Jurnal Ilmiah Teknik Elektro Komputer dan Informatika (JITEKI)
Vol. 10, No. 2, June 2024, pp. 201-216
ISSN: 2338-3070, DOI: 10.26555/jiteki.v10i2.28521

Effect of SMOTE Variants on Software Defect Prediction Classification Based on Boosting Algorithm

Rahmina Ulfah Aflaha, Rudy Herteno, Mohammad Reza Faisal, Friska Abadi, Setyo Wahyu Saputro
Computer Science Department, Lambung Mangkurat University, Banjarbaru, Indonesia

ARTICLE INFO

Article history:

Received March 08, 2024
Revised May 15, 2024
Published June 21, 2024

Keywords:

Software Defect Prediction;
Imbalance;
SMOTE Variants;
Boosting

ABSTRACT

Detecting software defects early on is critical for avoiding significant financial losses. However, building accurate software defect prediction models can be challenging due to class imbalance, where the data for defective modules is much less than for standard modules. This research addresses this issue using the imbalanced dataset NASA MDP. To address this issue, researchers have proposed new methods that combine data level balancing approaches with 14 variations of the SMOTE algorithm to increase the amount of defective module data. An algorithm-level approach with three boosting algorithms, Catboost, LightGBM, and Gradient Boosting, is applied to classify modules as defective or non-defective. These methods aim to improve the accuracy of software defect prediction. The results show that this new method can produce a more accurate classification than previous studies. The DSMOTE and Gradient Boosting pair with 0.9161 has the highest average accuracy (0.9161). The DSMOTE and Catboost model achieved the highest average AUC value (0.9637). The ADASYN kernel and Catboost showed the best ability to perform the average G-mean value (0.9154). The research contribution to software defect prediction involves developing new techniques and evaluating their effectiveness in addressing class imbalance.

Corresponding Author:

Rudy Herteno, Computer Science Department, Universitas Lambung Mangkurat, Banjarbaru, Indonesia
Email: rudy.herteno@ulm.ac.id

1. INTRODUCTION

In this digital age, software is becoming essential in various aspects of human life in the 21st century [1]. A minor defect in any software can lead to software failure. According to Huang & Strigini, in 2018, the cost of finding and fixing defects worldwide was estimated to be trillions of dollars [2]. Defect prevention is critical because once defects are created in the code, it is difficult to ensure that all such defects will be found and removed through testing. Therefore, the prediction of problematic modules at an early stage is more beneficial and also reduces the overall cost of the software [3]. Software defect prediction uses machine learning classifiers to identify defect-prone software modules [4]. By identifying problematic modules early, developers can focus on them during testing and reduce the chances of undetected defects.

However, building an effective defect prediction model can be complicated due to the problem of imbalanced data. This occurs when the amount of data in two different classes in a dataset is imbalanced, where there is a significant difference between the amount of data in the majority and minority classes. Meng and Li revealed that the majority class has significantly more data, while the minority class has significantly less data [5]. This imbalance can disrupt the learning process of the model and result in inaccurate predictions [6]. There are several solutions to overcome the problem of class imbalance, which are divided into three types of approaches: data level, algorithm level and hybrid [7]. At the data level, oversampling approaches are widely used to overcome class imbalance [8], and ensemble learning can be used at the algorithm level to improve model performance. In this study using 3 boosting algorithms namely Catboost, LightGBM, and Gradient Boosting.

Vardhan *et al.* found that Catboost is very effective in detecting software bugs. The study compared Catboost with other classification algorithms using the NASA MDP dataset. Catboost produced higher accuracy, with scores of 0.82 on CM1, 0.72 on JM1, 0.84 on KC2, and 0.87 on PC3. These results indicate that Catboost can help improve software quality [9]. Based on the research of Fahimuzzman Sohan *et al.* showed that Gradient Boosting is a very effective method for handling imbalanced datasets. This study uses the SeaCraft dataset, Gradient Boosting was compared with several other classification methods. The results show that Gradient Boosting produces the highest accuracy, which is 0.95 on imbalanced data [10]. In research [11], focused on predicting software defects using three classification models: Gradient Boosting LightGBM, XGBoost, and CatBoost. This research uses datasets CM1, JM1, PC1, KC1, and KC2. The results showed that LightGBM with GridSearchCV library gave the most dominant performance (98%) compared to other algorithms.

Research by Malhotra and Jain found that using a resampling technique before applying the Boosting method can help improve prediction models. One of the resampling techniques that can be used to address an unbalanced dataset is SMOTE [12]. Although SMOTE is commonly used in software defect prediction research, there is still relatively little research on its variants. In fact, SMOTE Variants can provide a wider range of solutions to overcome data imbalance problems in machine. Hassanat *et al.* evaluated and ranked over 70 over-sampling methods on three imbalanced real-world datasets. The MOT2LD (Rank 8), CURE SMOTE (Rank 9), Edge Det SMOTE (Rank 17), SDSMOTE (Rank 16), DSMOTE (Rank 1), and NT SMOTE (Rank 11) methods showed superior performance to SMOTE (Rank 24) based on the average error value [13]. Kovács evaluated the performance of several oversampling methods in improving classification performance. In SVM, GSMOTE, and CE classification, SMOTE showed higher AUC values than SMOTE, 0.9062 and 0.9056 compared to 0.8999. In Decision Tree classification, SMOTE IPF, Assembled SMOTE, and Lee also showed higher AUC values, 0.8828, 0.8834, and 0.8883, compared to 0.8809 in SMOTE. The results of this study show that oversampling can help improve classification performance, and some oversampling methods proved to be more effective than SMOTE [14]. Zhai *et al*. [15] proposed an OUPS oversampling method that uses an enhanced GAN to improve classification performance on imbalanced numerical datasets. The results show that OUPS is more effective than SMOTE in improving AUC.

According to the research on [16], DTS2, G-SMOTE, Edge Det SMOTE, and SDSMOTE datasets produce better results than SMOTE concerning AUC. The study revealed that G-SMOTE achieved an AUC value of 0.8857, while Edge Det SMOTE obtained an AUC value of 0.8798, and SDSMOTE obtained an AUC value of 0.8823On the other hand, SMOTE achieved an AUC value of only 0.8779. Ding *et al.* conducted research on state-of-the-art sampling techniques across nine image classification datasets, each with varying levels of imbalance, and the ADASYN Kernel produced higher AUC than SMOTE on several datasets, including Lung, Colon, Glioma, Haberman, Vehicle, CMC, Balance, and Zooscan [17]. Dudjak and Martinović evaluated the performance of several oversampling methods and classification algorithms. It was found that SMOTE D was the best oversampled for 5-NN classification, with an F1 score of 4.175, outperforming SMOTE with a score of 5.275 [18]. Research [19] shows the superiority of the CBSO method over SMOTE in handling unbalanced data. CBSO achieved better performance on the Glass, Yeast, and Pima datasets, with G-mean of 0.9536, 0.7683, and 0.7367, compared to SMOTE's 0.9412, 0.7298, and 0.7170. This significant increase in G-mean suggests that CBSOs are more effective in addressing minority classes in the data, resulting in more accurate classification models.

According to researches [13]-[19] study SMOTE Variants have performed better than the original SMOTE. However, using SMOTE Variants for software defect prediction with the NASA MDP dataset still requires further research. Another research study [12] suggests boosting after SMOTE can enhance model performance. Research [9]-[11] have demonstrated that boosting algorithms like CatBoost, LightGBM, and Gradient Boosting are highly effective in software defect prediction and produce accurate results.

The aim of this study is to identify the best combination of SMOTE and its Variants to enhance the performance of Catboost, LightGBM, and Gradient Boosting algorithms in predicting software defects, especially in cases where data imbalance is a problem. This research introduces a novel approach to software defect prediction that integrates 14 different variations of SMOTE with three boosting algorithms. This method is expected to address the issue of imbalanced datasets and generate more accurate classifications than previous research.

2. METHODS

The research flow carried out in the study can be seen Fig. 1. The first step is to process the initial data by converting categorical variables into numerical values. Then, the data is split into training and test data. To address any imbalances in the data, we apply SMOTE Variants before utilizing boosting algorithms such as

Catboost, LightGBM, and Gradient Boosting. To ensure optimal performance of the machine learning model in various scenarios, we evaluate its performance using ten cross-validation and various evaluation metrics such as accuracy, AUC, and g-mean.

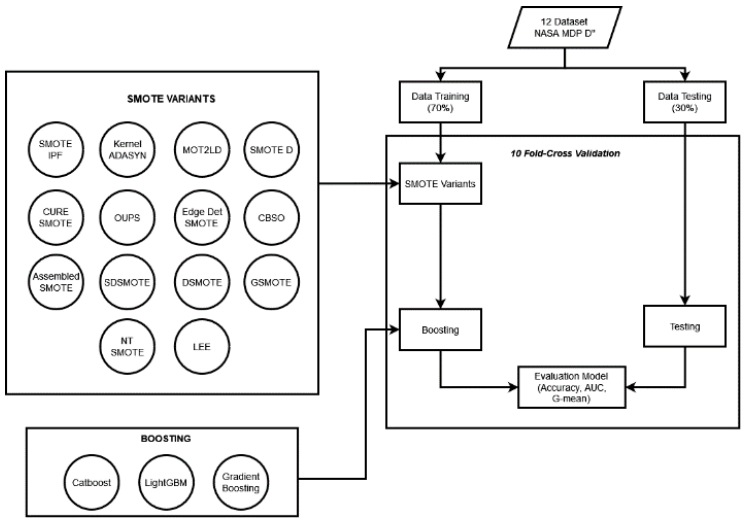

Fig. 1. Research Flow

2.1. Data Collection

NASA MDP (Metrics Data Program) is a public data repository widely used in software defect prediction research [20]. The dataset used in this study is the NASA MDP with version D" or a cleaned data version. It comprises datasets from 12 projects: CM1, JM1, KC1, KC3, MC1, MC2, MW1, PC1, PC2, PC3, PC4, and PC5. Table 1 contains details from the NASA MDP D dataset.

