The Significant Effects of Data Sampling Approaches on Software Defect Prioritization and Classification

Kwabena Ebo Bennin*, Jacky Keung*, Akito Monden†, Passakorn Phannachitta‡ and Solomon Mensah*
*Department of Computer Science, City University of Hong Kong, China
†Graduate School of Natural Science and Technology, Okayama University, Japan
‡College of Arts, Media and Technology, Chiang Mai University, Thailand
Email: kebennin2-c, Jacky.Keung, smensah2-c@my.cityu.edu.hk, monden@okayama-u.ac.jp, passakorn.p@cmu.ac.th

***Abstract*—Context: Recent studies have shown that performance of defect prediction models can be affected when data sampling approaches are applied to imbalanced training data for building defect prediction models. However, the magnitude (degree and power) of the effect of these sampling methods on the classification and prioritization performances of defect prediction models is still unknown. Goal: To investigate the statistical and practical significance of using resampled data for constructing defect prediction models. Method: We examine the practical effects of six data sampling methods on performances of five defect prediction models. The prediction performances of the models trained on default datasets (no sampling method) are compared with that of the models trained on resampled datasets (application of sampling methods). To decide whether the performance changes are significant or not, robust statistical tests are performed and effect sizes computed. Twenty releases of ten open source projects extracted from the PROMISE repository are considered and evaluated using the AUC, *pd*, *pf* and G-mean performance measures. Results: There are statistical significant differences and practical effects on the classification performance (*pd, pf* and G-mean) between models trained on resampled datasets and those trained on the default datasets. However, sampling methods have no statistical and practical effects on defect prioritization performance (AUC) with small or no effect values obtained from the models trained on the resampled datasets. Conclusions: Existing sampling methods can properly set the threshold between buggy and clean samples, while they cannot improve the prediction of defect-proneness itself. Sampling methods are highly recommended for defect classification purposes when all faulty modules are to be considered for testing.**

Keywords - Imbalanced data; Defect prediction; Sampling methods; Statistical significance; Empirical software engineering.

I. Introduction

Software defect prediction models predict buggy modules in a software system and aids in the efficient prioritization and allocation of scarce testing resources on the predicted buggy modules [31], [36], [14]. The performances of defect prediction models are known to be dependent on data used for training them [1], [11], [31]. The algorithms of most conventional prediction models are made for balanced datasets so as to maximize the overall classification accuracy [47], [51]. However, defect datasets are mostly imbalanced with the non-defective modules dominating the defective modules [14], [46], [12]. Models trained on imbalanced datasets will inadvertently predict most minority (defective) class instances of a new project as majority (non-defective) instances [39]. A high misclassification error is usually associated with the classifier on the rare occasions that the classifier predicts the minority class samples compared to predicting the majority class instances. Software quality teams are however, interested in a model with high prediction accuracy on the minority class [35], [33].

A popular approach for dealing with the imbalanced learning is the application of data sampling methods [2], [8], [13] to the dataset prior to training. The two primary methods, under-sampling and over-sampling alter the distribution of the data by reducing the majority class and increasing the minority class respectively. The first to conduct a study using sampling methods in Software Defect Prediction (SDP) were Kamei et al. [21] evaluated on an industrial dataset. They suggested that sampling methods can be effectively used for defect prediction. Subsequent studies [41], [38], [46] examined the effects of sampling methods on prediction models using open source projects from PROMISE repository. These previous studies all conclude that sampling methods does improve prediction performance but do not report whether the improvements are really practical or important for software defect prediction studies.

Most of these studies failed to perform statistical tests and compute effect sizes to support their results and conclusions (e.g. [21], [41], [38]) given that the results obtained could be heavily affected by the unstable variances and skewed class distribution of the datasets used as reported by Kitchenham et al. [24]. With the abundance of several sampling methods proposed in literature, knowing the actual degree and significant effect a sampling method could have on the performance of prediction models has real world benefits for Software Quality Assurance (SQA) teams. SQA teams will adopt the right sampling method for the efficient allocation of scarce resources for quality assurance activities such as code review efforts for testing all classified defective modules

978-1-5090-4039-1/17 $31.00 © 2017 IEEE
DOI 10.1109/ESEM.2017.50

or a subset (top-n) defective modules. Substantial evidence is thus required to validate the ability of sampling methods in improving performance of defect prediction models notwithstanding the prediction model or evaluation measure. Robust statistical significance tests together with an effect size metric that quantifies the strength or weakness of the relationship between investigated variables (sampling methods and prediction performance) is thus very necessary.

The main objective of this study is to investigate whether there are statistical differences and practical effects (with regards to prediction performances) when data sampling methods are applied to training data (resampled data) in building prediction models compared to the application of no sampling method (default data). We consider the practical effects with respect to the classification and prioritization of defective modules. As such, we structure our experiments to address the following two research questions:

1) **RQ1: Do sampling methods improve defect classification?** Yes, we observe a clear distinction between prediction performance (*pd, pf* and G-mean) for models trained on resampled data and models trained on default data based on the large and significant effect sizes obtained. Our results show that existing sampling methods could help in classifying all defective modules.
2) **RQ2: Do sampling methods improve defect prioritization?** No, Brunner's statistical test [6] results obtained on the AUC performance values show that, prediction models trained on resampled datasets could not outperform models trained on the default datasets. The effect sizes computed from Cliff's method with Hochberg's method [6] were insignificant indicating that existing sampling methods cannot improve the prediction of defect-proneness itself.

The experimental results show that all sampling methods significantly improved the G-mean and *pd* performance values whilst they increased *pf* for all prediction models with large effect sizes. Sampling methods had no statistical and practical effect on the AUC values. None of the sampling methods studied could concurrently increase *pd* and decrease *pf*. We encourage the use of sampling methods only for defective modules classification and not prioritization. To help improve defect prioritization, sampling methods which could concurrently achieve low *pf* and high *pd* is needed. Overall, the simple RUS method outperformed the complex SMOTE and it's variants. We thus recommend the use of RUS when large data samples of a particular project exist.

The organization of this paper is as follows. Section II reviews some previous studies on the application of sampling methods for defect prediction. Section III presents a brief background description of the sampling methods and the effect size measure used for the experiments and in Section IV, we describe the experimental methodology, data processing steps and performance evaluation measures. The empirical results and analysis is presented in Section V. Further analysis and discussions together with the threats to validity are presented in Section VI and conclusion and discussion of future works are presented in section VII.

