This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2876537, IEEE Transactions on Software Engineering

The Impact of Class Rebalancing Techniques on the Performance and Interpretation of Defect Prediction Models

Chakkrit Tantithamthavorn, *Member, IEEE,* Ahmed E. Hassan, *Senior Member, IEEE,* and Kenichi Matsumoto, *Senior Member, IEEE*

Abstract—Defect models that are trained on class imbalanced datasets (i.e., the proportion of defective and clean modules is not equally represented) are highly susceptible to produce inaccurate prediction models. Prior research compares the impact of class rebalancing techniques on the performance of defect models but arrives at contradictory conclusions due to the use of different choice of datasets, classification techniques, and performance measures. Such contradictory conclusions make it hard to derive practical guidelines for whether class rebalancing techniques should be applied in the context of defect models. In this paper, we investigate the impact of class rebalancing techniques on performance measures and the interpretation of defect models. We also investigate the experimental settings in which class rebalancing techniques are beneficial for defect models. Through a case study of 101 datasets that span across proprietary and open-source systems, we conclude that the impact of class rebalancing techniques on the performance of defect prediction models depends on the used performance measure and the used classification techniques. We observe that the optimized SMOTE technique and the under-sampling technique are beneficial when quality assurance teams wish to increase AUC and Recall, respectively, but they should be avoided when deriving knowledge and understandings from defect models.

Index Terms—Software quality assurance, software defect prediction, class rebalancing techniques, experimental design, empirical investigation.

1 Introduction

Defect prediction models play a critical role in the prioritization of SQA effort. Defect prediction models are trained using historical data to identify defect-prone software modules. From an SQA perspective, defect prediction models serve two main purposes. First, defect prediction models can be used to predict modules that are likely to be defect-prone in the future [2, 14, 25, 40, 55–57, 65, 99]. SQA teams can use defect prediction models in a prediction setting to effectively allocate their limited resources to the modules that are most likely to be defective. Second, defect prediction models can be used to understand the impact of various software metrics on the defect-proneness of a module [8, 49, 54, 55, 76, 77]. For example, one builds a prediction model using a code complexity metric with an assumption that more complex code shares an increasing relationship with defect-proneness. If the model shows that code complexity is the most important metric (i.e, top-rank metric). Such insights that are derived from defect prediction models can help software teams avoid past pitfalls that are associated with defective modules (e.g., developers should initiate a quality improvement plan that would carefully examine more complex code in an effort to avoid defects in future releases).

The performance and interpretation of a defect prediction model depend heavily on the data on which it was trained. Prior work raised concerns regarding defect prediction models that are trained on *imbalanced datasets* (i.e., datasets where the proportion of defective and clean modules is not equally represented). Such models are highly susceptible to producing inaccurate prediction results [27]. Indeed, when training a defect prediction model from an imbalanced dataset, traditional classification techniques often fail to accurately identify the minority class (i.e., defective modules).

To mitigate the risk of imbalanced datasets, prior studies apply *class rebalancing techniques* (i.e., techniques for rebalancing the proportion of defective and clean modules of the training corpus). Such techniques aim to produce an equal representation of two classes of software modules (i.e., defective and clean modules) prior to constructing a defect prediction model. Plenty of prior studies have shown a performance improvement when applying class rebalancing techniques in the machine learning area. For example, Chawla [9] and Seiffert *et al.* [73] show that the AUC performance can be substantially improved by up to 40% when applying class rebalancing techniques.

Recent defect prediction studies have compared the impact of class rebalancing techniques on the performance of defect prediction models [36, 48, 66, 68, 71, 81,

- *C. Tantithamthavorn is with the Faculty of Information Technology, Monash University, VIC, Australia. E-mail: chakkrit.tantithamthavorn@monash.edu.*
- *A. E. Hassan is with the School of Computing, Queen's University, Canada. E-mail: ahmed@cs.queensu.ca.*
- *K. Matsumoto is with the Graduate School of Information Science, Nara Institute of Science and Technology, Japan. E-mail: matumoto@is.naist.jp.*

Manuscript received August 1, 2017; revised August 12, 2018.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2876537, IEEE Transactions on Software Engineering

93]. For example, Kamei *et al.* [36] show the performance improvement of class rebalancing techniques on 2 defect datasets of proprietary systems. Recently, Malhotra *et al.* [48] show the impact of class rebalancing techniques on 6 defect datasets of open-source systems. In contrast, Riquelme *et al.* [66] argue that class rebalancing techniques have little impact on the performance of defect prediction models when they are trained on 4 datasets of NASA systems. Such contradictory conclusions make it hard to derive practical guidelines about whether class rebalancing techniques should be applied in the context of defect prediction models. Since prior work focuses on the different choice of datasets, classification techniques, and performance measures, it is likely that class rebalancing techniques may be useful for some specific contexts of defect prediction models.

Moreover, Turhan [92] points out that applying class rebalancing techniques may lead to bias in learned concepts (i.e., *concept drift*) — the resampled training dataset is not representative of the original dataset. Indeed, *concept drift* appears when the class distributions of the training and testing datasets are different. Thus, class rebalancing techniques may impact the interpretation of defect prediction models.

In this paper, we set out to investigate the impact of 5 popularly-used class rebalancing techniques (i.e., over-sampling, under-sampling, Default SMOTE, Optimized SMOTE, and ROSE techniques) on the performance and interpretation of defect prediction models. We train our defect prediction models using 7 commonly-used classification techniques, i.e., random forest (RF), logistic regression (LR), naive bayes (NB), neural network (AVNNet), C5.0 Boosting (C5.0), extreme gradient boosting (xGBTree), and gradient boosting method (GBM). We evaluate the performance of defect prediction models using 10 commonly-used performance measures, i.e., 3 threshold-independent (e.g., AUC) and 7 threshold-dependent (e.g., Precision, Recall, F-Measure) performance measures.

To better understand the impact of class rebalancing techniques on defect prediction models, we construct statistical models to study the relationship between the experimental factors (e.g., defective ratios, classification techniques) and the performance and interpretation of defect prediction models. Through a large-scale empirical study of 101 publicly-available defect datasets that span across open-source and proprietary systems that are collected from 5 different corpus, we record our observations with respect to 3 dimensions:

(1) The Nature of Imbalanced Defect Datasets
As little as 8% of defect datasets have a defective ratio between 45%-55%, suggesting that class imbalance is prominent in defect datasets, likely affecting the performance and interpretation of defect prediction models.

(2) Model Performance
Performance Analysis. The AUC measure is less impacted by the under-sampling, over-sampling, and default SMOTE techniques for defect prediction models. On the other hand, the AUC measure is substantially improved when the k SMOTE parameter is optimized, suggesting that future studies must optimized the SMOTE parameters.
Experimental Factors Analysis. The impact of the under-sampling, over-sampling, default SMOTE and optimized SMOTE techniques on the performance of defect prediction models depends on experimental settings. Defect prediction models yield the largest AUC improvement when applying the optimized SMOTE technique and the largest Recall improvement when applying the under-sampling technique.

(3) Model Interpretation
Interpretation Analysis. Regardless of class rebalancing techniques, the learned concepts are shifted (i.e., biasing the interpretation of defect prediction models). We find that as little as 23%-34%, 55%-62%, and 68%-71% of the top variables in the top importance rank of the re-balanced models appear in the top importance rank of the baseline models for the neural network, logistic regression, and random forest classifiers, respectively.
Experimental Factors Analysis. The impact of class rebalancing techniques on the interpretation of defect prediction models relies heavily on the used classification techniques, suggesting that researchers and practitioners should avoid such rebalancing when deriving knowledge and understandings from defect prediction models.

Our results lead us to conclude that the impact of class rebalancing techniques on the performance of defect prediction models depends on the used performance measure and the used classification techniques. While the commonly-used class rebalancing techniques (except the optimized SMOTE technique) substantially improve the Recall measure and decrease the Precision measure, they have little impact on the AUC measure. On the other hand, the commonly-used class rebalancing techniques negatively impact the interpretation of defect prediction models—i.e., we find that class rebalancing techniques shift the learned concepts to the interpretation of defect prediction models.

Based on our findings, we recommend that the optimized SMOTE technique and the under-sampling technique are beneficial when quality assurance teams wish to increase the ability to classify defective modules (i.e., AUC) and the completeness of identifying software defects (i.e., Recall), respectively, but they should be avoided when deriving knowledge and understandings from defect prediction models.

1.1 Novelty Statements

This paper presents the first empirical study to investigate (1) the impact of class rebalancing on defect prediction models using the largest number of commonly-used defect datasets (i.e., 101 defect datasets)—prior studies

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2876537, IEEE Transactions on Software Engineering

focus on less than 10 defect datasets; (2) the impact of class rebalancing techniques on the interpretation of defect prediction models; and (3) the experimental design settings where class rebalancing yields the largest benefits for defect prediction models.

1.2 Contributions

The contributions of our paper are as follows:

1) An empirical demonstration of the nature of class imbalance in 101 publicly-available defect datasets.
2) An empirical investigation of the impact of class rebalancing techniques on 10 commonly-used threshold-dependent and threshold-independent performance measures.
3) An empirical investigation of the impact of class rebalancing techniques on the interpretation of defect prediction models.
4) An in-depth examination of the impact of experimental factors (including class rebalancing techniques) on the performance and interpretation of defect prediction models.

1.3 Paper organization

The remainder of this paper is organized as follows. Section 2 illustrates the nature of class imbalance in defect datasets. Section 3 introduces class rebalancing techniques. Section 4 positions this paper with respect to the related work. Section 5 discusses the design of our case study, while Section 6 presents our results with respect to our two research questions. Section 7 revisits our RQ2 and RQ3 analysis for the optimized SMOTE technique. Section 8 offers practical guidelines for practitioners and researchers. Section 9 discusses the threats to the validity of our study. Finally, Section 10 draws conclusions.

2 The Nature of Imbalanced Defect Datasets

Motivation. Class imbalance refers to a classification problem where the classes (i.e., the proportion of defective and clean modules) are not represented equally. However, little is known about the nature of class imbalance in defect prediction datasets. Thus, we set out to investigate the following research question.

(RQ1) How imbalanced are defect prediction datasets?

Approach. In order to assess whether class imbalance is prominent in defect prediction studies, we analyze the defective ratio of 101 publicly-available defect datasets that have been popularly studied in prior defect prediction research. 76 datasets are downloaded from the Tera-PROMISE repository [51], 12 clean NASA datasets are provided by Shepperd *et al.* [74], 5 datasets are provided by Kim *et al.* [39] and Wu *et al.* [94], 5 datasets are provided by D'Ambros *et al.* [14, 15], and 3 datasets are provided by Zimmermann *et al.* [99]. Figure 1 shows a histogram of the defective ratios of the 101 defect datasets.

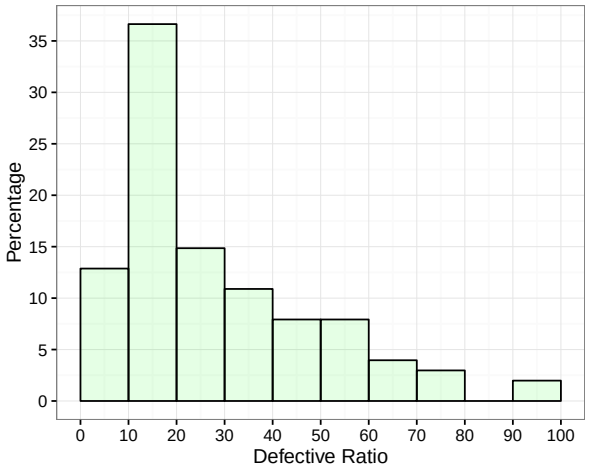


Fig. 1: A histogram of the defective ratios of the 101 publicly-available defect datasets.

Results. **64% of the defect datasets have a defective ratio below 30%.** Indeed, 38% of the defect datasets have a defective ratio between 10%-20%, suggesting that the majority of defect datasets are highly imbalanced. However, as little as 8% of defect datasets have a defective ratio between 45%-55%, suggesting that there are only few defect datasets that have a defective ratio of nearly 50% (i.e., balanced datasets). On the other hand, only 1% of defect datasets (i.e., `log4j-1.2`, `xalan-2.7`) have a defective ratio higher than 90%.

The majority of defect datasets (64%) that are popularly-used in the literature have a defective ratio below 30%, suggesting that class imbalance is prominent in defect datasets, likely affecting the performance and interpretation of defect prediction models.

3 Class Rebalancing Techniques for Defect Prediction Models

A plethora of class rebalancing techniques exist [28], e.g., (1) sampling methods for imbalanced learning, (2) cost-sensitive methods for imbalanced learning, (3) kernel-based methods for imbalanced learning, and (4) active learning for imbalanced learning. Since it is impractical to study all of these techniques, we select a manageable set of class rebalancing techniques for our study. As discussed by He *et al.* [28], we start from the four families of imbalance learning techniques. Based on a literature surveys by Hall *et al.* [21], Shihab [75], and Nam [58], we then select only the family of sampling techniques for the context of defect prediction.

We first select the three commonly-used techniques (i.e., over-sampling, under-sampling, and Default SMOTE [10]) that were previously used in the literature

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2876537, IEEE Transactions on Software Engineering

Fig. 2: An illustrative overview of the 4 studied class rebalancing techniques.

[36, 38, 61, 72, 81, 85, 93, 95–97]. Recent research shows that bootstrap resampling techniques tend to produce more accurate and reliable estimates in the context of software engineering [88]. Recently, Menardi *et al.* [50] show that a smoothed bootstrap resampling technique (ROSE) outperforms other techniques in a non-software engineering domain. Thus, we select the ROSE technique [45, 50] in our study. Figure 2 provides an illustrative overview of the 4 studied class rebalancing techniques. Below, we provide a description and a discussion of the 4 studied class rebalancing techniques for our study.

3.1 Over-Sampling Technique (OVER)

The over-sampling technique (a.k.a. up-sampling) randomly samples with replacement (i.e., replicating) *the minority class* (e.g., defective class) to be the same size as the majority class (e.g., clean class). The advantage of an over-sampling technique is that it leads to no information loss. Since oversampling simply adds replicated modules from the original dataset, the disadvantage is that the training dataset ends up with multiple redundant modules, leading to an overfitting. Thus, when applying the over-sampling technique, the performance of with-in defect prediction models is likely higher than the performance of cross-project defect prediction models.

3.2 Under-Sampling Technique (UNDER)

The under-sampling technique (a.k.a. down-sampling) randomly samples (i.e., reducing) *the majority class* (e.g., clean class) in order to reduce the number of majority modules to be the same number as the minority class (e.g., defective class). The advantage of an under-sampling technique is that it reduces the size of the training data when the original data is relatively large. However, the disadvantage is that removing modules may cause the training data to lose important information pertaining to the majority class.

3.3 Synthetic Minority Oversampling Technique (SMOTE)

The SMOTE technique [10] was proposed to combat the disavantages of the simple over-sampling and under-sampling techniques. The SMOTE technique creates artificial data based on the feature space (rather than the data space) similarities from the minority modules. The SMOTE technique starts with a set of minority modules (i.e., defective modules). For each of the minority defective modules of the training datasets, SMOTE performs the following steps:

(Step 1) Calculate the k-nearest neighbors.
(Step 2) Select N majority clean modules based on the smallest magnitude of the euclidean distances that are obtained from the k-nearest neighbors.

Finally, SMOTE combines the synthetic oversampling of the minority defective modules with the undersampling the majority clean modules.

3.4 Boostrap Random Over-Sampling Examples Technique (ROSE)

The ROSE technique [45] uses a smoothed-bootstrapping approach to draw artificial samples from the feature space neighbourhood around the minority class [18]. ROSE combines oversampling and undersampling by generating an augmented sample of the data (especially belonging to the rare class). The ROSE technique consists of four steps:

(Step 1) Resample the data of the majority class using a bootstrap resampling technique to remove modules of the majority class to a defective ratio of 50% (undersampling).
(Step 2) Resample the data of the minority class using a bootstrap resampling technique to repeat modules of the minority class to a defective ratio of 50% (oversampling).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2876537, IEEE Transactions on Software Engineering

(Step 3) Combine the data of the majority and minority classes from Steps 1 and 2 into a new training sample.

(Step 4) Generate a new synthetic data for both the majority and minority classes in its neighborhood [50, p. 101] based on the combined data from Step 3. The shape of the neighborhood is determined by the kernel density function with a Gaussian kernel K and a smoothing matrix $\mathbf{H}$ with a d dimension (i.e., d is the number of independent variables), where $\mathbf{H} = diag(h_1, ..., h_d)$ and h_d is defined as follows:

$$h_q = \left(\frac{4}{(d+2)n}\right)^{1/(d+4)} \times \hat{\sigma}_q; q = 1, ..., d. \quad (1)$$

, where $\hat{\sigma}_q$ is the standard deviation of the q^{th} dimension of the observations belonging to a given class [50, p. 102].

These four steps are repeated for each training sample in order to produce a new synthetic training sample of approximately equal size as the original dataset where the number of modules for both classes equally represent (i.e., a defective ratio of nearly 50%).

4 Related Work & Research Questions

Defect prediction models may produce inaccurate predictions and interpretation when they are trained on imbalanced datasets (i.e., a dataset where the proportion of defective and clean modules is not equally represented). Prior research investigated the impact of class rebalancing techniques on the performance of defect prediction models. For example, Kamei *et al.* [36] investigate the impact of class rebalancing techniques on 2 proprietary datasets using 4 classification techniques (i.e., linear discriminant analysis (LDA), logistic regression analysis (LRA), neural network (NN), and classification tree (CT)) and 3 performance measures (i.e., Precision, Recall, and F-Measure). Riquelme *et al.* [66] investigate the impact of class rebalancing techniques on 4 open-source datasets (i.e., CM1, KC1, KC2, PC1) using 2 classification techniques (i.e., Naive Bayes and C4.5) and 1 performance measure (i.e., AUC). Wang *et al.* [93] investigate the impact of class rebalancing techniques on 5 open-source datasets (i.e., CM1, KC3, PC1, PC3, MW1) using 2 classification techniques (i.e., Naive Bayes and AdaBoost) and 5 performance measure (i.e., PD, PF, Balance, G-mean, AUC). Tan *et al.* [81] investigate the impact of class rebalancing techniques on 7 commercial and open-source datasets using 7 classification techniques (i.e., Naive Bayes, Instance-based learning, Boosting, KNN, and SVM) and 3 performance measure (i.e., Precision, Recall, and F-Measure). However, prior work focus on a limited number of datasets and performance measures, which limits the generalization (i.e., external validity) of their conclusions (see Table 6).

TABLE 1: An overview comparison of our study with respect to prior work.

Study	#Classification	Datasets	Performance Measures
Kamei *et al.* [36]	4	2	P, R, and F1
Riquelme *et al.* [66]	2	4	AUC
Wang *et al.* [93]	2	5	PD, PF, Balance, G-mean, AUC
Tan *et al.* [81]	7	7	P, R, and F1
Agrawal *et al.* [1]	6	9	P, R, PF, AUC
Bennin *et al.* [4]	5	40	P, R, AUC, Balance, G-mean
Our study	7	101	10 performance measures

Indeed, the conclusions of prior research are contradictory. For example, Kamei *et al.* [36] find that class rebalancing techniques improve the F-measure performance by 7.8%-22.4%. However, Riquelme *et al.* [66] argue that class rebalancing techniques do not improve the percentage of correctly classified modules (i.e., Accuracy), but they do improve the AUC measure. A recent meta-analysis of 42 primary defect prediction studies [47] also demonstrates that class imbalance is not considered harmful when the minority class is above 20%. Such inconsistent conclusions make it hard to derive practical guidelines when applying class rebalancing techniques when constructing defect prediction models. To address the inconsistent conclusions and generalization issue of prior work, we address the following research question.

(RQ2) How do class rebalancing techniques impact the performance of defect prediction models?

In addtion to being used for predictions, prior research also uses defect prediction models to uncover past pitfalls that lead to defective modules. For example, Hassan [25] studies the impact of complexity of code changes on software quality. Shihab *et al.* [76] investigate the impact of code and process metrics on post-release defects. Bettenburg *et al.* [5] investigate the impact of social interactions on software quality. McIntosh *et al.* [49] investigate the impact of code review coverage and participation on softwar equality. Thongtanunam *et al.* [90] investigate the impact of code review ownership on software quality. Such an understanding of defect characteristics is essential to chart quality improvement plans.

Recently, Turhan [92] point out that class rebalancing techniques may lead to bias in the learned concepts (i.e., *concept drift*). Yet, no research investigates the impact of class rebalancing techniques on the interpretation of defect prediction models. Thus, we address the following research question.

(RQ3) How do class rebalancing techniques impact the interpretation of defect prediction models?

5 Case Study Design

In this section, we describe the design of our case study that we perform to address our research questions. Figure 3 provides an overview of the case study design that we apply to each studied dataset. We describe each step below.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2876537, IEEE Transactions on Software Engineering

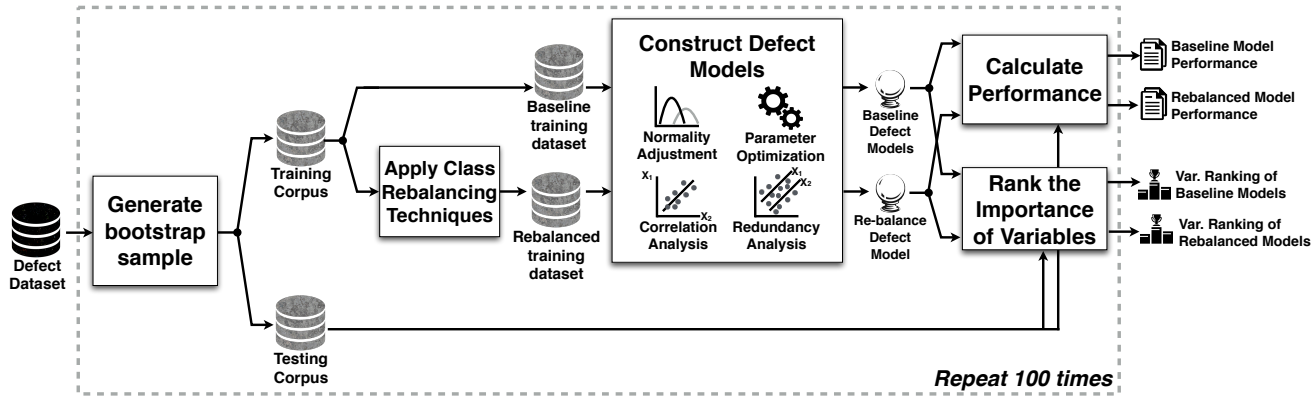


Fig. 3: An overview of our case study design.

5.1 Studied Datasets

In selecting the studied datasets, we identified two important criteria that need to be satisfied:

- **Criterion 1—Datasets from different corpora and domains**. Our recent work [87] shows the tendency of researchers to reuse experimental components (e.g., datasets, metrics, and classifiers) can introduce a bias in the reported results. To extend the generality of our conclusions, we choose to train our defect prediction models using datasets from as many different corpora and domains as possible.
- **Criterion 2—Publicly-available defect datasets**. Recently, replicability concerns are raised in our SE and medical discipline. For example, Robles *et al.* [67] point out that over 38% of 171 software engineering studies do not use publicly-available datasets nor provide their studied datasets. Moreover, Ioannidis *et al.* [30] raise concerns that the majority of medical studies in the highest ranked journals like Nature Genetic are not replicable. To foster the replication of our experiments, we choose to train our defect prediction models using datasets that are hosted in publicly-available data repositories.

To satisfy criterion 1 and 2, we opt to use the 101 publicly-available defect datasets that are described in Section 2. The 101 studied systems include proprietary and open source systems, with varying size, domain, and defective ratio.

5.2 Generate bootstrap samples

In order to ensure that the conclusions that we draw about our defect prediction models are robust, we use the out-of-sample bootstrap validation technique [88], which leverages aspects of statistical inference [17]. The out-of-sample bootstrap is made up of two steps:

(Step 1) A bootstrap sample of size N is randomly drawn with replacement from an original dataset, which is also of size N.

(Step 2) A model is trained using the bootstrap sample and tested using the rows that do not appear in the bootstrap sample. On average, 36.8% of the rows will not appear in the bootstrap sample, since it is drawn with replacement [17].

The out-of-sample bootstrap process is repeated 100 times, and the average out-of-sample performance is reported as the performance estimate.

5.3 Apply Class Rebalancing Techniques

In practice, *class rebalancing techniques should only be applied on training datasets, while* . In order to investigate the impact of class rebalancing techniques, we apply the 4 studied class rebalancing techniques (as described in Section 3) only on the training datasets, while the testing data is not rebalanced. To apply the over-sampling technique, we use the implementation of the `upSample` function that is provided by the `caret` R package [42]. To apply the under-sampling technique, we use the implementation of the `downSample` function that is provided by the `caret` R package [42]. To apply the Default SMOTE technique, we use the implementation of the `SMOTE` function that is provided by the `DMwR` R package [91] with the default k setting (i.e., $k = 5$). Section 7 provides an in-depth analysis on the impact of optimized SMOTE technique on defect models. To apply the ROSE technique, we use the implementation of the `ROSE` function that is provided by the `ROSE` R package [45].

5.4 Construct Defect Models

There are a plethora of classification techniques that have been studied in defect prediction domain [20, 21, 43, 58, 75, 86]. Since it is impractical to study all of these techniques, we would like to select a manageable set of classification techniques for our study. In selecting the classification techniques for our study, we select to study only the top-ranked classification techniques, according to our recent analysis on the ranking of classification techniques for defect prediction models when automated parameter optimization is applied [89]. We choose 7 classification techniques that appear at the top-2 ranked classification techniques. We construct defect prediction models using 7 classification techniques, i.e.,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2876537, IEEE Transactions on Software Engineering

TABLE 2: The definitions and descriptions of our studied threshold-depending performance measures.

Measures	Definition	Description
Precision (P) or Positive Predicted Values	$\frac{\text{TP}}{\text{TP}+\text{FP}}$	A proportion of modules that are correctly classified as defective
Recall (R), Probability of Detection (PD), True Positive Rate (TP_{rate}, Sensitivity)	$\frac{\text{TP}}{\text{TP}+\text{FN}}$	A proportion of defective modules that are correctly classified
F-Measure	$2 \times \frac{\text{P}\times\text{R}}{\text{P}+\text{R}}$	A harmonic mean of precision (P) and recall (R)
Matthews Correlation Coefficient (MCC)	$\frac{\text{TP}\times\text{TN}-\text{FP}\times\text{FN}}{\sqrt{(\text{TP}+\text{FP})(\text{TP}+\text{FN})(\text{TN}+\text{FP})(\text{TN}+\text{FN})}}$	A balanced measure based on true and false positives and negatives
G-mean	$\sqrt{\text{TP}_{\text{rate}} \times \text{TN}_{\text{rate}}}$	A geometric mean of a true positive rate and a true negative rate
G-measure	$\frac{2\times\text{PD}\times(1-\text{FP}_{\text{rate}})}{\text{PD}+(1-\text{FP}_{\text{rate}})}$	A harmonic mean of the probability of detection (PD) and a false positive rate (FP_{rate})
Accuracy	$\frac{\text{TP}+\text{TN}}{\text{TP}+\text{FN}+\text{FP}+\text{TN}}$	A proportion of correctly classified modules

Note: TP = True Positive, TN = True Negative, FP = False Positive, FN = False Negative, $\text{FP}_{\text{rate}} = \frac{\text{FP}}{\text{FP}+\text{TN}}$

Random Forest (RF), Logistic Regression (LR), Naive Bayes (NB), neural network (AVNNet), C5.0 Boosting (C5.0), extreme gradient boosting (xGBTree), and gradient boosting method (GBM).

Random forest constructs multiple decision trees from bootstrap samples. Logistic regression measures the relationship between a categorical dependent variable and one or more independent variables. Naive bayes is a probability-based technique that assumes that all of the predictors are independent of each other. Neural network is used to estimate or approximate functions that can depend on a large number of inputs, and that are generally unknown. C5.0 Boosting, extreme gradient boosting (xGBTree), and the gradient boosting method (GBM) perform multiple iterations, each with different example weights, and makes predictions using classifier voting.

Correlation Analysis. Jiarpakdee *et al.* [32] point out that 10%-67% of metrics of publicly-available defect datasets are redundant. Highly correlated independent variables may interfere with each other when a model is being interpreted. Indeed, our recent work [84, p. 288] [87] demonstrates that collinearity and multicollinearity issues can artificially inflate (or deflate) the impact of software metrics when interpreting defect prediction models. Recently, Jiarpakdee *et al.* [31, 33] point out that correlated metrics impact the ranking of the highest ranked metric of defect prediction models. Moreover, Jiarpakdee *et al.* [31, 33] point out that removing correlated metrics improves the consistency of the highest ranked metric regardless of how a model is specified and negligibly impacts the performance and stability of defect models. Thus, we perform correlation and redundancy analyses prior to training our defect prediction models. We measure the correlation between explanatory variables using Spearman rank correlation tests (ρ). We then use a variable clustering analysis [70] to construct a hierarchical overview of the correlation and remove explanatory variables with a high correlation. We select $|\rho| = 0.7$ as a threshold for removing highly correlated variables [37]. We perform this analysis iteratively until all clusters of surviving variables have $|\rho| < 0.7$.

Redundancy Analysis. While correlation analysis reduces collinearity among our variables, it does not detect all of the *redundant variables*, i.e., variables that do not have a unique signal with respect to the other variables. Redundant variables will interfere with each other, distorting the modelled relationship between the explanatory variables and the outcome. Therefore, we remove redundant variables prior to constructing our defect prediction models. In order to detect redundant variables, we fit preliminary models that explain each variable using the other explanatory variables. We use the R^2 value of the preliminary models to measure how well each variable is explained by the others.

We use the implementation of redundancy analysis as provided by the `redun` function of the `rms` R package [23]. The variable that is most well-explained by the other variables is iteratively dropped until either: (1) no preliminary model achieves an R^2 above a cutoff threshold (for this paper, we use the default threshold of 0.9), or (2) removing a variable would make a previously dropped variable no longer explainable, i.e., its preliminary model will no longer achieve an R^2 exceeding the threshold.

Parameter Settings. Since the studied classification techniques have configurable parameter settings, we apply Caret parameter optimization [42] prior to constructing defect prediction models as suggested by Tantithamthavorn [86].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2876537, IEEE Transactions on Software Engineering

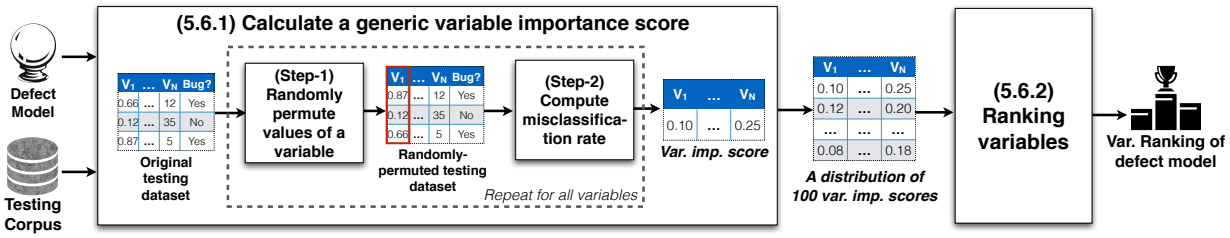


Fig. 4: An overview of our generic variable importance score calculation.

5.5 Calculate Performance

We apply the defect prediction models that we train using the training corpus to the untreated testing corpus (i.e., not rebalanced) in order to measure their performance. We use both threshold-independent and threshold-dependent performance measures to quantify the performance of our models. We describe the various performance measures that we used below.

5.5.1 Threshold-Independent Performance Measures

First, we use the *Brier score* [7, 69] to measure the distance between the predicted probabilities and the outcome. The Brier score is calculated as $\mathrm{B} = \frac{1}{N}\sum_{i=1}^{N}(f_t - o_t)^2$, where f_t is the predicted probability, o_t is the outcome for module t encoded as 0 if module t is clean and 1 if it is defective, and N is the total number of modules. The Brier score ranges from 0 (best classifier performance) to 1 (worst classifier performance), where a Brier score of 0.25 is a random-guessing performance.

Second, we use the *calibration slope* to measure the direction and spread of the predicted probabilities [13, 16, 22, 24, 52, 78, 80]. The calibration slope is the slope of a logistic regression model that is trained using the predicted probabilities of our original defect prediction model to predict whether a module will be defective or not [13]. A calibration slope of 1 indicates the best classifier performance (i.e., the predicted probabilities are consistent with modules labels) and a calibration slope of 0 (or less) indicates the worst classifier performance (i.e., the predicted probabilities are inconsistent with module's labels)

Third, we use the *Area Under the receiver operator characteristic Curve (AUC)* to measure the discriminatory power of our models, as suggested by recent research [16, 22, 29, 43, 79, 80]. The AUC is a threshold-independent performance metric that measures a classifier's ability to discriminate between defective and clean modules (i.e., do the defective modules tend to have higher predicted probabilities than clean modules?). AUC is computed by measuring the area under the curve that plots the true positive rate against the false positive rate, while varying the threshold that is used to determine whether a file is classified as defective or not. Values of AUC range between 0 (worst performance), 0.5 (random guessing performance), and 1 (best performance). We use the `val.prob` function of the `rms` R package [23] to calculate the Brier score, calibration slope, and AUC.

5.5.2 Threshold-Dependent Performance Measures

In order to calculate the threshold-dependent performance measures, the probabilities are transformed into a binary classification (defective or clean) using a default threshold value of 0.5, i.e., if a module has a predicted probability above 0.5, it is considered defective; otherwise, the module is considered clean. Using the threshold of 0.5, we compute nine threshold-dependent performance measures. Table 2 provides the definitions and descriptions of our 7 threshold-dependent performance measures.

5.6 Rank the Importance of Variables

To identify the most important variables in our built models, we compute the variable importance for each variable in our models. To do so, we develop a generic variable importance score that can be applied to any classifier [89]. Figure 4 provides an overview of the calculation of our variable importance measurement to generate ranks of the important variables for each of the baseline and rebalanced models.

5.6.1 Generic Variable Importance Score

The calculation of our variable importance score consists of 2 steps for each variable.

(Step 1) For each testing dataset, we first randomly permute the values of that particular variable, producing a randomly-permuted dataset.

(Step 2) We then compute the difference in the misclassification rates of defect prediction models that are trained using the original-unpermuted and the randomly-permuted datasets. The larger the difference, the greater the importance of that variable.

We repeat the Steps 1 and 2 for each variable in order to produce a variable importance score for all variables. Since the experiment is repeated 100 times, each variable will have several variable importance scores (i.e., one score for each of the repetitions).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2876537, IEEE Transactions on Software Engineering

5.6.2 Ranking Variables

To study the impact of the studied variables on our models, we apply the improved Scott-Knott Effect Size Difference (ESD) test (v2.0) [83, 88, 89]. The Scott-Knott ESD test clusters variables according to statistically significant differences in their mean variable importance scores ($\alpha = 0.05$).

Unlike our earlier version of the Scott-Knott ESD test (v1.0) that post-processes the groups that are produced by the Scott-Knott test, the Scott-Knott ESD test (v2.0) checks the magnitude of the difference throughout the clustering process by merging pairs of statistically distinct groups that have a negligible Cohen's d effect size difference for all of the treatments of those two groups. Cohen's d effect size [11] is an effect size estimate based on the difference between the two means divided by the standard deviation of the two datasets ($d = \frac{\bar{x}_1 - \bar{x}_2}{s.d.}$). The magnitude is assessed using the thresholds that are provided by Cohen [12], i.e. $|d| < 0.2$ "negligible", $|d| < 0.5$ "small", $|d| < 0.8$ "medium", otherwise "large".

The Scott-Knott ESD test also overcomes the confounding factor of overlapping groups that are produced by other post-hoc tests [20, 53], such as Nemenyi's test [60], which were used in prior studies [43]. We use the implementation of the Scott-Knott ESD test (v2.0) that is provided by the `ScottKnottESD` R package [83].

Finally, we produce rankings for the variables in the baseline models and rebalanced models. Thus, each variable has a rank for each type of model.

5.7 Statistical Analysis of the Experimental Settings

To better understand which of the experimental settings has the most impact on the performance of defect prediction models (i.e., RQ2 and RQ3), we build regression models to understand the relationship between experimental settings and outcome (e.g., performance difference). To study the importance of each configuration parameter, we perform an ANOVA analysis to examine the relative contribution (in terms of explanatory power) of each experimental settings to the regression model. Figure 5 shows an overview of our sensitivity analysis approach. We describe each step of our approach below.

(Step-1) Construct Models for Experimental Settings. We build regression models to explain the relationship that experimental settings have on the performance difference of defect prediction models. A regression model fits a line of the form $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n$ to the data, where y is the dependent variable and each x_i is an explanatory variable. We fit our regression models using the Ordinary Least Squares (OLS) technique using the `ols` function provided by the `rms` R package [23].

(Step-2) Assessment of Model Stability. We evaluate the fit of our models using the *Adjusted* R^2, which provides a measure of fit that penalizes the use of additional degrees of freedom. However, since the adjusted R^2 is measured using the same data that was used to train the model, it is inherently upwardly biased, i.e., "optimistic". We estimate the optimism of our models using the following bootstrap-derived approach [22].

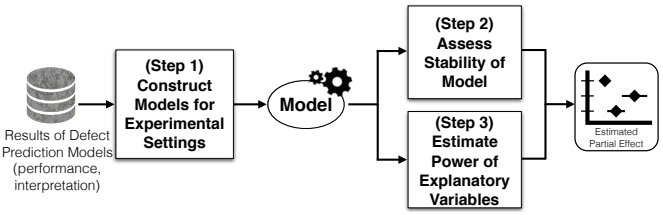


Fig. 5: An overview for our approach for statistical analysis of the experimental settings.

First, we build a model from a bootstrap sample, i.e., a dataset sampled with replacement from the original dataset, which has the same population size as the original dataset. Then, the optimism is estimated using the difference of the adjusted R^2 of the bootstrap model when applied to the original dataset and the bootstrap sample. Finally, the calculation is repeated 1,000 times in order to calculate the average optimism. This average optimism is subtracted from the adjusted R^2 of the model fit on the original data to obtain the optimism-reduced adjusted R^2. The smaller the average optimism, the higher the stability of the original model fit.

(Step-3) Estimate Power of Explanatory Variables. We perform an ANOVA analysis to examine the relative contribution (in terms of explanative power) of each experimental settings to the regression models using the Wald χ^2 maximum likelihood (a.k.a., "chunk") test. The larger the Wald χ^2 value, the larger the impact that a particular explanatory variable has on the response [22]. Finally, we present both the raw Wald χ^2 values, and its bootstrap 95 percentile confidence interval.

6 Case Study Results

In this section, we present the results of our case study with respect to the following two research questions.

(RQ2) How do class rebalancing techniques impact the performance of defect prediction models?

Approach. To address RQ2, we start with the performance distribution of defect prediction models that are trained using original (i.e., unbalanced) and re-balanced datasets. For each class rebalancing technique, we compute the difference in the performance of classifiers that are trained using using original and re-balanced datasets. We then use boxplots to present the distribution of the absolute performance for each of the 10 commonly-used performance measures.

Results. Figure 6 shows the absolute performance when applying class rebalancing techniques to defect prediction models for each of the 10 commonly-used performance measures. Figure 6 shows that the ROSE technique produces the least stable conclusions when applied to defect prediction models. Indeed, we observe that the ROSE technique has both positive and negative

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2876537, IEEE Transactions on Software Engineering

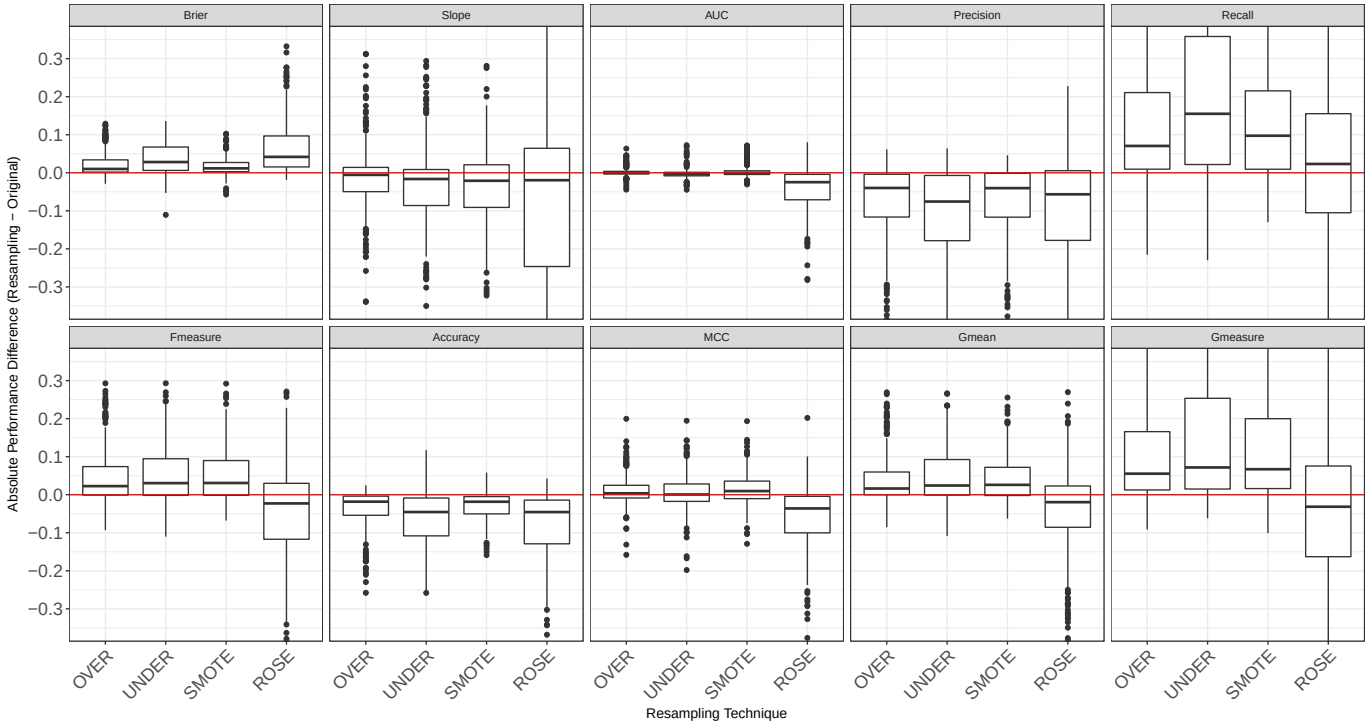


Fig. 6: The absolute performance difference when applying class rebalancing techniques to defect prediction models for each of the 10 commonly-used performance measures. A red line indicates a performance difference of zero (i.e., no improvement).

impact on the Recall, F-measure, G-measure, Gmean, and Slope of defect prediction models, suggesting that the ROSE technique should be avoid in future defect prediction studies. In the remainder of this section, we only focus on the under-sampling, over-sampling, and default SMOTE techniques.

Below, we structure the following findings with respect to the performance measures that (1) are not impacted; (2) are improved; and (3) are decreased by class rebalancing techniques.

The AUC measure is less sensitive to the under-sampling, over-sampling, and default SMOTE techniques. Looking at Figure 6, when applying over-sampling, under-sampling, and SMOTE techniques, we observe that the absolute performance values of AUC measure (i.e., `min-max`) vary from -4 to 7 percentage points. The absolute performance values that we observe are relatively smaller than common findings in the machine learning domain. For C5.0 classification technique, we observe that the maximum AUC improvement is by up to 5 percentage points when applying SMOTE technique to defect prediction models. We note that C5.0 classification technique is an improvement of C4.5 classifier in terms of speed and memory usage while sharing similar algorithms to C4.5 classifiers [41]. This observation contradicts the conclusions of Chawla [9], who found that C4.5 classifiers tends to achieve a 40% of AUC improvement when applying SMOTE to machine learning datasets with the default setting ($k = 5$). Our contradictory observation shows that domain-specifics play an important role—***the default setting that works well in machine learning domain might not always be optimal for software engineering domain.*** This finding suggests that the parameter optimization of the SMOTE technique may be of importance for defect models. Section 7 provides an in-depth analysis on the impact of optimized SMOTE technique on defect models.

Moreover, we also observe similar trends with the MCC measure which is less sensitive to class rebalancing techniques. We find that the distributions of the absolute performance of AUC and MCC are centered at zero. For example, when applying over-sampling, under-sampling, and SMOTE techniques, we find that the absolute performance of the AUC measure for 75 percentage points of the defect prediction models (i.e., 1^{st}-3^{rd} quantiles) vary from -0.8 to 0.5 percentage points. The absolute performance values of the MCC measure for 75 percentage points of the defect prediction models (i.e., 1^{st}-3^{rd} quantiles) vary from -1.7 to 3.6 percentage points. The distributions that are centered at zero indicate that the AUC and MCC measures are not impacted positively nor negatively by class rebalancing techniques for defect prediction models. The AUC measure is insensitive to class rebalancing techniques since it considers all probability thresholds for determining a module is defective or clean. On the other hand, the MCC measure is insensitive to class rebalancing techniques since it considers all aspects of the confusion

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2876537, IEEE Transactions on Software Engineering

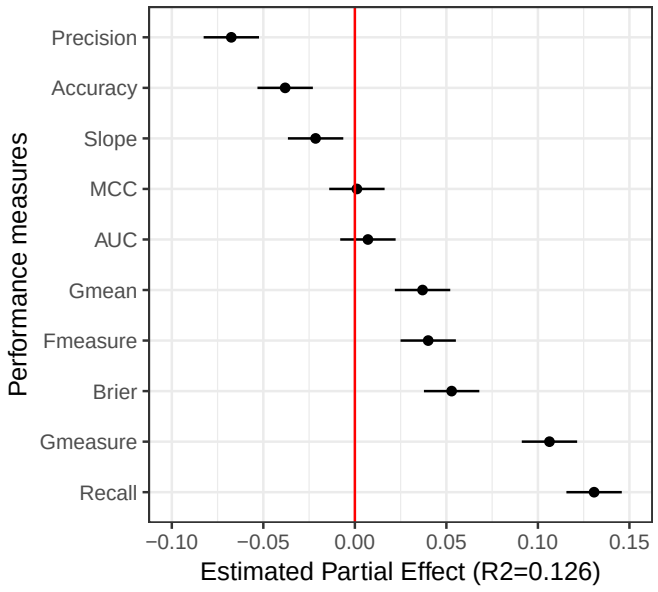


Fig. 7: Estimated partial effect plot of the relationship between performance measures and the magnitude of the performance difference with the 95% confidence interval.

metrics (i.e., true and false positives and negatives).

In contrast to the AUC measure which is rarely impacted, the other threshold-independent measures like the Brier and Slope measures are sensitive when applied class rebalancing techniques. The sensitivity of the Brier and Slope measures has to do with the computation of the Brier and Slope measures. Such computation relies heavily on the predicted probabilities (see Section 5.5.1). The Brier measure uses the predicted probabilities to compute the distance between the predicted probabilities and the outcome. The Slope measure uses the predicted probabilities to compute their directions and spreads.

The under-sampling, over-sampling, and default SMOTE techniques substantially improve the performance of defect prediction models by up to 69 percentage points for Recall, 60 percentage points for G-measure, for 27 percentage points of F-measure. We find that the proportion of defective modules that are correctly classified (i.e., Recall) improves by up to 69 percentage points when applying the over-sampling technique to the `mylyn` dataset when constructing a logistic regression classifier. Moreover, we find that the F-measure improves by up to 27 percentage points when applying under-sampling technique to the `prop-5` dataset prior to constructing a logistic regression classifier. These results indicate that class rebalancing techniques tend to have a positive impact on the Recall, G-measure, and F-measure when they are applied to defect prediction models.

On the other hand, The under-sampling, over-sampling, and default SMOTE techniques decrease the performance of defect prediction models by up to 57 percentage points for Precision, 73 percentage points for Accuracy. Interestingly, while class rebalancing techniques substantially increase Recall, they decrease Precision. We find that the decrease in the Precision measure has to do with an increased number of false positive (FP) modules (i.e., the number of clean modules that are misclassified), suggesting that the improvement of the F-measure performance has to do with the improvement of the proportion of defective modules that are correctly classified (i.e., Recall).

Statistical Analysis of the Performance Measures. To statistically (1) validate if the distributions of the performance measures are statistically different; and (2) investigate which performance measures have the largest and smallest impact, we construct a one-way ANOVA model of the distributions of the performance measures. The one-way ANOVA is a hypothesis test in which a single categorical variable or a single factor (i.e., performance measures) is considered when comparing the mean distributions of the performance values for all of the 10 studied performance measures. The one-way ANOVA model confirms that there is a significant difference in the means among the performance measures with a significant level of 0.05 (i.e., p-value < 0.05). We then plot the estimated partial effect of the absolute performance values with the 95% confidence interval (see Figure 7). The x-axis describes the effect of the performance differences, while the y-axis describes the performance measures. The effect values indicate the positive and negative magnitude of the absolute performance values, while an effect value of zero indicates that class rebalancing techniques have no impact to a particular performance measure. The estimated partial effect plot of Figure 7 confirms that AUC and MCC are insensitive to class rebalancing techniques. Moreover, Figure 7 also confirms that class rebalancing techniques yield the largest positive impact on Recall and the largest negative impact on Precision.

The AUC measure is less impacted by the under-sampling, over-sampling, and default SMOTE techniques for defect prediction models. Class rebalancing techniques impact Recall the most positively and impact Precision the most negatively.

Statistical Analysis of the Experimental Settings. To better understand the experimental settings where class rebalancing techniques yield the largest positive impact on performance measures, we construct a regression model to understand the relationship between the experimental settings (i.e., EPV, defective ratio, classification techniques, class rebalancing techniques, and metric family) and the performance difference of each performance measure using the high-level approach of Section 5.7. *EPV (Event-Per-Variables)*—a measure to evaluate the risk of overfitting—is the ratio

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2876537, IEEE Transactions on Software Engineering

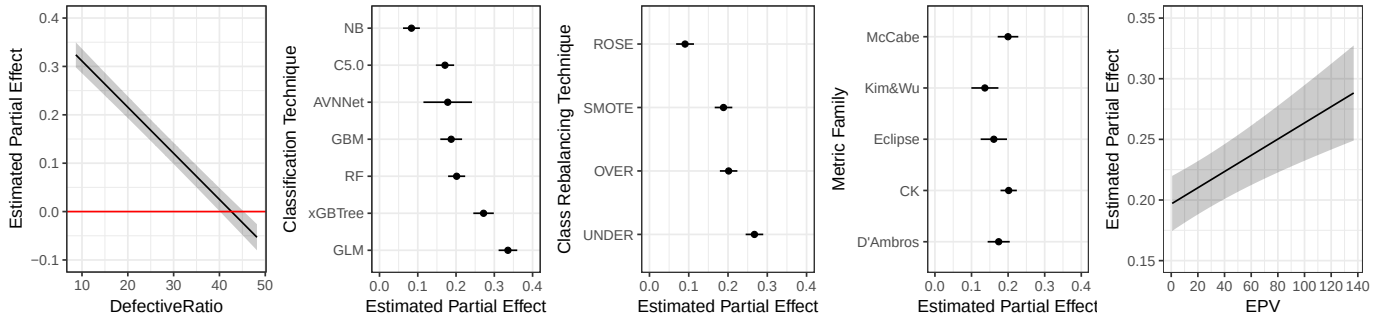


Fig. 8: Estimated partial effect plots of the relationship between the experimental settings (i.e., Defective Ratio, Classification Technique, Class Rebalancing Technique, Metric Family, and EPV) and the performance difference of the Recall measure. The grey areas and error bars indicate the 95% confidence interval. The effect values indicate the positive and negative magnitude of the absolute performance values, while an effect value of zero indicates that class rebalancing techniques have no impact to a particular performance measure.

of events to the number of independent variables used to train a model. Formally,

$$\text{EPV} = \frac{\text{\#events (e.g., \#defective modules)}}{\text{\#variables}} \quad (2)$$

where the event is the number of occurrences of the least frequently occurring class of the dependent variable (e.g., the numbers of defective modules), and the variables is the number of independent variables used to train the model (i.e., the number of software metrics) [88]. Recently, Tantithamthavorn *et al.* [88] demonstrated that models that are trained using datasets where the EPV is low (i.e., too few events are available relative to the number of independent variables) are especially susceptible to overfitting (i.e., being fit too closely to the training data). We include the EPV measure in our statistical models in order to control for a confounding factor of dataset characteristics. Since we experiment on 101 defect datasets where each dataset has different set of metrics, it is possible that some metrics are robust or sensitive to class rebalancing techniques. To statistically verify this assumption while controlling for other factors, we also include a Metric Family (i.e., D'Ambros, CK, Eclipse, Kim&Wu, and McCabe) into our statistical models. Tables 3 and 4 show the statistics of the regression model for Recall and Precision measures. In total, we analyze the results of the 2,828 experimental settings (i.e., 101 datasets x 7 classification techniques x 4 class rebalancing techniques) Below, we only discuss the analyses for the Precision and Recall measures, since we find that class rebalancing techniques have the largest impact on Recall and Precision—the results of F-Measure, AUC and MCC measures are included in the appendix.

The impact of the under-sampling, over-sampling, and default SMOTE techniques for defect prediction models relies heavily on the defective ratio of the defect datasets and the used classification techniques for Recall and Precision, respectively. Table 3 shows that *Defective Ratio* is the most influential experimental factor that impacts the Recall of defect prediction models.

TABLE 3: Statistics of the regression model of the relationship between the experimental factors and the performance difference of the *Recall* measure.

	Factor Analysis		
Adjusted R^2	0.62		
Optimism-reduced adjusted R^2	0.60		
Total Wald χ^2	1,350		
	D.F.	χ^2	p-value
Defective Ratio	1	46%	***
Classification Technique	6	33%	***
Class Rebalancing Technique	3	19%	***
Metric Family	4	1%	***
EPV	1	1%	***

Statistical significance of explanatory power according to Wald χ^2 likelihood ratio test: ∘ $p \geq 0.05$; * $p < 0.05$; ** $p < 0.01$; *** $p < 0.001$

Table 4 shows that *Classification Techniques* is the most influential experimental factor that impacts the Precision of defect prediction models. Both statistical models can explain the variability of the data with an R^2 value of 0.60 and 0.58 for the Recall and Precision measures. On the other hand, Tables 3 and 4 show that Metric Family and EPV have little impact on the performance of defect prediction models when applying class rebalancing techniques, which is similar to the results of F-Measure (see Table 7). Below, we discuss the impact of each experimental setting on the performance difference of class rebalancing techniques for defect prediction models.

For the Recall performance measure, the under-sampling, over-sampling, and default SMOTE techniques yield the largest benefits when they are applied to highly-imbalanced defect datasets with a defective ratio below 20%. The estimated partial effect plot of Figure 8 for defective ratios shows a negative ralationship between the defective ratios of defect datasets and the performance difference of defect prediction models. Thus, we plot the performance difference of the Recall measure for each range of the defective ratios in Fig-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2876537, IEEE Transactions on Software Engineering

TABLE 4: Statistics of the regression model of the relationship between the experimental factors and the performance difference of the *Precision* measure.

	Factor Analysis		
Adjusted R^2	0.59		
Optimism-reduced adjusted R^2	0.58		
Total Wald χ^2	1,261		
	D.F.	χ^2	p-value
Classification Technique	6	50%	***
Defective Ratio	1	42%	***
Class Rebalancing Technique	3	5%	***
Metric Family	4	3%	***
EPV	1	0%	∘

Statistical significance of explanatory power according to Wald χ^2 likelihood ratio test: ∘ $p \geq 0.05$; * $p < 0.05$; ** $p < 0.01$; *** $p < 0.001$

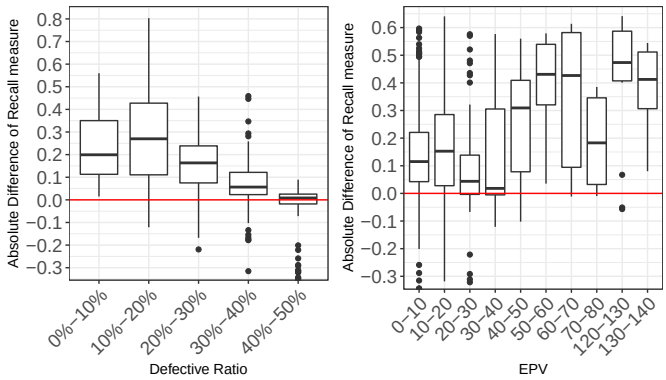


Fig. 9: The performance difference for the Recall measure for each range of defective and EPV ratios.

ure 9. We find that class rebalancing techniques tend to yield the largest performance improvement for defect prediction models when they are applied to datasets with a defective ratio below 20%. Specifically, for highly-imbalanced defect datasets (i.e., a defective ratio below 10%), we observe that class rebalancing techniques consistently improve the Recall of defect prediction models. On the other hand, for nearly-balanced defect datasets (i.e., a defective ratio between 40-50%), we find that class rebalancing techniques have little impact on the performance improvement.

Class rebalancing techniques yield the largest benefits when they are applied to defect datasets with an EPV ratio higher than 40. The estimated partial effect plot of Figure 8 for EPV ratios shows a positive rationship between the EPV ratios of defect datasets and the impact on the performance of defect prediction models. Thus, we plot the performance difference of the Recall measure for each range of the EPV ratios in Figure 9. We observe that class rebalancing techniques tend to yield the largest performance improvement for defect prediction models when they are applied to defect datasets with an EPV ratio higher than 40. Our finding is consistent with Blagus *et al.* [6]'s findings which points out that SMOTE does not perform well with high-dimensionality data.

Logistic regression is the most sensitive classifier to imbalanced defect datasets, while more advanced classification techniques like neural networks and random forest tend to be less sensitive. Figure 8, the estimated partial effect plot for classification techniques, shows that the impact of class rebalancing techniques on the performance of defect prediction models varies between the studied classification techniques. We find that logistic regression classifiers tend to yield the largest benefit, while naive bayes classifiers tend to yield the smallest benefit on the Recall performance measure. Thus, class rebalancing techniques should be applied to future defect prediction studies which making use of logistic regression.

The under-sampling technique improves Recall measure the most. Figure 8, the estimated partial effect plot for classification techniques, shows a positive rationship between the class rebalancing techniques and the performance of defect prediction models. We find that the under-sampling technique tends to improve Recall measure the most, while the ROSE technique tends to improve Recall measure the least. Similarly, Figure 6 shows that the Recall improvement is, on average, 18 percentage points for under-sampling technique, 10 percentage points for the SMOTE technique, 7 percentage points for the over-sampling technique, and 3 percentage points for the ROSE technique, indicating that the under-sampling technique should be used when the main objective of defect prediction model is the proportion of defective modules that are correctly classified (i.e., Recall).

The impact of the under-sampling, over-sampling, and default SMOTE techniques on the performance of defect prediction models depends on experimental settings. Defect prediction models yield the largest performance improvement when applying the under-sampling technique to logistic regression models using defect datasets that are highly-imbalanced with an EPV ratio higher than 40.

(RQ3) How do class rebalancing techniques impact the interpretation of defect prediction models?

Approach. To address RQ3, we start with the variable ranking of the 7 studied classification techniques on each of the 101 studied datasets for both classifiers that are trained with the original and rebalanced datasets. For each classification technique, we compute the difference in the ranks of the variables that appear in the top-three ranks of the classifiers that are trained using the original and rebalanced datasets. For example, if a variable v appears in the top rank in both the original and rebalanced models, then the variable would have a rank difference of 0. However, if v appears in the third rank in the rebalanced model, then the rank difference of v would be -2.
Results. **Class rebalancing techniques have a large impact on the interpretation of defect prediction models that are produced by popularly-used classification**

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2876537, IEEE Transactions on Software Engineering

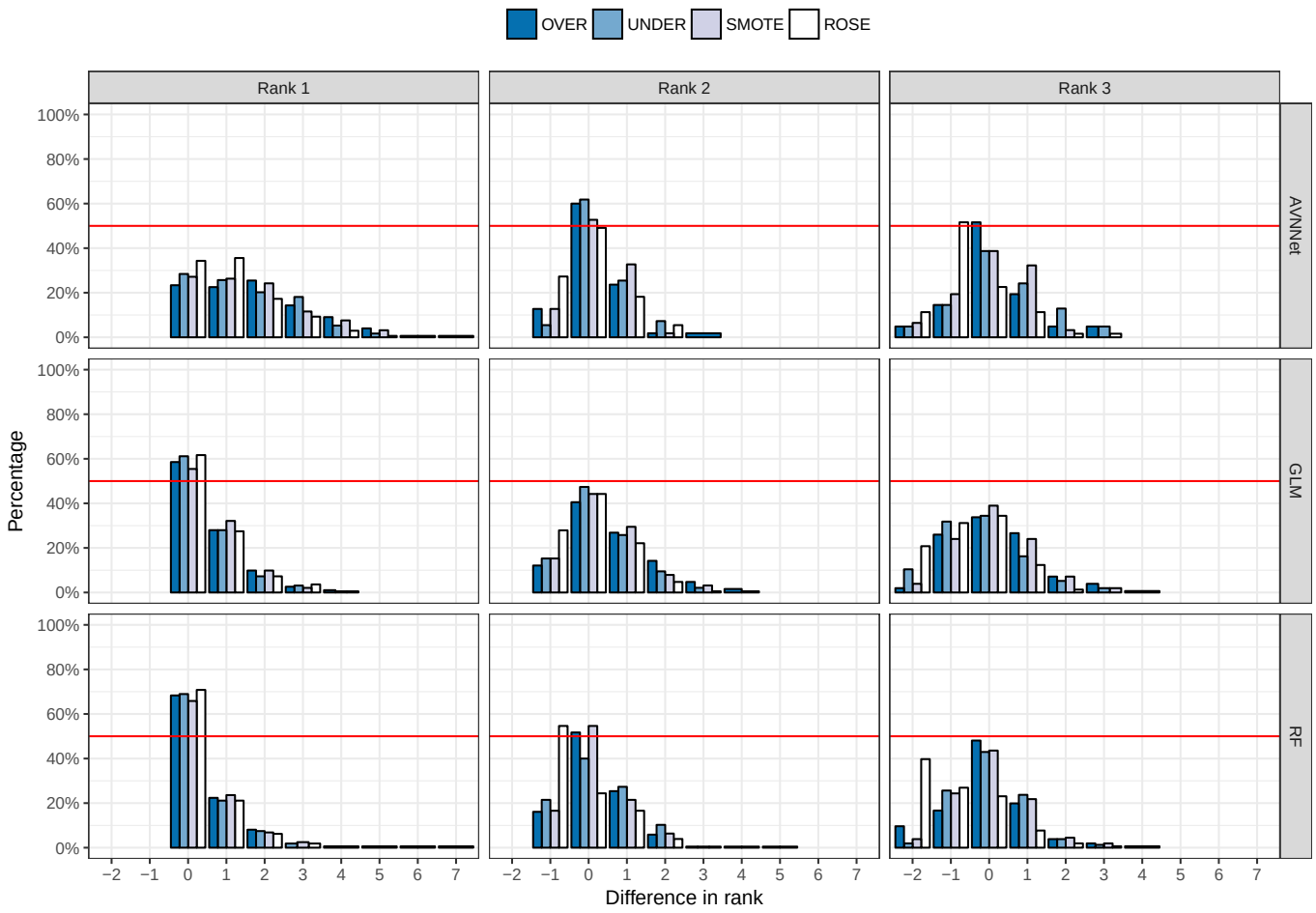


Fig. 10: The difference in the ranks for the variables according to their variable importance scores among the defect prediction models that are trained using original (i.e., baseline) and re-balanced datasets. The bars indicate the percentage of variables that appear in that rank in the re-balanced model while also appearing in that rank in the baseline model. The higher the percentage is, the least stable the interpretation of defect prediction models is.

techniques like random forest, logistic regression, and neural network. Figure 10 shows that as little as 23%-34%, 55%-62%, and 68%-71% of the top variables in the top importance rank of the rebalanced models appear in the top importance rank of the baseline models for neural network, logistic regression, and random forests classifiers, respectively. In other words, as much as 77%-66%, 45%-38%, and 32%-29% of the top variables in the top importance rank of the rebalanced models do not appear in the top importance rank of the original models. Moreover, the variables in the second and third ranks are even more unstable. ***This is the first empirical evidence that confirms the suspicious of Turhan [92] who point out that class rebalancing techniques shift the learned concepts (i.e., biasing the interpretation of defect prediction models)***, suggesting that class rebalancing techniques should be avoided in future defect prediction studies, especially, when deriving knowledge and understandings from these models.

Class rebalancing techniques shift the learned concepts (i.e., biasing the interpretation of defect prediction models). We find that as little as 23%-34%, 55%-62%, and 68%-71% of the top variables in the top importance rank of the rebalanced models appear in the top importance rank of the baseline models for neural network, logistic regression, and random forest classifiers, respectively.

Statistical Analysis on the Experimental Settings. To better understand the experimental setup where class rebalancing techniques have the largest and smallest impact on the interpretation of defect prediction models, we construct a regression model to understand the relationship between the experimental factors (i.e., EPV, defective ratio, classification techniques, class rebalancing techniques, and metric family) and the percentage of the most important variables appearing in the same rank using the high-level approach of Section 5.7. Table 5 shows the statistics of the regression model. Figure 7 shows the estimated partial effect of the relationship between performance measures and the percentage of the most important variables appearing in the same rank.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2876537, IEEE Transactions on Software Engineering

TABLE 5: Statistics of the regression model of the relationship between the experimental factors and the percentage of the most important variables appearing in the same rank.

	Factor Analysis		
Adjusted R^2	0.15		
Optimism-reduced adjusted R^2	0.14		
Total Wald χ^2	406		
	D.F.	χ^2	p-value
Classification Technique	6	79%	***
Metric Family	4	10%	***
Class Rebalancing Technique	3	5%	***
Defective Ratio	1	4%	***
EPV	1	1%	∘

Statistical significance of explanatory power according to Wald χ^2 likelihood ratio test: ∘ $p \geq 0.05$; * $p < 0.05$; ** $p < 0.01$; *** $p < 0.001$

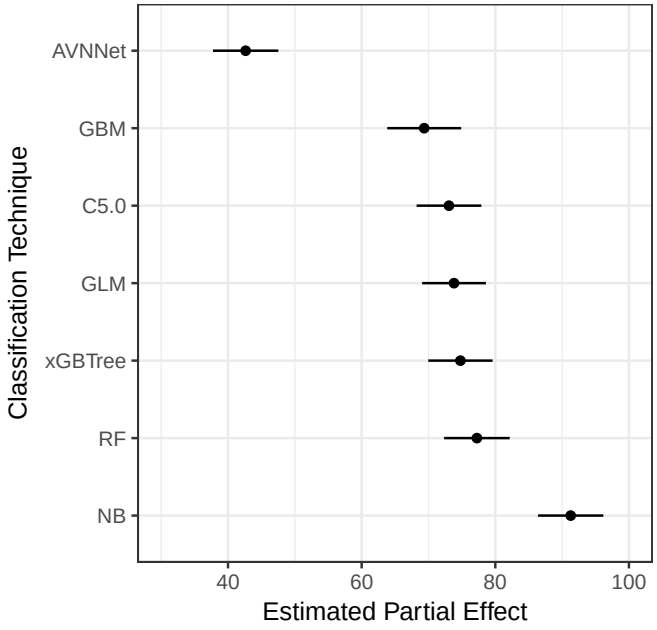


Fig. 11: Estimated partial effect plot of the relationship between the studied classification techniques and the percentage of the most important variables appearing in the same rank. The error bars indicate the 95% confidence interval. The figure shows that neural network is the most sensitive classification technique, while naive bayes is the least sensitive classification technique to class rebalancing techniques.

The impact of class rebalancing techniques on the interpretation of defect prediction models relies heavily on the used classification techniques. Table 5 shows that *classification technique* is the most influential experimental factor on the impact of class rebalancing techniques on the interpretation of defect prediction models. Figure 11 confirms that neural network classifiers are the most sensitive classification techniques when applying class rebalancing techniques, suggesting that neural network classification techniques should be avoided when interpreting insights from defect prediction models.

The impact of class rebalancing techniques on the interpretation of defect prediction models relies heavily on the used classification techniques, suggesting that researchers and practitioners should avoid class rebalancing techniques when deriving knowledge and understandings from defect prediction models.

7 An Investigation of the Impact of the Optimized SMOTE Technique on Defect Models

7.1 Motivation

Plenty of prior work show that the parameters of classification techniques have an impact on the performance of defect prediction models [86, 89]. Similarly, the SMOTE class rebalancing technique has a configurable parameter that needs to be specified. Similar to prior studies in software engineering [3, 4, 72, 95], the results of our RQ2 and RQ3 rely on one default parameter setting (i.e., $k = 5$). Recently, Agrawal and Menzies [1] pointed out that the k SMOTE parameter may be sensitive to the AUC performance of defect models. On the other hand, Bennin *et al.* [4] pointed out that the SMOTE technique does not improve the AUC performance of defect models. Thus, we set out to re-investigate the impact of the k SMOTE parameter on our 101 studied defect datasets. In addition to the contradictory conclusion, these two recent studies do not further investigate to what extent the k SMOTE parameter impacts the performance and interpretation of defect prediction models.

7.2 Approach

To identify the optimized SMOTE parameter settings, we use a grid-search optimization approach, as suggested by recent studies [86, 89]. A key benefit of applying a grid-search optimization technique is that the experimental design is highly controlled—i.e., the candidate parameter settings of the grid search technique do not vary for each dataset and bootstrap sample, while the candidate parameter settings of other advanced parameter optimization techniques often vary, leading to substantially more complex experimental settings.

To apply the optimized SMOTE technique, we start from a training dataset that is generated by the out-of-sample bootstrap (see Section 5.2). For each training dataset, we randomly generate a bootstrap sample and a validation sample for evaluating each of the 20 candidate k parameter settings (i.e., from 1 to 20). To evaluate each setting, we apply the SMOTE technique with the candidate setting and evaluate the performance of the candidate setting using the validation sample. *We note that the evaluation of k SMOTE parameter settings does not involve any testing datasets to avoid generating optimisitc performance estimates.* Finally, we identify an optimal k

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2876537, IEEE Transactions on Software Engineering

setting as the k setting that acheives the highest performance among the candidate settings.

We apply the Optimized SMOTE technique to all of the 101 studied defect datasets with 20 different k SMOTE settings (i.e., from 1 to 20) for all of the seven classification techniques. For each dataset and each classification technique, we identify the optimal setting if a k setting is the top-performing. We then measure the AUC improvement as the absolute difference between the AUC values of defect models when applying the optimal SMOTE and default SMOTE techniques.

7.3 Results

In contrast to our RQ2 and Bennin *et al.* [4] which conclude the SMOTE technique does not improve the AUC performance of defect models, we find that the AUC performance of defect models is substantially improved when the k SMOTE parameter is optimized. Figure 12a, which presents the AUC improvement of defect prediction models when the optimized SMOTE technique is applied for each of the studied classification techniques, comfirming that optimizing SMOTE parameter consistently improves the AUC performance of defect prediction models. This finding echoes *the importance of the parameter optimization on the SMOTE technique for defect models*, which is similar to Agrawal and Menzies [1].

In addition to Agrawal and Menzies [1], we observe that the optimization of the k SMOTE parameter yields the largest AUC improvement when it is applying along with AVNNet, GBM, RF, and C5.0 classifiers. Figure 12b presents boxplots of the AUC improvement of defect prediction models when the optimized SMOTE technique is applied to data when using each classification technique. Figure 12b shows that the performance of defect models improves by a median of 13-20 percentage points for AVNNet, GBM, RF, and C5.0 classifiers when the optimized SMOTE technique is applied. On the other hand, Figure 12b shows that defect models yield less improvement for GLM, NB, and xGBTree when the optimized SMOTE technique is applied.

Similar to RQ3, the optimized SMOTE technique has a large impact on the interpretation of defect prediction models that are produced by popularly-used classification techniques. Figure 12c shows that as little as 28%, 73%, and 68% of the top variables in the top importance rank of the rebalanced models appear in the top importance rank of the baseline models for neural network, logistic regression, and random forests classifiers, respectively. This finding confirms our suggestion that class rebalancing techniques should be avoided in future defect prediction studies, especially, when deriving knowledge and understandings from these models.

8 Practical Guidelines

Table 6 summarizes a comparison of prior findings in the literature with our findings. In this section, we offer practical guidelines for future defect prediction studies:

(1) **The optimized SMOTE technique and the undersampling technique are beneficial when quality assurance teams wish to increase the ability to classify defective modules (i.e., AUC) and the completeness of identifying software defects (i.e., Recall), respectively**, since Figure 7 shows that class rebalancing techniques substantially improve the proportion of defective modules that are correctly classified (i.e., Recall measure). Specifically, Figure 8 also shows that defect prediction models yield the largest improvement in Recall when applying the undersampling technique to logistic regression models using defect datasets that are highly-imbalanced with an EPV ratio higher than 40. Nevertheless, when applying class rebalancing techniques, the improvement of the Recall measure has to be sacrificed with a decrease in the Precision measure.

(2) **Class rebalancing techniques should be avoided when deriving knowledge and understandings from defect prediction models to initiate quality improvement plans**, since Figure 11 shows that commonly-used classification techniques like logistic regression, random forest, and neural networks are sensitive to class rebalancing techniques. Moreover, our statistical model (Table 5) of the experimental factors analysis confirms that class rebalancing techniques have a large impact on the interpretation of defect prediction models.

9 Threats to Validity

Like any empirical study design, experimental design settings may impact the results of our study [82]. Below, we discuss threats that may impact the results of our study.

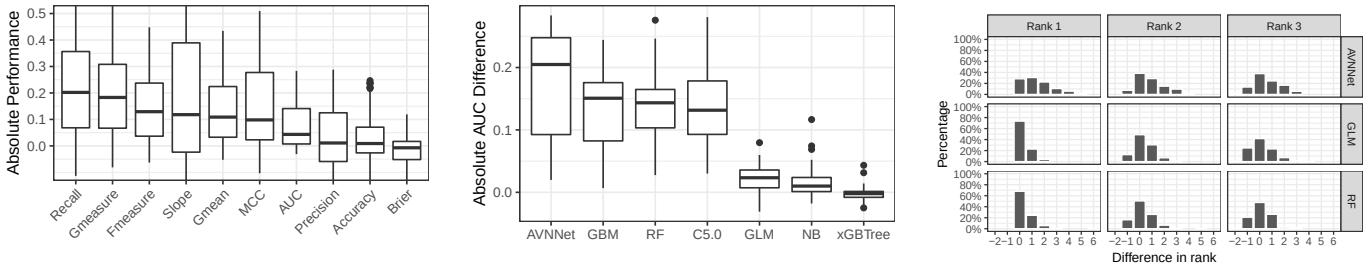
9.1 External Validity

We studied a limited number of proprietary and open-source software systems. Thus, our results may not generalize to all software systems. However, to the best of our knowledge, this study is among the largest empirical study on the impact of class rebalancing techniques for defect prediction models — our conclusions are drawn from the 101 publicly-available defect datasets.

The conclusions of our case study rely on one defect prediction scenario (i.e., within-project defect prediction models). However, there are a variety of defect prediction scenarios in the literature (e.g., cross-project defect prediction [98], just-in-time defect prediction [34], heterogenous defect prediction [59]). Therefore, the practical guidelines may differ when applying class rebalancing techniques to other scenarios. Thus, future research should revisit our study in other scenarios of defect prediction models.

Recent studies [35, 62–64] recommend the consideration of developer effort when evaluating the performance of defect prediction models. For example,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2876537, IEEE Transactions on Software Engineering



(a) The performance difference of defect prediction models when the optimized SMOTE technique is applied for each performance measure.

(b) The absolute AUC difference of defect prediction models when the optimized SMOTE technique is applied to each of the studied classification techniques.

(c) The difference in the ranks for the variables according to their variable importance scores among the defect prediction models that are trained using original and re-balanced datasets that are produced by the optimized SMOTE technique.

Fig. 12: The results of defect prediction models when the optimized SMOTE technique is applied.

TABLE 6: A comparison of prior findings in the literature to our findings.

Paper	Their finding	Our finding
Riquelme *et al.* [66]	Class rebalancing techniques do not improve the percentage of correctly classified modules (i.e., Accuracy), but they do improve the AUC measure for 4 NASA datasets.	Class rebalancing techniques do improve the percentage of correctly classified modules (i.e., Accuracy), but they have little improvement for the AUC measure (especially for datasets with a high EPV value).
Chawla [9]	SMOTE improves the AUC measure by up to 40% for machine learning datasets.	The AUC measure is less sensitive to class rebalancing techniques (especially for datasets with a high EPV value).
Wang *et al.* [93]	An advanced class rebalancing technique (i.e., AdaBoost.NC) yeilds similar AUC performance when compared to random forest for 10 NASA defect datasets.	The AUC measure is less sensitive to class rebalancing techniques (especially for datasets with a high EPV value).
Tan *et al.* [81]	Class rebalancing techniques improve the Precision measure but decrease the Recall measure for change classification models.	Class rebalancing techniques decrease the Precision measure but improve the Recall measure for defect classification models.
Kamei *et al.* [36]	Class rebalancing techniques improve the Recall and F-measures but decrease the Precision measure.	Class rebalancing techniques improve the Recall and F-measures but decrease the Precision measure.
Bennin *et al.* [4]	The default SMOTE technique does not improve the AUC performance of defect models	The AUC performance of defect models is substantially improved when the k SMOTE parameter is optimized.
Agrawal *et al.* [1]	The AUC performance of defect models is substantially improved when the k SMOTE parameter is optimized	In addition to Agrawal *et al.*, we observe that the optimization of the k SMOTE parameter yields the largest AUC improvement when it is applying along with AVNNet, GBM, RF, and C5.0 classifiers, but the optimized SMOTE technique has a large impact on the interpretation of defect models.

Kamei *et al.* [63] suggest to evaluate defect prediction models using the Area Under the Cost Effectiveness Curve (AUCEC). While we studied a large number of performance measures, i.e., 7 threshold-dependent and 3 threshold-independent measures, our results may not generalize to other performance measures (e.g., AUCEC). Since the AUCEC measure is not currently compatible with the Caret implementation [42]. Caret measures the model performance based on a `summaryFunction` function that only takes the observed and predicted values. Hence, we are unable to compute the AUCEC measure. Nonetheless, other performance measures can be explored in future work. We provide a detailed methodology for others who would like to re-examine our findings using unexplored performance measures.

9.2 Internal Validity

Recent work pointed out that class rebalancing techniques suffer from creating an artificial bias towards minority class [46]. Thus, Friedman *et al.* [19] suggested to use advanced classification techniques (e.g., penalized classification) to address the class imbalance problem for defect prediction models without applying a class rebalancing technique. Such penalization classifcation imposes an additional cost on the models for making classification mistakes on the minority class during training, while enabling the models to pay more attention to the minority class. To assess if a penalized classification technique addresses the class imbalance problem for defect prediction models, we built penalized logistic regression models for the 101 studied defect datasets using the `glmnet` function that is provided by the `glmnet` R package [26]. We then compared the AUC distributions with the other seven classification techniques when both

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2876537, IEEE Transactions on Software Engineering

class rebalancing techniques are applied and not applied. We find that the penalized logistic regression models have little improvement on the AUC performance. Figure 13 shows the AUC distributions of the 101 defect datasets for each of the studied seven classification technique, the five class rebalancing techniques, and the penalized logistic regression technique. The results of Figure 13 confirms that building random forest models with the optimized SMOTE technique tends to be the top-performing models for the AUC performance. We suspect that the top-performing random forest models when the optimized SMOTE technique is applied has to do with the random process for constructing multiple trees in order to mitigate imbalance datasets, the averaging calculation of the performance of a random forest model, and the synthetic samples of the minority class that are generated by SMOTE. For example, random forest may generate trees that are constructed with balanced samples [44]. In addition, the averaging of the performance from multiple trees may decrease the negative impact of class imbalance on trees that are constructed with imbalanced samples.

However, the goal of our paper is not to examine all classification techniques and class rebalancing techniques that were used in the literature. Nonetheless, other classification techniques and class rebalancing techniques [28] can be explored in future work. This paper provides a detailed methodology for others who would like to re-examine our findings using unexplored class rebalancing techniques and advanced classification techniques.

10 Conclusions

In this paper, we set out to investigate the impact of 4 popularly-used class rebalancing techniques, i.e., over-sampling, under-sampling, SMOTE, and ROSE techniques, on the performance and the interpretation of defect prediction models. We train our defect prediction models using 7 classification techniques and evaluate the performance of defect prediction models using 10 commonly-used performance measures. To better understand in which experimental settings class rebalancing techniques are beneficial for defect prediction models, we also construct statistical models to study the relationship between the experimental settings and the performance and interpretation of defect prediction models. Through a large-scale empirical study of 101 publicly-available defect datasets that span across open-source and proprietary systems that are collected from 5 different corpus, we record the following observations:

- As little as 8% of defect datasets have a defective ratio between 45%-55%, suggesting that class imbalance is prominent in defect datasets, likely affecting the performance and interpretation of defect prediction models.
- The AUC measure is less impacted by the under-sampling, over-sampling, and default SMOTE techniques for defect prediction models. On the other hand, the AUC measure is substantially improved when the k SMOTE parameter is optimized, suggesting that future studies must optimized the SMOTE parameters.
- The impact of the under-sampling, over-sampling, default SMOTE and optimized SMOTE techniques on the performance of defect prediction models depends on experimental settings. Defect prediction models yield the largest AUC improvement when applying the optimized SMOTE technique and the largest Recall improvement when applying the under-sampling technique.
- Unfortunately, class rebalancing techniques shift the learned concepts (i.e., biasing the interpretation of defect prediction models). We find that as little as 23%-34%, 55%-62%, and 68%-71% of the top variables in the top importance rank of the re-balanced models appear in the top importance rank of the baseline models for neural network, logistic regression, and random forest classifiers, respectively.
- The impact of class rebalancing techniques on the interpretation of defect prediction models relies heavily on the used classification techniques, suggesting that researchers and practitioners should avoid class rebalancing techniques when deriving knowledge and understandings from defect prediction models.

Based on our findings, we make the following suggestions for researchers and practitioners:

1) The optimized SMOTE technique and the under-sampling technique are beneficial when quality assurance teams wish to increase the ability to classify defective modules (i.e., AUC) and the completeness of identifying software defects (i.e., Recall), respectively.
2) Class rebalancing techniques should be avoided when deriving knowledge and understandings from defect prediction models to initiate quality improvement plans.

Acknowledgments

This work was supported by the Grant-in-Aid for JSPS Fellows (No. 16J03360), and the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] A. Agrawal and T. Menzies, "Is "better data" better than "better data miners"?" in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2018, p. To Appear.

[2] F. Akiyama, "An Example of Software System Debugging," in *Proceedings of the International Federation of Information Processing Societies Congress (IFIP'71)*, 1971, pp. 353–359.

[3] 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 Transactions on Software Engineering (TSE)*, p. To Appear, 2017.

[4] K. E. Bennin, J. W. Keung, and A. Monden, "On the relative value of data resampling approaches for software defect prediction," *Empirical Software Engineering*, pp. 1–35, 2018.

[5] N. Bettenburg and A. E. Hassan, *Studying the impact of social interactions on software quality*, apr 2012, vol. 18, no. 2. [Online]. Available: http://link.springer.com/10.1007/s10664-012-9205-0

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2876537, IEEE Transactions on Software Engineering

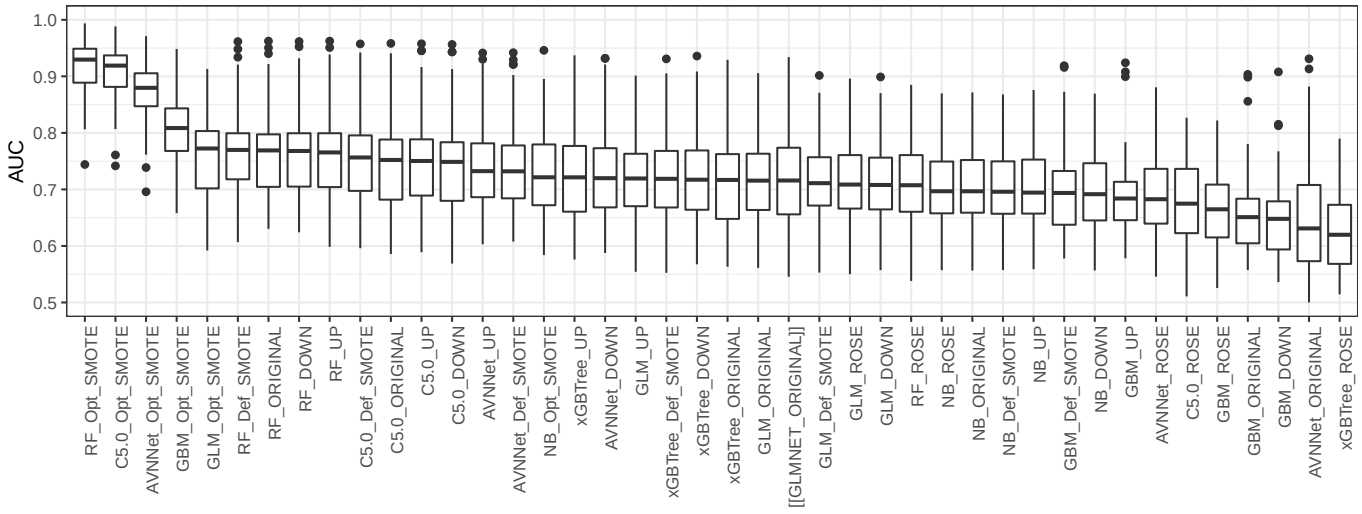


Fig. 13: The AUC distributions of the 101 defect datasets for each of the studied seven classification techniques, the five class rebalancing techniques, and the penalized logistic regression technique.

[6] R. Blagus and L. Lusa, "Smote for high-dimensional class-imbalanced data," *BMC Bioinformatics*, vol. 14, no. 1, p. 106, 2013. [Online]. Available: http://dx.doi.org/10.1186/1471-2105-14-106

[7] G. W. Brier, "Verification of Forecasets Expressed in Terms of Probability," *Monthly Weather Review*, vol. 78, no. 1, pp. 25–27, 1950.

[8] M. Cataldo, A. Mockus, J. Roberts, and J. Herbsleb, "Software Dependencies, Work Dependencies, and Their Impact on Failures," *IEEE Transactions on Software Engineering (TSE)*, vol. 35, no. 6, pp. 864–878, 2009.

[9] N. V. Chawla, "C4.5 and imbalanced data sets: investigating the effect of sampling method, probabilistic estimate, and decision tree structure," in *n Proceedings of the ICML03 Workshop on Class Imbalances*, vol. 3, 2003.

[10] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[11] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*, 1988.

[12] ——, "A power primer." *Psychological bulletin*, vol. 112, no. 1, p. 155, 1992.

[13] D. R. Cox, "Two Further Applications of a Model for Binary Regression," *Biometrika*, vol. 45, no. 3, pp. 562–565, 1958.

[14] M. D'Ambros, M. Lanza, and R. Robbes, "An Extensive Comparison of Bug Prediction Approaches," in *Proceedings of the Working Conference on Mining Software Repositories (MSR)*, 2010, pp. 31–41.

[15] ——, "Evaluating Defect Prediction Approaches: A Benchmark and an Extensive Comparison," *Empirical Software Engineering*, vol. 17, no. 4-5, pp. 531–577, 2012.

[16] S. den Boer, N. F. de Keizer, and E. de Jonge, "Performance of prognostic models in critically ill cancer patients - a review." *Critical care*, vol. 9, no. 4, pp. R458–R463, 2005.

[17] B. Efron, "Estimating the Error Rate of a Prediction Rule: Improvement on Cross-Validation," *Journal of the American Statistical Association*, vol. 78, no. 382, pp. 316–331, 1983.

[18] B. Efron and R. J. Tibshirani, *An Introduction to the Bootstrap*. Boston, MA: Springer US, 1993.

[19] J. Friedman, T. Hastie, and R. Tibshirani, "Regularization paths for generalized linear models via coordinate descent," *Journal of statistical software*, vol. 33, no. 1, p. 1, 2010.

[20] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2015, pp. 789–800.

[21] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A Systematic Literature Review on Fault Prediction Performance in Software Engineering," *IEEE Transactions on Software Engineering (TSE)*, vol. 38, no. 6, pp. 1276–1304, nov 2012.

[22] F. E. Harrell Jr., *Regression Modeling Strategies*, 1st ed. Springer, 2002.

[23] ——, "rms: Regression modeling strategies," http://CRAN.R-project.org/package=rms, 2015.

[24] F. E. Harrell Jr., K. L. Lee, and D. B. Mark, "Tutorial in Biostatistics Multivariable Prognostic Models : Issues in Developing Models, Evaluting Assumptions and Adequacy, and Measuring and Reducing Errors," *Statistics in Medicine*, vol. 15, pp. 361–387, 1996.

[25] A. E. Hassan, "Predicting Faults Using the Complexity of Code Changes," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2009, pp. 78–88.

[26] T. Hastie, "glmnet: Lasso and Elastic-Net Regularized Generalized Linear Models," https://cran.r-project.org/web/packages/glmnet/index.html, 2017.

[27] H. He and E. A. Garcia, "Learning from Imbalanced Data," *Transactions on Knowledge and Data Engineering (TKDE)*, vol. 21, no. 9, pp. 1263–1284, 2009.

[28] ——, "Learning from imbalanced data," *IEEE Transactions on knowledge and data engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[29] J. Huang and C. X. Ling, "Using AUC and accuracy in evaluating learning algorithms," *Transactions on Knowledge and Data Engineering*, vol. 17, no. 3, pp. 299–310, 2005.

[30] J. P. Ioannidis, D. B. Allison, C. A. Ball, I. Coulibaly, X. Cui, A. C. Culhane, M. Falchi, C. Furlanello, L. Game, G. Jurman *et al.*, "Repeatability of published microarray gene expression analyses," *Nature genetics*, vol. 41, no. 2, pp. 149–155, 2009.

[31] J. Jiarpakdee, C. Tantithamthavorn, and A. E. Hassan, "The impact of correlated metrics on defect models," *Under review at IEEE Transactions on Software Engineering (TSE)*, 2018, https://arxiv.org/abs/1801.10271.

[32] J. Jiarpakdee, C. Tantithamthavorn, A. Ihara, and K. Matsumoto, "A study of redundant metrics in defect prediction datasets," in *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE)*, 2016, pp. 51–52.

[33] J. Jiarpakdee, C. Tantithamthavorn, and C. Treude, "Autospearman: Automatically mitigating correlated metrics for interpreting defect models," in *Proceeding of the International Conference on Software Maintenance and Evolution (ICSME)*, 2018, p. To Appear.

[34] Y. Kamei, E. Shihab, B. Adams, a. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A Large-Scale Empirical Study of Just-in-Time Quality Assurance," *IEEE Transactions on Software Engineering*

(TSE), vol. 39, no. 6, pp. 757–773, 2013.

[35] Y. Kamei, S. Matsumoto, A. Monden, K.-i. Matsumoto, B. Adams, and A. E. Hassan, "Revisiting Common Bug Prediction Findings Using Effort-Aware Models," in *Proceedings of the International Conference on Software Maintenance (ICSM)*, 2010, pp. 1–10.

[36] Y. Kamei, A. Monden, S. Matsumoto, T. Kakimoto, and K.-i. 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, sep 2007. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4343747

[37] A. E. Kazdin, "The meanings and measurement of clinical significance." 1999.

[38] T. Khoshgoftaar, K. G. K. Gao, and N. Seliya, "Attribute Selection and Imbalanced Data: Problems in Software Defect Prediction," *Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on*, vol. 1, 2010.

[39] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with Noise in Defect Prediction," in *Proceeding of the International Conference on Software Engineering (ICSE)*, 2011, pp. 481–490.

[40] S. Kim, T. Zimmermann, E. J. Whitehead Jr., and A. Zeller, "Predicting Faults from Cached History," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2007, pp. 489–498.

[41] M. Kuhn, "C50: C5.0 decision trees and rule-based models," http://CRAN.R-project.org/package=C50, 2015.

[42] ——, "caret: Classification and regression training," http://CRAN.R-project.org/package=caret, 2015.

[43] S. Lessmann, S. Member, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings," *IEEE Transactions on Software Engineering (TSE)*, vol. 34, no. 4, pp. 485–496, 2008.

[44] A. Liaw and M. Wiener, "randomforest: Breiman and cutler's random forests for classification and regression," http://CRAN.R-project.org/package=randomForest, 2015.

[45] N. Lunardon, G. Menardi, and N. Torelli, "Rose: a package for binary imbalanced learning," *R Journal*, vol. 6, no. 1, pp. 79–89, 2014.

[46] L. Lusa *et al.*, "Class prediction for high-dimensional class-imbalanced data," *BMC bioinformatics*, vol. 11, no. 1, p. 523, 2010.

[47] Z. Mahmood, D. Bowes, P. C. Lane, and T. Hall, "What is the impact of imbalance on software defect prediction performance?" in *Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering*. ACM, 2015, p. 4.

[48] R. Malhotra and M. Khanna, "An empirical study for software change prediction using imbalanced data," *Empirical Software Engineering*, pp. 1–46, 2017.

[49] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "The Impact of Code Review Coverage and Code Review Participation on Software Quality," in *Proceedings of the Working Conference on Mining Software Repositories (MSR)*, 2014, pp. 192–201.

[50] G. Menardi and N. Torelli, "Training and assessing classification rules with imbalanced data," *Data Mining and Knowledge Discovery*, vol. 28, no. 1, pp. 92–122, 2014.

[51] T. Menzies, C. Pape, R. Krishna, and M. Rees-Jones, "The Promise Repository of Empirical Software Engineering Data," http://openscience.us/repo, 2015.

[52] M. E. Miller, S. L. Hui, and W. M. Tierney, "Validation techniques for logistic regression models." *Statistics in medicine*, vol. 10, no. 8, pp. 1213–1226, 1991.

[53] N. Mittas and L. Angelis, "Ranking and Clustering Software Cost Estimation Models through a Multiple Comparisons Algorithm," *IEEE Transactions on Software Engineering (TSE)*, vol. 39, no. 4, pp. 537–551, 2013.

[54] A. Mockus, "Organizational Volatility and its Effects on Software Defects," in *Proceedings of the International Symposium on Foundations of Software Engineering (FSE)*, 2010, pp. 117–127.

[55] A. Mockus and D. M. Weiss, "Predicting Risk of Software Changes," *Bell Labs Technical Journal*, vol. 5, no. 6, pp. 169–180, 2000.

[56] R. Moser, W. Pedrycz, and G. Succi, "A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2008, pp. 181–190.

[57] N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, and B. Murphy, "Change Bursts as Defect Predictors," in *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE)*, 2010, pp. 309–318.

[58] J. Nam, "Survey on software defect prediction," *Department of Compter Science and Engineerning, The Hong Kong University of Science and Technology, Tech. Rep*, 2014.

[59] J. Nam and S. Kim, "Heterogeneous defect prediction," in *Proceedings of the 2015 10th joint meeting on foundations of software engineering*. ACM, 2015, pp. 508–519.

[60] P. Nemenyi, "Distribution-free multiple comparisons," Ph.D. dissertation, Princeton University, 1963.

[61] L. Pelayo and S. Dick, "Applying novel resampling strategies to software defect prediction," *Annual Conference of the North American Fuzzy Information Processing Society - NAFIPS*, pp. 69–72, 2007.

[62] F. Rahman and P. Devanbu, "How, and Why, Process Metrics Are Better," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2013, pp. 432–441.

[63] F. Rahman, I. Herraiz, D. Posnett, and P. Devanbu, "Sample Size vs. Bias in Defect Prediction," in *Proceedings of the joint meeting of the European Software Engineering Conference and the symposium on the Foundations of Software Engineering (FSE)*, 2013, pp. 147–157.

[64] F. Rahman, D. Posnett, and P. Devanbu, "Recalling the "Imprecision" of Cross-Project Defect Prediction," in *Proceedings of the International Symposium on the Foundations of Software Engineering (FSE)*, 2012, pp. 61:1–61:11.

[65] F. Rahman, D. Posnett, A. Hindle, E. Barr, and P. Devanbu, "BugCache for Inspections: Hit or Miss?" in *Proceedings of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2011, pp. 322–331.

[66] J. Riquelme, R. Ruiz, D. Rodríguez, and J. Moreno, "Finding defective modules from highly unbalanced datasets," *Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos*, vol. 2, no. 1, pp. 67–74, 2008.

[67] G. Robles, "Replicating msr: A study of the potential replicability of papers published in the mining software repositories proceedings," in *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*. IEEE, 2010, pp. 171–180.

[68] D. Rodriguez, I. Herraiz, R. Harrison, J. Dolado, and J. C. Riquelme, "Preliminary comparison of techniques for dealing with imbalance in software defect prediction," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. ACM, 2014, p. 43.

[69] K. Rufibach, "Use of Brier score to assess binary predictions," *Journal of Clinical Epidemiology*, vol. 63, no. 8, pp. 938–939, 2010.

[70] W. S. Sarle, "The varclus procedure," in *SAS/STAT User's Guide. SAS Institute, Inc, 4th edition*, 1990.

[71] C. Seiffert, T. M. Khoshgoftaar, and J. Van Hulse, "Improving software-quality predictions with data sampling and boosting," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 39, no. 6, pp. 1283–1294, 2009.

[72] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Folleco, "An empirical study of the classification performance of learners on imbalanced and noisy software quality data," *Information Sciences*, vol. 259, pp. 571–595, feb 2014.

[73] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "Rusboost: A hybrid approach to alleviating class imbalance," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 40, no. 1, pp. 185–197, 2010.

[74] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the NASA software defect datasets," *IEEE Transactions on Software Engineering (TSE)*, vol. 39, no. 9, pp. 1208–1215, 2013.

[75] E. Shihab, "An Exploration of Challenges Limiting Pragmatic Software Defect Prediction," Ph.D. dissertation, Queen's Univer-

sity, 2012.

[76] E. Shihab, Z. M. Jiang, W. M. Ibrahim, B. Adams, and A. E. Hassan, "Understanding the Impact of Code and Process Metrics on Post-Release Defects: A Case Study on the Eclipse Project," in *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2010.

[77] E. Shihab, Y. Kamei, B. Adams, and A. E. Hassan, "High-Impact Defects: A Study of Breakage and Surprise Defects," in *Proceedings of the joint meeting of the European Software Engineering Conference and the symposium on the Foundations of Software Engineering (ESEC/FSE'11)*, 2011, pp. 300–310.

[78] E. W. Steyerberg, M. J. Eijkemans, F. E. Harrell Jr., and J. D. Habbema, "Prognostic modelling with logistic regression analysis: a comparison of selection and estimation methods in small data sets," *Statistics in Medicine*, vol. 19, pp. 1059–1079, 2000.

[79] E. W. Steyerberg, *Clinical prediction models: a practical approach to development, validation, and updating*. Springer Science & Business Media, 2008.

[80] E. W. Steyerberg, A. J. Vickers, N. R. Cook, T. Gerds, N. Obuchowski, M. J. Pencina, and M. W. Kattan, "Assessing the performance of prediction models: a framework for some traditional and novel measures," *Epidemiology*, vol. 21, no. 1, pp. 128–138, 2010.

[81] M. Tan, L. Tan, S. Dara, and C. Mayeux, "Online Defect Prediction for Imbalanced Data," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2015, pp. 99–108.

[82] C. Tantithamthavorn, "Towards a Better Understanding of the Impact of Experimental Components on Defect Prediction Modelling," in *Companion Proceedings of the International Conference on Software Engineering (ICSE)*, 2016, pp. 867–870.

[83] ——, "ScottKnottESD: The Scott-Knott Effect Size Difference (ESD) Test," https://cran.r-project.org/web/packages/ScottKnottESD/index.html, 2017.

[84] C. Tantithamthavorn and A. E. Hassan, "An experience report on defect modelling in practice: Pitfalls and challenges," in *In Proceedings of the International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP'18)*, 2018.

[85] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, A. Ihara, and K. Matsumoto, "The Impact of Mislabelling on the Performance and Interpretation of Defect Prediction Models," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2015, pp. 812–823.

[86] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated Parameter Optimization of Classification Techniques for Defect Prediction Models," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2016, pp. 321–322.

[87] ——, "Comments on "Researcher Bias: The Use of Machine Learning in Software Defect Prediction"," *IEEE Transactions on Software Engineering (TSE)*, 2016.

[88] ——, "An Empirical Comparison of Model Validation Techniques for Defect Prediction Models," *IEEE Transactions on Software Engineering (TSE)*, 2017.

[89] ——, "The Impact of Automated Parameter Optimization on Defect Prediction Models," *IEEE Transactions on Software Engineering (TSE)*, 2018.

[90] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, "Revisiting code ownership and its relationship with software quality in the scope of modern code review," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2016, pp. 1039–1050.

[91] L. Torgo, "DMwR: Functions and data for "Data Mining with R"," https://cran.r-project.org/web/packages/DMwR/index.html, 2013.

[92] B. Turhan, "On the dataset shift problem in software engineering prediction models," *Empirical Software Engineering*, vol. 17, no. 1-2, pp. 62–74, 2011.

[93] S. Wang and X. Yao, "Using Class Imbalance Learning for Software Defect Prediction," *IEEE Transactions on Reliability*, vol. 62, no. 2, pp. 434–443, 2013.

[94] R. Wu, H. Zhang, S. Kim, and S. C. Cheung, "ReLink: Recovering Links between Bugs and Changes," in *Proceedings of the joint meeting of the European Software Engineering Conference and the symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2011, pp. 15–25.

[95] X. Xia, D. Lo, E. Shihab, X. Wang, and X. Yang, "Elblocker: Predicting blocking bugs with ensemble imbalance learning," *Information and Software Technology*, vol. 61, pp. 93–106, 2015.

[96] X.-L. Yang, D. Lo, X. Xia, Q. Huang, and J.-L. Sun, "High-impact bug report identification with imbalanced learning strategies," *J. Comput. Sci. & Technol*, vol. 32, no. 1, 2017.

[97] X. Yang, D. Lo, Q. Huang, X. Xia, and J. Sun, "Automated identification of high impact bug reports leveraging imbalanced learning strategies," in *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*, vol. 1. IEEE, 2016, pp. 227–232.

[98] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction," in *Proceedings of the European Software Engineering Conference and the symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2009, pp. 91–100.

[99] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting Defects for Eclipse," in *Proceedings of the International Workshop on Predictor Models in Software Engineering (PROMISE)*, 2007, pp. 9–20.

APPENDIX

Tables 7, 8, 9 provide additional results with respect to RQ2.

TABLE 7: Statistics of the regression model of the relationship between the experimental factors and the performance difference of the *F-Measure* measure.

	Factor Analysis		
Adjusted R^2	0.50		
Optimism-reduced adjusted R^2	0.48		
Total Wald χ^2	837.51		
	D.F.	χ^2	p-value
Classification Technique	6	33%	***
Class Rebalancing Technique	3	33%	***
Defective Ratio	1	23%	***
EPV	1	8%	***
Metric Family	4	3%	***

Statistical significance of explanatory power according to Wald χ^2 likelihood ratio test: ∘ $p \geq 0.05$; * $p < 0.05$; ** $p < 0.01$; *** $p < 0.001$

TABLE 8: Statistics of the regression model of the relationship between the experimental factors and the performance difference of the *AUC* measure.

	Factor Analysis		
Adjusted R^2	0.36		
Optimism-reduced adjusted R^2	0.33		
Total Wald χ^2	525.54		
	D.F.	χ^2	p-value
Class Rebalancing Technique	3	77%	***
Classification Technique	6	17%	***
Metric Family	4	5%	***
Defective Ratio	1	0%	∘
EPV	1	0%	∘

Statistical significance of explanatory power according to Wald χ^2 likelihood ratio test: ∘ $p \geq 0.05$; * $p < 0.05$; ** $p < 0.01$; *** $p < 0.001$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2018.2876537, IEEE Transactions on Software Engineering

TABLE 9: Statistics of the regression model of the relationship between the experimental factors and the performance difference of the *MCC* measure.

	Factor Analysis		
Adjusted R^2	0.31		
Optimism-reduced adjusted R^2	0.28		
Total Wald χ^2	357.46		
	D.F.	χ^2	p-value
Classification Technique	6	59%	***
Class Rebalancing Technique	3	10%	***
Metric Family	4	10%	***
Defective Ratio	1	10%	***
EPV	1	11%	***

Statistical significance of explanatory power according to Wald χ^2 likelihood ratio test: ∘ $p \geq 0.05$; * $p < 0.05$; ** $p < 0.01$; *** $p < 0.001$

Chakkrit Tantithamthavorn is a lecturer at the Faculty of Information Technology, Monash University, Australia. Prior to that, he was a lecturer at School of Computer Science, the University of Adelaide, a research fellow at Queen's University, and a research fellow at Nara Institute of Science and Technology. During his Ph.D. study, he won one of the most prestigious and selective sources of national funding in Japan, i.e., a JSPS Research Fellowship for Young Researchers and a Grants-in-Aid for JSPS Fellow, and won a "Best Ph.D. Student Award". His work has been published at several top-tier software engineering venues, such as the IEEE Transactions on Software Engineering (TSE), the Springer Journal of Empirical Software Engineering (EMSE) and the International Conference on Software Engineering (ICSE). His Ph.D. thesis aims to improve the fundamentals of analytical modelling for software engineering in order to produce more accurate predictions and reliable insights. His research interests include empirical software engineering and mining software repositories (MSR). He received the B.E. degree in computer engineering from Kasetsart University, Thailand, the M.E. and Ph.D. degrees in Information Science from Nara Institute of Science and Technology, Japan. More about Chakkrit and his work is available online at http://chakkrit.com.

Ahmed E. Hassan is the Canada Research Chair (CRC) in Software Analytics, and the NSERC/BlackBerry Software Engineering Chair at the School of Computing at Queens University, Canada. His research interests include mining software repositories, empirical software engineering, load testing, and log mining. He received a PhD in Computer Science from the University of Waterloo. He spearheaded the creation of the Mining Software Repositories (MSR) conference and its research community. He also serves on the editorial boards of IEEE Transactions on Software Engineering, Springer Journal of Empirical Software Engineering, and PeerJ Computer Science. More about Ahmed and his work is available online at http://sail.cs.queensu.ca/.

Kenichi Matsumoto is a professor in the Graduate School of Information Science at Nara Institute of Science and Technology, Japan. He received the Ph.D. degree in information and computer sciences from Osaka University. His research interests include software measurement and software process. He is a fellow of the IEICE, a senior member of the IEEE, and a member of the ACM, and the IPSJ. More about Kenichi and his work is available online at http://se-naist.jp/.