Table 1. NASA MDP D"

Dataset	Attributes	Module	Defective	Non-Defective	Defective (%)
CM1	38	327	42	285	12.8
JM1	22	7720	1,612	6,108	20.8
KC1	22	1162	294	868	25.3
KC3	40	194	36	158	18.5
MC1	39	1952	36	1916	1.8
MC2	40	124	44	80	35.4
MW1	38	250	25	225	10
PC1	38	679	55	624	8.1
PC2	37	722	16	706	2.2
PC3	38	1053	130	923	12.3
PC4	38	1270	176	1094	13.8
PC5	39	1694	458	1236	27.0

2.2. Prepocessing

Preprocessing is essential for maximizing the accuracy of machine learning models fed with datasets [21]. When dealing with ordinal categorical features, where order matters, label encoding is the preferred technique [22]. This is exemplified by the "target class" attribute with nominal Boolean values ("Y" for defective and "N" for non-defective modules). Label encoding converts these qualitative labels ("Y" and "N") into numerical values in Table 2. This transformation enhances model comprehension and processing, ultimately improving training and evaluation efficiency.

Table 2. Lable Encoding

Label Encoding	*Description*
1	Defective (Y)
0	Non-Defective (N)

2.3. Data Splitting

The research data is divided into two parts: training data and testing data. This data separation is a common technique in data science and machine learning to objectively evaluate models. The research data was split into two parts, with a ratio of 70% for training data and 30% for test data. This ratio was chosen based on research [23] which demonstrated that the 70:30 ratio resulted in the best model performance compared to other ratios. To evaluate model performance, 10-fold cross-validation is used. This method divides the training data into ten subsets, and the model is trained and tested on nine subsets in turn. The average of the evaluation metrics from these ten iterations is used to estimate the model's performance on unseen data. The selection of 10-fold cross-validation was based on [23]-[25] studies showing that k=10 resulted in more accurate estimations, reduced bias, and overfitting.

2.4. SMOTE Variants

SMOTE (Synthetic Minority Oversampling Technique) is a resampling technique that generates new samples to increase the minority class by creating synthetic instances along the line segments [27]. Over the past decade, many variants for SMOTE have been proposed. To handle the issue of data class imbalance, this study utilizes 14 specifically selected SMOTE Variants. These variants were chosen after performing a comparative evaluation of 85 variants on large, unbalanced datasets, as referenced in [14]. SMOTE Variants used in this study is available in a Python package that can be accessed on the documentation page http://smote-variants.readthedocs.io.

a. SMOTE IPF consists of SMOTE and IPF (Iterative Filter Partition). SMOTE-IPF improves the oversampling process in SMOTE by removing noisy data points from the majority class to enhance model performance in imbalanced classification tasks [28].
b. Kernel ADASYN uses a kernel-based adaptive synthetic over-sampling approach to address data imbalance. These techniques rely on local information rather than the overall data distribution [29].
c. MOT2LD is a method used for taking measurements by determining the Local Density dataset in a Low Dimensional Space. Initially, the dataset is represented in a smaller dimension, meaning feature selection is done to find the best combination. The results of the representation are obtained by clustering the values of Local Minority Density and Local Majority Count, which indicate the required levels of samples to create new synthetic data [30].
d. OUPS is an oversampling technique that uses propensity score matching to select neighbors and generates synthetic samples through SMOTE [31].
e. SMOTE D algorithm selects minority instances based on their deviation of distances. This algorithm has the least impact on the performance of the majority class and it statistically outperforms other algorithms [18].
f. CURE SMOTE (Combination of Clustering Using Representatives Synthetic Minority Oversampling Technique) Experiments on the UCI imbalanced data show that the original Synthetic Minority Over-sampling Technique is effectively enhanced by the use of the combination of clustering using representative algorithm [32]. CURE-SMOTE uses the hierarchical clustering algorithm CURE to clear outlier data before applying SMOTE [33].
g. Edge Det SMOTE proposed edge detection algorithm Egde-Det, this method generates synthetic data based on the sample weight calculated by the overall magnitude of gradient.
h. CBSO is a cluster-based synthetic oversampling algorithm that effectively deals with imbalanced data problems. It has been used widely in data analysis and has proven effective with synthetic oversampling techniques like SMOTE and ADASYN, which use a KNN approach. The algorithm can also reduce the impact of imbalanced data on classification by oversampling only the minority examples near the borderline [34].
i. Assembled SMOTE is a minority oversampling technique that connects minority samples near the decision boundary with minority samples farther away, thereby improving data diversity and machine learning model performance in handling imbalanced data [35].
j. SDSMOTE is a method used to address the issue of imbalanced datasets with a significantly smaller minority class, often resulting in inaccurate models. This method creates synthetic minority class samples to tackle the problem. Unlike SMOTE, which randomly selects neighbors, SDSMOTE views the data as a solid graph and ensures that the overall distribution remains stable by carefully positioning new samples. This leads to more effective oversampling without compromising the integrity of the original data [36].
k. DSMOTE (Diversity and Separable Metrics in Over-Sampling Technique) method that improves the accuracy. Anomalous samples are removed from the negative class. The top three samples are then considered based on a criteria and synthetic data is generated based on these samples [37].

l. GSMOTE employs bootstrapping as part of its hybrid sampling technique, effectively addressing the challenge of highly skewed data distributions [38].
m. NT SMOTE is an up-sampling approach that utilizes the neighborhood triangular synthetic minority over-sampling technique to address the issue of unsatisfactory results in imbalanced risk prediction for minority class samples [39].
n. Lee generates a new synthetic observation with SMOTE, and includes a step to assess whether the observation is noise. For example, when k is 5 and the rejection level is 3, the resulting synthetic observations are removed for C with three or more of the five surrounding observations. However, in the case of F, with two or less majority class observations, the synthetic observations are maintained [40].

2.5. Algorithm Boosting

Boosting, an algorithm-level approach used in this research is Catboost, LightGBM, and Gradient Boosting. CatBoost is an algorithm was developed by Yan dex researchers for gradient boosting on decision trees, which can handle categorical features in the training phase [41]. In the decision tree, the label means will also be the criterion for node splitting, also known as greedy target variable statistics, and the formula is expressed as (1) [42]:

$$\hat{x}_k^i = \frac{\sum_{j=1}^{p-1}[x_{j,k} = x_{i,k}] . \gamma_i}{\sum_{j=1}^{n}[x_{j,k} = x_{i,k}]} \quad (1)$$

LightGBM is one of the boosting algorithms that use a "leaf-wise" algorithm to grow trees vertically. Microsoft launched LightGBM (Light Gradient Boosting Machine) in 2017 [43]. In addition, LightGBM uses a histogram-based method to find the best splitting candidate [44]. According to the level-wise growth strategy, the leaves on the same layer are simultaneously split. Leaves on same layer are indiscriminately treated, whereas they have different information gain. Information gain indicates the expected reduction in entropy caused by splitting the nodes based on attributes (2) [45].

$$IG(B,V) = \sum_{d=1}^{D} -p_d \log_2 p_{d,} - \sum_{v\in(V)} \frac{|B_v|}{B} En(B_v) \quad (2)$$

where p_d is the ration of B pertaining to category d, D is the number of categories, is the value of attribute V, and B_v is the subset of B for which attribute has value.

Gradient Boosting is one of the ensemble machine-learning methods introduced by Friedman in 2001 [46]. Gradient Boosting is applied to regression and classification problems; this model works by giving a certain weight to each data point [47]. Gradually, the learning process builds a model by combining several weak prediction models, usually a decision tree [48]. The GBT (gradient boosting tree) can be defined as the summation of nn regression-trees [49] in (3). Where every $f_i(x_t)$ is a decision tree (regression-tree).

$$F_n(x_t) = \sum_{i=1}^{n} f_i(x_t) \quad (3)$$

2.6. Evaluation

In Software Defect Prediction, performance assessment is usually calculated based on a confusion matrix [50]. An example of a confusion matrix can be seen in the following Table 3. Classification results are evaluated in terms of accuracy, AUC and G-mean. In the field of machine learning, accuracy has always been the primary metric used to measure the performance of conventional algorithms. However, when dealing with imbalanced datasets, where one class has significantly more samples than the others, the reliability of this metric decreases. In such cases, the accuracy can overestimate the classifier's ability to identify the majority class [51]. Therefore, this study has adopted more robust evaluation metrics such as AUC and G-mean to assess the performance of the classifier [34], [50].

Table 3. Confusion Matrix

	Predictive Negative	*Predictive Positive*
Actual Negatif	True Negative (TN)	False Positive (FP)
Actual Positive	False Negative (FN)	True Positive (TP)

a. Accuracy is the ratio of correctly classified modules to the total number of modules. Equation (4) used to calculate Accuracy is as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (4)$$

b. AUC Area Under Curve (AUC) is the percentage of area under the Receiver Operator Characteristic (ROC) curve. Equation (5) used to calculate Accuracy is as follows:

$$AUC = \frac{1 + TP_r - FP_r}{2} \quad (5)$$

c. G-mean (Geometric Mean) measures central tendency that calculates the average of sensitivity and specificity. Equation (6) used to calculate Accuracy is as follows:

$$G - mean = \sqrt{Precision \times Recall} \quad (6)$$

3. RESULTS AND DISCUSSION

This study assessed 15 classification models, each utilizing a boosting approach, specifically CatBoost, LightGBM, and Gradient Boosting. The initial model runs without SMOTE, while the other 14 models run with 14 different SMOTE variations. Subsequently, the accuracy, AUC, and G-mean values will be observed for each model.