II. Related Work

The problem of class imbalance makes it challenging to conduct experiments with datasets in the field of software defect prediction. Few studies have considered the effects of class imbalance and how to alleviate this issue in software defect prediction.

Kamei et al. [21] were the first to suggest the use of sampling methods as a preprocessing technique for SDP. They experimentally evaluated the effects of four sampling methods (ROS, SMOTE, RUS, ONESS) on four prediction models using two sets of industrial legacy software evaluated using the F1-value. Sampling methods improved performance for two models (Linear Discriminant Analysis and Logistic Regression) whilst the opposite was true for Neural Networks and Classification Trees. The best performing sampling method was different among the models and datasets. The limitation of this study is that they used a dataset which is not public and not readily available. Menzies et al. [37] empirically examined several training data for defect prediction models. They studied the effects of over-, under- and micro-sampling on prediction performance and concluded that these methods improve the information content within the training data and thus improve prediction performance. By conducting an empirical study considering five PROMISE projects, two prediction models and two sampling methods, Riquelme et al. [41] demonstrated that sampling methods especially SMOTE does improve the AUC performance when a fully balanced dataset was generated. Similarly, Pelayo and Dick [38] analysed the effects of SMOTE on C4.5 prediction model using four benchmarked datasets. They observed an improvement of at least 23% in the average geometric mean (G-mean).

A more comprehensive study by Wang and Yao [46] investigated the effects of different types of class imbalance learning methods comprising of sampling method (RUS), threshold moving and boosting-based ensemble algorithms together with two base classifiers (Naive Bayes and Random Forests). Using three evaluation measures and 10 PROMISE datasets, all methods improved performance. However, AdaBoost.NC, the ensemble technique had the best overall performance.

Our recent study [3] investigated the effects of sampling approaches on effort-aware prediction models. We conducted an empirical comparisons among 10 effort-aware models trained on 10 manually extracted open source software projects after resampling with four data sampling approaches and evaluated these models using the effort-aware measure Norm(P*opt*). Results showed that ROS outperformed the SMOTE approach and no sampling outperformed RUS when the dataset is highly imbalanced that is, has percentage of fault-prone modules (*Pfp*) less than 20%.

Most of these previous studies consider very few sampling methods for their comparative studies (e.g. [41], [38]) and fail to perform statistical tests and effect size computations to aid conclude the efficiency of these methods on the prediction

models. Benefits of using statistical test especially robust ones for evaluating software engineering empirical results were reported in the recent work of Kitchenham et al. [24]. They argue that these robust statistical test are very necessary given the fact that most real world data for software engineering experiments are either skewed, heavy-tailed or have unstable variances and could heavily impact the results produced from these datasets. The authors recommended non-parametric methods such as Brunner's test for statistical significance and Cliff's delta for effect size computation.

This study builds on and extends previous studies [21], [41], [38] by investigating additional oversampling approaches and using more prediction models. However, this study differentiates itself from [41], [38] in that, we use different datasets and conduct cross-release prediction. We further statistically evaluate the performance of the models trained on these resampled datasets to examine if the improvements were significant or not using robust statistical test recommended by Kitchenham et al. [24].

III. Background

In this section, we present a brief description of the sampling methods and the effect size metric adopted and used in our experiments. These methods are selected because they are the most commonly used in several defect prediction studies.

A. Data Sampling Approaches

Data sampling approaches are mostly considered as preprocessing procedures which are applied to imbalanced datasets by either increasing or decreasing data instances (samples) so as to alleviate the class imbalance before a prediction model is trained on the datasets. They comprise of over-sampling whereby minority data instances are increased and under sampling whereby majority data instances are decreased.

1) Random Over-sampling (ROS): The basic and simplest form of over-sampling is ROS. This method adds more minority instances to the datasets by randomly duplicating existing minority instances. It provides no new information to the trained classifier and is known to cause over-fitting. It is mostly preferred because it is easy to implement and requires no complex algorithm.

2) Random Under-sampling (RUS): RUS decreases the number of majority instances by randomly deleting already existing majority instances. As a simple and easy approach, the approach is repeated until the desired sampling rate is achieved. Due to the randomness of deleting, majority instances which provide vital information to the classifier in clearly identifying the decision boundary could be deleted.

3) Synthetic Minority Over-sampling TEchnique (SMOTE): SMOTE has been extensively applied in several defect prediction studies [42], [41]. Proposed by Chawla et al. [8], this technique over-samples the minority class in a dataset by generating new "synthetic" samples. This technique adopts the k-nearest neighbor approach to over-sample the minority class instances. To generate these "synthetic" samples, each minority class sample is considered and the new samples are introduced along the line segments that joins any of the *k* minority class nearest neighbors.

4) Adaptive Synthetic Sampling (ADASYN): ADASYN was proposed by He et al. [16] as an alternative over-sampling approach that focuses on the minority classes that are difficult to classify. Weights are assigned to the minority classes and dynamically adjusted so as to reduce the bias in the imbalanced dataset by considering the characteristics of the data distribution. The ADASYN algorithm incorporates a density distribution in automatically deciding the number of synthetic samples needed for each minority class instance resulting in unequal data samples being generated for each considered minority data instance.

5) Borderline-SMOTE: As a modified version of the SMOTE technique, this method proposed by Han et al. [15] mainly focuses on the harder-to-classify minority instances found on the border-line of a classifier. By adopting the K-nearest neighbor technique, minority data instances with more than a specified k-nearest neighbors of majority instances than minority instances are labeled and considered for over-sampling. The SMOTE technique is subsequently applied to the labeled instances. A major advantage of this method is the strengthening of the borderline between the majority and minority class samples.

