prediction is concerned and expected from practitioners [9-11] to make predictions more *explainable and actionable* [77, 78]. According to Jiarpakdee et al. [10], there are two main goals for interpretation of defect prediction models which are 1) learning from the past defects, and 2) explaining the current predictions. We summarize the studies which aims to make defect prediction explainable in Table 7 categorized by XAI techniques suggested by Guidotti et al. [15]. Besides, there are studies focused on factors and techniques which may have potential impact on the interpretation of defect prediction models [2, 25, 33, 34]. Further, interpretation of larger grain defect prediction model (e.g., file-level or commit-level) is also be used for giving predictions in a fine-grained level (e.g., line-level) based on features from source code [73, 74, 79].

Recent studies adopt local explanation techniques [14, 49, 75] and model inspection techniques [16, 54] to explain individual predictions of powerful black-box models, but they can only tell practitioners why a certain file (or commit) is buggy and what should be done to reduce the risk of such file (or commit). Model inspection techniques such as ANOVA and VarImp [10, 13] can help people understand the characteristics of past defects from historical data, but such techniques cannot provide more detailed information (e.g., direction of the relationship of each feature) about past defects. Interpretable models such as rule-based and tree-based transparent (or surrogate) models [17-22] can also be used to provide information about past defects, but none of them consider the impact of imbalanced data and resampling techniques. Thus, we investigate the impact of imbalanced data and resampling techniques on rule-based (and tree-based) models and propose a novel rule-based approach which can achieve better performance and interpretation.

7.3 Interpretable Machine Learning

The interpretability of machine learning has been more concerned by researchers during recent years. As it is described in Fig 1, there are 4 basic ideas for explainable machine learning, including model explanation, local explanation (also called outcome explanation), model inspection and transparent model construction. Prior studies on transparent model construction mainly improve the performance based on rule induction [50, 66-69]. Global model explanation studies generate extra artificial data points to build a more accurate and interpretable white-box model [48, 70]. Local explanation techniques which focus on the explanation of a single prediction such as LIME [43] and Anchors [71] have been proposed. Besides, model inspection techniques such as Partial Dependence Plot (PDP) [72] are also practical approaches for model interpretation.

8. Conclusion

In this paper, we investigate the impact of class imbalance problem, and its popular solution, resampling techniques on interpretable models, through an empirical study on 47 publicly available datasets and 3 rule-based interpretable defect prediction models. We confirm that concept drift [10, 11, 23] will be introduced by resampling, and we observe that the model complexity will also be impacted. Further, we find that constructing interpretable models on original data which is suggested by prior studies [11, 25] may not be feasible when the data is heavily imbalanced for defective instances. Thus, we suggest that rule-based interpretable models should be able to deal with imbalanced data without rebalanced data, which can fit the data well and achieve good operational performance.

To deal with imbalanced data without resampling, we also proposed an improved rule-based approach. Experiments show its effectiveness on original but imbalanced data, by increasing median AUC and F1 to 0.704 and 0.523, which outperforms other rule-based interpretable models with rebalanced data. Moreover, the interpretation is also improved by covering more defect instances and building simpler rules. We assume it would be feasible if one wants to achieve a balance between performance and interpretability in defect prediction practice.

Acknowledgement

This work was partly supported by National Natural Science Foundation of China (No.

62077029), Future Network Scientific Research Fund Project (No. FNSRFP-2021-YB-32), the Open Project Fund of Key Laboratory of Safety-Critical Software, Ministry of Industry and Information Technology (No. NJ2020022), CCF-Huawei Populus Grove Fund (No. CCF-HuaweiFM202209), and the Graduate Science Research Innovation Program of Jiangsu Normal University (No. 2021XKT1394).

References

[1] S. Pandey, R. Mishra, A. Tripathi, Machine learning based methods for software fault prediction: a survey, Expert Syst. Appl. 172 (2021), https://doi.org/10.1016/j.eswa.2021.114595.

