Impact of the Distribution Parameter of Data Sampling Approaches on Software Defect Prediction Models

Kwabena Ebo Bennin*, Jacky Keung* and Akito Monden†
*Department of Computer Science, City University of Hong Kong, China
†Graduate School of Natural Science and Technology, Okayama University, Japan
Email: kebennin2-c, Jacky.Keung@my.cityu.edu.hk, monden@okayama-u.ac.jp

***Abstract*—Sampling methods are known to impact defect prediction performance. These sampling methods have configurable parameters that can significantly affect the prediction performance. It is however, impractical to assess the effect of all the possible different settings in the parameter space for all the several existing sampling methods. A constant and easy to tweak parameter present in all sampling methods is the distribution of the defective and non-defective modules in the dataset known as *Pfp* (% of fault-prone modules). In this paper, we investigate and assess the performance of defect prediction models where the *Pfp* parameter of sampling methods are tweaked. An empirical experiment and assessment of seven sampling methods on five prediction models over 20 releases of 10 static metric projects indicate that (1) Area Under the Receiver Operating Characteristics Curve (AUC) performance is not improved after tweaking the *Pfp* parameter, (2) *pf* (false alarms) performance degrades as the *Pfp* is increased. (3) a stable predictor is difficult to achieve across different *Pfp* rates. Hence, we conclude that the *Pfp* parameter setting can have a large impact on the performance (except AUC) of defect prediction models. We thus recommend researchers experiment with the *Pfp* parameter of the sampling method since the distribution of training datasets vary.**

Keywords - Search based SE, Defect prediction, Sampling methods, Imbalanced Data, Preprocessing, Empirical software engineering.

I. Introduction

Prioritization of testing resources is a key challenge of any software engineering project. Allocating the same testing efforts to all modules of a software is not ideal since defects are not uniformly distributed within the software. Software quality teams have adopted defect prediction models to predicts buggy modules in software and this helps in prioritization of scarce testing resources [24], [26]. The effectiveness of these prediction models are however dependent on the data used in training them [2], [11], [24].

Defect datasets are typically imbalanced with the non-defective samples dominating the defective samples [13], [31], [12]. In SE research, the prevalent and easy approach for solving the class imbalance issue is the application of data sampling methods [3], [10], [5]. These sampling methods have configurable parameters that control the characteristics of the datasets produced for training classifiers. For example, the number of neighbors (K) considered for data generation in SMOTE can be configured prior to being applied on the training data. Different sampling methods have different configurable parameters and it is impractical to configure all these parameters of the various sampling methods during model construction. Nevertheless, a constant parameter for all sampling method is the amount of resampling or percentage of fault-prone modules (*Pfp*) which controls the amount of data added to or removed from the minority class samples or the majority class samples respectively.

Our literature analysis reveal that prior defect prediction studies that apply sampling methods as preprocessing techniques do not consider the impact of these parameters and its effect on prediction performance. Agrawal and Menzies [1] note that when the parameters of SMOTE are tuned based on the data distribution, significant improvements could be observed. Most sampling methods apply a fixed balanced resampling rate since an optimal *Pfp* rate for these existing sampling methods is a challenge. Our previous study [4] shows that the application of a fixed *Pfp* for sampling models may degrade performance of prediction models due to the difference in imbalance ratio and size (number of samples) of training datasets.

In this paper, we investigate and asses the significant effect of tuning the *Pfp* rate of sampling methods on performances of defect prediction models. Different *Pfp* rates are selected and configured for each sampling method before being applied to the training datasets prior to classifier training. We conduct an extensive experiment on seven sampling methods and five prediction models evaluated with five performance measures across 20 releases of 10 open-source software projects. We further perform a mini ranking experiment to obtain the best performing predictors. Our analysis indicate that sampling methods have a strong correlation with *Pfp*. The stable ranking experiments show that a stable predictor is difficult to achieve across different evaluation measures. No predictor was stable across the *Pfp* rates. In conclusion, we observe that performance of all sampling methods studied can be improved when the parameters are tuned and encourage the tuning of this parameter.