B. Effect Size

Any indicator that measures the magnitude of a treatment effect is referred to as an effect size. Usually, the p-values obtained from statistical significance tests are affected by the sample size [45] hence, the need for effect sizes which provide a more practical significance and objective measure of the experimental effect, regardless of the statistical significance of the test statistics used [34], [24]. Effect sizes provides an appropriate evaluation measure for determining the magnitude and importance of the statistical results obtained [43]. The benefits of effect sizes has been extensively reviewed and reported by Tomczak and Tomczak [45]. Cliff's delta (δ) [9], [48] has been reported to outperform other effect size metrics such as Cohen's *d* and Hedges *g* statistics [28]. Cliff's method with Hochberg's method proposed by Brunner et al. [6] for controlling multiple tests was adopted for computing the effect sizes in our study. This method was recently recommended by Kitchenham et al. [23] as a more robust and appropriate choice for non-normal data since it is reliable when ties do exist within the data. The method, related to Cliff's delta uses the midranks ($\bar{R}$) of the treatments to calculate the test metric (p-hat($\widehat{P}$)) which represents the effect size and can be used to compute the delta value ($\widehat{\delta}$). A p-value is also produced which helps indicate if the effect size is statistically significant or not. The mathematical definitions of the effect size metric adopted for this study are presented as follows:

$$\bar{R}_j = \frac{i}{n_j} \sum_{i=1}^{n_j} R_{ij} \tag{1}$$

$$\widehat{P} = \frac{1}{N}(\bar{R}_1 - \bar{R}_2) + \frac{1}{2} \tag{2}$$

$$\widehat{\delta} = 1 - 2\widehat{P} \tag{3}$$

IV. Experimental Setting

This section describes the research questions, datasets, prediction models, performance measures and comparison criteria used for the study. In emulation of real-life setting, we used two sets of project datasets for training and testing. Detailed but simple experiments were conducted at different resampling rates (*Pfp*) to aid in the examination and analyses of the effects of sampling methods on the chosen prediction models.

A. Research Questions

Our empirical experiments are designed to address the following two research questions:

RQ1: Do sampling methods improve defect classification?

RQ2: Do sampling methods improve defect prioritization?

As we pointed out earlier in Section II, previous studies report that sampling methods do improve performance of defect prediction models but the practical effects and implications of theses sampling methods are mostly not reported. Jiang et al. [18] reports that the type of evaluation measure used for performance assessment of prediction models could have a practical value for software engineers. They report that measures such as the G-mean, F-measure, *pd, pf* and precision offer comparison of classification performance but not sufficient for model selection. RQ1 adopts some of these measures to examine the practical effects of sampling methods on classification performance on prediction models. On the other hand, for more safety critical systems with very limited resources available for verification and validation activities, graphical model evaluation approaches such as the AUC will aid in fault resolution and prioritization activities. RQ2 examines the effect of sampling methods on defect prediction models for software verification and prioritization activities.

B. Datasets

Twenty (20) versions of ten (10) open source software project datasets were extracted from the PROMISE data repository. Table I shows a summary of description of these datasets. These projects donated by Jurezcko and Madeyski [19], and Jurezcko and Spinellis [20] were written in JAVA where a module corresponds to a class. Twenty (20) static code metrics measured using BugInfo and ckjm[1] tools were extracted from each module. A non-defective module is labeled as zero and defective modules are labeled with the number of bugs present in the module. Table II presents the static code metrics. Previous or old release datasets (version A) were used as the train datasets whilst the latest or newer releases (version B) were used as the testing datasets. The datasets selected consist of very low and high imbalanced ratio or percentage of fault-prone modules (*Pfp*) with the training datasets having a *Pfp* ranging between 3.8-17.2%. These datasets have also been widely used in several empirical studies in the defect prediction domain [17], [19], [20], [4]

[1] http://www.spinelis.gr/sw/ckjm

$$Recall(pd) = \frac{TP}{TP + FN} \tag{4}$$

$$pf = \frac{FP}{FP + TN} \tag{5}$$

$$g - mean = \sqrt{\frac{TP}{TP + FN} * \frac{TN}{TN + FP}} \tag{6}$$

C. Evaluation Criteria

Evaluation of the predictive models are done considering the confusion matrix (Table III). From the table, TP is the number of true positives, FP is the number of false positives, FN is the number of false negatives and TN is the number of true negatives. We use Recall (*pd*), AUC, Geometric-mean and *pf*. These measures are known to be stable for evaluating performance of prediction models on imbalanced datasets and ranges between the values of 0-100%. Recall (*pd*) measures how much of the defective modules were actually detected. A higher *pd* denotes better performance. Probability of false alarm (*pf*) measures the rate of predicted modules that were non-defective or wrongly predicted as defective. A low or zero value for *pf* implies a better prediction model. For a highly imbalanced dataset, the Geometric-mean (G-mean) proposed by Kubat and Matwin [29] is preferred as it considers both the *pf* and *pd* values and does not value one measure over the other. Measures such as precision and F-measure were excluded since precision has been refuted by Menzies et al. [35] because they are unstable for highly imbalanced datasets. We also use the AUC measure because it does not sacrifice one class over the other and optimally handles the tradeoffs between the true and false positive error rates. The mathematical definitions of *pd*, *pf* and G-mean measures are presented in equations 4-6.

D. Experimental Setup

For the experiments, implementation of the ROS, RUS, SMOTE resampled datasets, all prediction models construction and evaluations were executed using the open source statistical processing tool, R [44]. The library packages WRS [49] and Caret [30] were used for the statistical tests and prediction model construction respectively. The ADASYN and Borderline-SMOTE generated datasets were implemented following the instructions given by the authors using MATLAB tool. We use all features or metrics for each training dataset as suggested by Menzies et al. [36] . For this study, five prediction models (Table IV) widely used in defect prediction studies [31], [32] were chosen. The parameter configurations for each of the prediction models are also presented in Table IV. We considered the Random Forests [31], C4.5 decision tree [40], K-NN algorithm [10], NNET [22] and SVM [50]. It should be noted that we predicted the presence of bugs in a module and not the number of bugs in a module. The experiments conducted are described below.

Table I
SUMMARY OF SELECTED 20 IMBALANCED DATASETS EXTRACTED FROM PROMISE DATA REPOSITORY

Project Name	Short Name	Version A				Version B			
		Release	Module	# Defects	Pfp(%)	Release	Module	# Defects	Pfp(%)
Ant1	ANT1	1.3	125	20	16	1.4	178	40	22
Ant2	ANT2	1.5	293	32	10.9	1.6	351	92	26.2
Camel1	CAM1	1.0	339	13	3.8	1.2	608	216	35.5
Camel2	CAM2	1.4	872	145	17	1.6	965	188	19.5
Forrest	FORR	0.7	29	5	17.2	0.8	32	2	6.2
Ivy	IVY	1.4	241	16	7	2.0	352	40	11.4
Jedit	JEDIT	4.2	367	48	13.1	4.3	492	11	2.2
Synapse	SYNA	1.0	157	16	10.2	1.1	222	60	27
Xalan	XALA	2.4	723	110	15.2	2.5	803	387	48.2
Xerces	XERC	1.2	440	71	16.1	1.3	453	69	15.2

Table II
DESCRIPTION OF STATIC CODE METRICS [17]

Abbreviation	Description
WMC	Weighted methods per class
DIT	Depth of Inheritance Tree
NOC	Number of Children
CBO	Coupling between object classes
RFC	Response for a Class
LCOM	Lack of cohesion in methods
LCOM3	Lack of cohesion in methods, different from LCOM
NPM	Number of Public Methods
DAM	Data Access Metric
MOA	Measure of Aggregation
MFA	Measure of Functional Abstraction
CAM	Cohesion Among Methods of Class
IC	Inheritance Coupling
CBM	Coupling Between Methods
AMC	Average Method Complexity
Ca	Afferent couplings
Ce	Efferent couplings
CC	McCabe's cyclomatic complexity
Max(CC)	Maximum value of CC methods of the investigated class
Avg(CC)	Arithmetic mean of the CC value in the investigated class
LOC	Lines of Code
Bug	Number of detected bugs in the class

Table III
CONFUSION MATRIX FOR TWO-CLASS CLASSIFICATION PROBLEM

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

We examined the performance of the defect prediction models when the 5 different sampling methods as described in Section III-A are applied to each project data at different *Pfp* rates (default, 20, 30, 40, 50) to create 5 independent training data. The default value indicates that no sampling method is applied and will be represented as NOS in our experiments. We stop resampling at 50% because our pilot experiments shows that application of sampling methods above 50% could cause serious over-fitting and RUS could totally deplete the training data. Considering 10 test and 10 training datasets, we conducted 10 cross-project experiments per each *Pfp* making 50 experiments for each prediction model in all

Table IV
PREDICTIVE MODELS AND THEIR HYPERPARAMETER CONFIGURATIONS

Model	Overview	Hyperparameters
C4.5	J48 Decision Tree	c ={0.05,0.1,0.2,0.3,0.4, 0.5,0.6,0.7}
NNET	3-layer Neural Network	size = {4,..,28}, decay = {0.1,0.2}
SVM	Support Vector Machine	c ={2^{-6},. . .,2^{10}}
RF	Random Forest	mtry= { 10, 50,100, 200, 250,500, 1000}
KNN	K-Nearest Neighbor	c ={2*(0,.....,7) + 1}

for each sampling method. After obtaining the training data, a 10-fold cross validation method is applied during the model construction. This method ensures that for a particular model training, the dataset is split into 10 folds where 90% of the dataset is used for training and the remaining 10% is used for testing. It is iterated until all folds in the dataset have been used once as a test dataset. After model training, the model is tested on the separate test data sets (version B) and evaluated using the performance measures discussed in Section IV-C. Algorithm 1 presents the procedure followed for the experiments. From the algorithm, steps 1 to 3 initializes the parameters used for the experiment. These include the input variables such as the source and target data, sampling methods, prediction models and *Pfp* rates. The process continues from step 4 through to step 17 where the classifiers are constructed on the resampled or modified training data (P_{pfp}) and then tested on the test data. For both SMOTE and ADASYN, we set *k*=5 as used by the original authors of these approaches [16], [8].

E. Performance Comparison

We compare the results of the performance measures of each prediction model on the original unmodified dataset (simple predictors) and the resampled data (advanced predictors). A superior sampling method should produce better results on the prediction model, notwithstanding the performance measure used. Sampling methods generating high values for *pd*, G-mean, AUC and low value for *pf* is thus preferred. We adopted the *win-tie-loss* statistics, one of the most prevalent statistical evaluation method used in several Software effort estimation

Algorithm 1 Outline of Experiment

Procedure Begin

1) For source projects P_{source}(Old version)={P_1 , . ., P_n } and target projects Q_{test}(New version)={Q_1 , . ., Q_j }, **repeat**
2) Initialize pf, pd, auc, g-mean = $\varnothing$; sampling=[SMOTE, ADASYN, BORDERLINE-SMOTE, ROS, RUS];
3) Initialize pfp=[default, 20, 30, 40, 50]; learner=[C4.5, SVM, KNN, NNET, RF] ;
4) **for** $j = 1, ..., length(P_{source})$ **do**
5) Select each train data $P_j$and its respective test data Q_j
6) **for** $k = 1, ..., length(sampling)$ **do**
7) **for** $i = 1, ..., length(pfp)$ **do**
8) Generate modified dataset ($P_{pfp[i]}$) from P_{source} where $P_{pfp[i]}$ = resample(P_j, sampling[k], pfp[i]);
9) **for** $l = 1, ..., length(learner)$ **do**
10) Train classifier learner$_l$ on modified data $P_{pfp[i]}$ using 10-fold cross-validation
11) Classify data points in Q_j using the classifier learner$_l$
12) Compute: *pf*, *pd*, AUC, G-mean for each classifier learner$_l$
13) **end for**
14) **end for**
15) **end for**
16) **end for**
17) **until** P_{source} is empty (i.e., the training data is exhausted)

End

studies [25], [26]. This procedure performs the statistical test by name Brunner's test [6] to find the significant differences in the performances of the models. This test is known to be more robust in comparison to other non-parametric tests [24]. Three counters by name *wins*, *ties* and *losses* are initially set for each predictor. For two pairs of selected predictors (combination of prediction model and sampling method), the statistical test is performed across all datasets for each performance measure and the *win*, *tie* or *loss* value is reported based on the significance value ($p < 0.05$). If the performance distributions of 2 predictors are not significantly different, the *ties* counters of both predictors are increased by 1; otherwise the *wins* counter of the significant predictor is increased by 1 and the *losses* counter of the other predictor also increased by 1. Lastly, we compute the effect size if a predictor is significant and outperformed other predictors using Cliff's method with Hochberg's method proposed by Brunner et al. [6] (Section III-B). The rationale behind Cliff's method is "What is the probability that a random observation from the set of simple predictors is greater than an observation from the set of advanced predictors?". The p-value obtained from this method helps indicate if the effect size computed is statistically significant or not. We interprete the effect sizes using the proposed magnitude labels of Kraemer and Kupfer [27] - small ($\widehat{\delta}$<=0.112), medium (0.112>$\widehat{\delta}$<0.428) and large ($\widehat{\delta}$>=0.428).

V. Experimental Results

We present the prediction results of sampling methods applied to the dataset at different *Pfp* rates. Due to space limitation, the median values of each performance metric and prediction model across all datasets are presented in Figure 1. In this figure, we compare the performance of the prediction models on the default dataset (NOS) with the resampled datasets. To statistically evaluate the effects of applying sampling methods on the prediction performance, we extract the total count of losses from the win-tie-loss statistic procedure explained in Section IV-E. Using this statistical procedure, we compared the prediction performance value between prediction models trained on NOS datasets (simple predictor) and prediction models trained on each sampling method applied to all datasets (advanced predictors) across 4 (20,30,40,50 %) *Pfp* rates. The total losses generated across all datasets for each prediction model and performance measure is presented in Table V. For a record in the table, the minimum loss is 0, which either indicates a predictor was never outperformed by its respective predictor being compared, and the maximum losses is 4 indicating the predictor lost across all 4 *Pfp* rates. Records with the lowest or zero losses indicate better performance. For more practical results, we present the effect sizes computed for each pairwise comparison between the simple predictor and each other advanced predictors in Figure 2. Similar to the *win-tie-loss* statistic procedure, we compute the effect size per each performance measure across all 4 *Pfp* values. If the effect size was significant based on the p-value, we report the magnitude of significance based on magnitude labels of Kraemer and Kupfer [27]. Based on the results presented in the Figures and table, we answer the following two research questions.

RQ1: Do sampling methods improve defect classification?

In this section, we measure classification performance using G-mean, *pd* and *pf* evaluation measures. From Figure 1, all sampling methods have an impact on the performance values of the models. It is evident that sampling methods performed better than no sampling across all prediction models except SVM with regards to the G-mean and *pd* performance values. With no sampling (NOS) as the baseline, all sampling methods had positive impacts on the prediction models except the ROS where it had a negative impact on the SVM model regarding the G-mean and *pd* values. However, the *pf* values were consistently high with the application of sampling methods. For statistical significance, we observe from Table V that for all performance measures except *pf*, application of different sampling methods did improve performance. On rare occasions did NOS outperform the other sampling methods. The losses are mostly reduced to a 0 or 1 after applying sampling methods. The large differences in the total losses after applying sampling methods across all performance measures indicates the positive influence of sampling methods. Several large effect sizes were observed for the G-mean and *pd* evaluation measures across

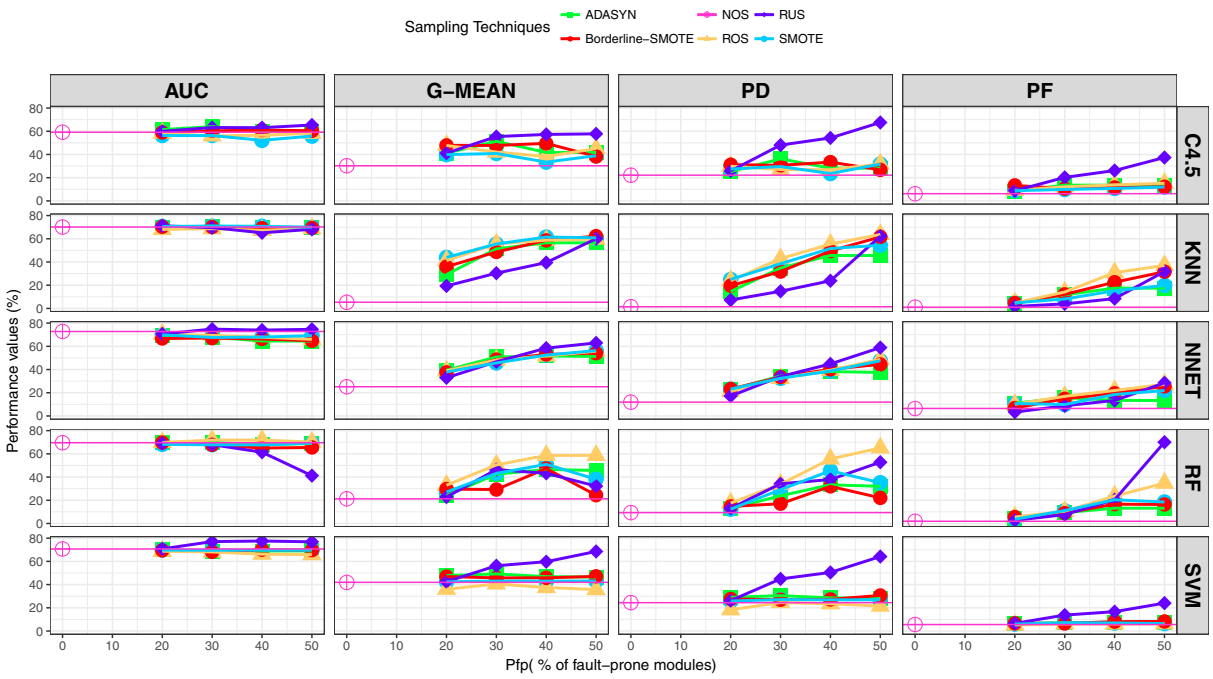

Figure 1. Performance plots (median values) for the sampling methods on different prediction models at different *Pfp* values across all ten cross-release project datasets

Table V
PERFORMANCE IN TERMS OF LOSSES AGGREGATED ACROSS FOUR *Pfp* VALUES ON THE 10 DATASETS. LOWER LOSSES VALUES INDICATES HIGHER PERFORMANCE. RECORDS HIGHLIGHTED IN LIGHT GRAY INDICATE THAT APPLICATION OF SAMPLING METHODS RESULTS IN BETTER PERFORMANCE WHILST THOSE IN DARK GRAY INDICATE OTHERWISE.

PERF	MODEL	TOTAL LOSSES									
		NOS	ADASYN	NOS	BSMOTE	NOS	ROS	NOS	RUS	NOS	SMOTE
AUC	C4.5	0	0	0	0	0	0	2	0	0	0
	KNN	0	0	0	0	0	1	0	1	1	0
	NNET	0	1	0	3	0	0	0	0	0	0
	SVM	0	0	0	0	0	0	0	1	0	0
	RF	0	0	0	0	0	0	0	0	0	0
GMEAN	C4.5	0	0	0	0	0	0	3	0	0	0
	KNN	4	0	4	0	4	0	4	0	4	0
	NNET	4	0	4	0	4	0	3	0	4	0
	SVM	3	0	1	0	3	0	2	0	2	0
	RF	0	0	0	0	0	1	3	0	0	0
PD	C4.5	1	0	1	0	0	0	3	0	0	0
	KNN	4	0	4	0	4	0	4	0	4	0
	NNET	4	0	4	0	4	0	3	0	4	0
	SVM	3	0	2	0	4	0	3	0	3	0
	RF	0	0	0	0	0	1	3	0	0	0
PF	C4.5	0	3	0	1	0	4	0	3	0	2
	KNN	0	4	0	4	0	4	0	1	0	4
	NNET	0	2	0	3	0	4	0	3	0	3
	SVM	0	4	0	3	0	4	0	3	0	4
	RF	0	0	0	0	0	0	0	3	0	1
Total Losses		23	14	20	14	23	19	33	15	22	14

most prediction models (Figure 2). The sampling methods had large and positive impacts on the KNN, NNET and SVM models as the *Pfp* rate increased. We can conclude based on the statistical and practical significance that applying sampling methods on prediction models does improve the classification performance of prediction models.

Application of sampling methods improved the G-mean and *pd* performance measures of prediction models and were statistically significant with large effect sizes in comparison to prediction models trained on datasets with no sampling method applied. Existing sampling methods can properly set the threshold between buggy and clean groups

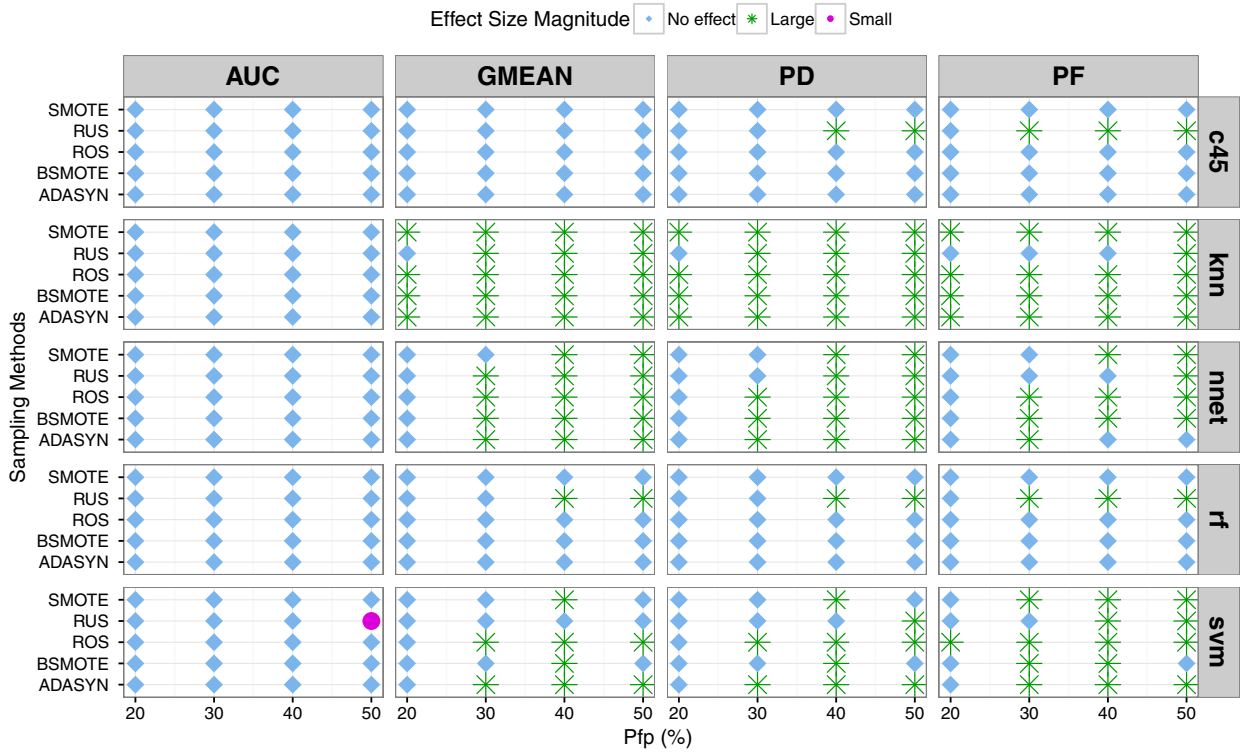

Figure 2. Performance plots for the effect sizes computed for each sampling methods compared to the no sampling method on different prediction models across all ten cross-project datasets per each *Pfp* values.

RQ2: Do sampling methods improve defect prioritization?

From Figure 1, for most of the prediction models, the AUC performance values of NOS prediction models seem to be no different from prediction models trained on datasets with sampling methods applied. The *pf* values were consistently low for the NOS models. The statistical tests results presented in Table V show very few losses even after applying sampling methods to the prediction models for the AUC measure. NOS significantly outperformed all sampling methods across all models regarding the *pf* performance measure. RUS and ROS produced the worst and significant increase in *pf*.

The effect size results (Figure 2) for the *pf* measure indicates that application of sampling methods on prediction models does increase the false alarms (*pf*) significantly since the effect sizes computed were large. From the figure, it was observed that all results obtained excluding one (RUS on SVM at *Pfp* of 50) were not practically significant when evaluated with the AUC measure. Application of sampling methods cannot improve the prediction of defect-proneness due to the high *pf* values making it difficult for engineers to rank and choose the top-n ranked modules for testing purposes.

Application of sampling methods significantly increased the *pf* performance values of prediction models with large effect sizes and had no significant effect on the AUC performance value in comparison to prediction models trained on datasets with no sampling method applied. Sampling methods do not significantly improve the detection of defective modules.

VI. Discussion

Practical Implications of Results: Our empirical results show the positive effects sampling methods have on prediction performance. With models trained on resampled datasets statistically outperforming the models trained on default dataset with regards to the G-mean and *pd*, it implies that most or all defective modules are correctly classified. However, with the high *pf* values, these modules will be contaminated with non-defective modules. Sampling methods are useful for Engineers/SQA teams that wish to classify faulty/clean modules so as to test all faulty modules.

With insignificant AUC results and high *pf* values produced by prediction models trained on resampled data, existing sampling methods are not useful to prioritize modules for quality assurance activities (such as testing and code review). Engineers/SQA teams that wish to pick top n% faulty modules for testing will struggle to select those faulty modules due to

the contamination of non-defective modules in the obtained results.

Comparisons with past findings: We revisited some previous studies and compared our findings to theirs. Riquelme et al. [41] reported that SMOTE does improve the AUC performance of prediction models. Our findings from Figure 2 does show some performance changes with respect to the AUC values but after conducting statistical tests and effect size computations, our results however reveals that although some sampling methods had significant AUC values, the effect sizes computed were impractical or insignificant across all models. We can thus conclude that evaluation of prediction models using the AUC measure is not affected by the sampling methods.

Compared with previous work, our results agree that sampling methods does improve prediction performance values of G-mean and *pd* values. Simple methods such as the ROS and RUS had very positive influence on all prediction models and could be used for fast computations. The results obtained with statistical significance from this experiments clearly shows the positive impacts sampling methods do have on prediction models and does reinforce that of previous studies [21], [38]: the use of resampling approaches does improve the classification performance compared to not using resampling approaches. With large effect sizes obtained for the G-mean and *pd* values, we recommend future software prediction studies to consider using sampling methods for classification purposes.

Threats to Validity As an empirical study, there are several potential limitations. We discuss below the internal and external threats to validity of the study. Conditions under which the experiments were performed relates to the internal threats to validity. The configuration parameters chosen for each prediction model had an impact on our results. Secondly, the amount of resampling or *Pfp* is a potential threat. With no agreed amount of resampling to be used for sampling methods in literature, we used low to high *Pfp* rates so as to eliminate any bias towards some datasets which are not severely imbalanced. Considering only open source software projects with static code metrics poses an external threat. These metrics extracted with automated softwares are easy to collect but the results cannot be generalized to all other projects that have different metrics apart from static code or projects in the commercial domain. Also, the number of projects considered could have an effect on our results. We however considered a wide range and sufficient sizes of projects and therefore produced robust results, which we expect to be similar for other unconsidered projects. We intend to consider commercial projects in future studies. A limited number of prediction models were also considered. However, these are widely used models in several defect prediction studies [41], [5], [38]. Considering different types of prediction models could have different implications.

VII. Conclusion and Future work

The purpose of this paper was to investigate the statistical and practical significance of using resampled data for constructing defect prediction models. Employing five different sampling techniques, we analyzed the performance of these sampling methods on prediction models evaluated with four different measures. There is an interesting relation between the choice of evaluation measure and the sampling method used. Evaluation measure that do not rely on the confusion matrix like the AUC was not influenced by the sampling method whilst the converse was true for those computed from a confusion matrix. Sampling methods had a significant and positive influence on prediction models when evaluated with the G-mean and *pd* performance measures. Effect sizes computed for the pairwise comparison between the simple predictors (models with no sampling method) and advanced predictors (models with sampling methods) were very large implying the importance of sampling methods in improving the classification performance of prediction models. Consequently, these sampling methods also significantly increases the *pf* which is very undesirable and impractical for defect prediction studies. The use of sampling methods for prioritization purposes is thus not recommended.

This work could be improved in several areas. We intend to investigate other simple and complex sampling methods [4], [7] and analyse its effect on the intra- and inter- release prediction performance using more prediction models and other metric based datasets. An in-depth analysis on the best rate of sampling (*Pfp*) to use for sampling methods is left for future studies.

VIII. Acknowledgement

This work is supported in part by the General Research Fund of the Research Grants Council of Hong Kong (No. 11208017 and 11214116) and the research funds of City University of Hong Kong (No. 7004683 and 7004474)

References

[1] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *Journal of Systems and Software*, vol. 83, no. 1, pp. 2–17, 2010.

[2] R. Barandela, J. S. Sánchez, V. Garcıa, and E. Rangel, "Strategies for learning in class imbalance problems," *Pattern Recognition*, vol. 36, no. 3, pp. 849–851, 2003.

[3] K. E. Bennin, J. Keung, A. Monden, Y. Kamei, and N. Ubayashi, "Investigating the effects of balanced training and testing datasets on effort-aware fault prediction models," in *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*, vol. 1. IEEE, 2016, pp. 154–163.

[4] 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*, 2017.

[5] K. E. Bennin, K. Toda, Y. Kamei, J. Keung, A. Monden, and N. Ubayashi, "Empirical evaluation of cross-release effort-aware defect prediction models," in *Software Quality, Reliability and Security (QRS), 2016 IEEE International Conference on*. IEEE, 2016, pp. 214–221.

[6] E. Brunner, U. Munzel, and M. L. Puri, "The multivariate nonparametric behrens–fisher problem," *Journal of Statistical Planning and Inference*, vol. 108, no. 1, pp. 37–53, 2002.

[7] C. Bunkhumpornpat, K. Sinapiromsaran, and C. Lursinsap, "Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2009, pp. 475–482.

[8] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, pp. 321–357, 2002.

[9] N. Cliff, "Dominance statistics: Ordinal analyses to answer ordinal questions." *Psychological Bulletin*, vol. 114, no. 3, p. 494, 1993.

[10] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[11] M. DAmbros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, no. 4-5, pp. 531–577, 2012.

[12] N. E. Fenton and N. Ohlsson, "Quantitative analysis of faults and failures in a complex software system," *IEEE Transactions on Software Engineering*, vol. 26, no. 8, pp. 797–814, 2000.

[13] S. García and F. Herrera, "Evolutionary undersampling for classification with imbalanced datasets: Proposals and taxonomy," *Evolutionary computation*, vol. 17, no. 3, pp. 275–306, 2009.

[14] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "The misuse of the nasa metrics data program data sets for automated software defect prediction," in *Proceedings of 15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011)*. IET, 2011, pp. 96–103.

[15] H. Han, W.-Y. Wang, and B.-H. Mao, "Borderline-smote: a new over-sampling method in imbalanced data sets learning," in *Advances in Intelligent Computing*. Springer, 2005, pp. 878–887.