[2] C. Tantithamthavorn, S. McIntosh, A.E. Hassan, A. Ihara, K. Matsumoto, The impact of mislabeling on the performance and interpretation of defect prediction models, Proceedings of the 2015 ACM/IEEE 37th International Conference on Software Engineering (ICSE), 2015, pp. 812-823,

[3] S. Wang, T. Liu, L. Tan, Automatically learning semantic features for defect prediction, in: Proceedings of the 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), 2016, pp. 297-308,

[4] J. Li, P. He, J. Zhu, M.R. Lyu, Software defect prediction via convolutional neural network, in: Proceedings of 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), 2017, pp. 318-328, https://doi.org/10.1109/QRS.2017.42.

[5] S. Wang, X. Yao, Using class imbalance learning for software defect prediction, IEEE Trans. Reliab. 62 (2) (2013) 434-443.

[6] C. Seiffert, T.M. Khoshgoftaar, J. Van Hulse, A. Folleco, An empirical study of the classification performance of learners on imbalanced and noisy software quality data, Inf. Sci. 259 (2014) 571-595,

[7] X. Xia, D. Lo, E. Shihab, X. Wang, X. Yang, ELBlocker: Predicting blocking bugs with ensemble imbalance learning, Inf. Softw. Technol. 61 (2015) 93-106,

[8] S. Kim, H. Zhang, R. Wu, L. Gong, Dealing with noise in defect prediction, in: Proceedings of 2011 ACM/IEEE 33rd International Conference on Software Engineering (ICSE), 2011, pp. 481-490.

[9] Z. Wan, X. Xia, A.E. Hassan, D. Lo, J. Yin, X. Yang, Perceptions, expectations, and challenges in defect prediction, IEEE Trans. Softw. Eng. 46(11) (2020) 1241-1266,

[10] J. Jiarpakdee, C. Tantithamthavorn, J. Grundy, Practitioners' perceptions of the goals and visual explanations of defect prediction models, in: Proceedings of 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), 2021, pp. 432-443,

[11] C. Tantithamthavorn, A.E. Hassan, An experience report on defect modelling in practice: pitfalls and challenges, in: Proceedings of the IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), 2018, pp. 286-295.

[12] N. C. Shrikanth, T. Menzies, Assessing practitioner beliefs about software defect prediction, in: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), 2020, pp. 182-190, https://doi.org/10.1145/3377813.3381367.

[13] G.K. Rajbahadur, S. Wang, G. Ansaldi, Y. Kamei, A.E. Hassan, The impact of feature importance methods on the interpretation of defect classifiers, IEEE Trans. Softw. Eng., 2021, https://doi.org/10.1109/TSE.2021.3056941.

[14] D. Rajapaksha, C. Tantithamthavorn, C. Bergmeir, W. Buntine, J. Jiarpakdee, J. Grundy, SQAPlanner: Generating data-informed software quality improvement plans," IEEE Trans. Softw. Eng., 2021, https://doi.org/10.1109/TSE.2021.3070559.

[15] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, D. Pedreschi, A survey of methods for explaining black box models, ACM Comput. Surv. 51(5) (2018) 1-42,

[16] T. Mori, N. Uchihira, Balancing the trade-off between accuracy and interpretability in software defect prediction, Empir. Softw. Eng. 24(2) (2019) 779-825,

[17] J. Moeyersoms, E.J. Fortuny, K. Dejaeger, B. Baesens, D. Martens, Comprehensible software fault and effort prediction: A data mining approach, J. Syst. Software 100 (2015) 80-90,

[18] T. Diamantopoulos, A. Symeonidis, Towards interpretable defect-prone component analysis using genetic fuzzy systems, in: Proceeding of 2015 IEEE/ACM International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), 2015, pp. 32-38,

[19] A. Monden, J. Keung, S. Morisaki, Y. Kamei, K. Matsumoto, A heuristic rule reduction approach to software fault-proneness prediction, in: Proceedings of 2012 19th Asia-Pacific Software Engineering Conference (APSEC), 2012, pp. 838-847,

[20] K. Dejaeger, T. Verbraken, B. Baesens Toward comprehensible software fault prediction models using Bayesian network classifiers, IEEE Trans. Softw. Eng. 39 (2) (2013) 237-257,

[21] P. Singh, N.R. Pal, S. Verma, O.P. Vyas, Fuzzy rule-based approach for software fault prediction, IEEE Trans. Syst. Man. Cybern: Systems. 47 (5) (2017) 826-837,

[22] P. Singh, S. Verma, ACO based comprehensive model for software fault prediction, Int. J. Knowl-Based. Intell. Eng. Sys. 24 (1) (2020) 63-71, http://doi.org/10.3233/KES-200029.