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 describes the experimental methodology, sampling methods and performance

978-1-5386-3681-7/17 $31.00 © 2017 IEEE
DOI 10.1109/APSEC.2017.76

evaluation measures. The empirical results and analysis is presented in Section IV. The threats to validity are presented in Section V and conclusion and discussion of future works are presented in section VI.

II. Related Work

Several studies have assessed the impact of sampling methods on defect prediction performance. There are however contradictory findings about the performance of these sampling methods especially concerning SMOTE.

These studies report the positive impact sampling methods have on prediction models. Kamei et al. [19] experimentally evaluated the effects of four sampling methods on four prediction models using two sets of industrial legacy software evaluated using the F1-value. They observed that sampling methods improved the performance of two out of four models. Similarly, Menzies et al. [27], Riquelme et al. [29], Pelayo and Dick [28] and Wang and Yao [31] concluded that sampling methods improve prediction performance. These studies applied the sampling methods using the default parameter values. In our recent study [4], we investigated the effects of four sampling approaches on ten effort-aware prediction models where we tuned the resampling parameters. Results showed that ROS outperformed the SMOTE approach and no sampling outperformed RUS when the dataset is highly imbalanced. The recent study by Agrawal and Menzies [1] where they tuned several parameters of SMOTE showed that performance can be further improved when these parameters are tuned according to the distribution of the training data. This study differentiates itself from previous studies in that, we assess the performance of more than five sampling methods when the *Pfp* parameter is tweaked. Additionally, we conduct cross-release predictions with five performance measures. 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. [20].

III. Experimental Setting

To assess the effect of tweaking the distribution parameter of sampling methods on prediction performance, we briefly discuss the sampling methods, datasets, performance measures and comparison criteria used for the study. In emulation of real-life setting, two sets of project datasets are used for the experiments in order to examine the impact of applying sampling methods on the performance of the cross-release project prediction models. 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. Sampling Methods

Data sampling approaches were proposed with the aim of increasing the sample size of the minority class or decreasing the sample size of the majority class referred to as oversampling and undersampling respectively. Seven sampling methods were considered for this experiments.

RUS: Random Undersampling is the simplest form of undersampling. Majority instances of the dataset are randomly selected and deleted whereas the minority instances are left intact. Information crucial to the classifier during training could be lost since instances are deleted. It is however very fast to execute for large datasets.

ROS: As a very simple and effective oversampling method, Random Oversampling increases the minority sample by randomly selecting and duplicating minority instances. This method provides no "new" information to the classifier.

SMOTE: Proposed by Chawla et al. [10], 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. SMOTE is one of the most popularly used sampling method in SE studies.

ADASYN: An adaptive oversampling method proposed by He et al. [15], ADASYN 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 data.

Borderline-SMOTE: A modification of SMOTE proposed by Han et al. [14], this method mainly focuses on the harder-to-classify minority instances found on the border-line of a classifier. SMOTE is then applied to the instances labelled as "bordeline".

Safe-Level SMOTE: Originally proposed by Bunkhumpornpat et al. [9], this method aims to avoid the generation of synthetic minority samples in the region of the majority instances. Samples are labelled as "Safe-Level" before the conventional SMOTE method is applied. Safe-level is defined as the number of majority instances among the k nearest neighbor of a minority instance.

MAHAKIL: This is a recent oversampling approach based on the chromosomal theory of inheritance developed by Bennin et al. [5]. MAHAKIL aims to generate diverse minority instances by obtaining different traits from two dissimilar instances. MAHAKIL considers two unique and dissimilar instances (i.e., based on Mahalanobis distance values) for the generation of new instances. MAHAKIL alleviates overgeneralization of prediction models since synthetic data instances created are not clustered into a specific region of the minority class of the dataset.