[16] H. He, Y. Bai, E. Garcia, S. Li *et al.*, "Adasyn: Adaptive synthetic sampling approach for imbalanced learning," in *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*. IEEE, 2008, pp. 1322–1328.

[17] Z. He, F. Shu, Y. Yang, M. Li, and Q. Wang, "An investigation on the feasibility of cross-project defect prediction," *Automated Software Engineering*, vol. 19, no. 2, pp. 167–199, 2012.

[18] Y. Jiang, B. Cukic, and Y. Ma, "Techniques for evaluating fault prediction models," *Empirical Software Engineering*, vol. 13, no. 5, pp. 561–595, 2008.

[19] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*. ACM, 2010, p. 9.

[20] M. Jureczko and D. Spinellis, "Using object-oriented design metrics to predict software defects," *Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej*, pp. 69–81, 2010.

[21] Y. Kamei, A. Monden, S. Matsumoto, T. Kakimoto, and K.-i. Matsumoto, "The effects of over and under sampling on fault-prone module detection," in *First International Symposium on Empirical Software Engineering and Measurement, 2007. ESEM 2007.* IEEE, 2007, pp. 196–204.

[22] T. M. Khoshgoftaar, A. S. Pandya, and D. L. Lanning, "Application of neural networks for predicting program faults," *Annals of Software Engineering*, vol. 1, no. 1, pp. 141–154, 1995.

[23] B. Kitchenham, "Robust statistical methods: why, what and how: keynote," in *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*. ACM, 2015, p. 1.

[24] B. Kitchenham, L. Madeyski, D. Budgen, J. Keung, P. Brereton, S. Charters, S. Gibbs, and A. Pohthong, "Robust statistical methods for empirical software engineering," *Empirical Software Engineering*, pp. 1–52, 2016.

[25] E. Kocaguneli, T. Menzies, A. Bener, and J. W. Keung, "Exploiting the essential assumptions of analogy-based effort estimation," *IEEE Transactions on Software Engineering*, vol. 38, no. 2, pp. 425–438, 2012.

[26] E. Kocaguneli, T. Menzies, and J. W. Keung, "On the value of ensemble effort estimation," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1403–1416, 2012.

[27] H. C. Kraemer and D. J. Kupfer, "Size of treatment effects and their importance to clinical research and practice," *Biological psychiatry*, vol. 59, no. 11, pp. 990–996, 2006.

[28] J. D. Kromrey, K. Y. Hogarty, J. M. Ferron, C. V. Hines, M. R. Hess *et al.*, "Robustness in meta-analysis: An empirical comparison of point and interval estimates of standardized mean differences and cliffs delta," in *Joint Statistical Meetings. Minneapolis, MN*, 2005.

[29] M. Kubat, S. Matwin *et al.*, "Addressing the curse of imbalanced training sets: one-sided selection," in *ICML*, vol. 97. Nashville, USA, 1997, pp. 179–186.

[30] M. Kuhn, J. Wing, S. Weston, A. Williams, C. Keefer, A. Engelhardt, T. Cooper, and Z. Mayer, "caret: classification and regression training. r package version 6.0-24," 2014.

[31] S. Lessmann, 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*, vol. 34, no. 4, pp. 485–496, 2008.

[32] Y. Ma, L. Guo, and B. Cukic, "A statistical framework for the prediction of fault-proneness," *Advances in Machine Learning Application in Software Engineering, Idea Group Inc*, pp. 237–265, 2006.

[33] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Information and Software Technology*, vol. 54, no. 3, pp. 248–256, 2012.

[34] L. Madeyski, *Test-driven development: An empirical evaluation of agile practice*. Springer Science & Business Media, 2010.

[35] T. Menzies, A. Dekhtyar, J. Distefano, and J. Greenwald, "Problems with precision: A response to comments on data mining static code attributes to learn defect predictors," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, p. 637, 2007.

[36] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 2–13, 2007.

[37] T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic, and Y. Jiang, "Implications of ceiling effects in defect predictors," in *Proceedings of the 4th international workshop on Predictor models in software engineering*. ACM, 2008, pp. 47–54.

[38] L. Pelayo and S. Dick, "Applying novel resampling strategies to software defect prediction," in *Fuzzy Information Processing Society, 2007. NAFIPS'07. Annual Meeting of the North American*. IEEE, 2007, pp. 69–72.

[39] F. Provost, "Machine learning from imbalanced data sets 101," in *Proceedings of the AAAI 2000 workshop on imbalanced data sets*, 2000, pp. 1–3.

[40] J. R. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.

[41] 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.

[42] A. A. Shanab, T. M. Khoshgoftaar, R. Wald, and A. Napolitano, "Impact of noise and data sampling on stability of feature ranking techniques for biological datasets," in *Information Reuse and Integration (IRI), 2012 IEEE 13th International Conference on*. IEEE, 2012, pp. 415–422.

[43] G. M. Sullivan and R. Feinn, "Using effect sizeor why the p value is not enough," *Journal of graduate medical education*, vol. 4, no. 3, pp. 279–282, 2012.

[44] R. C. Team, "R: A language and environment for statistical computing," *Vienna, Austria: R Foundation for Statistical Computing*, 2012.

[45] M. Tomczak and E. Tomczak, "The need to report effect size estimates revisited. an overview of some recommended measures of effect size," *Trends in Sport Sciences*, vol. 21, no. 1, pp. 19–25, 2014.

[46] 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.

[47] G. M. Weiss and F. Provost, "The effect of class distribution on classifier learning: an empirical study," *Rutgers Univ*, 2001.

[48] R. Wilcox, *Modern statistics for the social and behavioral sciences: A practical introduction*. CRC press, 2011.

[49] R. Wilcox and F. Schoenbrodt, "Wrs: A compiled package of rr wilcoxrobust statistics functions," *R package version 0*, vol. 1, 2009.

[50] F. Xing, P. Guo, and M. R. Lyu, "A novel method for early software quality prediction based on support vector machine," in *Software Reliability Engineering, 2005. ISSRE 2005. 16th IEEE International Symposium on*. IEEE, 2005, pp. 10–pp.

[51] K. Yoon and S. Kwek, "A data reduction approach for resolving the imbalanced data issue in functional genomics," *Neural Computing and Applications*, vol. 16, no. 3, pp. 295–306, 2007.