[23] B. Turhan, On the dataset shift problem in software engineering prediction models, Empir. Softw. Eng. 17(2) (2011) 62-74,

[24] A.J. Storkey, When training and test sets are different: characterizing learning transfer, in Dataset shift in machine learning, Chapter 1. Cambridge, MA, The MIT Press, 3-28, 2009.

[25] C. Tantithamthavorn, A.E. Hassan, K. Matsumoto, The impact of class rebalancing techniques on the performance and interpretation of defect prediction models, IEEE Trans. Softw. Eng. 46(11) (2020) 1200-1219,

[26] Z.C. Lipton, The mythos of model interpretability, in: Proceedings of 2016 ICML workshop on human interpretability in machine learning (WHI), 2016,

[27] K.E. Bennin, J. Keung, and A. Monden, On the relative value of data resampling approaches for software defect prediction, Empir. Softw. Eng. 24 (2019), 602-636,

[28] M. D'Ambros, M. Lanza, R. Robbes, Evaluating defect prediction approaches: a benchmark and an extensive

comparison, Empir. Soft. Eng. 17 (2012) 531-577,
[29] M. Jureczko, L. Madeyski, Towards identifying software project clusters with regard to defect prediction, In: Proceedings of the 6th International Conference on Predictive Models in Software Engineering (PROMISE), 2010, pp. 1-10.
[30] R. Wu, H. Zhang, S. Kim, S. Cheung, ReLink: recovering links between bugs and changes, in: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE), 2011, pp. 15-25.
[31] M. Shepperd, Q. Song, Z. Sun, Data quality: some comments on the nasa software defect datasets, IEEE Trans. Softw. Eng. 39(9) (2013) 1208-1215,
[32] J. Petric, D. Bowes, T. Hall, B. Christianson, N. Baddoo, The jinx on the nasa software defect data sets, in: Proceedings of the International Conference on Evaluation and Assessment in Software Engineering (EASE), 2016, pp. 13-17.
[33] J. Jiapakdee, C. Tantithamthavorn, C. Treude, The impact of automated feature selection techniques on the interpretation of defect models, Empir. Softw. Eng. 25 (2020) 3590-3638.
[34] J. Jiarpakdee, C. Tantithamthavorn, A.E. Hassan, The impact of correlated metrics on the interpretation of defect models, IEEE Trans. Softw. Eng. 47 (2) (2021) 20-331.
[35] H.C. Kraemer, G.A. Morgan, N.L. Leech, J.A. Gliner, J.J. Vaske, and R.J. Harmon, Measures of clinic significance, J. Amer. Acad. Child Adolescent Psychiatry, 42 (12) (2003) 1524-1529.
[36] W. S. Sarle, The varclus procedure, in: SAS/STAT User's Guide, SAS Institute, Inc, 4th Edition, 1990.
[37] B. Efron, Estimating the error rate of a prediction rule: Improvement on cross-validation, J. Amer. Statistical Assoc, 78 (382) (1983) 316-331.
[38] B. Efron, R.J. Tibshirani, An Introduction to the Bootstrap, MA, USA, Springer, 1993.
[39] C. Tantithamthavorn, S. McIntosh, A.E. Hassan, and K. Matsumoto, An empirical comparison of model validation techniques for defect prediction models, IEEE Trans. Softw. Eng. 43 (1) (2017) 1-18.
[40] N.V. Chawla, K.W. Bowyer, L. O. Hall, and W.P. Kegelmeyer, SMOTE: Synthetic minority over-sampling technique, J. Artif. Intell. Res. 16 (2002) 321-357.
[41] Y. Kamei, A. Monden, S. Matsumoto, T. Kakimoto, K. Matsumoto, The effects of over and under sampling on fault prone module detection, in: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement (ESEM), 2007, pp. 196-204.
[42] A. Agrawal, T. Menzies, Is 'better data' better than 'better data miners'? On the benefits of tuning SMOTE for defect prediction, in: Proceedings of 2018 ACM/IEEE 40th International Conference on Software Engineering (ICSE), 2018, pp. 1050-1061.
