2016 IEEE 40th Annual Computer Software and Applications Conference

Investigating the Effects of Balanced Training and Testing Datasets on Effort-Aware Fault Prediction Models

Kwabena Ebo Bennin
and Jacky Keung
Department of Computer Science
City University of Hong Kong
Hong Kong, China
kebennin2-c@my.cityu.edu.hk
Jacky.Keung@cityu.edu.hk

Akito Monden
Department of Computer Science
Division of Industrial Innovation Sciences
Okayama University
Okayama, Japan
monden@okayam-u.ac.jp

Yasutaka Kamei
and Naoyasu Ubayashi
Graduate School and Faculty of Information
Kyushu University
Kyushu, Japan
{kamei, ubayashi}@ait.kyushu-u.ac.jp

***Abstract*—To prioritize software quality assurance efforts, fault prediction models have been proposed to distinguish faulty modules from clean modules. The performances of such models are often biased due to the skewness or class imbalance of the datasets considered. To improve the prediction performance of these models, sampling techniques have been employed to rebalance the distribution of fault-prone and non-fault-prone modules. The effect of these techniques have been evaluated in terms of accuracy/geometric mean/F1-measure in previous studies; however, these measures do not consider the effort needed to fix faults. To empirically investigate the effect of sampling techniques on the performance of software fault prediction models in a more realistic setting, this study employs Norm(P_{opt}), an effort-aware measure that considers the testing effort. We performed two sets of experiments aimed at (1) assessing the effects of sampling techniques on effort-aware models and finding the appropriate class distribution for training datasets (2) investigating the role of balanced training and testing datasets on performance of predictive models. Of the four sampling techniques applied, the over-sampling techniques outperformed the under-sampling techniques with Random Over-sampling performing best with respect to the Norm(P_{opt}) evaluation measure. Also, performance of all the prediction models improved when sampling techniques were applied between the rates of (20-30)% on the training datasets implying that a strictly balanced dataset (50% faulty modules and 50% clean modules) does not result in the best performance for effort-aware models. Our results also indicate that performances of effort-aware models are significantly dependent on the proportions of the two types of the classes in the testing dataset. Models trained on moderately balanced datasets are more likely to withstand fluctuations in performance as the class distribution in the testing data varies.**

Keywords - class imbalance, sampling techniques, software quality, software fault prediction, empirical study

I. Introduction

With the increasing pressures of expediting software projects that is always increasing in size and complexity to meet rapidly changing business needs, quality assurance activities such as fault prediction models have thus become extremely important. The main purpose of a fault prediction model is the effective allocation or prioritization of quality assurance effort (test effort and code inspection effort) [29], [32]. Construction of these prediction models are mostly dependent on historical or previous software project data referred to as a dataset.

However, a prevalent problem in data mining is the skewness of a dataset [14], [38]. Fault prediction datasets are not excluded from this phenomenon. Most datasets have the majority of the instances being either clean or not faulty [38], [42] and conventional learning methods are primarily designed for balanced datasets [7]. Common classifiers such as Neural Networks (NN), Support Vector Machines (SVM) and decision trees work best towards optimizing their objective functions that leads to the maximum overall accuracy - ratio of correctly identified faults to the number of total predicted faults. The use of imbalanced datasets for training a classifier will most likely generate a classifier that tend to over-predict the presence of the majority class but a lower probability of predicting the minority or faulty modules. Should the model predict the minority class, it usually comes with a high rate of error than predicting the majority class [21], [38]. This therefore affects the prediction performance of the classifiers and in machine learning, is referred to as learning from imbalanced datasets.

Several methods have been proposed in machine learning for dealing with the class imbalanced issue such as random over and under sampling [13], [7] creating synthetic data [18], [19], application of cleaning techniques for data sampling [44] and cluster-based sampling [21]. With a significant amount of literature in machine learning for imbalanced datasets, very few studies have tackled it in the area of fault prediction. The first of such studies by Kamei et al. [23] showed that sampling techniques improved the prediction performance of linear and logistics models whilst other two models (neural network and classification tree) did not have a better performance upon application of the sampling techniques.

Interestingly, sampling techniques applied to datasets during fault prediction are mostly evaluated in terms of Accuracy, AUC, F1-measure, Geometric Mean Accuracy [23], [10],

0730-3157/16 $31.00 © 2016 IEEE
DOI 10.1109/COMPSAC.2016.144

IEEE computer society

Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on September 17,2026 at 02:33:26 UTC from IEEE Xplore. Restrictions apply.

[38], [41], [39], [46] just to name a few; however, these measures ignore the effort needed to fix faults, i.e., they do not distinguish between a predicted fault in a small module and a predicted fault in a large module. Nickerson et al [36] concludes that to evaluate the performance of classifiers on imbalanced datasets, accuracy or its inverse error rate should never be used. Chawla et al. [6] also alludes to the conclusion that simple predictive accuracy might not be appropriate for an imbalanced dataset.

The goal of this research is to improve the prediction performance of fault-prone module prediction models, applying over and under sampling approaches to rebalance number of fault-prone modules and non-fault-prone modules in the training dataset and to find the appropriate distribution or proportion of faulty and non-faulty modules that results in the best performance. The experiment focuses on the use of Norm(P_{opt}), which is an effort-aware measure proposed by Kamei et al. [22] to evaluate the effect of over/under sampling on prediction models to find out if the over/under sampling is still effective in a more realistic setting.

Ten datasets are extracted from sampled open-source softwares (OSS) datasets with various degrees of sizes and distribution of fault-prone modules and 10 effort-aware prediction models are constructed considering different P_{fp} (percentage of fault-prone modules) values for 4 sampling techniques. Ideally, a balanced dataset should have P_{fp}= 50% but this might not be the best choice for each model and as such eight (8) P_{fp} values of (default,20,30,40,50,60,70 and 80)% are applied. In total, we conducted 800 or $10\times10\times8$ experiments for each of the four sampling techniques and compared the performance of these models to identify the most influential sampling approaches and required P_{fp} value for effort-aware fault prediction model. Our results shows that over-sampling with a P_{fp} of 20% to 30% improved the performances for all effort-aware models but the Relevance Vector Machine (RVM) model. We thus recommend over-sampling with P_{fp} between (20-30)% but caution should be exercised when applying it for use on RVM models since over-sampling by replication does not present new information to aid in re-estimating the RVM's parameters.

The organization of this paper is as follows. Section II reviews the most closely related work in connection with imbalanced dataset and sampling techniques and discusses the background of effort-aware and evaluation of models on imbalanced datasets. Section III presents a discussion on our research objectives. In Section IV, we describe the experimental methodology and performance evaluation. The results are presented in Section V and a discussion on threats to validity and future work of our study is in Section VI. Finally, a summary of the overall work is presented in section VII.

II. Background and Related Work

This section introduces the class imbalance learning problem and current research in this area of software engineering. We further discuss effort-aware models and evaluation measures for class imbalance and effort-aware models.

A. Class Imbalance learning

The class imbalance issue has been tackled by the research communities in two ways. Reducing the classification cost by assigning distinct costs to fit dataset [37], [11] and resampling the original dataset by over-sampling the minority class [7] or under-sampling the majority class [21], [26]. Over-sampling refers to the process of increasing the number of minority cases in a dataset. Several methods such as Random Over Sampling (ROS), Synthetic Minority Over Sampling Techniques (SMOTE), ADASYN [19] and MWMOTE [3] have been proposed. Oversampling is known to cause overfitting [42] hence care is taken when applying such techniques. Under sampling on the hand is the converse of over sampling whereby we reduce the majority cases in a dataset by deletion. Random Under Sampling (RUS) and One-Sided Selection (OSS) are some techniques available for under sampling. Chawla et al. [6] argues that removal of repeated data points could substantially destabilize the different class distributions. This may therefore boost or reduce classifier performance due to the class imabalance problem.

In the area of software fault prediction, learning from imbalanced fault datasets has received little attention. Kamei et al. [23], the first to apply sampling techniques to fault-proneness models experimentally evaluated the effects of four sampling techniques (ROS, SMOTE, RUS, OSS) applied to four fault-proneness models (Linear Discriminant Analysis-LDA, Logistic Regression-LR, Neural Networks-NN and Classification Tree-CT). Using two module sets of an industry legacy software and F1-Values as accuracy measure, all four sampling techniques improved the LR and LDA whilst the contrary was true for the NN and CT. They concluded that, the most appropriate sampling technique varied among the models.

Pelayo and Dick [38] applied SMOTE algorithm to determine if classification performance could be improved for defect-prone modules using the C4.5 decision tree model and the cost of misclassification. Using the average geometric mean classification accuracy as the performance indicator, they observed an improvement of at least 23% on four datasets. Riquelme et al. [41] analyses two balancing techniques with two predictive models on five datasets from the PROMISE repository aimed at improving the performance measure. They conclude that the balancing techniques may not necessarily improve the percentage of correctly classified intances, they however improve the AUC measure of classifying better buggy instances which are of more interest to the researcher.

Wang and Yao [46] investigated how the class imbalance learning methods improves software fault prediction. They applied three class imbalance learning methods of three main types (undersampling, threshold-moving and boosting-based ensembles) and compared with two popular base classifiers (Naive Bayes and Random Forest) on ten datasets from the PROMISE repository. Measuring classification performance

Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on September 17,2026 at 02:33:26 UTC from IEEE Xplore. Restrictions apply.

using balance, G-mean and AUC, Adaboost.NC- an ensemble predictive model, produced the best overall performance.

All of these studies considered datasets and fault prediction models of five or less. This study extensively applies four sampling techniques to ten open source datasets using the Normalised P_{opt}, an effort-aware evaluation measure to assess the performance and effects of sampling approaches on ten fault prediction models. It also aims at finding the appropriate class distribution for training datasets during application of sampling techniques.

B. Effort-Aware Classifiers

Fault prediction, which is a method of predicting the number of defects in a software is valuable in organizing a project's scarce test resources. Given limited resources for software testing, fault predictors could focus test engineers on the modules most likely to be faulty [45], [1]. For predictive models to be practical in an industrial setting, effort aware models have been proposed [1]. The intuition behind the effort-aware models is that time spent on reviewing a potentially buggy file is dependent on the file size and complexity, therefore in discovering bugs, less effort is required for small modules than for large modules [2]. The effort required to inspect a file or module during fault prediction has thus been considered in several recent studies. In a study by Mende and Koschke [30], authors proposed an effort-aware evaluation measure and considered lines of code as a proxy for effort. Kamei et al. [22] revisited common bug prediction findings using effort-aware performance measure, which was an upgrade on Mende and Koschke's performance measure and discovered that process metrics outperformed product metrics when testing effort was considered. Focusing on predicting fault-prone parts of large legacy system, Arisholm and Briand [1] assessed the cost-effectiveness of using an effort-aware logistic model considering the verification effort to predict the fault-proneness of the legacy system. Similarly, Menzies et al. [33] considered effort during the fault prediction construction and found that learners which considered effort out-performed other simple learners.

C. Evaluation Performance Measures

Most classification algorithms used for fault prediction studies attains acceptable or better classifaction accuracy when the dataset is rightly balanced [28]. Performance of prediction models have been typically evaluated using Accuracy [10], [35]. Menzies et al. [32] and Chawla et al. [6] argues that evaluating predictive models trained on class imbalanced dataset with accuracy is unsuitable as a result of the differing cost of error for the classes in the dataset or misclassification cost. Misclassification cost is very crucial and critical during software testing therefore using accuracy will be devastating to the investigator or business. For example, a usual software fault dataset could contain 95% bug free modules and 5% buggy modules. A model trained on this dataset could produce a 95% accuracy for the bug free modules. However, this predictive accuracy would be relevant if it reflects a higher rate of the minority modules (faulty) and allows a small error rate for majority bug free modules. Performance measures such as Receiver Operating Characteristics (ROC) curve, Area Under the ROC Curve (AUC), F-measure, Recall, Precision and G-mean have thus been considered for evaluating models trained on imbalanced datasets in several studies [23], [10], [38], [41], [39], [46], because these measures in one way or another manages the misclassification error rates . However, for a fault prediction model to be beneficial in an industrial setting, a software quality ranking system provides such benefits. Gray et al. [16] reports that using a software quality ranking system, an ordered list of the most likely error-prone modules could be produced. In effect, special consideration could be given to such modules in the form of detailed inspection (effort) in a descending order of level of defectiveness. Arisholm et al. [2] also suggests that evaluation and comparison of fault-proneness prediction models should not only consider their prediction accuracy but also take into consideration the potential cost-effectiveness of applying such models. For this study we use the Norm(P_{opt}) evaluation metric proposed by Kamei et al [22]. This metric will be discussed in Section IV.

III. Research Questions

We presume effort-aware prediction models will be improved if the suitable sampling technique together with the appropriate class distribution is chosen for an imbalanced fault dataset. We thus embark on an experimentation to compare different effort-aware model performances when sampling techniques are applied to the training dataset. The objectives of this paper are to empirically demonstrate the effect (positive and negative) of sampling techniques on the performance of software effort-aware fault prediction models and the effectiveness of having the same class proportions in both the training and testing dataset. To achieve the objectives as mentioned above, three research questions are stated:

RQ1. Is sampling approach effective on effort-aware prediction models? Several sampling approaches have been proposed with each model claiming to be more effective than another. In the domain of fault prediction, we sought to find which sampling approach has greater influence on effort-aware models.

RQ2. Could the right distribution of faulty and clean classes be identified during sampling for fault prediction? The search for the right distribution is a fascinating and prevalent problem which emerges during class imbalance learning. Intutitively, it may seem reasonable that making the dataset as balanced as possible would produce the best performance but that may not always be the case.

RQ3. Should the resampled training dataset have same class distribution as the testing dataset and which effort-aware models are more affected by imbalanced distributions? Menzies et al. [31] reports that resampling could produce a training set that contains classes of buggy/non-buggy = 1, although the test set still retains the original distribution. It may seem reasonable that training a predictive model on a dataset whose class distribution has been tuned on the testing dataset's distribution or similar to that of the test dataset should

Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on September 17,2026 at 02:33:26 UTC from IEEE Xplore. Restrictions apply.

perform better. We explore this domain to investigate the role a "balanced" testing dataset has on a predictive model.

IV. Experimental Setting

We begin this section by describing the datasets used in the study. We conducted a large study with 10 datasets which are publicly available. Sampling techniques, model construction and evaluation were executed using the open source statistical processing tool R [43]. The library packages Caret [27] and Unbalanced [40] were also used.

A. Datasets

We experiment on 10 different OSS project datasets available online to ensure the reproducibility of our experiments. Table I presents a summary of these projects. With varying degree in size and class distributions, these datasets provide an extensive domain for the investigation of the effects of sampling techniques on the performance of the effort-aware models to be constructed.

The class proportion of the faulty or minority modules ranges from as low as 5.8% to maximum of 17.46%. In emulation of a real-life setting, we extracted two versions of each project whereas the old version is used for training and the new version for testing [9]. Considering commit logs from version control tools (CVS, SVN), we obtained the number of bugs in source code files through thorough analysis of files and commits. If files are modified for bug fixes within "X" months after release, we identify the files that have post-defect. If commits include certain keywords such as "bug" and "fix", we conclude the commits are for bug fixes. We extracted seven process metrics proposed by Moser at al. [34] from these repositories and found the fault densities for each module. Table II shows the summary of the metrics extracted.

B. Sampling Techniques

For this study, Sampling in this context refers to a preprocessing procedure to correct the imbalanced target classes in a dataset by either increasing or decreasing the data samples before a model could be built for the dataset. We apply four sampling techniques RUS, ROS, SMOTE, PSMOTE at different *percentage of fault prone modules (P_{fp})* values. P_{fp} refers the total number of minority class divided by the total number of data points in a dataset. The Sampling techniques considered in this study are discussed below.

Table II
Measurement Metrics

Name	Description
Codechurn	Number of changed rows (number of additional lines + number of deleted rows)
LOCAdded	Additional number of rows (Lines of codes)
LOCDeleted	Deleted rows (Lines of codes)
Revisions	Revision number
Age	Last number of days since it was fixed
BugFixes	Number of corrected faults
Refactorings	Number of refactorings

RUS: Randomly remove clean or majority modules from the data until we reach the desired P_{fp} value whilst the minority modules are kept intact.

ROS: Randomly increase the minority or faulty modules until the desired P_{fp} value is attained. The increase is done by duplicating already existing minority modules.

SMOTE: This approach synthetically over-samples the minority class by considering each class instance and introduces a synthetic example along the line of segments which joins any of the k nearest neighbors [7]. The k nearest neighbor is randomly selected depending on the P_{fp} value required.

Purely SMOTE (PSMOTE): The original SMOTE algorithm oversamples the minority and undersamples the majority in a bid of shifting the initial bias of the learner from the majority class (negative) towards the minority class (positive) [7]. However, due to the effects of undersampling - removing important data points, PSMOTE applies the algorithm only to oversample the minority whilst keeping the majority classes intact. This is due to the fact that we do not want to lose any important module. When a P_{fp} value is applied to oversample the minority so as to become the majority class, the once majority class is still kept intact whilst the previous minority class is oversampled untill it becomes the majority class.

C. Evaluation Criteria

Evaluation of the predictive models is done using the Normalised P_{opt} referred to as Norm(P_{opt}). The Norm(P_{opt}) proposed by Kamei et al. [22], which is an upgrade of the P_{opt} evaluation measure proposed by Mende and Koschke [30] is an effort-aware ranking that takes into account the effort needed to review a file using the LOC metric as a proxy for effort. To calculate P_{opt} value, we first find $\triangle_{opt}$ (Figure 1) which is defined as the difference between the LOC-based cummulative lift chart of the optimal model and that of the prediction model. From the lift chart in Figure 1, the number of defects is represented on the y-axis and the cummulative size of the modules or LOC is represented on the x-axis.

Intuitively, more time is required to review a larger file than a small file, hence smaller files should always be prioritized if the predicted defects are same for both files thus P_{opt}= 1-$\triangle_{opt}$, which implies a small value for $\triangle_{opt}$ will produce a larger P_{opt} denoting the best performing classifier. For a standard comparison of the models, the P_{opt} values are normalised since the fault densities are dependent on the LOC for each module. The normalised P_{opt} is defined as equation (1)

$$Norm(P_{opt}) = \frac{P_{opt} - min(P_{opt})}{max(P_{opt}) - min(P_{opt})} \quad (1)$$

From equation (1), max(P_{opt}) and min(P_{opt}) are computed based on the lift chart when all modules are ordered by fault densities. Evidently, max(P_{opt}) will always be 1 since $\triangle_{opt}$= 0 for the best curve. To find min(P_{opt}), modules are ordered increasingly on the x-axis using their actual fault densities similar to finding the worst case of prediction.

Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on September 17,2026 at 02:33:26 UTC from IEEE Xplore. Restrictions apply.

Table I
THE NUMBER OF FILES AND MODULES OF TWO VERSIONS OF PROJECTS USED IN EXPERIMENTS

Project Name	Version A (Old)					Version B(New)				
	Release	Date	Module	Defects	P_{fp}	Release	Date	Module	Defects	P_{fp}
Clam Antivirus (CLAM)	1.0	2010/03/31 - 2010/09/27	1597	93	5.82	1.1	2011/02/07 - 2011/08/06	8723	56	0.64
eCos (ECOS)	2.0	2003/05/20 - 2006/05/04	630	110	17.46	3.0	2009/03/30 - 2011/08/25	3480	67	1.93
Helma (HELM)	1.3.1	2003/08/28 - 2004/02/24	312	38	12.18	1.4.0	2004/04/01 - 2004/09/28	100	17	17
NetBSD (NBSD)	4.0	2007/12/19 - 2008/06/16	6781	548	8.08	5.0	2009/04/29 - 2009/10/26	10960	299	2.73
OpenBSD (OBSD)	4.5	2009/05/01 - 2009/07/30	1706	275	16.12	4.6	2009/10/18 - 2010/01/16	5964	158	2.65
OpenCms (OCMS)	7.0.0	2007/07/05 - 2008/01/01	1727	193	11.18	8.0.0	2011/05/08 - 2011/10/13	2821	93	3.3
openNMS (ONMS)	1.7.0	2009/01/15 - 2009/04/15	1203	123	10.22	1.8.0	2010/06/07 - 2010/09/05	1416	112	7.91
Scilab (SCIL)	5.2.0	2009/12/15 - 2010/06/13	2636	239	9.07	5.3.0	2010/12/15 - 2011/06/13	1248	244	19.55
Spring Security (SPRI)	2.0.0	2008/04/14 - 2008/10/11	926	126	13.61	3.0.0	2009/12/22 - 2010/06/20	639	132	20.66
XORP (XORP)	1.0	2004/07/08 - 2005/01/04	1696	158	9.32	1.1	2005/04/13 - 2005/10/10	1755	217	12.36

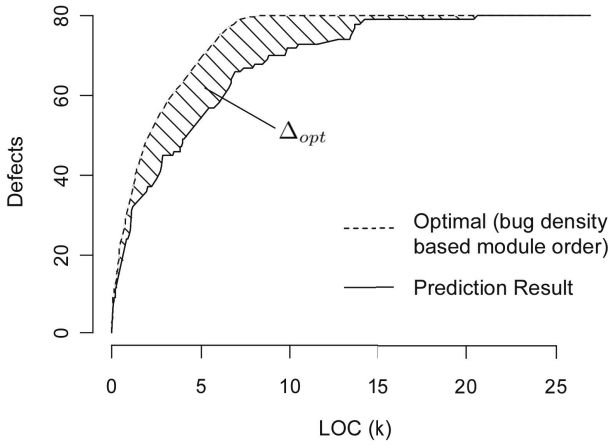

Figure 1. An example of LOC-based cummulative lift chart, Kamei et al. [22]

D. Experimental Method

To answer the research questions raised aboved, two sets of empirical experiments are conducted. We compare and examine the effects of four different sampling techniques at different P_{fp} values on ten effort-aware prediction models (Table III) using the procedure in Algorithm 1 for experiment 1. We furthermore investigate the influence a balanced testing dataset has on the performance on an effort-aware model after resampling the training datasets in experiment 2. The fault density for each module is predicted and not the presence or absence of defects. This produces an ordered list of modules and as noted by D'Ambros et al. [9], it helps in resource allocation.

Table III
PREDICTION METHODS AND HYPERPARAMETERS IN OUR EXPERIMENTS

Category	Model	Description	Hyper parameters Used
Statistical approach	LM	Linear regression	No
	LARS	Least Angle Regression [12]	No
	RVM	Relevance Vector Machine [4]	No
Neighborhood Law	K-NN	k neighborhood method [29]	k={1,. . . . . , 15}
	K*	k neighborhood method using distance based on entropy [8]	No
Neural Net	NNET	3-layer neural net [24]	size = {4,..,28}, decay = {0.1,0.2}
The tree model	M5	classification tree [15]	No
	RPART	CART [25]	cp= {0.05, 0.1, ..., 0.7}
Collective Learning	GBM	classification using boosting tress [5]	n.tree= { 10,50, 100, 200 }
	RF	Random Forest [17]	mtry= { 10, 50, 100, 200, 250, 500, 1000}

We refer to a resampled training dataset as a dataset which has been preprocessed using a sampling technique to an appropriate P_{fp} value. A 10-fold cross-validation method is employed during the model construction on the resampled training datasets. The resampled training dataset is split into ten partitions with nine partitions being used for training and the single one partition used for validation. This is done several times to ensure that the optimal parameters are found and

Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on September 17,2026 at 02:33:26 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1 Outline of Experiment 1

Given a training set comprised of clean(c) and buggy(b) data, W_{train}(Version A) = (W_c , W_b), $|W_b| < |W_c|$
and Z_{test}(Version B) be set of testing data set

Procedure Begin

1) For training dataset W_{train}\{W_1 , . ., W_n \} and Z_{test}\{Z_1 , . ., Z_n \} **repeat**
2) Initialize Norm(Popt) = ∅; samp=[ROS, RUS, SMOTE, PSMOTE];
3) Initialize pfp=[default,20,30,40,50,60,70,80]; C=[GBM, K*, KNN, LARS, LM, M5, NNET, RF, RPART, RVM]
4) **for** $k = 1, ..., length(samp)$ **do**
5) **for** $i = 1, ..., length(pfp)$ **do**
6) Generate N balanced training sets from W_{train} : $W_{pfp[1]}$, . ., $W_{pfp[n]}$, where $W_{pfp[i]}$ = resample(W, samp[k], pfp[i]);
7) Train classifier C_i on modified data $W_{pfp[i]}$
8) Classify data points in Z_{test} using the classifier C_i
9) Compute: Norm(P_{opt}) for each classifier C_i
10) **end for**
11) **end for**
12) **until** W_{train} is empty (i.e., the training data is exhausted)

End

selected after validating them on each fold of data when the default set parameters was used initially for building models. Since all the datasets have P_{fp} values less than 20% by default, we use P_{fp} values of 20% and above during the resampling process to generate seven independent training datasets (train-20, train-30, train-40, train-50, train-60, train-70, train-80). Default parameter configurations are used for the SMOTE algorithm where k-nearest neighbor is five (k=5).

Experiment 2 follows the procedure of experiment 1 but makes modification at step 8 in Algorithm 1 where the Z_{test} is modified to reflect the different P_{fp} values in relation to **RQ3**'s assertion of the effects a balanced testing dataset could have on the performance of predictive models. We consider five datasets and five effort-aware models based on their high Norm(P_{opt}) performance from experiment 1 and create seven independent testing datasets (test-20, test-30, test-40, test-50, test-60, test-70, test-80) using the same number of testing data points but variating the class distribution (P_{fp} from 20%-80%) to test the seven trained models constructed for each effort-aware model after applying ROS to the training datasets (train-20, train-30, train-40, train-50, train-60, train-70, train-80) in Algorithm 1.

V. Empirical Results and Discussion

RQ1 presents the comparison of performances of sampling techniques and its effects on the models. **RQ2** closely investigates the best proportion of class distribution to be used during sampling and **RQ3** examines the effect of a balanced testing dataset on an effort-aware fault prediction model.

RQ1. *Is sampling approach effective on effort-aware fault prediction models?*

We compare the performance of each sampling technique on the effort-aware models to the performance of the effort-aware models when no sampling technique is applied to the dataset which is denoted as NOS. Figure 2 shows the average Norm(P_{opt}) evaluation measure across all ten datasets for each sampling technique on the effort-aware fault prediction models. Effort-aware prediction results on individual datasets are not displayed due to space restrictions. From the plots, we observe the performance of the sampling techniques and using no sampling (NOS) as a threshold, it is evident only ROS performed relatively better. The other three sampling techniques negatively affected the performances of the models. All the models considered in this study were clearly induced or affected by the sampling approaches. ROS performs better in all the models except RVM compared to SMOTE and this could be explained by the fact that ROS merely duplicates the instances instead of introducing new information to the classifier hence the RVM model, a probalistic model whose functional form is equivalent to the Support Vector Machine (SVM) [4] could only have a decrease in performance because it requires new information to re-estimate its parameters. Study by Wei et al. [47] also confirmed the negative effect of ROS on the performance of SVM and RVM. The over-sampling techniques also had a positve effects on the NNET and RPART models since most of the Norm(P_{opt}) values were above the threshold line. Interestingly, the performance of most models when no sampling (NOS) was applied to the technique was better in comparison to the other three techniques (RUS, SMOTE, PSMOTE). Table IV presents the summarised results of the average Norm(P_{opt}) for each model and sampling technique across all the P_{fp} values considered for this study. The first column displays the no sampling approach (NOS) on the models, the last row shows the overall average of each sampling technique across all models and the best Norm(P_{opt}) for each sampling technique is bolded and underlined. Clearly, the over-sampling approaches outperforms the under-sampling approaches with ROS performing best with an average of 65.407 across all models. The SMOTE and PSMOTE achieved very similar Norm(P_{opt}) values. Similarly,

Table IV
Average Norm(P_{opt}) of Effort-Aware Models among different Sampling Techniques across all P_{fp} values

Models	NOS	RUS	ROS	SMOTE	PSMOTE
GBM	66.34	58.27	**68.11**	65.93	64.3
K*	65.31	63.72	**70.09**	63.21	63.06
KNN	58.84	58.32	**61.75**	57.81	58.56
LARS	68.19	61.54	**72.14**	63.74	62.06
LM	63.37	55.25	**69.72**	60.34	57.93
M5	67.44	64.77	**69.22**	61.02	63.15
NNET	54.97	53.19	**68.84**	57.97	57.81
RF	65.44	60.07	**68.13**	63.28	63.52
RPART	55.64	56.93	**62.83**	58.06	59.13
RVM	49.55	47.64	43.24	**50.52**	48.08
AVERAGE	61.509	57.97	**65.407**	60.188	59.76

Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on September 17,2026 at 02:33:26 UTC from IEEE Xplore. Restrictions apply.

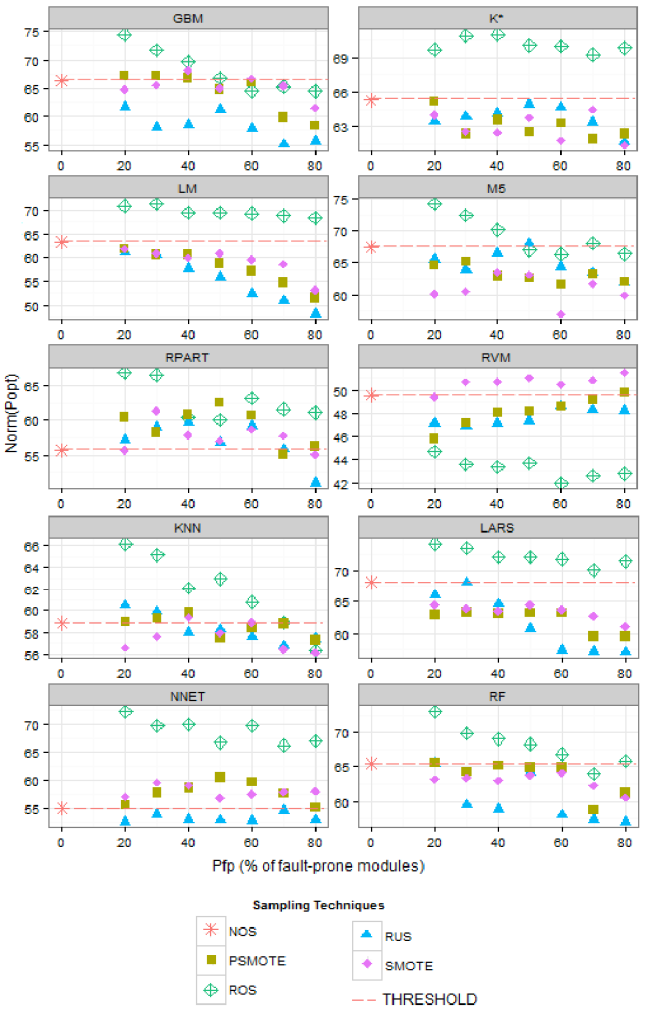

Figure 2. Average Norm(P_{opt}) performances across all ten datasets for the sampling techniques on different Effort-Aware Models at different P_{fp} values

NOS also outperforms the under-sampling techniques which implies that under-sampling approach should not be applied alone if the Norm(P_{opt}) evaluation measure is employed but could be considered when combined with an over-sampling approach.

RQ2. *Could the appropriate distribution of classes be identified during sampling for fault prediction?*

In light of the analysis presented in RQ1, it will be of great essence to examine the effort-aware predictive models and find the appropriate class distribution required during sampling. Examining the results from the plot in Figure 3, it could be observed that performance of sampling techniques on the effort-aware models varied across different P_{fp} values for different sampling techniques. For most of the models, the best performance is achieved during the initial P_{fp} values. For all the models, except LM and K*, ROS performs best when P_{fp} is 20%. Similarly, RUS and PSMOTE attains better performance for most of the models when P_{fp} is either 20% or 30%. From Figure 3, it is evident that at P_{fp} values of 20% and 30%, most models performed best in comparison to the other P_{fp} values. However, for SMOTE, the best performance seems to alternate at different P_{fp} values with four of the models achieving best performances at 40%. This is not unusual since this techniques generate synthetic instances which are new as compared to the ROS which duplicates already existing instances hence the synthetic technique will have a better performance when more are being generated. A stable P_{fp} value for SMOTE could not be attained but majority of the models achieved best performance at P_{fp} values less than 50%. Generally, most models exhibit better performances when the sampling approaches are applied at P_{fp} level less than 50% and thus conclude that most models performs best when sampling is applied at a P_{fp} level of 20-30%.

RQ3. *Should the resampled training dataset have the same class distribution as the testing dataset and which effort-aware models are more affected by imbalanced distributions?*

To find out whether the performance of the effort-aware models trained on the resampled training datasets were influenced by the proportion of the class distribution in the testing dataset, we explored further by conducting experiment 2. For each of the five datasets considered, Figure 4 shows the Norm(P_{opt}) performance of the seven training set models tested by seven different testing datasets for each model and dataset. Norm(P_{opt}) values are displayed on the y-axis and the P_{fp} values of the testing dataset are represented on the x-axis. The plots in Figure 4 reveals an interesting trend in the Norm(P_{opt}) performance achieved by all models.

Observing the K* model, the Norm(P_{opt}) curves for all the datasets except HELM are not significantly slanted, they are almost flat when the percentage of the fault prone modules (P_{fp}) increases from 20% to 80%. On the HELM dataset however, there seems to be a positive trend with four of the models increasing as the P_{fp} values increases in the testing datasets. The model trained with dataset train-20 achieves the best performance when P_{fp} is 80 for the test dataset. LARS and LM models display similar results across all datasets. For both models, we observe that a combination of test-80 with train-80 achieves the best performance on three of the datasets showing a steady rise in performance as the P_{fp} in the test dataset is increasing. Similarly, they both display a sharp increase in performance as P_{fp} increases from 20% to 30% and maintains a stable and no significant change in performance afterwards with the train-20 model being the only exception which exhibits a flat or stable performance for the HELM dataset. Also for the OCMS dataset, models trained with train-50 and above exhibit some significant increase in performance in the initial stages as P_{fp} increases from 20% to 50% and stalls afterward for the LARS model whilst it the converse for the LM model which maintains the same performance until P_{fp} reaches 50% where performance displays a significant increase afterwards. With train-20 and train-80 performing well on the three models considered so far, the train-20 model however performs poorly whilst the train-80 still performs better for the NNET model. It is also interesting to note that

Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on September 17,2026 at 02:33:26 UTC from IEEE Xplore. Restrictions apply.

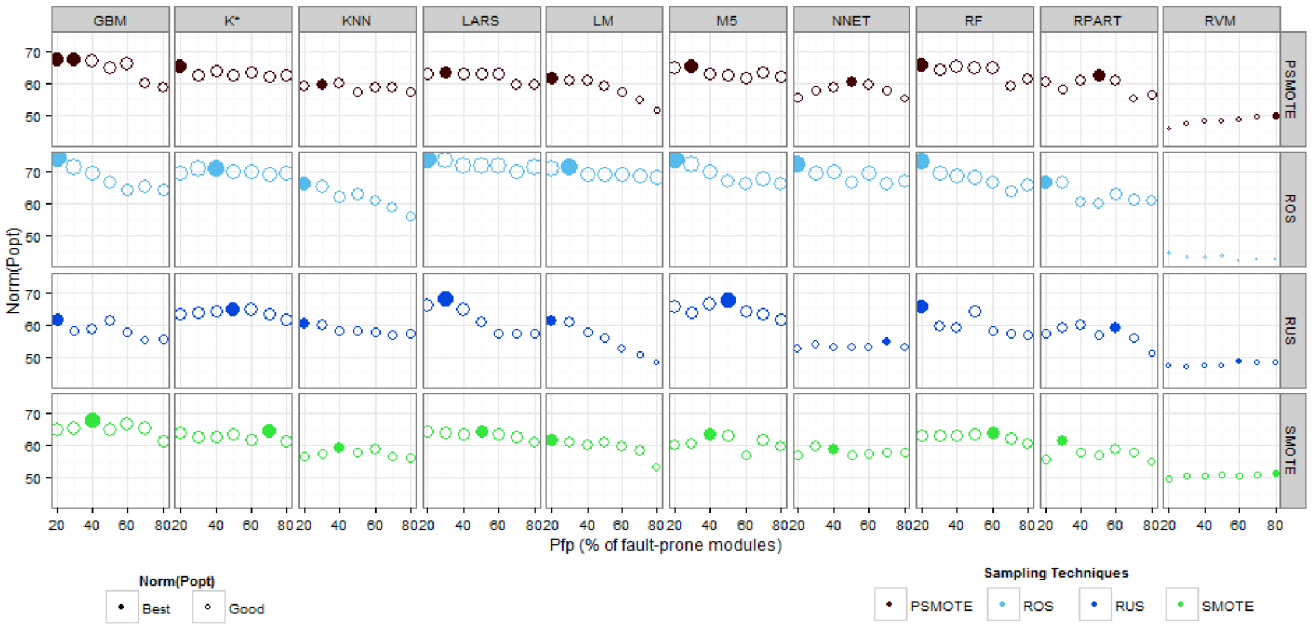

Figure 3. Average performance of Effort-Aware Models across different P_{fp} values

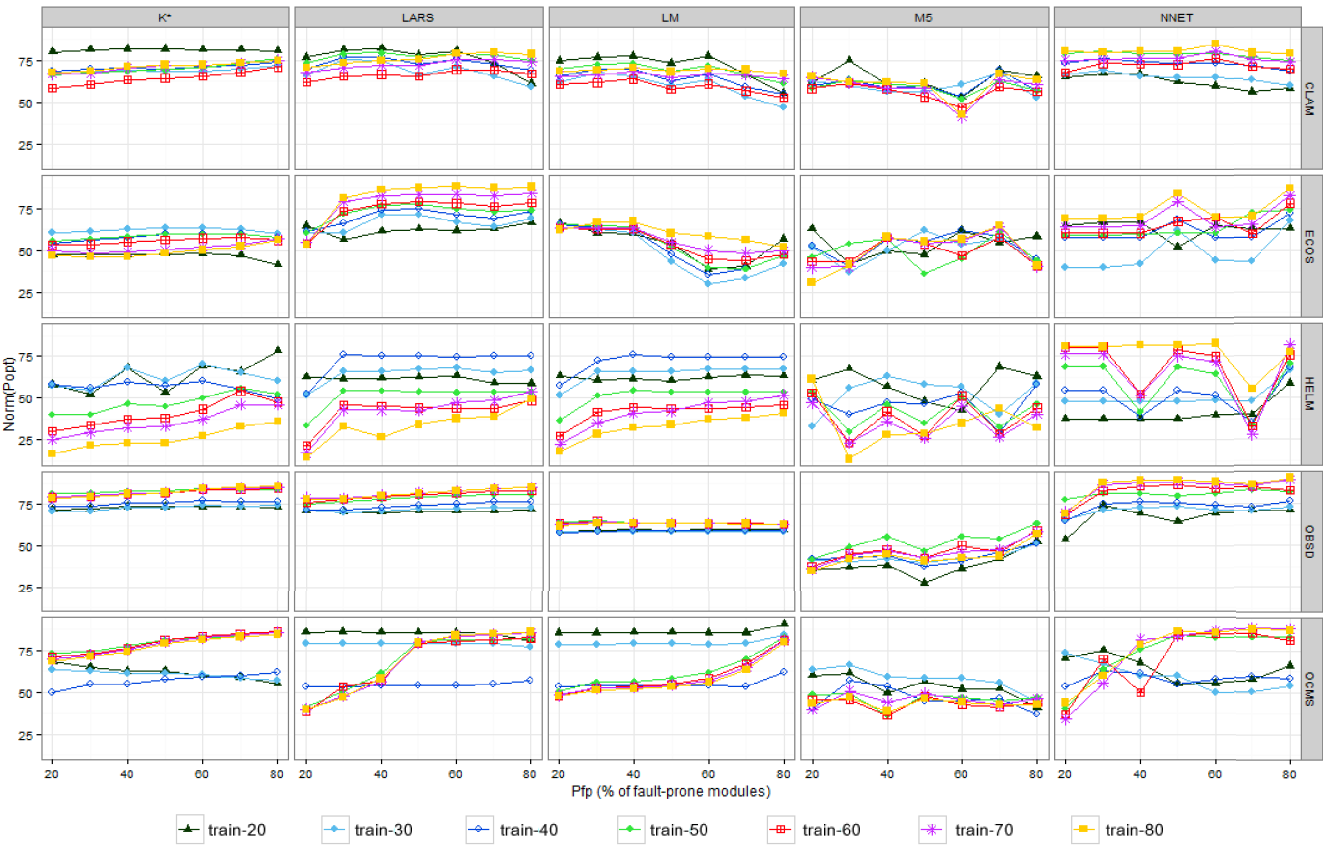

Figure 4. Effort-Aware Models trained on 7 different datasets (train-20, train-30, train-40, train-50, train-60, train-70 and train-80) tested by seven different testing datasets, increasing from 20% faulty (x-axis=20) to 80% faulty (x-axis=80)

Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on September 17,2026 at 02:33:26 UTC from IEEE Xplore. Restrictions apply.

train-20, train-30 and train-80 fails to react at test-40 when all the models seem to fall or decrease dramatically for the HELM dataset. Train-20 and train-30 exhibits resistance to change but finally increases at test-80. On all datasets, models trained on P_{fp} values 50% and above performs better than those below 50%. The M5 model overall seems to be the most affected model by class distributions in the test dataset. There could not be established a stable trend in performance as the Norm(P_{opt}) curves oscillate when the P_{fp} values in the test data kept on increasing.

In summary, we could conclude the following: (1) initial increase in the P_{fp} of the testing dataset results in most of the models having a significant increase/decrease in performance. (2) equal P_{fp} value in both training and testing dataset does not achieve the maximum Norm(P_{opt}) in most cases (3) models trained on a moderately rebalanced training datasets such as train-20 and train-30 are the most resistant to performance fluctuations notwithstanding the proportions of two class distributions in the testing datasets.

VI. 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.

This study obtains the number of faults in source code files through the analysis of files and commits in software repositories. This process is commonly used in fault prediction research [34], [10], [22], but has the limitation that faults not recorded in CVS log comments cannot be collected. Further research is required to improve the accuracy of faults collection from repositories. Considering only process metrics, we cannot generalize our results to all areas of fault prediction. However, process metrics has been shown in literature as a better option for software fault prediction [34], [22], [30], [10]. Consideration of other metrics such as the static code metrics or product metrics is left for future studies. Our study was also limited to ten purely OSS datasets and ten effort-aware models which could also be a source of bias to the results. However, the depth and size of our datasets were large enough for a comprehensive experiment. We acknowledge the availability of diverse effort-aware predictive models which could have been considered but we argue that the models selected covers different domains of techniques ranging from statistical, data mining and machine learning techniques hence these models are representative of all the several models that exist in literature.

It has been proven that ensemble classifiers perform better than single base classifiers for datasets with different characteristics [20], we aim to replicate this study considering ensemble effort-aware classifiers in future studies. There exist several proposed sampling techniques in literature but our study focused on four sampling techniques which are commonly applied in the domain of class imbalanced learning due to their ease of implementation. We however, intend to consider more of these techniques in future studies.

VII. Conclusions

We have assessed and examined the role of balanced and imbalanced training and testing datasets on the performance of software defect prediction models considering ten datasets and ten effort-aware models evaluated using an effort-aware measure, Norm(P_{opt}). Of interest to us were how the percentage of target variable distribution in a fault dataset affects the performance of a predictive model with the background knowledge that rate of faulty modules in real world fault dataset is unstable and highly imbalanced. Assessing the performance of four sampling techniques on ten effort-aware predictive models, our results indicate that moderate application of sampling technique was required to achieve better results for most effort-aware models considered since the best percentage or rate of fault-prone modules in a dataset (P_{fp}) was found to be between the ranges of 20-30%. In comparison of the best sampling technique, the over-sampling approaches outperformed the under-sampling approach with Random Over-Sampling edging out the others.

Our results indicate no sampling is better than applying under-sampling technique to a faulty dataset when the minority class of the original dataset is less than 20% of the total dataset. Under-sampling could be considered when it is combined with other over-sampling techniques. Most studies have reported the positve effects of applying sampling techniques to fault prediction models but our study contrasted that view when we evaluated using the Norm(P_{opt}). Random over-sampling (ROS) proved to be the only positive approach. Lastly, the study showed that performance of trained models is highly dependent on the class distribution in a testing dataset and thus extra caution should be considered when evaluating and comparing effort-aware models across different testing datasets with varied proportions of buggy and clean instances. We have demonstrated in a practical setting, the effects of sampling approach on fault prediction models and intend to apply more advanced sampling techniques on very large datasets for future studies.

VIII. Acknowledgment

This research is supported by the City University of Hong Kong research funds (Project No. 7200354, 7004222, 7004474).

References

[1] E. Arisholm and L. C. Briand, "Predicting fault-prone components in a java legacy system," in *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*. ACM, 2006, pp. 8–17.

[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] S. Barua, M. M. Islam, X. Yao, and K. Murase, "Mwmote–majority weighted minority oversampling technique for imbalanced data set learning," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 26, no. 2, pp. 405–425, 2014.

[4] C. M. Bishop and M. E. Tipping, "Variational relevance vector machines," in *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 2000, pp. 46–53.

Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on September 17,2026 at 02:33:26 UTC from IEEE Xplore. Restrictions apply.

[5] P. Bühlmann and T. Hothorn, "Boosting algorithms: Regularization, prediction and model fitting," *Statistical Science*, pp. 477–505, 2007.

[6] N. V. Chawla, "Data mining for imbalanced datasets: An overview," in *Data Mining and Knowledge Discovery Handbook*. Springer, 2010, pp. 875–886.

[7] 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.

[8] J. G. Cleary, L. E. Trigg *et al.*, "K*: An instance-based learner using an entropic distance measure," in *Proceedings of the 12th International Conference on Machine learning*, vol. 5, 1995, pp. 108–114.

[9] M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," in *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*. IEEE, 2010, pp. 31–41.

[10] M. DŠAmbros, 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.

[11] P. Domingos, "Metacost: A general method for making classifiers cost-sensitive," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1999, pp. 155–164.

[12] B. Efron, T. Hastie, I. Johnstone, R. Tibshirani *et al.*, "Least angle regression," *The Annals of statistics*, vol. 32, no. 2, pp. 407–499, 2004.

[13] A. Estabrooks, T. Jo, and N. Japkowicz, "A multiple resampling method for learning from imbalanced data sets," *Computational Intelligence*, vol. 20, no. 1, pp. 18–36, 2004.

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

[15] E. Frank, Y. Wang, S. Inglis, G. Holmes, and I. H. Witten, "Using model trees for classification," *Machine Learning*, vol. 32, no. 1, pp. 63–76, 1998.

[16] 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 *Evaluation & Assessment in Software Engineering (EASE 2011), 15th Annual Conference on*. IET, 2011, pp. 96–103.

[17] L. Guo, Y. Ma, B. Cukic, and H. Singh, "Robust prediction of fault-proneness by random forests," in *Software Reliability Engineering, 2004. ISSRE 2004. 15th International Symposium on*. IEEE, 2004, pp. 417–428.

[18] 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.

[19] 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.

[20] S. Hussain, J. Keung, A. A. Khan, and K. E. Bennin, "Performance evaluation of ensemble methods for software fault prediction: An experiment," in *Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference*. ACM, 2015, pp. 91–95.

[21] W. Jindaluang, V. Chouvatut, and S. Kantabutra, "Under-sampling by algorithm with performance guaranteed for class-imbalance problem," in *Computer Science and Engineering Conference (ICSEC), 2014 International*. IEEE, 2014, pp. 215–221.

[22] 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 *Software Maintenance (ICSM), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–10.

[23] 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 *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*. IEEE, 2007, pp. 196–204.

[24] 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.

[25] T. M. Khoshgoftaar and N. Seliya, "Comparative assessment of software quality classification techniques: An empirical case study," *Empirical Software Engineering*, vol. 9, no. 3, pp. 229–257, 2004.

[26] 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.

[27] 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.

[28] I. H. Laradji, M. Alshayeb, and L. Ghouti, "Software defect prediction using ensemble learning on selected features," *Information and Software Technology*, vol. 58, pp. 388–402, 2015.

[29] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *Software Engineering, IEEE Transactions on*, vol. 34, no. 4, pp. 485–496, 2008.

[30] T. Mende and R. Koschke, "Revisiting the evaluation of defect prediction models," in *Proceedings of the 5th International Conference on Predictor Models in Software Engineering*. ACM, 2009, p. 7.

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

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

[33] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect prediction from static code features: current results, limitations, new approaches," *Automated Software Engineering*, vol. 17, no. 4, pp. 375–407, 2010.

[34] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*. IEEE, 2008, pp. 181–190.

[35] J. Nam, "Survey on software defect prediction," *Master's Thesis*, 2009.

[36] A. Nickerson, N. Japkowicz, and E. Milios, "Using unsupervised learning to guide resampling in imbalanced data sets," in *Proceedings of the Eighth International Workshop on AI and Statitsics*, 2001, pp. 261–265.

[37] M. Pazzani, C. Merz, P. Murphy, K. Ali, T. Hume, and C. Brunk, "Reducing misclassification costs," in *Proceedings of the Eleventh International Conference on Machine Learning*, 1994, pp. 217–225.

[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] S. L. Phung, A. Bouzerdoum, and G. H. Nguyen, "Learning pattern classification tasks with imbalanced data sets," 2009.

[40] A. D. Pozzolo, O. Caelen, and G. Bontempi, "unbalanced: Racing for unbalanced methods selection," 2015, r package version 2.0. [Online]. Available: http://CRAN.R-project.org/package=unbalanced

[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] Z. Sun, Q. Song, and X. Zhu, "Using coding-based ensemble learning to improve software defect prediction," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 42, no. 6, pp. 1806–1817, 2012.

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

[44] I. Tomek, "Two modifications of cnn," *IEEE Trans. Syst. Man Cybern.*, vol. 6, pp. 769–772, 1976.

[45] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, vol. 14, no. 5, pp. 540–578, 2009.

[46] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *Reliability, IEEE Transactions on*, vol. 62, no. 2, pp. 434–443, 2013.

[47] L. Wei, Y. Yang, R. M. Nishikawa, and Y. Jiang, "A study on several machine-learning methods for classification of malignant and benign clustered microcalcifications," *Medical Imaging, IEEE Transactions on*, vol. 24, no. 3, pp. 371–380, 2005.

Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on September 17,2026 at 02:33:26 UTC from IEEE Xplore. Restrictions apply.