3.1. Classification with Catboost

The software defect prediction model with the Catboost algorithm was measured, and the results are presented in Table 4, Table 5, and Table 6. Table 4 displays the accuracy results, Table 5 presents the AUC, and Table 6 shows the G-mean results. Using SMOTE Variants in software defect prediction models has proven more effective than Boosting models without SMOTE Variants. Higher accuracy, AUC scores, and G-mean were observed in all datasets, and based on the test results in Table 4, Table 5, and Table 6.

Table 4. Accuracy Results of SMOTE Variants in Catboost Classification

	CM1	JM1	KC1	KC3	MC1	MC2	MW1	PC1	PC2	PC3	PC4	PC5
Without SMOTE	0.8678	0.7946	0.7718	0.4199	0.9799	0.7042	0.8873	0.9129	0.9770	0.8553	0.8956	0.7778
SMOTE IPF	0.9347	0.8636	0.8348	0.8955	0.9908	0.8205	0.9464	0.9578	0.9804	0.9272	0.9485	0.8391
Kernel ADASYN	0.9446	0.8626	0.8290	0.9045	0.9915	0.8295	0.9430	0.9667	0.9814	0.9264	0.9491	0.8385
MOT2LD	0.9160	0.8650	0.8215	0.9095	0.9885	0.7627	0.9375	0.9552	0.9911	0.9281	0.9466	0.8405
OUPS	0.8392	0.8590	0.7928	0.8455	0.9849	0.7409	0.8614	0.9178	0.9578	0.8976	0.9233	0.8235
SMOTE D	0.9176	0.8679	0.8271	0.8927	0.9904	0.7848	0.9241	0.9592	0.9892	0.9255	0.9365	0.8461
CURE SMOTE	0.9146	0.8659	0.8430	0.9045	0.9901	0.7568	0.9369	0.9522	0.9833	0.9287	0.9381	0.8529
Edge Det SMOTE	0.9422	0.8622	0.8233	0.8864	0.9908	0.7932	0.9431	0.9589	0.9833	0.9294	0.9452	0.8345
CBSO	0.9473	0.8615	0.8143	0.9000	0.9901	0.8121	0.9276	0.9511	0.9814	0.9257	0.9452	0.8379
Assembled SMOTE	0.9297	0.8621	0.8290	0.8955	**0.9919**	0.8114	0.9430	0.9622	0.9824	0.9332	0.9504	0.8402
SDSMOTE	0.9322	0.8619	0.8232	0.9182	0.9915	0.7311	0.9464	0.9622	0.9833	0.9279	0.9517	0.8356
DSMOTE	0.9222	0.8778	0.8602	0.8773	0.9901	0.8106	0.9403	0.9567	0.9853	0.9325	0.9471	0.8645
G SMOTE	0.9422	0.8651	0.8315	0.8909	**0.9919**	0.8136	0.9337	0.9633	0.9843	0.9416	0.9452	0.8420
NT SMOTE	0.9196	0.8622	0.8290	0.8909	0.9901	0.7939	0.9556	0.9556	0.9814	0.9226	0.9356	0.8420
Lee	0.9146	0.8665	0.8364	0.9091	0.9897	0.8023	0.9462	0.9533	0.9843	0.9287	0.9381	0.8460

Assembled SMOTE and G-SMOTE demonstrated the highest accuracy performance, achieving a score of 0.9919 in the MC1 dataset. This indicates that combining these two techniques improved the model's ability to classify instances within this dataset accurately. On the other hand, the Without SMOTE pair in the KC3 dataset recorded the lowest accuracy value, with a score of 0.4199. This suggests that the dataset may need to be more balanced, making it easier for the model to accurately classify instances without using techniques such as SMOTE to address this imbalance. In addition, CBSO and Catboost outperformed other variants in terms of AUC, with a score of 0.9993 in the PC2 dataset, and Assembled SMOTE again demonstrated the best performance in terms of G-mean, with a score of 0.9938 in the MC1 dataset. However, the CM1, MW1, and PC2 datasets recorded the lowest AUC and G-mean values, NaN, without SMOTE. This highlights the importance of handling class imbalance for reliable model evaluation, especially when dealing with datasets dominated by a majority class.

Table 5. AUC Results of SMOTE Variants in Catboost Classification

	CM1	JM1	KC1	KC3	MC1	MC2	MW1	PC1	PC2	PC3	PC4	PC5
Without SMOTE	NaN	0.7102	0.6879	0.7797	0.8888	0.7953	NaN	0.8634	NaN	0.8211	0.9296	0.8037
SMOTE IPF	0.9889	0.9239	0.8895	0.9686	0.9993	0.8630	0.9904	0.9925	0.9992	0.9829	0.9912	0.9226
Kernel	0.9881	0.9251	0.8939	0.9694	0.9995	0.8669	0.9896	0.9947	0.9991	0.9841	0.9911	0.9244
ADASYN												
MOT2LD	0.9703	0.9241	0.8833	0.9565	0.9975	0.8546	0.9717	0.9896	0.9982	0.9808	0.9915	0.9145
OUPS	0.9253	0.9172	0.8605	0.9048	0.9984	0.8025	0.9456	0.9778	0.9937	0.9703	0.9845	0.9068
SMOTE D	0.9757	0.9210	0.8906	0.9600	0.9973	0.8593	0.9795	0.9902	0.9995	0.9773	0.9895	0.9249
CURE	0.9669	0.9198	0.8899	0.9467	0.9981	0.8569	0.9845	0.9898	0.9989	0.9782	0.9895	0.9253
SMOTE												
Edge Det	0.9852	0.9244	0.8915	0.9507	0.9990	0.8638	0.9864	0.9947	0.9993	0.9834	0.9908	0.9193
SMOTE												
CBSO	0.9899	0.9198	0.8792	0.9663	0.9993	0.8514	0.9813	0.9909	**0.9998**	0.9811	0.9900	0.9213
Assembled	0.9828	0.9243	0.8924	0.9505	0.9994	0.8695	0.9886	0.9934	0.9991	0.9839	0.9910	0.9243
SMOTE												
SDSMOTE	0.9859	0.9248	0.8888	0.9663	0.9994	0.8427	0.9928	0.9938	0.9993	0.9847	0.9917	0.9185
DSMOTE	0.9775	0.9383	0.9195	0.9375	0.9970	0.9218	0.9684	0.9884	0.9983	0.9805	0.9919	0.9454
G SMOTE	0.9889	0.9240	0.8947	0.9637	0.9993	0.8469	0.9765	0.9955	0.9995	0.9856	0.9918	0.9282
NT SMOTE	0.9802	0.9238	0.8937	0.9587	0.9986	0.8495	0.9771	0.9917	0.9990	0.9796	0.9901	0.9217
Lee	0.9710	0.9232	0.8878	0.9593	0.9979	0.8743	0.9758	0.9893	0.9982	0.9780	0.9904	0.9238

Table 6. G-mean Results of SMOTE Variants in Catboost Classification

	CM1	JM1	KC1	KC3	MC1	MC2	MW1	PC1	PC2	PC3	PC4	PC5
Without SMOTE	NaN	0.3466	0.4311	0.7692	0.4388	0.5949	NaN	0.3599	NaN	0.2575	0.6533	0.5785
SMOTE IPF	0.9589	0.8325	0.8151	0.8925	0.9932	0.8288	0.9597	0.9690	0.9385	0.9385	0.9625	0.8438
Kernel ADASYN	0.9672	0.8311	0.8186	0.9098	0.9936	0.8399	0.9581	0.9764	0.9376	0.9376	0.9648	0.8495
MOT2LD	0.9208	0.8391	0.8038	0.9182	0.9894	0.7601	0.9378	0.9527	0.9418	0.9418	0.9543	0.8444
OUPS	0.8446	0.8205	0.7665	0.8408	0.9844	0.7480	0.8667	0.9218	0.8979	0.8979	0.9375	0.8231
SMOTE D	0.9231	0.8225	0.7892	0.8915	0.9881	0.7389	0.9144	0.9535	0.9187	0.9187	0.9328	0.8197
CURE SMOTE	0.9021	0.8209	0.7958	0.8828	0.9865	0.7506	0.9401	0.9464	0.9201	0.9201	0.9318	0.8203
Edge Det SMOTE	0.9661	0.8331	0.8029	0.9046	0.9925	0.8011	0.9488	0.9669	0.9392	0.9392	0.9606	0.8364
CBSO	0.9656	0.8257	0.7965	0.9089	0.9929	0.8092	0.9447	0.9642	0.9366	0.9366	0.9603	0.8430
Assembled SMOTE	0.9526	0.8296	0.8158	0.9007	**0.9938**	0.8041	0.9550	0.9732	0.9446	0.9446	0.9663	0.8437
SDSMOTE	0.9525	0.8296	0.8075	0.9189	0.9936	0.7379	0.9640	0.9722	0.9419	0.9419	0.9648	0.8382
DSMOTE	0.9013	0.8360	0.8174	0.8544	0.9860	0.7608	0.9322	0.9438	0.9176	0.9176	0.9378	0.8398
G SMOTE	0.9535	0.8344	0.8145	0.8901	0.9930	0.7873	0.9413	0.9693	0.9520	0.9520	0.9597	0.8494
NT SMOTE	0.9294	0.8298	0.8042	0.8803	0.9879	0.7857	0.9522	0.9545	0.9216	0.9216	0.9396	0.8370
Lee	0.8970	0.8394	0.8182	0.9130	0.9912	0.7815	0.9420	0.9437	0.9351	0.9351	0.9427	0.8497

3.2. Classification with LightGBM

A software defect prediction model was evaluated using the LightGBM algorithm, and the findings are presented in three tables: Table 7 shows the accuracy results, Table 8 presents the AUC (Area Under the Curve) results, and Table 9 displays the G-mean results. The test results in Table 7, Table 8, and Table 9 showed that the combination of Assembled SMOTE and the LightGBM algorithm achieved the highest accuracy of 0.9915 on the MC1 dataset. This indicates that using these two techniques together is particularly effective in improving the model's ability to correctly categorize instances in this dataset, which may have a significant class imbalance. However, CBSO performed better than the others in AUC, scoring 0.9993 on the PC2 dataset. CBSO is particularly effective in distinguishing between positive and negative classes in this dataset, leading to high accuracy in this metric. The ADASYN kernel achieved the highest score of 0.9925 on the MC1 dataset for G-mean, indicating that ADASYN is particularly effective in balancing the class distribution and enhancing the model's ability to classify instances in this dataset correctly. The KC3 dataset obtained an accuracy score of 0.3960 without using SMOTE. It scored the lowest value of NaN for AUC and G-mean on the CM1, MW1, and PC2 datasets without SMOTE. This indicates that the model encountered issues during the evaluation of these datasets, potentially due to class imbalance or extreme values, leading to NaN values in the metrics.

Table 7. Accuracy Results of SMOTE Variants in LightGBM Classification

	CM1	JM1	KC1	KC3	MC1	MC2	MW1	PC1	PC2	PC3	PC4	PC5
Without SMOTE	0.8551	0.7885	0.7500	0.3960	0.9799	0.6931	0.8928	0.9129	0.9674	0.8567	0.8822	0.7536
SMOTE IPF	0.9322	0.8629	0.8208	0.9136	0.9908	0.6955	0.9462	0.9667	0.9853	0.9309	0.9401	0.8414
Kernel	0.9372	0.8615	0.8281	0.9136	0.9915	0.7576	0.9430	0.9656	0.9853	0.9310	0.9459	0.8425
ADASYN												
MOT2LD	0.9291	0.8631	0.8088	0.9097	0.9889	0.7445	0.9406	0.9540	0.9891	0.9422	0.9426	0.8376
OUPS	0.8744	0.8616	0.8018	0.8773	0.9849	0.6515	0.8864	0.9433	0.9745	0.9112	0.9201	0.8235
SMOTE D	0.9200	0.8687	0.8279	0.8842	0.9879	0.7856	0.9303	0.9593	0.9853	0.9255	0.9319	0.8410
CURE SMOTE	0.9221	0.8656	0.8306	0.8909	0.9890	0.7402	0.9493	0.9467	0.9873	0.9204	0.9311	0.8379
Edge Det SMOTE	0.9272	0.8631	0.8241	0.9000	0.9901	0.7576	0.9399	0.9656	0.9853	0.9302	0.9394	0.8385
CBSO	0.9272	0.8644	0.8085	0.9000	0.9901	0.7492	0.9275	0.9511	0.9873	0.9279	0.9414	0.8345
Assembled SMOTE	0.9347	0.8636	0.8249	0.8773	**0.9915**	0.8197	0.9464	0.9644	0.9824	0.9325	0.9465	0.8368
SDSMOTE	0.9272	0.8649	0.8348	0.9000	0.9904	0.7492	0.9556	0.9644	0.9824	0.9363	0.9465	0.8362
DSMOTE	0.9146	0.8783	0.8602	0.8591	0.9901	0.8114	0.9309	0.9533	0.9853	0.9294	0.9517	0.8668
G SMOTE	0.9323	0.8636	0.8142	0.8909	0.9904	0.7500	0.9463	0.9667	0.9833	0.9370	0.9414	0.8414
NT SMOTE	0.9197	0.8633	0.8150	0.8864	0.9890	0.7583	0.9525	0.9567	0.9824	0.9241	0.9291	0.8327
Lee	0.8970	0.8650	0.8191	0.9000	0.9897	0.8038	0.9240	0.9544	0.9853	0.9256	0.9362	0.8385

Table 8. AUC Results of SMOTE Variants in LightGBM Classification

	CM1	JM1	KC1	KC3	MC1	MC2	MW1	PC1	PC2	PC3	PC4	PC5
Without SMOTE	NaN	0.6941	0.6771	0.8136	0.8915	0.7566	NaN	0.8624	NaN	0.8023	0.9165	0.7773
SMOTE IPF	0.9829	0.9223	0.8944	0.9661	0.9993	0.7957	0.9892	0.9919	0.9983	0.9824	0.9910	0.9234
Kernel	0.9768	0.9218	0.8944	0.9696	0.9993	0.8019	0.9888	0.9948	0.9981	0.9843	0.9912	0.9227
ADASYN												
MOT2LD	0.9783	0.9218	0.8826	0.9665	0.9971	0.8219	0.9713	0.9869	0.9985	0.9819	0.9899	0.9165
OUPS	0.9416	0.9192	0.8728	0.9271	0.9990	0.7649	0.9616	0.9840	0.9973	0.9748	0.9832	0.9110
SMOTE D	0.9788	0.9181	0.8885	0.9582	0.9967	0.8594	0.9799	0.9897	0.9993	0.9764	0.9873	0.9193
CURE SMOTE	0.9701	0.9188	0.8851	0.9387	0.9973	0.8321	0.9759	0.9892	0.9979	0.9753	0.9877	0.9193
Edge Det SMOTE	0.9780	0.9218	0.9031	0.9548	0.9988	0.8194	0.9931	0.9958	0.9986	0.9823	0.9901	0.9216
CBSO	0.9841	0.9194	0.8779	0.9743	0.9992	0.8238	0.9871	0.9920	**0.9993**	0.9832	0.9896	0.9183
Assembled SMOTE	0.9745	0.9207	0.8994	0.9502	0.9991	0.8617	0.9878	0.9934	0.9982	0.9835	0.9905	0.9265
SDSMOTE	0.9833	0.9233	0.9018	0.9727	0.9992	0.7942	0.9912	0.9938	0.9985	0.9858	0.9907	0.9211
DSMOTE	0.9778	0.9353	0.9211	0.9344	0.9968	0.9045	0.9559	0.9869	0.9981	0.9799	0.9901	0.9414
G SMOTE	0.9841	0.9221	0.8946	0.9600	0.9985	0.8288	0.9856	0.9944	0.9990	0.9837	0.9908	0.9277
NT SMOTE	0.9760	0.9203	0.8896	0.9596	0.9982	0.8306	0.9789	0.9902	0.9983	0.9801	0.9887	0.9201
Lee	0.9640	0.9213	0.8901	0.9526	0.9973	0.8739	0.9741	0.9881	0.9984	0.9787	0.9878	0.9212

Table 9. G-mean Results of SMOTE Variants in LightGBM Classification

	CM1	JM1	KC1	KC3	MC1	MC2	MW1	PC1	PC2	PC3	PC4	PC5
Without SMOTE	NaN	0.3691	0.5224	0.7626	0.5055	0.6272	NaN	0.4281	NaN	0.3972	0.6610	0.5851
SMOTE IPF	0.9529	0.8324	0.8158	0.9187	0.9913	0.6847	0.9540	0.9724	0.9886	0.9369	0.9490	0.8405
Kernel	0.9555	0.8288	0.8209	0.9112	**0.9925**	0.7739	0.9459	0.9719	0.9885	0.9334	0.9555	0.8412
ADASYN												
MOT2LD	0.9269	0.8344	0.8056	0.9158	0.9872	0.7550	0.9419	0.9487	0.9861	0.9475	0.9496	0.8399
OUPS	0.8746	0.8231	0.7884	0.8702	0.9850	0.6596	0.8829	0.9415	0.9804	0.9084	0.9263	0.8214
SMOTE D	0.9278	0.8242	0.8135	0.8835	0.9869	0.7595	0.9238	0.9522	0.9865	0.9201	0.9286	0.8245
CURE SMOTE	0.9180	0.8230	0.8059	0.8713	0.9852	0.7486	0.9464	0.9407	0.9856	0.9083	0.9233	0.8179
Edge Det SMOTE	0.9476	0.8317	0.8203	0.8950	0.9906	0.7559	0.9390	0.9694	0.9896	0.9350	0.9486	0.8413
CBSO	0.9441	0.8279	0.7964	0.8956	0.9910	0.7446	0.9356	0.9563	0.9905	0.9320	0.9502	0.8360
Assembled SMOTE	0.9518	0.8308	0.8241	0.8673	0.9921	0.8366	0.9509	0.9690	0.9871	0.9405	0.9550	0.8394
SDSMOTE	0.9453	0.8324	0.8323	0.8936	0.9915	0.7689	0.9590	0.9703	0.9861	0.9412	0.9577	0.8394
DSMOTE	0.8993	0.8366	0.8353	0.8499	0.9867	0.7889	0.9193	0.9434	0.9829	0.9160	0.9455	0.8470
G SMOTE	0.9441	0.8320	0.8117	0.8894	0.9908	0.7469	0.9509	0.9668	0.9836	0.9377	0.9456	0.8426
NT SMOTE	0.9273	0.8280	0.8081	0.8705	0.9870	0.7511	0.9506	0.9495	0.9810	0.9196	0.9306	0.8298
Lee	0.8853	0.8357	0.8120	0.8917	0.9890	0.7898	0.9242	0.9445	0.9855	0.9267	0.9397	0.8430

3.3. Classification with Gradient Boosting

The software defect prediction model with the Gradient Boosting algorithm was measured, and the results are presented in Table 10, Table 11, and Table 12. Table 10 displays the accuracy results, Table 11 presents

the AUC (Area Under the Curve), and Table 12 shows the G-mean results. Based on the results of the SMOTE Variants model test using the Gradient Boosting algorithm in Table 10, Table 11, and Table 12. DSMOTE achieved the highest accuracy with a score of 0.9893 on the MC1 dataset. This excellent performance demonstrates the effectiveness of DSMOTE in addressing class imbalance issues within the MC1 dataset. For the PC2 dataset, CBSO achieved the highest AUC value of 0.9990, indicating its ability to distinguish between positive and negative classes effectively. Additionally, the ADAYSN Kernel had the best G-mean score of 0.9902 on the MC1 dataset, highlighting its effectiveness in balancing precision and recall. It is important to note that pairs without SMOTE in datasets CM1, MW1, and PC2 exhibit the lowest AUC and G-mean values, denoted by NaN, suggesting that these datasets may require alternative strategies to address class imbalance effectively. In Fig. 2, the accuracy values of different models are displayed. The highest accuracy is achieved by the DSMOTE and gradient boosting models, with a value of 0.9161, while the lowest accuracy is attained by the models Without SMOTE and Gradient Boosting, with a value of 0.6696. This significant difference in accuracy highlights the crucial role of SMOTE in addressing class imbalance and improving the overall performance of machine learning models. The difference between the two values is a substantial 36.82%, emphasizing the significant impact that SMOTE, especially when combined with gradient boosting, can have on model accuracy. The results presented in Fig. 3 show that using SMOTE for class imbalance correction significantly improved the AUC. The best-performing combination was using DSMOTE with the Catboost algorithm, which achieved the highest AUC of 0.9637. On the other hand, the model trained without SMOTE and using LightGBM resulted in the lowest AUC score of 0.5993. This represents a substantial improvement of 60.82% and highlights the effectiveness of using SMOTE to address class imbalance and enhance the overall performance of machine learning models for software defect prediction.

Table 10. Accuracy Results of SMOTE Variants in Gradient Boosting Classification

	CM1	JM1	KC1	KC3	MC1	MC2	MW1	PC1	PC2	PC3	PC4	PC5
Without SMOTE	0.8415	0.7900	0.7453	0.4423	0.9792	0.7944	0.8480	0.9088	0.9693	0.8633	0.8933	0.7611
SMOTE IPF	0.9272	0.8315	0.7969	0.9091	0.9882	0.8038	0.9432	0.9467	0.9794	0.9166	0.9401	0.8201
Kernel	0.9271	0.8376	0.7969	0.8909	0.9882	0.8106	0.9465	0.9522	0.9814	0.9180	0.9375	0.8206
ADASYN												
MOT2LD	0.9002	0.8421	0.8097	0.9195	0.9882	0.8573	0.9178	0.9495	0.9871	0.9219	0.9373	0.8293
OUPS	0.8719	0.8460	0.7837	0.8818	0.9746	0.7750	0.8992	0.9278	0.9569	0.9044	0.9227	0.8149
SMOTE D	0.9026	0.8655	0.8149	0.8879	0.9875	0.8205	0.9239	0.9471	0.9823	0.9240	0.9371	0.8341
CURE	0.9020	0.8658	0.8208	0.8727	0.9882	0.7568	0.9496	0.9478	0.9804	0.9234	0.9311	0.8362
SMOTE												
Edge Det	0.9397	0.8381	0.8085	0.8955	0.9842	0.7932	0.9340	0.9511	0.9794	0.9173	0.9336	0.8172
SMOTE												
CBSO	0.9272	0.8487	0.7912	0.9000	0.9827	0.8212	0.9148	0.9478	0.9843	0.9120	0.9388	0.8160
Assembled	0.9195	0.8384	0.8134	0.8864	0.9864	0.8394	0.9337	0.9489	0.9794	0.9234	0.9414	0.8166
SMOTE												
SDSMOTE	0.9347	0.8400	0.7994	0.8773	0.9867	0.7667	0.9527	0.9467	0.9804	0.9233	0.9407	0.8270
DSMOTE	0.9322	0.8789	0.8536	0.8773	**0.9893**	0.8659	0.9245	0.9511	0.9833	0.9317	0.9459	0.8593
G SMOTE	0.9272	0.8402	0.8043	0.9136	0.9882	0.8303	0.9147	0.9533	0.9794	0.9249	0.9420	0.8241
NT SMOTE	0.9246	0.8524	0.8125	0.8727	0.9886	0.7939	0.9494	0.9456	0.9775	0.9150	0.9356	0.8270
Lee	0.8994	0.8436	0.8241	0.9136	0.9890	0.8030	0.9335	0.9456	0.9814	0.9234	0.9375	0.8247

Table 11. AUC Results of SMOTE Variants in Gradient Boosting Classification

	CM1	JM1	KC1	KC3	MC1	MC2	MW1	PC1	PC2	PC3	PC4	PC5
Without SMOTE	NaN	0.7036	0.6774	0.7627	0.8892	0.8421	NaN	0.8129	NaN	0.8091	0.9239	0.7723
SMOTE IPF	0.9784	0.9038	0.8705	0.9621	0.9985	0.8695	0.9831	0.9891	0.9982	0.9749	0.9889	0.8967
Kernel	0.9732	0.9063	0.8647	0.9643	0.9985	0.8569	0.9841	0.9911	0.9989	0.9763	0.9875	0.9016
ADASYN												
MOT2LD	0.9725	0.9070	0.8718	0.9637	0.9974	0.8881	0.9669	0.9864	0.9980	0.9768	0.9902	0.9037
OUPS	0.9291	0.9091	0.8480	0.9387	0.9977	0.7662	0.9627	0.9800	0.9957	0.9702	0.9834	0.8966
SMOTE D	0.9711	0.9190	0.8801	0.9468	0.9974	0.8732	0.9736	0.9867	0.9988	0.9766	0.9872	0.9136
CURE	0.9665	0.9175	0.8798	0.9436	0.9978	0.8299	0.9801	0.9860	0.9980	0.9758	0.9871	0.9131
SMOTE												
Edge Det	0.9742	0.9050	0.8732	0.9402	0.9977	0.8304	0.9849	0.9894	0.9983	0.9770	0.9868	0.8951
SMOTE												
CBSO	0.9810	0.9107	0.8640	0.9565	0.9983	0.8819	0.9788	0.9858	**0.9990**	0.9747	0.9869	0.9030
Assembled	0.9789	0.9063	0.8708	0.9435	0.9987	0.8762	0.9822	0.9873	0.9985	0.9757	0.9872	0.9018
SMOTE												
SDSMOTE	0.9800	0.9049	0.8690	0.9586	0.9988	0.8435	0.9861	0.9894	0.9990	0.9776	0.9884	0.9010
DSMOTE	0.9793	0.9370	0.9121	0.9353	0.9970	0.9219	0.9651	0.9837	0.9980	0.9816	0.9908	0.9390
G SMOTE	0.9795	0.9076	0.8773	0.9476	0.9984	0.8658	0.9749	0.9919	0.9987	0.9787	0.9896	0.9038
NT SMOTE	0.9759	0.9132	0.8745	0.9528	0.9982	0.8486	0.9733	0.9883	0.9978	0.9781	0.9885	0.9033
Lee	0.9636	0.9062	0.8691	0.9533	0.9981	0.8519	0.9706	0.9856	0.9980	0.9738	0.9883	0.9046

Table 12. G-mean Results of SMOTE Variants in Gradient Boosting Classification

	CM1	JM1	KC1	KC3	MC1	MC2	MW1	PC1	PC2	PC3	PC4	PC5
Without SMOTE	NaN	0.3191	0.4231	0.7852	0.5632	0.7148	NaN	0.4619	NaN	0.3841	0.6558	0.5462
SMOTE IPF	0.9451	0.8067	0.7810	0.8976	0.9899	0.8006	0.9486	0.9570	0.9875	0.9283	0.9535	0.8227
Kernel ADASYN	0.9446	0.8132	0.7798	0.8984	**0.9902**	0.8140	0.9536	0.9611	0.9875	0.9272	0.9499	0.8285
MOT2LD	0.9060	0.8253	0.7825	0.9206	0.9888	0.8235	0.9206	0.9459	0.9851	0.9290	0.9426	0.8281
OUPS	0.8638	0.8143	0.7598	0.8786	0.9734	0.7632	0.8927	0.9331	0.9657	0.9075	0.9346	0.8143
SMOTE D	0.9076	0.8165	0.7848	0.8892	0.9859	0.7724	0.9111	0.9418	0.9839	0.9164	0.9325	0.8052
CURE SMOTE	0.8941	0.8193	0.7744	0.8604	0.9851	0.7403	0.9465	0.9377	0.9821	0.9111	0.9220	0.8030
Edge Det SMOTE	0.9606	0.8148	0.7842	0.8923	0.9868	0.7732	0.9447	0.9611	0.9866	0.9260	0.9465	0.8129
CBSO	0.9400	0.8192	0.7753	0.9112	0.9833	0.7947	0.9247	0.9527	0.9900	0.9177	0.9507	0.8166
Assembled SMOTE	0.9371	0.8154	0.7955	0.8981	0.9899	0.8099	0.9307	0.9600	0.9846	0.9326	0.9530	0.8175
SDSMOTE	0.9452	0.8147	0.7768	0.8793	0.9890	0.7545	0.9522	0.9593	0.9870	0.9297	0.9519	0.8244
DSMOTE	0.9174	0.8321	0.8149	0.8694	0.9860	0.8439	0.9243	0.9411	0.9835	0.9180	0.9349	0.8420
G SMOTE	0.9340	0.8175	0.7855	0.9052	0.9896	0.8281	0.9192	0.9601	0.9827	0.9345	0.9521	0.8268
NT SMOTE	0.9254	0.8277	0.7914	0.8620	0.9867	0.7705	0.9409	0.9467	0.9766	0.9146	0.9374	0.8241
Lee	0.8940	0.8239	0.8114	0.9025	0.9902	0.7640	0.9255	0.9376	0.9826	0.9266	0.9416	0.8237