[43] M.T. Ribeiro, S. Singh, C. Guestrin, Why should I trust you? Explaining the predictions of any classifier, in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2016, pp.1135-1144,
[44] J. Huysmans, K. Dejaeger, C. Mues, J. Vanthienen, and B. Baesens, An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models, Decis. Supp. Syst. 51 (1) (2011) 141-154,
[45] A.A. Freitas, Comprehensible classification models: A position paper, ACM SIGKDD Explor. Newslett. 15 (1) (2014) 1-10,
[46] J.R. Quinlan, C4.5: Programs for Machine Learning, San Mateo, CA: Morgan Kaufmann, 1993.
[47] W. Cohen, Fast effective rule induction, in: Proceedings of the 12th International Conference on Machine Learning (ICML), 1995, pp. 115-123,
[48] E.J. Fortuny, D. Martens, Active learning-based pedagogical rule extraction, IEEE Trans. Neural Netw. Learn. Syst. 26 (11) (2015) 2664-2677,
[49] J. Jiarpakdee, C. Tantithamthavorn, H.K. Dam, J. Grundy, An empirical study of model-agnostic techniques for defect prediction models, IEEE Trans. Softw. Eng., http://dx.doi.org/10.1109/TSE.2020.2982385.
[50] L. Dong, X. Ye, G. Yang, Two-stage rule extraction method based on tree ensemble model for interpretable loan evaluation, Inf. Sci. 573 (2021) 46-64,
[51] T. Fawcett, An introduction to ROC analysis, Pattern Recognit. Lett. 27 (2005) 861-874.
[52] J. Huang, C.X. Ling, Using AUC and accuracy in evaluating learning algorithms, IEEE Trans. Knowl. Data Eng. 17 (3) (2005) 299-310,
[53] A.J. Scott, M. Knott, A cluster analysis method for grouping means in the analysis of variance, Biometrics, 30 (3) (1974) 507-512,
[54] G. Esteves, E. Figueiredo, A. Veloso, M. Viggiato, N. Ziviani, Understanding machine learning software defect predictions. Autom. Softw. Eng. 27 (2020) 369–392, https://doi.org/10.1007/s10515-020-00277-4.
[55] Y. Zhou, Y. Yang, H. Lu, L. Chen, Y. Li, Y. Zhao, J. Qian, B. Xu, How far we have progressed in the journey? An examination of cross-project defect prediction. ACM Trans. Softw. Eng. Methodol, 27 (1) (2018) 1-58,
[56] Q. Huang, X. Xia, D. Lo, Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction, Empir. Softw. Eng. 24 (2019) 2823-2862,
[57] T. Mende, R. Koschke, Effort-aware defect prediction models, in Proceedings of 2010 14th European conference on software maintenance and reengineering (CSMR), pp. 107–116.
[58] L. Gong, S. Jiang, L. Bo, L. Jiang, J. Qian, A novel class-imbalance learning approach for both within-project and cross-project defect prediction, IEEE Trans. Reliab. 69 (11) (2020) 40-54,
[59] K.E. Bennin, J. Keung, P. Phannachitta, A. Monden, and S. Mensah, MAHAKIL: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction, IEEE Trans. Softw. Eng. 44 (6) (2017) 534-550.
[60] S. Feng, J. Keung, X. Yu, Y. Xiao, K.E. Bennin, M.A. Kabir, and M. Zhang, "COSTE: Complexity-based Over-Sampling TEchniques to alleviate the class imbalance problem in software defect prediction," Inf. Softw. Technol. 129 (2021), http://dx.doi.org/10.1016/j.infsof.2020.106432.
[61] M. Liu, L. Miao and D. Zhang, Two-stage cost-sensitive learning for software defect prediction, IEEE Trans. Reliab. 63 (2) (2014) 676-686,
[62] R. Yedida, T. Menzies, On the value of oversampling for deep learning in software defect prediction, IEEE Trans. Softw. Eng. (2021), http://dx.doi.org/10.1109/TSE.2021.3079841.
[63] G.G. Cabral, L.L. Minku, E. Shihab and S. Mujahid, Class imbalance evolution and verification latency in just-in-time software defect prediction, in: Proceedings of 2019 IEEE/ACM 41st International Conference on

Software Engineering (ICSE), 2019, pp. 666-676.