B. Datasets

Twenty (20) versions of ten (10) JAVA open source software project datasets donated by Jurezcko and Madeyski [17], and Jurezcko and Spinellis [18] were extracted from the PROMISE data repository. A summary description of the projects are presented in Table I. 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%.

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

$$Recall(pd) = \frac{TP}{TP + FN} \tag{1}$$

$$pf = \frac{FP}{FP + TN} \tag{2}$$

$$g - mean = \sqrt{\frac{TP}{TP + FN} * \frac{TN}{TN + FP}} \tag{3}$$

$$balance(bal) = 1 - \frac{\sqrt{(0 - pf)^2 + (1 - pd)^2}}{\sqrt{2}} \tag{4}$$

Table II
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}

C. Evaluation Criteria

Five commonly used performance metrics are adopted for the study. The AUC, g-mean, balance, *pd* and *pf*. The first three metrics measure the performance of the overall classifier while the last two are used for assesing the performance of the minority(defective) class. Higher AUC, g-mean, balance and *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. Detailed explanations of these measures can be found in [16]. The mathematical definitions of *pd*, *pf*, balance and g-mean measures are presented in equations 1-4.

D. Experimental Setup

The experiments reports the performance of the defect prediction models when 7 different sampling methods are applied to each project data at different *Pfp* rates (default, 20, 30, 40, 50). This results in five independent training dataset (resampled data) per each sampling method and 350 (5 *Pfp* * 10 datasets * 7 sampling methods) cross-release predictions in total. We represent the default distribution value as No Sampling (NONE) in our experiments. In agreements with the assumptions of classifiers that the dataset should be balanced, we stop resampling at 50%. Also, resampling above 50% could totally deplete the dataset for RUS. Five commonly used prediction models in defect prediction [24], [25], [6] are selected for the experiment. They are presented in Table II together with the parameter configurations used. The experiments were executed with the open source statistical processing tool (R) [30]. The Caret [23] and WRS [32] library packages were used for the prediction model construction and statistical analyses respectively. According the recommendations of Menzies et al. [26], all the features (metrics) of the datasets are used during the model training. We strictly apply the parameters used by the original authors of all synthetic oversampling methods except the distribution values.

During the model construction, the 10-fold cross-validation method is applied to each resampled data. 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. The constructed model is tested on the separate test dataset (version B) and evaluated using the performance measures discussed in Section III-C.

E. Statistical Performance Comparison

The *win-tie-loss* evaluation procedure mostly used in several Software effort estimation studies [21], [22] is adopted for the performance comparison. This procedure employs a statistical test for a pairwise comparison between predictors . Brunner's statistical test [8] is applied for significant differences since this test was recently recommended by Kitchenham et al. [20] as a more robust non-parametric test. Three counters *wins*, *ties* and *losses* are initially set for each predictor. For any two pairs of selected predictors, 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. Figure 1 presents the *win-tie-loss* procedure used.

$win_k = 0, tie_k = 0, loss_k = 0$
$win_l = 0, tie_l = 0, loss_l = 0$
if *Brunner-test*$(Perf_k, Perf_l, 0.95)$ says they are the same **then**
  $tie_k = tie_k + 1$
  $tie_l = tie_l + 1$
else
  if better$(Perf_k, Perf_l)$ **then**
  $win_k = win_k + 1$
  $loss_l = loss_l + 1$
else
  $win_k = win_k + 1$
  $loss_l = loss_l + 1$
  end if
end if

Figure 1. Pseudocode for win-tie-loss computation between predictors *k* and *l* with performance measure values $Perf_k$ and $Perf_l$. Note that *pf* values are negated before comparison since the lower the *pf* value, the better.

IV. Experimental Results and Discussions

The experiment evaluates the effects of changing the *Pfp* on the prediction performance of seven sampling methods. The performance of each prediction model when sampling methods are applied are compared to the prediction model with no sampling (default *Pfp*) across the *Pfp* values. Figure 2 presents a plot of median performance values for each sampling method and prediction model across all ten static-metric datasets. By using the NONE as the benchmark, we observe change in performances as *Pfp* increases. AUC performance values are not significantly improved as *Pfp* increases. The AUC values for all sampling methods were almost the same as NONE. All other performance values improved with the exception of *pf* as *Pfp* increases for all the sampling methods.

For statistical significant results, we present the prediction results of sampling methods applied to the dataset at different *Pfp* rates by extracting the total count of wins and losses from the win-tie-loss statistic procedure explained in Figure 1. For each *Pfp* rate, comparisons are made among the five performance values of each sampling method and prediction model for every single dataset. We present the win-tie-loss results where each predictor is compared with 39 (5 models * 8 sampling methods - 1) other predictors for each *Pfp* rate in Table III. We focus on the wins and losses of each predictor and thus exclude the tie values from the results. From the table, the best ranked predictors differed at each *Pfp* rate. A summary of the results from the tables are presented as follows:

1) Across all *Pfp* rates, combination of no sampling method (NONE) with all the five prediction models were consistently ranked in the bottom half of the table, suggesting that prediction models perform worse when no sampling (NONE) methods are applied to imbalanced datasets.
2) A stable predictor could not be observed across all four *Pfp* rates. The best predictor varied across the *Pfp* rates.
3) Sampling methods significantly improved the KNN prediction model whilst the C4.5 prediction model is the least impacted by sampling methods.

The win-tie-loss results indicate sampling methods are more effective in improving performance of prediction models. However, the best sampling method to use differs with respect to the *Pfp* rate applied and evaluation metric. It is thus difficult to find a stable *Pfp* rate to use for defect prediction research and we recommend using different predictors based on the evaluation metric used. For each dataset and sampling method, several experiments should be conducted with different *Pfp* values where (default>*Pfp*<=50). The *Pfp* value resulting in the highest performance should be used. Our results is in agreement with the observations made by Bowes et al. [7] that there are uncertainties in the predictions made by classifiers. The instability in the prediction performances among the classifiers caused by the *Pfp* rates provides practical implications for practitioners. In safety critical scenarios where false alarms are costly, *Pfp* rates similar to the default distribution rate of the dataset is recommended for most sampling methods. From our results, we recommend KNN prediction model in combination with MAHAKIL or Safe-level SMOTE since it performed very well statistically irrespective of the *Pfp* rate across all performance measures. The results suggest that the number of minority samples introduced into the training data has significant effect on prediction performance.

V. Threats to Validity

As an empirical study, there are several potential limitations. We considered only 10 open-source projects with static metrics for the experiments. The results may not generalize for commercial projects and a replication study using commercial projects is left for a future study. The default imbalance ratio and project sizes span from low to high values. These imbalance values were considered so as to reduce any bias of specific datasets dominating and influencing the results. The effect of sampling methods on very large project sizes will be explored in a future work. We carefully selected a large number of performance measures and applied the robust statistical test recommended by Kitchenham et al. [20] for the experiment. The threat to statistical conclusion is thus limited in our study. We considered a limited number of prediction models. These models are however widely used in defect prediction studies. We acknowledge different prediction models could have different implications. The configuration parameters chosen for the prediction models could also have an impact on the prediction results.

VI. Conclusion and Future work

Sampling methods applied to training datasets prior to model construction are controlled by configurable parameters. Exploring all the accessible parameter for the various sampling methods is however impractical. Recent studies conclude that performance of prediction models are influenced by the

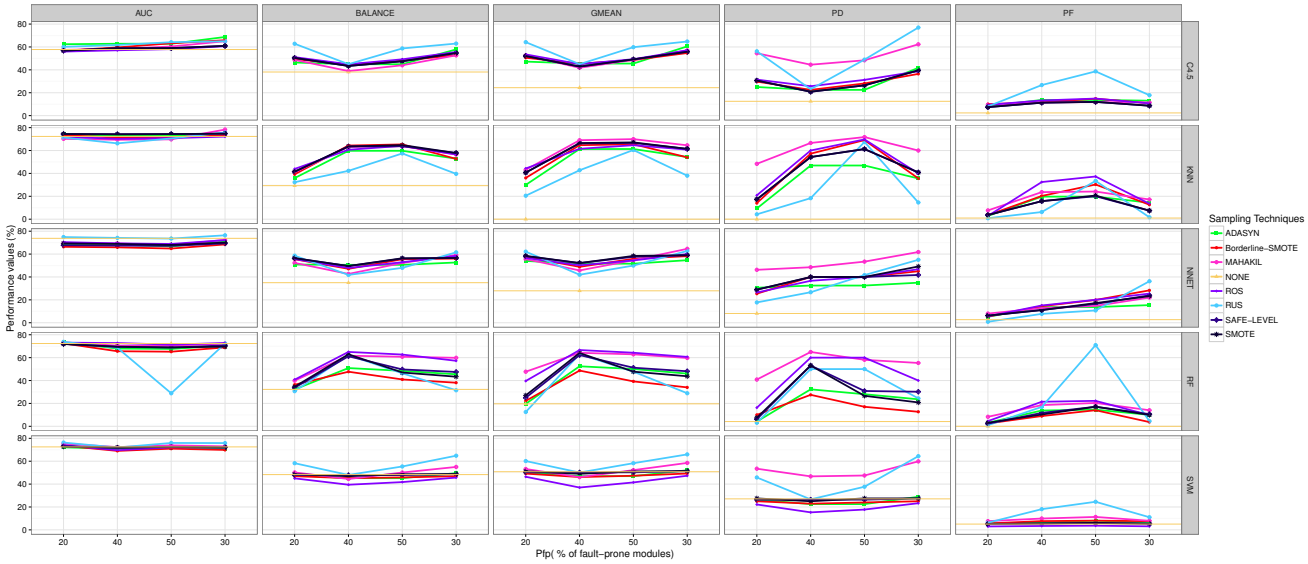

Figure 2. Performance plots (median values) for all performance measures of the sampling techniques on different Prediction Models at different *Pfp* values across all ten cross-release datasets

Table III
PERFORMANCE IN TERMS OF WINS, LOSSES, AND WINS-LOSSES AGGREGATED FROM ALL FIVE PERFORMANCE MEASURES ON THE 10 DATASETS. HIGHER WINS AND WINS-LOSSES INDICATE HIGHER PERFORMANCE, WHERE AS LOWER LOSSES INDICATE HIGHER PERFORMANCE. THE RANKS ARE ORDERED BY WINS-LOSSES

Pfp 20					*Pfp* 30					*Pfp* 40					*Pfp* 50				
Model	**Sampling**	**Wins**	**Losses**	**Wins-Losses**	**Model**	**Sampling**	**Wins**	**Losses**	**Wins-Losses**	**Model**	**Sampling**	**Wins**	**Losses**	**Wins-Losses**	**Model**	**Sampling**	**Wins**	**Losses**	**Wins-Losses**
RF	MAHAKIL	51	8	43	KNN	SAFE-LEVEL	64	6	58	KNN	SAFE-LEVEL	96	14	82	KNN	MAHAKIL	117	23	94
RF	ADASYN	46	9	37	KNN	SMOTE	64	6	58	KNN	SMOTE	96	14	82	RF	RUS	99	22	77
RF	BORDERLINE	46	9	37	SVM	MAHAKIL	66	15	51	KNN	MAHAKIL	83	23	60	KNN	SMOTE	92	17	75
SVM	MAHAKIL	49	12	37	RF	RUS	59	15	44	RF	RUS	75	15	60	KNN	SAFE-LEVEL	91	17	74
RF	SAFE-LEVEL	46	10	36	KNN	MAHAKIL	69	31	38	KNN	BORDERLINE	73	26	47	SVM	MAHAKIL	75	20	55
RF	NONE	46	10	36	RF	ADASYN	36	7	29	SVM	MAHAKIL	68	23	45	KNN	BORDERLINE	83	31	52
NNET	MAHAKIL	38	6	32	KNN	ADASYN	40	17	23	KNN	ADASYN	65	21	44	SVM	ROS	70	23	47
RF	RUS	41	10	31	NNET	RUS	30	7	23	SVM	ROS	62	21	41	KNN	ADASYN	62	23	39
KNN	MAHAKIL	42	12	30	RF	SAFE-LEVEL	37	14	23	RF	MAHAKIL	49	12	37	NNET	SMOTE	64	25	39
RF	SMOTE	40	10	30	RF	MAHAKIL	38	15	23	NNET	RUS	35	6	29	NNET	MAHAKIL	59	26	33
KNN	SAFE-LEVEL	36	10	26	RF	NONE	45	24	21	KNN	ROS	65	40	25	NNET	RUS	55	23	32
KNN	SMOTE	36	10	26	SVM	ROS	30	9	21	C45	RUS	45	24	21	KNN	ROS	68	37	31
NNET	RUS	34	8	26	KNN	ROS	40	20	20	RF	SMOTE	43	32	11	RF	MAHAKIL	56	26	30
NNET	ADASYN	26	12	14	RF	BORDERLINE	32	15	17	SVM	SAFE-LEVEL	27	16	11	C45	RUS	60	32	28
KNN	BORDERLINE	19	9	10	SVM	RUS	26	9	17	SVM	SMOTE	28	17	11	NNET	SAFE-LEVEL	56	29	27
KNN	ROS	22	13	9	NNET	MAHAKIL	41	26	15	RF	ADASYN	40	30	10	KNN	RUS	47	27	20
NNET	ROS	20	11	9	RF	SMOTE	27	13	14	RF	SAFE-LEVEL	44	34	10	NNET	ROS	50	38	12
NNET	BORDERLINE	21	13	8	NNET	SAFE-LEVEL	26	15	11	NNET	SMOTE	38	30	8	C45	MAHAKIL	47	45	2
SVM	ROS	14	12	2	SVM	SAFE-LEVEL	22	11	11	RF	NONE	49	46	3	RF	BORDERLINE	41	41	0
NNET	SAFE-LEVEL	16	15	1	NNET	ADASYN	20	11	9	SVM	RUS	16	14	2	RF	SMOTE	46	52	-6
NNET	SMOTE	16	16	0	C45	RUS	30	22	8	NNET	ADASYN	22	22	0	SVM	SAFE-LEVEL	20	28	-8
SVM	RUS	41	43	-2	NNET	SMOTE	22	14	8	NNET	MAHAKIL	38	40	-2	RF	NONE	48	57	-9
RF	ROS	24	29	-5	NNET	ROS	20	18	2	RF	ROS	38	42	-4	SVM	SMOTE	20	30	-10
C45	ADASYN	18	34	-16	NNET	BORDERLINE	21	20	1	SVM	ADASYN	25	31	-6	RF	ADASYN	43	54	-11
SVM	SMOTE	26	42	-16	RF	ROS	26	25	1	RF	BORDERLINE	36	44	-8	RF	SAFE-LEVEL	44	55	-11
SVM	SAFE-LEVEL	26	44	-18	SVM	SMOTE	21	20	1	NNET	BORDERLINE	21	33	-12	NNET	BORDERLINE	47	61	-14
C45	RUS	24	44	-20	KNN	BORDERLINE	26	29	-3	NNET	ROS	22	34	-12	NNET	ADASYN	27	48	-21
SVM	NONE	45	65	-20	C45	ADASYN	21	31	-10	NNET	SAFE-LEVEL	22	36	-14	SVM	ADASYN	33	56	-23
SVM	BORDERLINE	16	37	-21	C45	MAHAKIL	31	48	-17	C45	MAHAKIL	37	58	-21	RF	ROS	37	66	-29
C45	BORDERLINE	36	58	-22	SVM	ADASYN	18	36	-18	C45	BORDERLINE	20	42	-22	C45	SAFE-LEVEL	24	57	-33
C45	ROS	35	59	-24	C45	SAFE-LEVEL	14	39	-25	SVM	BORDERLINE	15	51	-36	C45	SMOTE	24	57	-33
C45	SAFE-LEVEL	21	46	-25	C45	SMOTE	14	39	-25	KNN	RUS	32	69	-37	C45	ROS	22	72	-50
C45	SMOTE	21	46	-25	C45	BORDERLINE	17	49	-32	C45	ADASYN	21	63	-42	C45	BORDERLINE	23	74	-51
C45	MAHAKIL	38	66	-28	C45	ROS	12	59	-47	C45	NONE	27	75	-48	C45	ADASYN	31	83	-52
C45	NONE	8	36	-28	C45	NONE	19	68	-49	SVM	NONE	48	101	-53	C45	NONE	27	80	-53
KNN	ADASYN	19	48	-29	KNN	RUS	37	91	-54	C45	ROS	13	77	-64	SVM	NONE	46	99	-53
SVM	ADASYN	21	55	-34	SVM	NONE	45	103	-58	KNN	NONE	50	115	-65	NNET	NONE	36	92	-56
KNN	RUS	42	77	-35	NNET	NONE	27	96	-69	NNET	NONE	33	98	-65	KNN	NONE	47	113	-66
NNET	NONE	18	62	-44	SVM	BORDERLINE	14	83	-69	C45	SAFE-LEVEL	13	79	-66	SVM	BORDERLINE	16	83	-67
KNN	NONE	42	112	-70	KNN	NONE	47	118	-71	C45	SMOTE	13	79	-66	SVM	RUS	15	128	-113

imbalance ratio of the training datasets. This imbalance ratio (*Pfp*) can be controlled when sampling methods are applied. To assess the effect of *Pfp* on sampling and prediction performance, we examine the statistical significant performance of seven sampling methods on five prediction models across 20 releases of 10 open source projects. Sampling methods

had a significant and positive influence on prediction models across all *Pfp* values. AUC values were not significantly improved by sampling methods whilst sampling methods significantly increases the false alarms (*pf*) as *Pfp* increased. The experimental results indicate that *Pfp* has significant effect on prediction performance. This suggest that not only is the sampling method important but the *Pfp* selected is important as well. It is however difficult to find a stable *Pfp* value to use for defect prediction studies. This depends on several factors with the default imbalance ratio of the dataset being the major factor. We suggest researchers apply different *Pfp* values and select the rate that produces the best performance. This work could be improved in several areas. We intend to investigate in detail how and why some sampling methods perform better than others. An in-depth analysis on the best rate of sampling (*Pfp*) to use for sampling methods is also left for future studies.

VII. Acknowledgement

and This work is supported in part by the General Research Fund of the Research Grants Council of Hong Kong (No. 11208017 and 11214116), the research funds of City University of Hong Kong (No. 7004683) and JSPS KAKENHI Grant number 17K00102.

References

[1] A. Agrawal and T. Menzies, “" better data" is better than" better data miners"(benefits of tuning smote for defect prediction),” *arXiv preprint arXiv:1705.03697*, 2017.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] D. Bowes, T. Hall, and J. Petrić, “Software defect prediction: do different classifiers find the same defects?” *Software Quality Journal*, Feb 2017. [Online]. Available: https://doi.org/10.1007/s11219-016-9353-3

[8] 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.

[9] 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.

[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*, pp. 321–357, 2002.

[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] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

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

[31] 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.

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