In Fig. 4, the average G-means value differs significantly between the Without SMOTE and SMOTE models, highlighting the effectiveness of SMOTE in addressing class imbalance. The highest G-mean value of 0.9154 is achieved by the ADAYSN Kernel and Catboost pair, indicating its suitability for handling imbalanced datasets. Conversely, the lowest G-mean value is 0.3691, observed in the Without SMOTE and Catboost pair, emphasizing the importance of incorporating SMOTE. The difference between the two values is a staggering 147.96%, emphasizing the significant impact of SMOTE on model performance.

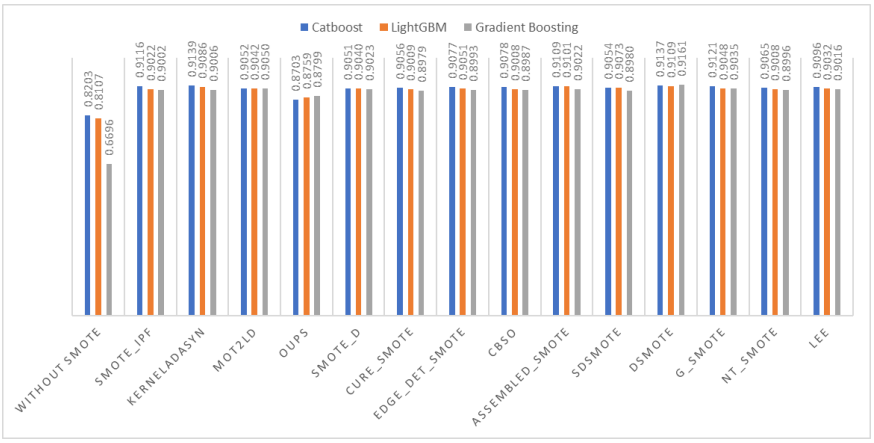

Fig. 2. Comparison of Average Accuracy

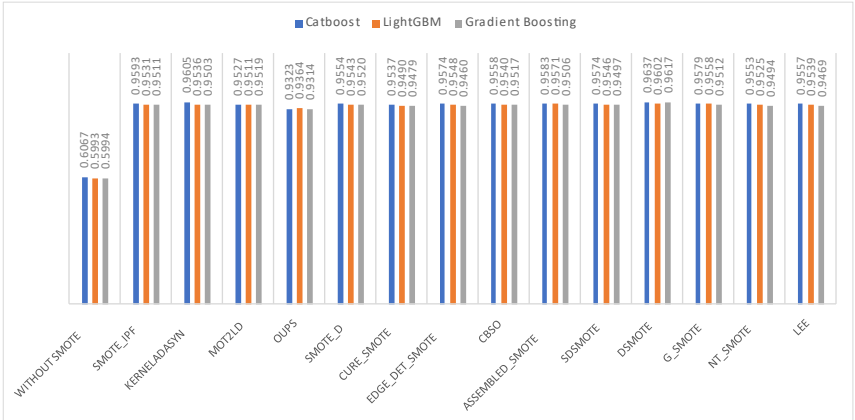

Fig. 3. Comparison of Average AUC

Based on the research results, it is crucial to use SMOTE to address unbalanced datasets. This is supported by the fact that several datasets, including CM1, MW1, and PC2, resulted in a value of NaN when SMOTE was not used. NaN (Not a Number) is a specific undefined value for numeric data types in computing. The occurrence of NaN in the dataset is likely due to class imbalance. Additionally, the choice of k-fold cross-validation parameters, particularly the number of folds (k), can potentially influence the occurrence of NaN values and, consequently, impact algorithm performance evaluation.During our research, we discovered a NaN value in the data set, which prompted us to investigate the issue further. As a result, we carried out a study to examine the impact of different K-fold values on the CM1, MW1, and PC2 datasets. The results of our experiments show that using various K-fold values improve performance metrics, as shown in Table 13. However, after careful consideration, we decided to use the 10-fold cross-validation. This aligns with previous research [23]-[25] highlighting the advantages of 10-fold cross-validation in minimizing bias and ensuring a more trustworthy evaluation of model generalizations. Comparison of model without SMOTE performance with different K-fold values shown in Table 13.

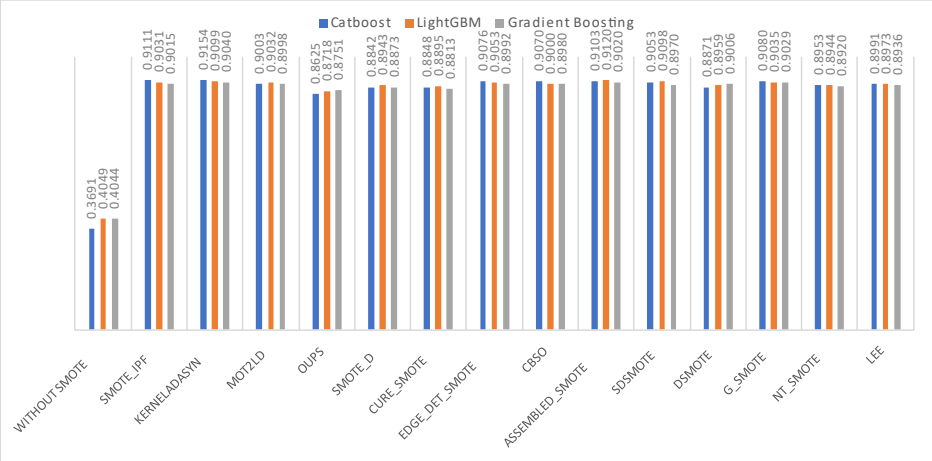

Fig. 4. Comparison of Average G-mean

Table 13. Comparison of Model Without SMOTE Performance with Different K-fold Values

Algorithm	K-fold Value	CM1		MW1		PC2	
		AUC	G-mean	AUC	G-mean	AUC	G-mean
Catboost	K=3	0.8248	0.1454	0.6808	0.4368	0.9034	0.000
	K=5	0.7926	0.1408	0.6902	0.2838	0.9022	0.000
LightGBM	K=3	0.8158	0.2045	0.7023	0.4253	0.8726	0.000
	K=5	0.8429	0.3634	0.7068	0.3913	0.8942	0.000
Gradient Boosting	K=3	0.8281	0.4491	0,6225	0.4397	0.8877	0.000
	K=5	0.7890	0.3440	0.6779	0.3832	0.9047	0.000

This study aims to compare the performance of the proposed model with models from previous studies, as presented in Table 14. However, direct comparison of the results with [13], [18], [19] studies is limited due to differences in the evaluation matrices. Therefore, we have chosen to compare our research to studies [14]-[17] based on their AUC values and methodological similarity to our study. We selected the studies with the highest AUC values to represent the best performance of the SMOTE Variant in those studies.

The analysis conducted in Table 14 indicates that the proposed method, which combines SMOTE Variants and Boosting Algorithms, can potentially enhance binary classification performance. This is supported by the AUC value, which is superior to other research methods. However, the performance is not consistently better than other methods, such as OUPS method, and this method's potential appears promising. It is worth noting that factors such as hyperparameter settings, data characteristics, and the choice of boosting algorithm can affect performance.

The proposed software defect prediction model offers many benefits to improve software quality and the efficiency of the development process. The model can help developers predict the likelihood of defects in software modules, enable early detection, and prioritize which modules should be tested first. The results of this study can serve as a foundation for further research and are expected to encourage research in this field to grow and produce better solutions.

Table 14. Comparison of AUC Results with Previous Studies

Research	Algorithm	AUC
[14]	CURE SMOTE + SVM	0.9068
	GSMOTE + SVM	0.9062
	LEE + DT	0.8834
	Assembled SMOTE+ DT	0.8834
	SMOTE IPF + DT	0.8828
[15]	OUPS	**0.97**
[16]	GSMOTE	0.8857
	SDSMOTE	0.8823
[17]	Kernel ADASYN	0.7902
Our Research	CURE SMOTE + Catboost	**0.9537**
	GSMOTE + Catboost	**0.9579**
	LEE + Catboost	**0.9557**
	Assembled SMOTE+ Catboost	**0.9583**
	SMOTE IPF + Catboost	**0.9593**
	OUPS + LightGBM	0.9364
	GSMOTE + Catboost	**0.9579**
	SDSMOTE + Catboost	**0.9574**

4. CONCLUSION

The research suggests that the Boosting model, when used with various variations of SMOTE, can predict software defects in the NASA MDP D" dataset. The study evaluates performance using accuracy, AUC, and G-mean metrics. It concludes that using SMOTE Variants with the Boosting algorithm can enhance classification performance for software defect prediction. The combination of DSMOTE with Gradient Boosting achieved the highest average accuracy of 0.9161. Additionally, DSMOTE and Catboost had the highest average AUC value of 0.9637, while Kernel ADASYN and Catboost showed the best ability to achieve an average G-mean value of 0.9154.

The study suggests that SMOTE Variants and Algorithm Boosting can effectively enhance software defect prediction. However, further investigation is required to address certain limitations. The results reveal a concerning presence of NaN values for AUC and G-mean across multiple boosting algorithms on datasets CM1, MW1, and PC2. This issue may be due to the 10-fold cross-validation methodology, where data stratification was not adequately ensured during fold creation. As a result, some folds may have imbalanced class distributions, leading to models encountering unseen data during training and generating undefined metrics (NaN).

Future research should focus on developing new classification methods, testing newer variations of SMOTE, and evaluating model performance on various datasets. This could involve exploring different techniques for handling class imbalance, such as oversampling minority classes or undersampling majority classes and assessing the effectiveness of these methods on a range of datasets. Additionally, further research can consider other factors, such as the value of k in cross-validation or the percentage distribution of datasets, to better understand how these variables impact model performance.

REFERENCES

[1] J. Liu, J. Ai, M. Lu, J. Wang, and H. Shi, "Semantic feature learning for software defect prediction from source code and external knowledge," *Journal of Systems and Software*, vol. 204, Oct. 2023, https://doi.org//10.1016/j.jss.2023.111753.

[2] F. Huang and L. Strigini, "HEDF: A Method for Early Forecasting Software Defects Based on Human Error Mechanisms," *IEEE Access*, vol. 11, pp. 3626–3652, 2023, https://doi.org//10.1109/ACCESS.2023.3234490.

[3] C. Manjula and L. Florence, "Deep neural network based hybrid approach for software defect prediction using software metrics," *Cluster Comput*, vol. 22, pp. 9847–9863, Jul. 2019, https://doi.org//10.1007/s10586-018-1696-z.

[4] H. Aljamaan and A. Alazba, "Software defect prediction using tree-based ensembles," in *PROMISE Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering, Co-located with ESEC/FSE,* pp. 1–10, 2020, https://doi.org//10.1145/3416508.3417114.

[5] D. Meng and Y. Li, "An imbalanced learning method by combining SMOTE with Center Offset Factor," *Appl Soft Comput*, vol. 120, May 2022, https://doi.org//10.1016/j.asoc.2022.108618.

[6] S. Goyal, "Handling Class-Imbalance with KNN (Neighbourhood) Under-Sampling for Software Defect Prediction," *Artif Intell Rev*, vol. 55, no. 3, pp. 2023–2064, Mar. 2022, https://doi.org//10.1007/s10462-021-10044-w.

[7] N. U. Niaz, K. M. N. Shahariar, and M. J. A. Patwary, "Class Imbalance Problems in Machine Learning: A Review of Methods And Future Challenges," in *ACM International Conference Proceeding Series*, Association for Computing Machinery, pp. 485–490, Mar. 2022, https://doi.org//10.1145/3542954.3543024.

[8] M. M. Ahsan, M. S. Ali, and Z. Siddique, "Enhancing and improving the performance of imbalanced class data using novel GBO and SSG: A comparative analysis," *Neural Networks*, vol. 173, p. 106157, May 2024, https://doi.org//10.1016/j.neunet.2024.106157.

[9] M. Vardhan, K. Banerjee, and D. Aggarwal, "A Systematic Approach for the Detection of Software Bug Using Catboost," in *International Conference on Machine Learning, Big Data, Cloud and Parallel Computing, COM-IT-CON*, pp. 414–419, 2022, https://doi.org//10.1109/COM-IT-CON54601.2022.9850519.

[10] M. F. Sohan, M. I. Jabiullah, S. S. M. M. Rahman, and S. M. H. Mahmud, "Assessing the Effect of Imbalanced Learning on Cross-project Software Defect Prediction," in *10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pp. 1–6, 2019, https://doi.org//10.1109/ICCCNT45670.2019.8944622.

[11] M. N. Uddin, B. Li, M. N. Mondol, M. M. Rahman, M. S. Mia, and E. L. Mondol, "SDP-ML: An Automated Approach of Software Defect Prediction employing Machine Learning Techniques," in *Proceedings of International Conference on Electronics, Communications and Information Technology, ICECIT*, pp. 1-4, 2021, https://doi.org//10.1109/ICECIT54077.2021.9641218.

[12] R. Malhotra and J. Jain, "Handling Imbalanced Data using Ensemble Learning in Software Defect Prediction," in *10th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pp. 300–304, 2020, https://doi.org/10.1109/Confluence47617.2020.9058124.

[13] A. S. Tarawneh, A. B. Hassanat, G. A. Altarawneh, and A. Almuhaimeed, "Stop Oversampling for Class Imbalance Learning: A Review," *IEEE Access*, vol. 10, pp. 47643–47660, 2022, https://doi.org//10.1109/ACCESS.2022.3169512.

[14] G. Kovács, "An empirical comparison and evaluation of minority oversampling techniques on a large number of imbalanced datasets," *Applied Soft Computing Journal*, vol. 83, Oct. 2019, https://doi.org//10.1016/j.asoc.2019.105662.

[15] J. Zhai, J. Qi, and S. Zhang, "Imbalanced data classification based on diverse sample generation and classifier fusion," *International Journal of Machine Learning and Cybernetics*, vol. 13, no. 3, pp. 735–750, Mar. 2022, https://doi.org//10.1007/s13042-021-01321-9.

[16] K. Teh, P. Armitage, S. Tesfaye, D. Selvarajah, and I. D. Wilkinson, "Imbalanced learning: Improving classification of diabetic neuropathy from magnetic resonance imaging," *PLoS One*, vol. 15, no. 12, Dec. 2020, https://doi.org//10.1371/journal.pone.0243907.

[17] H. Ding *et al.*, "KA-Ensemble: towards imbalanced image classification ensembling under-sampling and over-sampling," *Multimed Tools Appl*, vol. 79, no. 21–22, pp. 14871–14888, Jun. 2020, https://doi.org//10.1007/s11042-019-07856-y.

[18] M. Dudjak and G. Martinović, "In-Depth Performance Analysis of SMOTE-Based Oversampling Algorithms in Binary Classification," *International Journal of Electrical and Computer Engineering Systems*, vol. 11, no. 1, pp. 13–23, 2020, https://doi.org//http://dx.doi.org/10.32985/ijeces.11.1.2.

[19] C. Liu *et al.*, "Constrained Oversampling: An Oversampling Approach to Reduce Noise Generation in Imbalanced Datasets with Class Overlapping," *IEEE Access*, vol. 10, pp. 91452-91465, 2021, https://doi.org//10.1109/ACCESS.2020.3018911.

[20] A. Adorada, P. W. Wirawan, and K. Kurniawan, "The Comparison of Feature Selection Methods in Software Defect Prediction," in *ICICoS Proceeding: 4th International Conference on Informatics and Computational Sciences*, pp. 1-6, Nov. 2020, https://doi.org//10.1109/ICICoS51170.2020.9299022.

[21] N. Biswas, K. M. M. Uddin, S. T. Rikta, and S. K. Dey, "A comparative analysis of machine learning classifiers for stroke prediction: A predictive analytics approach," *Healthcare Analytics*, vol. 2, Nov. 2022, https://doi.org//10.1016/j.health.2022.100116.

[22] M. K. Dahouda and I. Joe, "A Deep-Learned Embedding Technique for Categorical Features Encoding," *IEEE Access*, vol. 9, pp. 114381–114391, 2021, https://doi.org//10.1109/ACCESS.2021.3104357.

[23] I. Kaur and A. Kaur, "Comparative analysis of software fault prediction using various categories of classifiers," *International Journal of System Assurance Engineering and Management*, vol. 12, no. 3, pp. 520–535, Jun. 2021, https://doi.org//10.1007/s13198-021-01110-1.

[24] M. Banga, A. Bansal, and A. Singh, "Proposed approach to predict software faults detection using Entropy," *International Journal of System Assurance Engineering and Management*, vol. 11, pp. 301–312, Jul. 2020, https://doi.org//10.1007/s13198-019-00934-2.

[25] A. Adorada, P. W. Wirawan, and K. Kurniawan, "The Comparison of Feature Selection Methods in Software Defect Prediction," in *ICICoS Proceeding: 4th International Conference on Informatics and Computational Sciences*, pp. 1-6, Nov. 2020. https://doi.org//10.1109/ICICoS51170.2020.9299022.