[64] X. Jing, F. Wu, X. Dong, B. Xu, An improved SDA-based defect prediction framework for both within-project and cross-project class-imbalance problems," IEEE Trans. Softw. Eng. 43 (4) (2017) 321-339,

[65] H.B. Yadav, D.K. Yadav, A fuzzy logic-based approach for phase-wise software defects prediction using software metrics, Inf. Softw. Technol. 63 (2015) 44-57, 2015,

[66] M. Hudec, E. Minarikova, R. Mesiar, A. Saranti, A. Holzinger, Classification by ordinal sums of conjunctive and disjunctive functions for explainable AI and interpretable machine learning solutions, Knowl. Based Syst. (2021), http://dx.doi.org/10.1016/j.knosys.2021.106916.

[67] K. Shehzad, Simple hybrid and incremental post-pruning techniques for rule induction, IEEE Trans. Knowl. Data Eng. 25 (2) (2013) 476-480,

[68] H. Liu, M. Cocea, Induction of classification rules by gini-index based rule generation, Inf. Sci. 436 (2018) 227-246,

[69] H. Liu, S. Chen, M. Cocea, Heuristic target class selection for advancing performance of coverage-based rule learning, Inf. Sci. 479 (2019) 164-179,

[70] A. Saadallah, K. Morik, Active sampling for learning interpretable surrogate machine learning models, in: Proceedings of 2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA), 2020, pp. 264-272,

[71] M.T. Ribeiro, S. Singh, C. Guestrin, Anchors: High-precision model-agnostic explanations, in: Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018, pp.1527-1535,

[72] J.H. Friedman, Greedy function approximation: A gradient boosting machine, Ann. Stat. (2001) 1189-1232,

[73] C. Pornprasit, C. Tantithamthavorn, DeepLineDP: Towards a Deep Learning Approach for Line-Level Defect Prediction, IEEE Trans. Softw. Eng. (2022), http://dx.doi.org/10.1109/TSE.2022.3144348.

[74] C. Pornprasit, C. Tantithamthavorn, JITLine: A Simpler, Better, Faster, Finer-grained Just-In-Time Defect Prediction, in: Proceedings of 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), 2021, pp. 369-379,

[75] C. Pornprasit, C. Tantithamthavorn, J. Jiarpakdee, M. Fu, P. Thongtanunam, PyExplainer: Explaining the Predictions of Just-In-Time Defect Models, in Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2021, pp. 407-418,

[76] D. Chen, W. Fu, R. Krishna, T. Menzies, Applications of psychological science for actionable analytics, in: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), 2018, pp. 456-467.

[77] C. Tantithamthavorn, J. Jiarpakdee, J. Grundy, Actionable Analytics: Stop Telling Me What It Is; Please Tell Me What To Do, IEEE Softw. 38 (4) (2021) 115-120,

[78] C. Tantithamthavorn, J. Jiarpakdee, J. Grundy, Explainable AI for Software Engineering, arXiv preprint, arXiv:2012.01614v1.

[79] S. Wattanakriengkrai, P. Thongtanunam, C. Tantithamthavorn, H. Hata, K. Matsumoto, Predicting Defective Lines Using a Model-Agnostic Technique, IEEE Trans. Softw. Eng., 48 (5) (2022) 1480-1496.

[80] N. Cliff, Ordinal Methods for Behavioral Data Analysis, London, U.K.: Psychology Press, 2014.

[81] T. Zimmermann, R. Premraj, A. Zeller, Predicting defects for eclipse, in: Proceedings of Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007), 2007, pp. 9-20.

[82] T. Menzies, C. Pape, R. Krishna, and M. Rees-Jones, The promise repository of empirical software engineering data (Online), 2015, http://openscience.us/repo.

Yuxiang Gao: Conceptualization, Methodology, Software, Experiment, Writing; Yi Zhu:Writing – Review & Editing, Supervision; Yu Zhao: Conceptualization, Writing – Review & Editing.

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐ The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: