Accredited SINTA 2 Ranking
Decree of the Director General of Higher Education, Research, and Technology, No. 158/E/KPT/2021
Validity period from Volume 5 Number 2 of 2021 to Volume 10 Number 1 of 2026

Published online at: **http://jurnal.iaii.or.id**

JURNAL RESTI
(Rekayasa Sistem dan Teknologi Informasi)
Vol. 8 No. 5 (2024) 631 - 643 e-ISSN: 2580-0760

An Investigation Towards Resampling Techniques and Classification Algorithms on CM1 NASA PROMISE Dataset for Software Defect Prediction

Agung Fatwanto[1*], Muh Nur Aslam[2], Rebbecah Ndugi[3], Muhammad Syafrudin[4]
[1, 2]Informatics Department, Faculty of Science and Technology, UIN Sunan Kalijaga, Yogyakarta, Indonesia
[3]Faculty of Mathematics and Computer Science, St. Petersburg State University, St. Petersburg, Russia
[4]Department of Artificial Intelligence and Data Science, Sejong University, Seoul, Republic of Korea
[1]agung.fatwanto@uin-suka.ac.id, [2]23206051031@student.uink-suka.ac.id, [3]rebbecahndungi94@gmail.com, [4]udin@sejong.ac.kr

Abstract

Software defect prediction is a practical approach to improving the quality and efficiency of software testing processes. However, establishing robust and trustworthy models for software defect prediction is quite challenging due to the limitation of historical datasets that most developers are capable of collecting. The inherently imbalanced nature of most software defect datasets also posed another problem. Therefore, an insight into how to properly construct software defect prediction models on a small, yet imbalanced, dataset is required. The objective of this study is therefore to provide the required insight by way of investigating and comparing a number of resampling techniques, classification algorithms, and evaluation measurements (metrics) for building software defect prediction models on CM1 NASA PROMISE data as the representation of a small yet unbalanced dataset. This study is comparative descriptive research. It follows a positivist (quantitative) approach. Data were collected through observation towards experiments on four categories of resampling techniques (oversampling, under sampling, ensemble, and combine) combined with three categories of machine learning classification algorithms (traditional, ensemble, and neural network) to predict defective software modules on CM1 NASA PROMISE dataset. Training processes were carried out twice, each of which used the 5-fold cross-validation and the 70% training and 30% testing data splitting (holdout) method. Our result shows that the combined and oversampling techniques provide a positive effect on the performance of the models. In the context of classification models, ensemble-based algorithms, which extend the decision tree classification mechanism such as Random Forest and eXtreme Gradient Boosting, achieved sufficiently good performance for predicting defective software modules. Regarding the evaluation measurements, the combined and rank-based performance metrics yielded modest variance values, which is deemed suitable for evaluating the performance of the models in this context.

Keywords: *software defect prediction; machine learning; classification algorithm; imbalanced data; resampling*

How to Cite: A. Fatwanto, M. Nur Aslam, R. Ndugi, T. Huguet, and M. Syafrudin, "An Investigation Towards Resampling Techniques and Classification Algorithms on CM1 NASA PROMISE Dataset for Software Defect Prediction", *J. RESTI (Rekayasa Sist. Teknol. Inf.)* , vol. 8, no. 5, pp. 631 - 643, Oct. 2024.
DOI: https://doi.org/10.29207/resti.v8i5.5910

1. Introduction

The need for new software system applications is rapidly increasing to meet the never-ending user demand. The newly demanded software system applications are also becoming increasingly more complex and complicated. This condition obviously has an impact on raising the required time and cost of software development due to the need for a comprehensive testing process.

Arar and Ayan, in their study, found that 23% of software development costs are spent on quality assurance and software testing alone [1]. One effort that can be made to reduce the time and cost of software testing is by way of predicting potential defects in software modules [2]. Machine learning algorithms can be utilized to develop these predicting models. Predicting defective modules helps software developers allocate resources more efficiently by focusing on

Received: 07-07-2024 | Accepted: 11-09-2024 | Published Online: 14-10-2024

modules likely to contain defects [3]. The main purpose of predicting defects within software modules is to determine which modules will be prioritized for further in-depth testing, either manually or automatically [4].

The method for finding defective/faulty software modules is known as Software Defect Prediction (SDP). On some occasions, it is also known as Software Fault Prediction (SFP). SDP/SFP is employed to assist in identifying potential defective/faulty software modules and calculating the number of potential defects on components/modules of the system being developed [5]. Predicting the number of potential errors can provide better insight into how much effort and resources are required to eliminate errors from software modules.

SDP/SFP have been and still attract a lot of researchers to study this particular topic such as reported in [6], [7]. A number of studies on SDP/SFP especially regarding the use of resampling techniques to handle imbalanced datasets have been conducted previously. A study as reported in [8], describes the application of undersampling techniques to minority classes (in this case the defective software module class) to resample the imbalanced dataset on software defects prediction. A study as reported in [9] employed SMOTE to handle imbalanced datasets in order to improve the classification performance. Another study on the impact of rebalancing techniques on the performance of defect prediction models is reported in [10].

Studies on the application of machine learning-based classification processes for SDP/SFP have also been conducted by several researchers. One study discusses the application of the kNN algorithm combined with the Random Walk Over Sampling technique to handle class imbalance for predicting defective software modules [11]. In addition, there is another study that discusses the optimization of Naïve Bayes using Gain Ratio to improve the accuracy of defective software module prediction [12].

A series of studies on the use of the boosting and stacking-based algorithm for SDP/SFP have also been conducted. In these studies, it was reported that this ensemble algorithm provides sufficiently good prediction results [13], [14], [15].

Nevertheless, as far as our exploration is concerned, there is still no study which covers a comprehensive investigation of the effect of resampling techniques and classification algorithms performance on small yet imbalanced datasets that are normally found in software defects data. As an effort to provide a relatively comprehensive picture, this study experimented with ten resampling techniques which represent four types of resampling categories, namely oversampling, undersampling, ensemble, and combination; combined with the use of eight classification algorithms which represent three types of classification algorithm categories, namely traditional, ensemble, and neural network. The result will then be evaluated for their performance using a set of measures which consist of single, combined, and rank performance measures.

As for the framework of our research, we put forward three research questions as follows:

RQ1: *What type of resampling techniques would provide a potentially positive effect on the performance of the software defects prediction model on CM1 NASA PROMISE data (as the representation of a small yet imbalanced dataset)?*

RQ2: *What type of classification algorithms would provide better performance for predicting defective software modules on CM1 NASA PROMISE data (as the representation of a small yet imbalanced dataset)?*

RQ3: *What type of evaluation measurements (metrics) are deemed suitable for assessing the performance of software defect prediction models on CM1 NASA PROMISE data (as the representation of a small yet imbalanced dataset)?*

The output of this study is expected to provide comprehensive insight into the effect of using particular resampling techniques and machine learning classification algorithms on software defect prediction. This insight is especially of interest for software developers which can act as their guideline for developing software defect prediction models.

2. Research Methods

This study is descriptive comparative research. It follows an empirical quantitative approach. Data were collected through observations towards experiments on resampling techniques and classification algorithms. The obtained classification results were then used to measure the models' performance. The flow of our research is depicted in Figure 1.

The research began with the data collection process. It was then followed by the experimental stage. We conducted our experiment on Google Colab. We employed Python 3.10.12 and used a number of libraries such as imblearn 0.10.1 for resampling, sklearn 1.2.2 for classification (LogisticRegression, SVC, KNeighbors, GaussianNB, DecisionTree, RandomForest, and MLP), and xgboost 2.0.3 for XGB classifier.

The experimental stage started with data preprocessing, in which the imbalanced dataset was resampled using ten techniques representing four types of resampling categories, namely UnderSampling, OverSampling, EnsembleSampling and CombineSampling. The new resampled datasets were then used as the material for training, validation, and testing processes of eight classification algorithms representing three types of machine learning categories, namely traditional, ensemble, and neural network models.

The training, validation, and testing processes were carried out twice, firstly using the 5-fold cross-validation and secondly using the holdout method (70%

training, 30% testing). The results of the testing process are then evaluated using 15 measurements (metrics) representing three types of performance measurement categories, namely single, combined, and rank performance measurements. The values of these measurements are then used as the basis for performing a descriptive comparative analysis towards the effect of resampling techniques and classification algorithms performance.

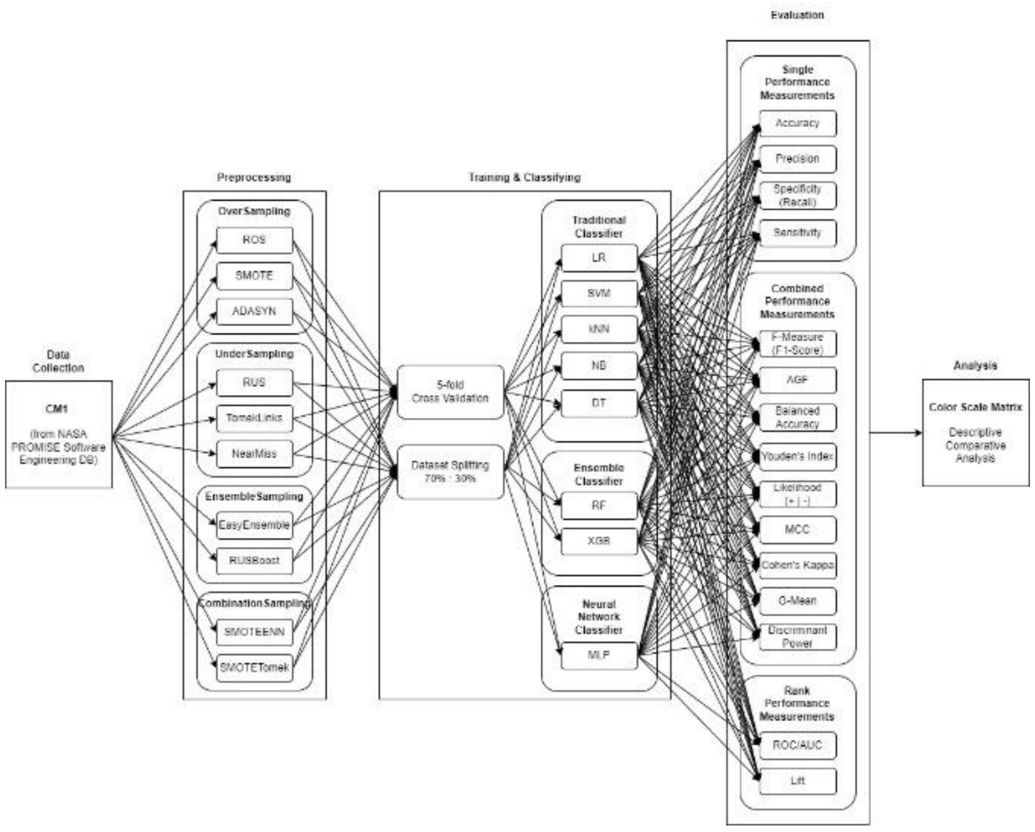

Figure 1. Research Method

2.1 Data Collection

This study used a software defect prediction dataset from the NASA PROMISE Software Engineering Database. This is a public dataset and can be obtained from the following link: http://promise.site.uottawa.ca/SERepository/datasets-page.html or https://github.com/ApoorvaKrisna/NASA-promise-dataset-repository. Compared to those from the NASA Metrics Data Program (MDP), all software defect prediction datasets from PROMISE have fewer variables (features) [16]. These datasets are hence deemed more suitable to be chosen for this type of study since fewer variables are easier and cheaper to collect hence would be obviously in favour of developers and relatively feasible to be practised on typical software development projects.

Out of 13 software defect prediction datasets from the NASA PROMISE Software Engineering Database, there are five datasets which contain the fewest variables, namely: CM1, JM1, KC1, KC2, and PC1. This study specifically used the CM1 dataset mainly because CM1 bears the characteristics that best represent the common problematic condition faced by most developers in collecting software defect datasets on typical projects. The size of most software systems found on typical medium-sized business projects are around hundreds of modules. It is categorized as small in the context of the size of the machine learning dataset. Yet, most software defect datasets are inherently imbalanced where positive cases (the occurrence of defects in modules) are way too small. Compared to other software defect datasets in the NASA PROMISE Software Engineering Database, CM1 best represents both characteristics. It has the smallest amount of data and also has the most extreme proportion of class imbalance between defective and non-defective classes (or having the most extreme defect rate). We also considered not combining CM1 with some other datasets to train the models in our experiment since they were collected from different projects which were potentially having different characteristics. An overview of the software defect prediction dataset from the NASA PROMISE Software Engineering Database is shown in Table 1.

Table 1. The NASA PROMISE software defect prediction datasets

No	Nane	# of Var	# of Data	[+ ; -]	% of Defects
1	CM1	22	498	[49 ; 449]	9.83%
2	JM1	22	10,885	[8779 ; 2106]	80.65%
3	KC1	22	2,109	[326 ; 1783]	15.45%
4	KC2	22	522	[107 ; 415]	20.5%
5	PC1	22	1,109	[1032 ; 77]	93.05%

Software Defect Prediction dataset from the NASA PROMISE Software Engineering Database has a total of 22 variables data. It consists of 21 independent variables and one dependent variable data. The independent variables data are in the form of quantitative measurements of the program source code. Four variables' data are measurements based on McCabe's theory, 11 variables data are measures based on Halstead's theory, and the other six variables' data are additional measurements. The dependent variable is the defect classification label in binary form. All independent variables have numeric data types (interval & ratio), while the independent variables have binary (binomial) data types (i.e. true = defect or false = not defect). The explanation of each variable in the dataset used for this study is shown in Table 2.

Table 2. List of variables in CM1 dataset

No.	Variable	Base	Type
1	Line of Code	McCabe	Numeric
2	Cyclomatic Complexity	McCabe	Numeric
3	Essential Complexity	McCabe	Numeric
4	Design Complexity	McCabe	Numeric
5	Halstead Operator and Operand	Halstead	Numeric
6	Halstead Volume	Halstead	Numeric
7	Halstead Program Length	Halstead	Numeric
8	Halstead Difficulty	Halstead	Numeric
9	Halstead Intelligence	Halstead	Numeric
10	Halstead Effort	Halstead	Numeric
11	Halstead Delivered Bug	Halstead	Numeric
12	Halstead Time Estimator	Halstead	Numeric
13	Halstead Line Count	Halstead	Numeric
14	Halstead Comments Count	Halstead	Numeric
15	Halstead Blank Line Count	Halstead	Numeric
16	Line of Code and Comments	Misc.	Numeric
17	Unique Operators	Misc.	Numeric
18	Unique Operands	Misc.	Numeric
19	Total Operators	Misc.	Numeric
20	Total Operands	Misc.	Numeric
21	Branch Count	Misc.	Numeric
22	Defects	Misc.	Boolean

2.2 Preprocessing

Prior to performing the experiment, first, the raw dataset will normally go through pre-processing activities. During pre-processing, several treatments will generally be carried out to the raw dataset, including i) a data cleaning process to find empty and inconsistent data items, ii) a data transformation (normalization) process to standardize the whole data so that they will have the same basis unit in order to make the data analysis process become feasible to perform, iii) resampling process to balance the proportion of class distribution, especially for the imbalanced dataset. After the data is deemed sufficient to be used as the material for training the model, it is continued to the experimental stage. Specifically in this study, the only preprocessing activity performed was the resampling process.

One of the main activities of this study was to investigate the effect of using several resampling techniques on small yet imbalanced datasets for predicting defective software modules. The resampling techniques experimented in this research represent four categories, namely oversampling, undersampling, ensemble, and combination. The experimented resampling techniques include RandomOverSampling (ROS), Synthetic Minority Oversampling Technique (SMOTE), and Adaptive Synthetic Sampling Approach (ADASYN) three of which represent the oversampling technique category; RandomUnderSampling (RUS), TomekLinks, and NearMiss in which three of them represent the undersampling technique category; EasyEnsemble and RUSBoost in which both represent ensemble technique category; and Synthetic Minority Oversampling Technique Edited Nearest Neighbors (SMOTEENN) and SMOTETomek in which both represent combination techniques category.

2.3 Training and Classification

In addition to investigating the effect of several resampling techniques on small yet imbalanced datasets, this study also investigated the performance of several machine learning classification algorithms, especially for predicting defective software modules. The experiment of eight algorithms for predicting defective software modules was carried out twice. The first experiment applies a k-fold cross-validation strategy (with k=5), and the second experiment applies a dataset-splitting strategy with a proportion of 70% for training and 30% for testing. Our reason why choosing k=5 for the k-fold cross-validation and 70%: 30% proportion for the holdout strategy was following the study as reported in [17], and [18] respectively. In our experiment, we chose to leave the default hyperparameter arrangement as originally set by all employed libraries as is in order to provide an authentic description of each algorithm's performance.

There are eight classification algorithms experimented in this study, i.e. Logistic Regression (LR), Support Vector Machine (SVM), k-nearest Neighbors (kNN) with k=5, Naïve Bayes (NB), Decision Tree (DT), Random Forest (RF), ExtremeGradientBoosting (XGB), and Multi-Layer Perceptron (MLP). The first five algorithms represent the traditional machine learning algorithm category, RF and XGB represent the ensemble category, while the last algorithm represents the neural network category.

2.4 Evaluation

Evaluation of the effect of resampling techniques combined with classification algorithms was measured using three types of measurements. First, based on single performance measures, which consist of accuracy, precision, sensitivity (recall), and specificity. Second, based on a combination of performance measures, which consist of F-Measure (F1-Score), Adjusted F-Measure (AGF), balanced accuracy, Youden's Index, positive and negative likelihood, Matthew's Correlation Coefficient (MCC), Cohen's Kappa, geometric mean (G-mean), and discriminant power. Third, based on rank performance measures, which consist of Receiver Operating Characteristic

(ROC) Curve / Area Under the Curve (AUC) and lift charts.

For single performance measures, accuracy is normally the most frequently reported value for classification tasks. Accuracy measures the overall classification correctness of a model. Precision measures model exactness in performing classification, while sensitivity (recall) measures the model’s effectiveness in classifying the minority class, by assessing the model's accuracy against positive cases. Meanwhile, specificity measures the model's effectiveness in classifying the majority class (negative), namely assessing the model's accuracy against negative cases. These single performance measures have values ranging from 0 to 1 where the larger the values mean better classifier models.

As for the combined measure, F-Measure (F1-Score) measures a harmonic mean of the precision and sensitivity. The F-Measure value shows the balance between precision and sensitivity. If either the precision or sensitivity value is 0, then the F-Measure is also 0. F-Measure is a good indicator in cases of fairly balanced datasets. For cases of imbalanced datasets, AGF is a better alternative. AGF measures all raw elements of the confusion matrix and gives more weight to patterns that are correctly classified in the minority class (positive cases). High F-Measure and AGF values indicate that the model is a good classifier of the minority class (positive cases). Balanced accuracy is the average accuracy obtained from both positive and negative cases. It is the arithmetic mean of sensitivity and specificity. The balanced accuracy value is slightly lower than the (single measurement) accuracy value if the classification performance is equally good in positive and negative cases since it places same weight to both cases. However, the balanced accuracy value will sharply decrease compared to the (single measurement) accuracy value if the model’s high accuracy is benefited from the distribution of majority class in the dataset. Youden's Index measures the model's ability to avoid misclassification. This index places equal weight on the model's performance in both positive and negative cases. A high Youden's Index value indicates that the model is a good classifier. Likelihood is a ratio of the model's classification performance measurements. Positive likelihood measures the ratio of the model's probability of predicting true positive cases as positive with the probability of predicting true negative cases as positive. While negative likelihood measures the ratio of the model's probability of predicting true negative cases as positive with the probability of predicting true negative cases as negative. A high positive likelihood value indicates good model performance in positive cases, and conversely, a low negative likelihood value indicates good model performance in negative cases. MCC measures the correlation coefficient between the classification prediction results and the observed conditions. MCC is one of the measures that is least affected by imbalanced data. MCC values range from -1 to +1. An MCC value of +1 indicates that the model is able to predict perfectly, a value of 0 indicates that the model's predictive ability is equal to random prediction, and a value of -1 indicates that the model has the worst predictive performance. Cohen's Kappa measures the degree of accuracy that is likely to occur purely by chance. Cohen's Kappa values also range from -1 to +1. A value of +1 indicates that there is a perfect match between the model's prediction and the actual class, a value of 0 indicates that there is no match between the model's prediction and the actual class, and a value of -1 indicates a complete mismatch between the model's prediction and the actual class. G-mean measures the balance between classification performance on positive and negative cases. This measure is very good to use as a guide to avoid overfitting in negative cases and underfitting in positive cases. A low G-mean value indicates poor performance in classifying positive cases even if the negative cases have been correctly classified. Discriminant power measures the combination of sensitivity and specificity. A discriminant power value above 3 indicates a good performing model, a value between 2-3 indicates fair performance, a value between 1-2 indicates limited performance, and a value less than 1 indicates a poor classifier.

As for the rank performance measure, ROC measures the balance between sensitivity and specificity along a continuum using a curve. AUC is the area under the ROC curve. A ROC/AUC value of 1 indicates a good classifier, while a value of 0.5 indicates poor model performance. Lift charts are tools that can be used to measure model effectiveness by calculating the ratio between the outcomes obtained. Lift charts assess the model's ability to detect events of interest in the data. For example, if there are n events of interest in the data to be classified, a good model will be able to place a higher score for the n events than data the non-events data. A model with perfect performance will produce n events of interest as the n highest-rank data. A high lift chart value indicates good performance.

3. Results and Discussions

The experiment as has been conducted in this study had successfully produced a number of insightful data. Section 3.1 presents all gathered data from the experiment in detail.

3.1 Results

Our experiment started with resampling the original CM1 dataset from the NASA PROMISE database. The dataset which was originally consist of 49 positive and 449 negative samples was resampled using ten different techniques. These resampling processes yielded various class proportions as presented in Table 3.

Based on the data as presented in Table 3, it can be seen that oversampling techniques had doubled the dataset size. They did it by randomly duplicating the positive samples (in the case of ROS) or creating synthetic new positive samples (in the case of SMOTE and ADASYN).

Table 3. Class Proportion on Dataset after Resampling

No.	Technique	Positive (Defect)	Negative (No Defect)	Total
1	Original Dataset	49	449	498
2	ROS	449	449	898
3	SMOTE	449	449	898
4	ADASYN	465	449	914
5	RUS	49	49	98
6	Tomek Links	49	432	481
7	NearMiss	49	49	98
8	EasyEnsemble	49	449	498
9	RUSBoost	49	449	498
10	SMOTEENN	232	205	437
11	SMOTETomek	400	400	800

Meanwhile, two of the undersampling techniques sharply decreased the dataset size. RUS randomly deleted negative samples to balance with the size of the positive samples, while NearMiss deleted a lot of negative samples which have close proximity to the positive samples. As for Tomek Links, it deleted only a few negative samples that are paired (have the closest distance) with those from the positive class.

Both ensemble techniques that were tried in our experiment did not change the size and proportion of the original dataset. It seemed because ensemble techniques basically only split the dataset into a number of balanced subsets and then train the models on each subset. They therefore did not either add or delete any samples in the dataset.

As for the combined techniques, i.e. SMOTEENN and SMOTETomek, they yielded relatively balanced class proportions. The total size of the resampled datasets produced by these two techniques is however significantly different. In addition to adding synthetic positive samples, SMOTEENN deleted more negative samples than SMOTETomek. It mainly because SMOTEENN delete those whose most of their neighbors are from the positive cases while SMOTETomek only delete those whose pairs (closest neighbor) are from positive cases.

Subsequently, after finishing with the resampling processes, our experiment was continued with training, validating, and testing eight different algorithms. This part of the process was performed twice. First, it was performed on a 5-fold cross validation session. Second, it was done on 70% : 30% holdout session. The performance of the experimented algorithms was then evaluated using 16 measurements (metrics). Table 4 presents the result of the testing to the classification models.

Table 4. Testing Result of the Classification Models

R	A	5-Fold				Holdout			
		TP	TN	FP	FN	TP	TN	FP	FN
A	1	1,2	87,8	2	8,6	**0**	87	1	12
	2	**0**	**89,8**	**0**	9,8	**0**	88	**0**	12
	3	**0**	88	1,8	9,8	**0**	88	**0**	12
	4	3,2	81,4	8,4	6,6	2	86	2	10
	5	2,6	80,2	9,6	7,2	4	82	6	8
	6	0,4	88,6	1,2	9,4	**0**	87	1	12
	7	1,8	86	3,8	8	1	85	3	11
	8	1,4	77,2	12,6	8,4	**0**	88	**0**	12
B	1	67,2	71,4	18,4	22,6	100	109	30	31
	2	35,4	73,6	16,2	54,4	33	77	19	51
	3	88,6	63,6	26,2	1,2	131	83	56	**0**
	4	28,8	78,2	11,6	**61**	44	120	19	87
	5	89	79,8	10	0,8	131	128	11	**0**
	6	89,8	84,4	5,4	**0**	131	128	11	**0**
	7	89,8	82,4	7,4	**0**	131	126	13	**0**
	8	72,8	46,4	43,4	17	117	54	85	14
C	1	66,8	74	15,8	23	96	114	25	35
	2	37	73,6	16,2	52,8	37	76	20	47
	3	71,6	57,8	32	18,2	96	86	53	35
	4	34,2	78,2	11,6	55,6	48	120	19	83
	5	81,8	75,4	14,4	8	121	110	29	10
	6	87	78,6	11,2	2,8	127	123	16	4
	7	87,8	80,8	9	2	123	123	16	8
	8	63,8	45,6	**44,2**	26	52	120	19	79
D	1	72,2	71,4	18,4	20,8	106	109	33	27
	2	36,8	72,2	17,6	56,2	32	76	18	57
	3	73,4	57	32,8	19,6	110	75	67	23
	4	34	76,8	13	59	43	124	18	90
	5	85,6	76,4	13,4	7,4	124	119	23	9
	6	**91,6**	79,4	10,4	1,4	127	123	19	6
	7	90	78,6	11,2	3	129	120	22	4
	8	63,2	46,8	43	29,8	**133**	30	**112**	**0**
E	1	5,8	6,2	3,6	4	6	12	5	7
	2	3,4	6,2	3,6	6,4	**0**	11	1	8
	3	4,8	5,4	4,4	5	7	6	11	6
	4	3,8	7,4	2,4	6	5	14	3	8
	5	5,2	6	3,8	4,6	8	13	4	5
	6	6,2	7	2,8	3,6	9	14	3	4
	7	6	6,4	3,4	3,8	8	15	2	5
	8	2,8	7	2,8	7	**0**	17	**0**	13
F	1	1,2	84	2,4	8,6	1	123	5	16
	2	**0**	86,4	**0**	9,8	**0**	85	**0**	12
	3	0,2	84,4	2	9,6	**0**	127	1	17
	4	2,8	79,4	7	7	5	123	5	12
	5	2,2	77,8	8,6	7,6	8	122	6	9
	6	0,8	84,8	1,6	9	1	126	2	16
	7	2	81,8	4,6	7,8	2	122	6	15
	8	0,2	86	0,4	9,6	**0**	128	**0**	17
G	1	8,6	9,8	**0**	1,2	11	17	**0**	2
	2	4,8	9,8	**0**	5	1	12	**0**	7
	3	8,2	9,8	**0**	1,6	11	17	**0**	2
	4	8,4	9,8	**0**	1,4	11	17	**0**	2
	5	8,8	8,6	1,2	1	11	15	2	2
	6	8,8	9,4	0,4	1	11	16	1	2
	7	8,8	9,4	0,4	1	11	16	1	2
	8	9	**2,8**	7	0,8	1	17	**0**	12
H	1	1,2	87,8	2	8,6	**0**	133	1	16
	2	**0**	**89,8**	**0**	9,8	**0**	88	**0**	12
	3	**0**	88	1,8	9,8	**0**	131	3	16
	4	3,2	81,4	8,4	6,6	2	126	8	14
	5	2,8	81,2	8,6	7	2	126	8	14
	6	0,4	89,2	0,6	9,4	**0**	130	4	16
	7	1,8	86	3,8	8	**0**	130	4	16
	8	**0**	85,8	4	9,8	1	133	1	15
I	1	1,2	87,8	2	8,6	**0**	133	1	16
	2	**0**	**89,8**	**0**	9,8	**0**	88	**0**	12
	3	**0**	88	1,8	9,8	**0**	131	3	16
	4	3,2	81,4	8,4	6,6	2	126	8	14
	5	2,8	82	7,8	7	1	124	10	15
	6	0,6	88,4	1,4	9,2	**0**	131	3	16
	7	1,8	86	3,8	8	**0**	130	4	16
	8	0,8	71,6	18,2	9	**0**	**134**	0	16
J	1	38,4	35	6	8	47	52	16	17
	2	21,6	35,4	5,6	24,8	20	39	7	22
	3	44	35,6	5,4	2,4	55	57	11	9
	4	24,6	38,4	2,6	21,8	31	64	4	33
	5	43	37,2	3,8	3,4	61	59	9	3

R	A	5-Fold				Holdout			
		TP	TN	FP	FN	TP	TN	FP	FN
	6	45,8	38,4	2,6	0,6	61	59	9	3
	7	45	38,4	2,6	1,4	63	62	6	1
	8	29,2	30,8	10,2	17,2	54	38	30	10
K	1	60,8	66,2	13,8	19,2	87	103	22	28
	2	32,8	66,8	13,2	47,2	33	73	7	47
	3	66,4	54,4	25,6	13,6	89	89	36	26
	4	31	71,6	8,4	49	43	113	12	72
	5	71,2	70,4	9,6	8,8	104	113	12	11
	6	77,8	71,8	8,2	2,2	113	113	12	2
	7	78,4	73	7	1,6	115	113	12	**0**
	8	38	58,8	21,2	42	**0**	125	**0**	**115**

Note:
Resampling Technique (R): Without Resampling (A), ROS (B), SMOTE (C), ADASYN (D), RUS (E), Tomek Links (F), NearMiss (G), EasyEnsemble (H), RUSBoost (I).
Algorithm (A): LR (1), SVM (2), kNN (3), NB (4), DT (5), RF (6), XGB (7), NN/MLP (8).
Red-Colored No.: Best Value; *Blue-Colored No.: Worst Value*

Following the result of our experiment, we observed several interesting findings. *First*, some resampling techniques seemed to have a positive effect on the models' performance. This is apparent, especially when oversampling and combined techniques. Other types of techniques, however, do not likely show any positive effect on models' performance. *Second*, ensemble algorithms (in this case RF and XGB) achieved sufficiently better performance. This result was consistent on both 5-fold cross-validation and 70%: 30% holdout training sessions. One traditional machine learning algorithm, i.e. DT, also scored relatively high performance compared to others. MLP, as the representation of a neural network algorithm, provides low performance in this context.

3.2 Discussions

In order to achieve the objective of this study, our research was designed in a framework to answer three research questions. This discussion section is therefore elaborated around these three research questions.

RQ1: *What type of resampling techniques would provide a potentially positive effect on the performance of the software defects prediction model on CM1 NASA PROMISE data (as the representation of a small yet imbalanced dataset)?*

Based on the result of our experiment, it is obviously evidenced that some type of resampling techniques did provide positive effect on models' performance in classifying the defective software modules on the CM1 NASA PROMISE dataset. It found that most models apparently scored low performance when trained using the original dataset which is highly imbalanced.

The case was different when the models were trained using resampled datasets, especially those processed with oversampling and combined techniques. Most of the models' performance tends to improve when trained using resampled datasets produced by these two types of methods. There is also one undersampling technique, i.e. NearMiss, which also provides a positive effect on the models' performance. However, the effect of the use of these resampling techniques was not strongly similar. This gradation is visualized in a color scale matrix as depicted in Figure 2.

The result from the two types of training sessions, i.e. 5-fold cross-validation and 70%: 30% holdout, as depicted in the colour scale matrix in Figure 2 showing a strongly consistent pattern. A more highlighted (darker) area in the matrix represents a higher effect on the models' performance. It seems that oversampling techniques (i.e. ROS, SMOTE and ADASYN) and combined techniques (i.e. SMOTEENN and SMOTETomek) provide sufficiently positive effects. Interestingly, there was one undersampling technique, i.e. NearMiss, which also offers a good effect on models' performance.

The resampling process of the ROS technique was carried out by duplicating randomly selected samples from the minority class so that the number of samples from the minority class increases and can balance the samples from the majority class [19]. ROS, therefore, does not add any new information to the dataset since it basically only duplicates existing data. For the context of training with small yet imbalanced datasets such as on software defect data, however, models' capability had gained improvement in detecting positive cases benefiting from the addition of more positive samples even though they are basically duplicated with no new information added.

In a slightly different context, the SMOTE technique carried out the resampling process by creating new sample instances based on a convex combination of adjacent samples (creating new synthetic samples by randomly selecting samples from the minority class and interpolating the randomly selected samples with a number of k samples which have the closest Euclidean distance), hence later the model might better recognize important minority classes [20]. However, SMOTE risks producing unrealistic instances and increases the chance of overfitting. With this mechanism, hence, SMOTE provided a relatively similar positive effect to models' performance as ROS even though with lesser power.

Similar to SMOTE, the ADASYN technique carried out the resampling process by creating new synthetic samples from the minority class which are considered more difficult to learn (more difficult to classify correctly) rather than uniformly resampling all samples from the minority class, by way of placing different weight distributions to the two types of classes [21]. However, ADASYN risks in producing instances that do not reflect the underlying distribution of the minority class, increasing the chance of overfitting and is sensitive to noise and outliers in the dataset, thus affecting the quality of the produced new synthetic samples. Considering this mechanism, ADASYN offers a fairly similar positive effect on models' performance as SMOTE.

Most of the undersampling techniques had shown relatively minor effects in our experiment. Except for the case of NearMiss, other undersampling techniques did not provide any valuable improvement to the models' performance.

The resampling process of the RUS technique was carried out by deleting randomly selected data samples from the majority class so that the number of samples from the majority class decreases and can be balanced with the samples from the minority class [19]. However, RUS risks in eliminating samples that may contain valuable information for the training process. Our experiment showed that having a more balanced smaller dataset for the training processes indeed improves models' performance on some measurements such as precision, sensitivity (recall), and G-mean. However, the performance did not show significant improvement based on other metrics.

This case is quite similar with the TomekLinks technique. The TomekLinks technique carried out the resampling process by identifying pairs of samples having the closest Euclidean distance where each of them belongs to a different class (majority/negative and minority/positive) and then deleting part of the paired sample that belongs to the majority class [22]. In some cases, samples from the minority class are also going to be deleted. By deleting these samples, the distance between different classes becomes farther, so that the boundaries for decision-making between two different classes also become clearer. TomekLinks is able to reduce noise in the dataset. However, TomekLinks risks deleting too many samples from the minority class, which potentially leads to underfitting of the training process. Another drawback of TomekLinks is that it normally cannot completely produce balanced data so sometimes it needs to be combined with other resampling techniques. Our experiment proved this theory. The resampling result of TomekLinks still had a relatively imbalanced proportion as can be seen in Table 3. It is, therefore, the effect of TomekLinks on models' performance was lesser compared to two other undersampling techniques (i.e. RUS and NearMiss).

NearMiss, interestingly, provides a positive effect in our experiment. The resampling process of the NearMiss technique was carried out by identifying and removing several samples from the majority class that are close to a set of samples from the minority class [23]. Reducing samples from the majority class will help the model focus on the most relevant and important samples in recognizing and predicting the minority class [24]. However, although not as strong as RUS, NearMiss also risks eliminating samples that may contain valuable information for the training process. By considering this mechanism, it is only a consequence that NearMiss eventually provided a stronger positive effect than RUS.

Among other types of resampling techniques, our experiment showed that both ensemble techniques had almost no effect on the models' performance since they did not add the size of the dataset nor change the class proportion as can be observed in Table 3. The resampling process of the EasyEnsemble technique was carried out by dividing the imbalanced dataset into several balanced subsets, and then training the model on each subset [25]. Each training process on these subsets will provide predictions on the test data and all of the prediction results will be combined using a voting or averaging mechanism to produce the final prediction. With this kind of mechanism, the training process was actually only dependent on several smaller balanced datasets. However, the mechanism to split the original dataset into a number of subsets might not yield as appropriate proportion as those datasets resampled with undersampling techniques.

A quite similar context is applied to the RUSBoost technique. The resampling process of the RUSBoost technique was carried out by combining the RUS with the Boosting technique in order to improve the model's performance in recognizing the minority class [26]. Initially, samples from the majority class are reduced using the undersampling technique. The dataset is then divided into several balanced subsets and training is performed on each subset. Each training process assigns weights (which may be different) to each sample. Each training process on these subsets will provide predictions on the test data and all of the prediction results will be combined using a voting or averaging mechanism to produce the final prediction [25]. The effect of the RUSBoost technique on models' performance was to some extent similar to EasyEnsemble due to their relatively similar resampling process mechanism.

Both combine resampling techniques, i.e. SMOTEENN and SMOTETomek, provide a relatively positive effect on the models' performance in our experiment. This positive effect was mostly perhaps due to taking the benefit of the advantage of applying an oversampling approach as part of these techniques. The resampling process of the SMOTEENN technique was carried out by oversampling the minority class using SMOTE, then followed by undersampling the majority class using ENN to eliminate samples that have the potential to cause confusion or errors in classification by removing samples from the majority class where most of its k closest samples (k = 3) belong to the minority class [27].

Meanwhile, the resampling process of the SMOTETomek technique was carried out by oversampling the minority class using SMOTE, then followed by undersampling the majority class using TomekLinks by randomly selecting samples from the majority class and deleting those with paired closest sample belonging to the minority class [22].

As evidenced in our result, oversampling techniques were observed to have provided a sufficiently positive effect on models' performance. Since the combined technique also applies an oversampling approach as part of their mechanism, they also yielded relatively similar positive effects as oversampling techniques.

The result of our experiment confirmed the study as reported in [9] where SMOTE provided a positive impact on the performance of defect prediction models. This result, however, is different from that of another study as reported in [10] where RUS combined with LR yielded the largest performance improvement of defect prediction models. Nevertheless, as reported in [8], RUS risks removing some instances that could provide more important information for the construction of the prediction models.

In summary, our experiment showed that the oversampling-based and combined approach (which part of its mechanism is oversampling), provided a positive effect on models' performance in predicting defective modules based on the CM1 NASA PROMISE dataset. On the other hand, ensemble-based resampling techniques such as EasyEnsemble and RUSBoost did not prove to provide a positive effect on models' performance hence not suitable to be applied for developing defective software module prediction models. It is important to note, however, that this finding was based on the CM1 NASA PROMISE dataset hence it might have a potential bias to the condition of a particular dataset.

RQ2: *What type of classification algorithms would provide better performance for predicting defective software modules on CM1 NASA PROMISE data (as the representation of a small yet imbalanced dataset)?*

The result of our experiment on testing the classification algorithms for predicting defective software modules based on the CM1 NASA PROMISE dataset that was performed twice, each of which using 5-fold cross-validation and 70%: 30% holdout training, yielded a sufficiently consistent pattern as represented in the colour scale matrix depicted in Figure 3. As can be observed in Figure 3, ensemble algorithms (i.e. RF and XGB) and DT showed higher performance based on most of the applied measurements compared to others.

For the case of traditional machine learning algorithms, our experiment showed that DT achieved sufficiently higher performance compared to their other peers. LR also showed slightly higher performance than SVM, kNN, and NB. LR is an algorithm specifically designed for binary (binomial) classification. This algorithm is quite robust against the effects of small data noise and is insignificantly affected by small multicollinearity [28]. LR can also be used to perform binary classification for high-dimensional data by testing its regression coefficients. LR can effectively control the error rate when testing data [29]. By considering this context, it is reasonable that LR yielded fairly better performance than some other algorithms experimented in this study.

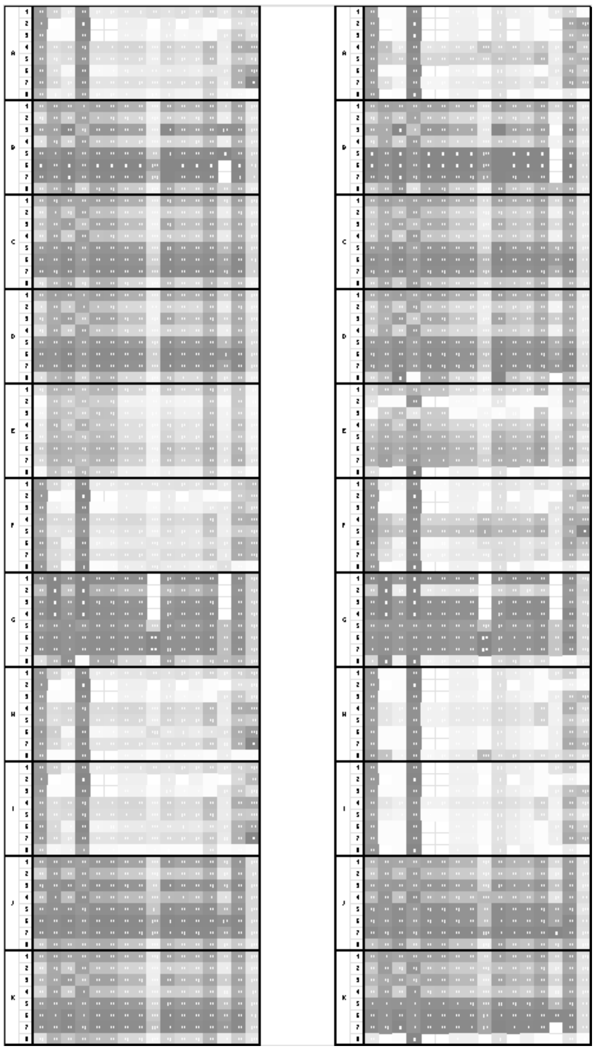
A. Performance from 5-Fold CV B. Performance from Holdout

Figure 2. Color Scale Matrix Comparing the Effect of Resampling Techniques on Models' Performance *(Note: A: Without Resampling, B: ROS, C: SMOTE, D: ADASYN, E: RUS, F: TOMEKLinks, G: NearMiss, H: EasyEnsemble, I: RUSBoost, J: SMOTEENN, K: SMOTETomek)*

In a relatively similar context to LR, SVM is also an algorithm specifically designed for binary (binomial) classification. SVM trains the model and classifies data based on their degree of polarity, by creating a hyperplane that has the farthest distance between each different class [23], [30]. SVM is able to handle the multidimensional classification of complex data. Our experiment, however, showed that SVM did not achieve as high performance as LR. Small differences in the degree of polarity within the dataset may explain why this particular condition occurred.

kNN, which is also categorized as a traditional machine learning algorithm, is basically a pattern recognition algorithm that can be used to perform classification by grouping data based on the class majority of the k closest neighbors (we set k=5) [31]. Training the kNN model using a small yet imbalanced dataset was rather ineffective since its classification mechanism which is based on the surrounding neighbors tends to detect negative cases. When this model is trained on the resampled dataset, the classification tendency will be more or less influenced by their proportion characteristics. This condition is shown in Figure 3 where there are a number of relatively lighter colored gradations on some applied resampling techniques for the kNN area compared to others.

NB, as another traditional machine learning algorithm, employs Bayes' theorem to perform classification. NB calculates the probability of each class and classifies the data based on the one with the highest value [32]. In this regard, NB is not as dependent as kNN on the proportion characteristics of the datasets, as also observed in Figure 3.

Compared to other traditional machine learning algorithms, DT achieved the highest performance in our experiment. DT is a classification algorithm that works like a flow diagram. Data is classified into two (or more) categories at each stage of the classification process, from the "root", "stem", and "branch", to "leaf" where the categories become increasingly similar. The classification process starts from the “root” of the DT and recursively progresses until it reaches the “leaf” with the class label. At each node, a split condition is applied to decide whether the input value should proceed to the left or right subtree until it reaches the leaf node [33]. The root node and each internal node divide the training data into disjoint subsets so that the search space can be significantly and efficiently pruned when all sequences are used as potential features [34]. DT is widely used to create classification models similar to human-like reasoning that are easy to understand and easier to explain than other classifiers including Artificial Neural Networks (ANN) and vector machine classification [35]. The inherent feature selection process implemented by DT was likely to contribute positively to its better performance compared to other traditional algorithms.

Ensemble type of models, i.e. RF and XGB, even achieved slightly better performance compared to DT in our experiment. RF is an algorithm that can be used for classification and regression [36]. This algorithm is a development of DT. RF is a type of bagging ensemble algorithm consisting of several decision trees. Each decision tree is trained with a dataset using the bootstrap aggregation (bagging) process. The training process of a DT uses the classification and regression tree (CART) algorithm. For classification, each node of the DT is designed to minimize impurity. The final decision of the random forest is made by voting on the classification of all DTs [37]. This mechanism explains why RF achieved slightly better performance than DT.

XGB, as another ensemble algorithm, is also a development of DT. It employs gradient-boosted DT and applies a generation community learning approach as a mechanism to improve performance by preventing overfitting in the training process [13]. Having this kind of mechanism made XGB yield comparatively similar performance to RF and DT. Since XGB (and also RF) are developed based on DT, they were likely also gaining performance improvement benefits from the inherently implemented feature selection process of DT.

As for the neural network type of algorithm, in this case, MLP, it achieved unsatisfactory performance in our experiment. This condition is reflected in the color scale matrix as depicted in Figure 2 where the area for MLP is significantly darker meaning that MLP's performance was relatively lower compared to others. MLP is a type of feedforward neural network architecture that consists of at least three layers and is connected to a non-linear activation function. As an extension to the single-layer perceptron (which can only distinguish linearly separated data), MLP is able to distinguish data that cannot be separated linearly [38]. This type of model, however, normally requires a large size of the dataset to achieve sufficiently good performance [39]. It is hence not suitable to be applied as a defective software modules predictor based on a small yet imbalanced dataset such as in the context of this study.

A limitation of our study is that we used the CM1 dataset which contains highly correlated features particularly those derived from Halstead's metrics or other complexity measures. Highly correlated features can impact the effectiveness of oversampling and undersampling techniques by reducing diversity, increasing redundancy, and affecting the balance of the feature space. Since the intention of our research design was to provide as natural software defect prediction model construction process description as possible, we used the original CM1 dataset instead of pre-processing it to reduce its dimension. In real practices, however, highly correlated features can be managed using dimensionality reduction techniques or feature selection methods to reduce redundancy and improve model efficiency when needed.

In summary, our experiment showed that ensemble algorithms that extend the DT classification mechanism such as RF and XGB, achieved sufficiently good performance for predicting defective software modules based on the CM1 NASA PROMISE dataset. DT, as the foundational platform of these two ensemble algorithms, also achieved relatively good performance. On the other hand, MLP apparently not suitable for this type of task. Similar to the result of the experimented resampling techniques, it is important to note that this finding was based on the CM1 NASA PROMISE dataset hence it might have a potential bias to the condition as of the particular dataset.

RQ3: *What type of evaluation measurements (metrics) are deemed suitable for assessing the performance of software defect prediction models on CM1 NASA PROMISE data (as the representation of a small yet imbalanced dataset)?*

Our study employs 16 different measurements (metrics) each representing one out of three types of measurement, namely single, combined, and rank performance measures. The reasons behind the use of these various different measurements were twofold.

First, we plan to get a more comprehensive evaluation of the experimented variables in our study (i.e. the effect of various different resampling techniques and the performance of a number of classification algorithms). Second, we intend to get an insight into what type of measurement could provide a more descriptive assessment for the case of classification performance on defective software modules based on small yet imbalanced datasets.

In order to elaborately analyze these metrics, we calculate both the population and sample variance to the collected models' performance data on each measure (metric). Table 7 summarizes the calculated variance value of each measure.

The higher variance value means that the measures (metrics) could provide a more sensitive capability in detecting smaller differences in models' performance rates. Measurements (metrics) having higher variance values, hence, could offer a more detailed description of models' performance.

For single performance measures, accuracy is normally the most frequently reported value for classification tasks. However, in the case of imbalanced datasets, the predictive value of accuracy can be misleading for the evaluation process [40]. It is evidenced in our experiment where accuracy has the lowest values on both population and sample variance compared to other measures (metrics). This is mainly because accuracy places more weight on the majority than the minority class (by placing true negative as the dividend in the formula) in calculating models' performance. As for precision and sensitivity, both resulted in modest variance values. It is basically due to the relatively equal weight placed on the majority and minority classes by both measures (metrics) in calculating models' performance. In a quite similar context to accuracy, specificity has the second lowest variance values among all measures (metrics). Since it also places more weight on the majority class in calculating the performance, it suffers low variance in accuracy when used to assess models' performance based on imbalanced datasets. Figure 2 and Figure 3 clearly show that the gradation range of accuracy and specificity is not as refined as in other measures on the colour scale matrix. Meanwhile, most of the combined performance measures have relatively modest variance values in our experiment. The explanation for this condition is that all combined performance measures place relatively equal weight on calculating the models' performance by way of combining single performance measures in their formulas. By combining the single performance measures to their calculating formulas, the combined performance measures more or less have placed equal weight on both majority and minority classes in calculating models' performance. Most of them eventually yielded relatively modest variance values in our experiment. A different condition occurred to the positive and negative Likelihood scores which resulted in very high variance values. It is mainly because likelihood scores have relatively longer-range values due to their formulas.

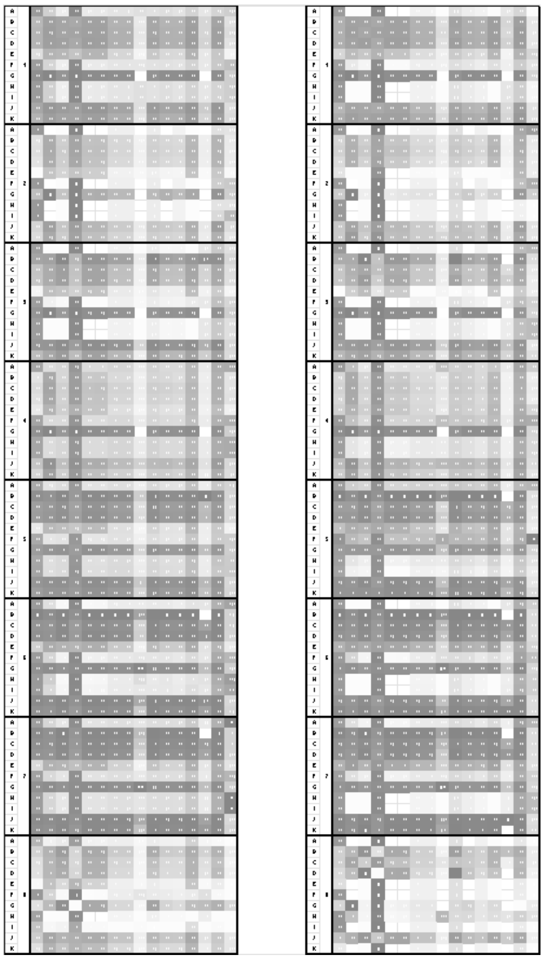
A. Performance from 5-Fold CV   B. Performance from Holdout

Figure 3. Colour Scale Matrix Comparing the Performance of Eight Different Classification Algorithms *(Note: 1: LR, 2: SVM, 3: kNN, 4: NB, 5: DT, 6: RF, 7: XGB, 8: MLP)*

Table 7. Variance from the 16 Measurement (Metric)

No.	Measure	5-Fold		Holdout	
		Var.P	Var.S	Var.P	Var.S
1	Accuracy	**0,017**^{1}	**0,018**^{1}	**0,016**^{1}	**0,016**^{1}
2	Precision	0,086	0,087	0,109	0,111
3	Sensitivity	0,121	0,122	0,148	0,15
4	Specificity	**0,02**^{2}	**0,02**^{2}	**0,022**^{2}	**0,022**^{2}
5	F-Measure	0,09	0,091	0,079	0,08
6	AGF	0,089	0,09	0,077	0,078
7	Blcd. Accry.	**0,027**^{3}	**0,027**^{3}	**0,028**^{3}	**0,028**^{3}
8	Youden's Id	0,107	0,108	0,111	0,113
9	+ Likelhd	**20,00**^{1}	**20,26**^{1}	**14,38**^{1}	**14,58**^{1}
10	- Likelhd	**0,134**^{3}	**0,135**^{3}	**0,154**^{3}	**0,156**^{3}
11	MCC	0,102	0,103	0,11	0,111
12	Chn. Kappa	0,105	0,106	0,111	0,113
13	G-mean	0,082	0,083	0,121	0,123
14	DP	0,092	0,094	0,063	0,064
15	ROC/AUC	0,035	0,035	0,041	0,042
16	Lift Chart	**0,376**^{2}	**0,38**^{2}	**0,513**^{2}	**0,519**^{2}

Note:
1, 2, 3: Rank Order
Red-Colored No.: High Value; *Blue-Colored No.: Low Value*

As for the rank performance measure, in this case is ROC/AUC and Lift Chart, they yielded relatively high variance values on our experiment with Lift even having the second highest values on the chart. Again, it is due to the equally placed weight on both majority and minority classes in calculating the models' performance.

In summary, based on our experiment, both combined and rank-based performance measures (metrics) are deemed suitable for evaluating the models' performance for predicting defective software modules based on the CM1 NASA PROMISE dataset. On the other hand, single performance measures, especially accuracy and specificity are not a good candidate for measuring this type of task. It is also important to note that if the dataset is imbalanced (with a majority of non-defective instances), specificity should be strongly considered as a metric since it puts more penalty on false positives measure.

4. Conclusions

Our study came up with three conclusions. *First*, some types of resampling techniques, especially oversampling and combined approaches, provide sufficiently positive effects on models' performance for predicting defective software modules. One undersampling technique, i.e. NearMiss, also potentially provides a positive effect on models' performance. Contrarily, ensemble-based resampling techniques did not show any positive effect on models' performance hence not suitable for this context. *Second*, ensemble-based algorithms which extend the DT classification mechanism such as RF and XGB, achieved sufficiently good performance for predicting defective software modules. DT, as the foundational platform of these two ensemble algorithms, also achieved relatively good performance. On the other hand, MLP apparently came with disappointing performance for this context. *Third*, combined and rank-based performance measures (metrics) are deemed suitable for evaluating the models' performance for predicting defective software modules. On the contrary, single performance measures, especially accuracy and specificity are not a reliable measure for this context. These three conclusions drawn from our study however cannot be generalized to other contexts since we only employed CM1 NASA PROMISE dataset. We suggest extending this research using more datasets to obtain more statistically trustworthy results and completing the experimented algorithms with a representation of stack-based ensemble in addition to bagging and boosting-based ensemble algorithms that have been tried in this study.

Acknowledgements

The author would like to thank UIN Sunan Kalijaga Yogyakarta for supporting of this study.

References

[1] Ö. F. Arar and K. Ayan, "Software defect prediction using cost-sensitive neural network," *Appl. Soft Comput. J.*, vol. 33, pp. 263–277, 2015, doi: 10.1016/j.asoc.2015.04.045. https://doi.org/10.29207/resti.v4i5.2391.

[2] H. Alsawalqah, H. Faris, I. A. B, and L. Alnemer, "Hybrid SMOTE-Ensemble Approach," *Adv. Intell. Syst. Comput.*, vol. 1, 2017, doi: 10.1007/978-3-319-57141-6.

[3] L. Qiao, X. Li, Q. Umer, and P. Guo, "Deep learning based software defect prediction," *Neurocomputing*, vol. 385, pp. 100–110, 2020, doi: 10.1016/j.neucom.2019.11.067.

[4] R. B. Bahaweres, K. Zawawi, D. Khairani, and N. Hakiem, "Analysis of statement branch and loop coverage in software testing with genetic algorithm," *Int. Conf. Electr. Eng. Comput. Sci. Informatics*, vol. 2017-Decem, no. September, pp. 19–21, 2017, doi: 10.1109/EECSI.2017.8239088.

[5] F. Matloob *et al.*, "Software Defect Prediction Using Ensemble Learning : A Systematic Literature Review," *IEEE Access*, vol. 9, no. October, pp. 98754–98771, 2021, doi: 10.1109/ACCESS.2021.3095559.

[6] M. N. M. Rahman, R. A. Nugroho, M. R. Faisal, F. Abadi, and R. Herteno, "Optimized multi correlation-based feature selection in software defect prediction," *TELKOMNIKA Telecommun Comput El Control J.*, vol. 98, no. 3, pp. 598-605, 2024, doi: https://doi.org/10.12928/TELKOMNIKA.v22i3.25793.

[7] U. S. Bhutamapuram and R. Sadam, "With-in-project defect prediction using bootstrap aggregation based diverse ensemble learning technique," *J. of King Saud University - Computer and Information Sciences.*, p. 4832864, 15 pages 2021, https://doi.org/10.1155/2021/4832864.**S.**

[8] S. Feng, J. Keung, Y. Xiao, P. Zhang, X. Yu and X. Cao, "Improving the undersampling technique by optimizing the termination condition for software defect prediction," *Expert Sys. with App.*, vol. 235, p. 121084, 2024, https://doi.org/10.1016/j.eswa.2023.121084.

[9] A. Saifudin, S. W. H. L. Hendric, B. Soewito, F. L. Gaol, E. Abdurachman, and Y. Heryadi, "Tackling Imbalanced Class on Cross-Project Defect Prediction Using Ensemble SMOTE," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 662, no. 6, pp. 0–10, 2019, doi: 10.1088/1757-899X/662/6/062011.

[10] C. Tantithamthavorn, A. E. Hassan and K. Matsumoto, "The Impact of Class Rebalancing Techniques on the Performance and Interpretation of Defect Prediction Models," *IEEE Transactions on Software Engineering*, vol. 46, no. 11, pp. 1200-1219, 1 Nov. 2020, doi: 10.1109/TSE.2018.2876537.

[11] Anna, "Penerapan k-nearest neighbor menggunakan pendekatan random walk over-sampling menangani ketidakseimbangan kelas pada prediksi cacat software," *Master Thesis, STMIK Nusa Mandiri*, 2018, [Online]. Available: https://repository.bsi.ac.id/repo/files/354032/download/Full-Tesis-Anna.pdf.

[12] M. Sonhaji Akbar and S. Rochimah, "Prediksi Cacat Perangkat Lunak Dengan Optimasi Naive Bayes Menggunakan Pemilihan Fitur Gain Ratio," *J. Sist. Dan Inform.*, vol. 11, no. 1, pp. 147–155, 2018.

[13] S. K. Pemmada, J. Nayak, H. S. Behera, and D. Pelusi, "Light Gradient Boosting Machine in Software Defect Prediction: Concurrent Feature Selection and Hyper Parameter Tuning," in *Intelligent Sustainable Systems*, J. S. Raj, Y. Shi, D. Pelusi, and V. E. Balas, Eds., Singapore: Springer Nature Singapore, 2022, pp. 427–442.

[14] A. Alazba and H. Aljamaan, "Software Defect Prediction Using Stacking Generalization of Optimized Tree-Based Ensembles," *Appl. Sci.*, vol. 12, no. 9, 2022, doi: 10.3390/app12094577.

[15] Y. Al-Smadi, M. Eshtay, A. Al-Qerem, S. Nashwan, O. Ouda, and A. A. Abd El-Aziz, "Reliable prediction of software defects using Shapley interpretable machine learning models," *Egypt. Informatics J.*, vol. 24, no. 3, p. 100386, 2023,doi: https://doi.org/10.1016/j.eij.2023.05.011.

[16] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the nasa software defect datasets," *IEEE*

Trans. On Soft. Eng., vol. 39, no. 9, pp. 1208–1215, 2013, doi: 10.1109/TSE.2013.11.

[17] S. Raschka, “Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning,” *arXiv*, primaryClass={cs.LG}, eprint 1811.12808, 2012, url=https://arxiv.org/abs/1811.12808.

[18] Q. H. Nguyen, H-B. Ly, L. S. Ho, N. Al-Ansari, H. V. Le, V. Q. Tran, I. Prakash and B. T. Pham, “Influence of Data Splitting on Performance of Machine Learning Models in Prediction of Shear Strength of Soil,” *Math. Problems in Eng.*, vol. 29, issue. 4, pp. 565-570, 2019, doi: 10.1145/3459665.

[19] C. Yang, E. A. Fridgeirsson, J. A. Kors, J. M Reps and P. R. Rijnbeek, “Impact of random oversampling and random undersampling on the performance of prediction models developed using observational health data,” *J. of Big Data.*, vol. 11, no. 7, pp. 2196-1115, 2024, https://doi.org/10.1186/s40537-023-00857-7.

[20] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: Synthetic Minority Over-sampling Technique,” *J. Artif. Intell. Res.*, vol. 16, no. Sept. 28, pp. 321–357, 2002, https://arxiv.org/pdf/1106.1813.pdf%0Ahttp://www.snopes.com/horrors/insects/telamonia.asp.

[21] H. He, Y. Bai, E. A. Garcia and S. Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, Hong Kong, 2008, pp. 1322-1328, doi: 10.1109/IJCNN.2008.4633969.

[22] T. Sasada, Z. Liu, T. Baba, K. Hatano and Y. Kimura, " A Resampling Method for Imbalanced Datasets Considering Noise and Overlap," *Procedia Computer Science (24 Int. Conf. on Knowledge-based & Intelligent Information & Engineering System)*, vol. 176, 2020, pp. 420-429, doi: 10.1016/j.procs.2020.08.043.

[23] Y. Tang, Y. Q. Zhang, and N. V. Chawla, “SVMs modeling for highly imbalanced classification,” *IEEE Trans. Syst. Man, Cybern. Part B Cybern.*, vol. 39, no. 1, pp. 281–288, 2009, doi: 10.1109/TSMCB.2008.2002909.

[24] F. Wang *et al.*, “Imbalanced data classification algorithm with support vector machine kernel extensions,” *Evol. Intell.*, vol. 12, no. 3, pp. 341–347, 2019, doi: 10.1007/s12065-018-0182-0.

[25] P. Mooijman, C. Catal, B. Tekinerdogan, A. Lommen, and M. Blokland, “The effects of data balancing approaches: A case study,” *Appl. Soft Comput.*, vol. 132, p. 109853, 2023, doi: 10.1016/j.asoc.2022.109853.

[26] W. Zheng *et al.*, “Machine learning for imbalanced datasets: Application in prediction of 3d-5d double perovskite structures,” *Comput. Mater. Sci.*, vol. 209, no. March, 2022, doi: 10.1016/j.commatsci.2022.111394.

[27] M. Xing *et al.*, “Predict DLBCL patients’ recurrence within two years with Gaussian mixture model cluster oversampling and multi-kernel learning,” *Comput. Methods Programs Biomed.*, vol. 226, p. 107103, 2022, doi: 10.1016/j.cmpb.2022.107103

[28] J. Bakerman, K. Pazdernik, G. Korkmaz, and A. G. Wilson, “Dynamic logistic regression and variable selection: Forecasting and contextualizing civil unrest,” *Int. J. Forecast.*, vol. 38, no. 2, pp. 648–661, 2022, doi: 10.1016/j.ijforecast.2021.07.003.

[29] M. Fang and D. T. N. Huy, “Building a cross-border e-commerce talent training platform based on logistic regression model,” *J. High Technol. Manag. Res.*, vol. 34, no. 2, p. 100473, 2023, doi: 10.1016/j.hitech.2023.100473.

[30] Y. Zhang, “Support Vector Machine Classification Algorithm and Its Application,” *Information Computing and Applications (ICICA 2012. Communications in Computer and Information Science)*, vol. 308, part II, pp. 179–186, 2012, Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-34041-3_27.

[31] P. Cunningham and S. J. Delany, “k-Nearest Neighbour Classifiers - A Tutorial,” *ACM Computing Surveys.*, vol. 54, no. 6, pp. 1–25, 2021, doi: 10.1145/3459665.

[32] Wickramasinghe and H. Kalutarage “Naive Bayes: applications, variations and vulnerabilities: a review of literature with code snippets for implementation”, *Soft Comput*, vol. 25, pp. 2277–2293, 2021, https://doi.org/10.1007/s00500-020-05297-6.

[33] S. Tangirala, “Evaluating the impact of GINI index and information gain on classification using decision tree classifier algorithm,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 11, no. 2, pp. 612–619, 2020, doi: 10.14569/ijacsa.2020.0110277.

[34] Z. He, Z. Wu, G. Xu, Y. Liu, and Q. Zou, “Decision Tree for Sequences,” *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 1, pp. 251–263, 2023, doi: 10.1109/TKDE.2021.3075023.

[35] F. Zou, “Research on data cleaning in big data environment,” *Proc. - 2022 Int. Conf. Cloud Comput. Big Data Internet Things, 3CBIT 2022*, pp. 145–148, 2022, doi: 10.1109/3CBIT57391.2022.00037.

[36] Z. Jin, J. Shang, Q. Zhu, C. Ling, W. Xie, and B. Qiang, “RFRSF: Employee Turnover Prediction Based on Random Forests and Survival Analysis,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 12343 LNCS, pp. 503–515, 2020, doi: 10.1007/978-3-030-62008-0_35.

[37] J. Lee *et al.*, “Data-driven disruption prediction using random forest in KSTAR,” *Fusion Eng. Des.*, vol. 199, no. December 2023, p. 114128, 2024, doi: 10.1016/j.fusengdes.2023.114128.

[38] J. Singh and R. Banerjee, "A Study on Single and Multi-layer Perceptron Neural Network," *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC)*, Erode, India, 2019, pp. 35-40, https://doi.org/10.1016/j.ijmst.2019.06.009.

[39] Y. Pu, D. B. Apel, V. Liu and H. Mitri, “Machine learning methods for rockburst prediction-state-of-the-art review,” *Int. J. of Mining Sci. and Tech.*, vol. 29, issue. 4, pp. 565-570, 2019, doi: 10.1145/3459665.

[40] J. S. Akosa, “Predictive Accuracy: A Misleading Performance Measure for Highly Imbalanced Data,” *in Proceeding*, 2017, https://api.semanticscholar.org/CorpusID:43504747.