[26] S. M. Malakouti, M. B. Menhaj, and A. A. Suratgar, "The usage of 10-fold cross-validation and grid search to enhance ML methods performance in solar farm power generation prediction," *Clean Eng Technol*, vol. 15, Aug. 2023, https://doi.org//10.1016/j.clet.2023.100664.
[27] A. M. Akbar, R. Herteno, S. W. Saputro, M. R. Faisal, and R. A. Nugroho, "Optimizing Software Defect Prediction Models: Integrating Hybrid Grey Wolf and Particle Swarm Optimization for Enhanced Feature Selection with Popular Gradient Boosting Algorithm," *Journal of Electronics, Electromedical Engineering, and Medical Informatics*, vol. 6, no. 2, pp. 169–181, Apr. 2024, https://doi.org//10.35882/jeeemi.v6i2.388.
[28] N. A. Azhar, M. S. Mohd Pozi, A. M. Din, and A. Jatowt, "An investigation of SMOTE based methods for imbalanced datasets with data complexity analysis," *IEEE Trans Knowl Data Eng*, vol. 35, no. 7, pp. 6651–6672, Jul. 2023, https://doi.org//10.1109/TKDE.2022.3179381.
[29] T. Murad, S. Ali, and M. Patterson, "Exploring the Potential of GANs in Biological Sequence Analysis," *Biology (Basel)*, vol. 12, no. 6, Jun. 2023, https://doi.org//10.3390/biology12060854.
[30] G. A. Pradipta, R. Wardoyo, A. Musdholifah, and I. N. H. Sanjaya, "Radius-SMOTE: A New Oversampling Technique of Minority Samples Based on Radius Distance for Learning from Imbalanced Data," *IEEE Access*, vol. 9, pp. 74763–74777, 2021, https://doi.org//10.1109/ACCESS.2021.3080316.
[31] D. Liu, S. Zhong, L. Lin, M. Zhao, X. Fu, and X. Liu, "Feature-level SMOTE: Augmenting fault samples in learnable feature space for imbalanced fault diagnosis of gas turbines," *Expert Syst Appl*, vol. 238, Mar. 2024, https://doi.org//10.1016/j.eswa.2023.122023.
[32] S. Wang, Y. Dai, J. Shen, and J. Xuan, "Research on expansion and classification of imbalanced data based on SMOTE algorithm," *Sci Rep*, vol. 11, no. 1, Dec. 2021, https://doi.org//10.1038/s41598-021-03430-5.
[33] X. W. Liang, A. P. Jiang, T. Li, Y. Y. Xue, and G. T. Wang, "LR-SMOTE — An improved unbalanced data set oversampling based on K-means and SVM," *Knowl Based Syst*, vol. 196, May 2020, https://doi.org//10.1016/j.knosys.2020.105845,
[34] F. Dai, Y. Song, W. Si, G. Yang, J. Hu, and X. Wang, "Improved CBSO: A distributed fuzzy-based adaptive synthetic oversampling algorithm for imbalanced judicial data," *Inf Sci (N Y)*, vol. 569, pp. 70–89, Aug. 2021, https://doi.org//10.1016/j.ins.2021.04.017.
[35] S. Szeghalmy and A. Fazekas, "A Comparative Study of the Use of Stratified Cross-Validation and Distribution-Balanced Stratified Cross-Validation in Imbalanced Learning," *Sensors*, vol. 23, no. 4, Feb. 2023, https://doi.org//10.3390/s23042333.
[36] Y. Geng, J. Sui, and Q. Zhu, "Rumor Detection of Sina Weibo Based on SDSMOTE and Feature Selection," in *IEEE 4th International Conference on Cloud Computing and Big Data Analytics*, pp. 120–125, 2019, https://doi.org/10.1109/ICCCBDA.2019.8725715.
[37] G. S. Thejas, Y. Hariprasad, S. S. Iyengar, N. R. Sunitha, P. Badrinath, and S. Chennupati, "An extension of Synthetic Minority Oversampling Technique based on Kalman filter for imbalanced datasets," *Machine Learning with Applications*, vol. 8, p. 100267, Jun. 2022, https://doi.org//10.1016/j.mlwa.2022.100267.
[38] T. Watthaisong, K. Sunat, and N. Muangkote, "Comparative Evaluation of Imbalanced Data Management Techniques for Solving Classification Problems on Imbalanced Datasets," *Statistics, Optimization and Information Computing*, vol. 12, no. 2, pp. 547–570, Mar. 2024, https://doi.org//10.19139/soic-2310-5070-1890.
[39] L. Wang, Y. Chen, H. Jiang, and J. Yao, "Imbalanced credit risk evaluation based on multiple sampling, multiple kernel fuzzy self-organizing map and local accuracy ensemble," *Applied Soft Computing Journal*, vol. 91, Jun. 2020, https://doi.org//10.1016/j.asoc.2020.106262.
[40] C. Lee and J. Kim, "Hybrid Oversampling Technique Based on Star Topology and Rejection Methodology for Classifying Imbalanced Data," in *IEEE International Conference on Data Mining Workshops, ICDMW*, pp. 1217–1226, 2022, https://doi.org//10.1109/ICDMW58026.2022.00033.
[41] J. Tanha, Y. Abdi, N. Samadi, N. Razzaghi, and M. Asadpour, "Boosting methods for multi-class imbalanced data classification: an experimental review," *J Big Data*, vol. 7, no. 1, Dec. 2020, https://doi.org//10.1186/s40537-020-00349-y.
[42] W. Chang, X. Wang, J. Yang, and T. Qin, "An Improved CatBoost-Based Classification Model for Ecological Suitability of Blueberries," *Sensors*, vol. 23, no. 4, Feb. 2023, https://doi.org//10.3390/s23041811.
[43] D. Zhang and Y. Gong, "The Comparison of LightGBM and XGBoost Coupling Factor Analysis and Prediagnosis of Acute Liver Failure," *IEEE Access*, vol. 8, pp. 220990-221003, 2020, https://doi.org//10.1109/ACCESS.2020.3042848.
[44] F. Alzamzami, M. Hoda, and A. El Saddik, "Light Gradient Boosting Machine for General Sentiment Classification on Short Texts: A Comparative Evaluation," *IEEE Access*, vol. 8, pp. 101840–101858, 2020, https://doi.org//10.1109/ACCESS.2020.2997330.
[45] W. Liang, S. Luo, G. Zhao, and H. Wu, "Predicting hard rock pillar stability using GBDT, XGBoost, and LightGBM algorithms," *Mathematics*, vol. 8, no. 5, May 2020, https://doi.org//10.3390/MATH8050765.
[46] J. Yoon, "Forecasting of Real GDP Growth Using Machine Learning Models: Gradient Boosting and Random Forest Approach," *Comput Econ*, vol. 57, no. 1, pp. 247–265, Jan. 2021, https://doi.org//10.1007/s10614-020-10054-w.
[47] D. K. Thai, T. M. Tu, T. Q. Bui, and T. T. Bui, "Gradient tree boosting machine learning on predicting the failure modes of the RC panels under impact loads," *Eng Comput*, vol. 37, no. 1, pp. 597–608, Jan. 2021, https://doi.org//10.1007/s00366-019-00842-w.

[48] E. K. Sahin, "Assessing the predictive capability of ensemble tree methods for landslide susceptibility mapping using XGBoost, gradient boosting machine, and random forest," *SN Appl Sci*, vol. 2, no. 7, Jul. 2020, https://doi.org//10.1007/s42452-020-3060-1.
[49] U. Singh, M. Rizwan, M. Alaraj, and I. Alsaidan, "A machine learning-based gradient boosting regression approach for wind power production forecasting: A step towards smart grid environments," *Energies (Basel)*, vol. 14, no. 16, Aug. 2021, https://doi.org//10.3390/en14165196.
[50] S. Feng *et al.*, "COSTE: Complexity-based OverSampling TEchnique to alleviate the class imbalance problem in software defect prediction," *Inf Softw Technol*, vol. 129, Jan. 2021, https://doi.org//10.1016/j.infsof.2020.106432.
[51] D. Chicco and G. Jurman, "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation," *BMC Genomics*, vol. 21, no. 1, Jan. 2020, https://doi.org//10.1186/s12864-019-6413-7.
[52] W. Xia *et al.*, "High-resolution remote sensing imagery classification of imbalanced data using multistage sampling method and deep neural networks," *Remote Sens (Basel)*, vol. 11, no. 21, Nov. 2019, https://doi.org//10.3390/rs11212523.
[53] J. H. Ri, G. Tian, Y. Liu, W. hua Xu, and J. gang Lou, "Extreme learning machine with hybrid cost function of G-mean and probability for imbalance learning," *International Journal of Machine Learning and Cybernetics*, vol. 11, no. 9, pp. 2007–2020, Sep. 2020, https://doi.org//10.1007/s13042-020-01090-x.

BIOGRAPHY OF AUTHORS

Rahmina Ulfah Aflaha is an undergraduate studying Computer Science at Lambung Mangkurat University. Her research focuses on predicting software defects. She can be contacted at email: miraaflaha@gmail.com.

Rudy Herteno is currently a lecturer in the Faculty of Mathematics and Natural Science, at Lambung Mangkurat University. He received his bachelor's degree in Computer Science from Lambung Mangkurat University and a master's degree in Informatics from STMIK Amikom University. His research interests include software engineering, software defect prediction, and deep learning. Email: rudy.herteno@ulm.ac.id.

Mohammad Reza Faisal received the B.Sc. and M.Eng. degrees in physics and informatics from Bandung Institute of Technology, Bandung, Indonesia, in 2004 and 2013. He also received a B.Eng. degree in informatics from Pasundan University, Bandung, Indonesia, in 2002 and a Ph.D. in computer science from Kanazawa University, Ishikawa, Japan, in 2018. He is currently a lecturer in the Computer Science Department, Faculty of Mathematics and Natural Sciences, Lambung Mangkurat University in Banjarbaru, Indonesia. His research interests include artificial intelligence applications, text mining, and software engineering. He can be contacted at email: reza.faisal@ulm.ac.id.

Friska Abadi received his bachelor's degree in computer science from Lambung Mangkurat University, Banjarbaru, Indonesia, in 2011. He also received a master's degree in informatics from STMIK Amikom, Yogyakarta, in 2016. He is currently a lecturer in the Computer Science Department, Faculty of Mathematics and Natural Sciences, Lambung Mangkurat University, Banjarbaru, Indonesia. His research interests include data mining and software engineering. He can be contacted at email: friska.abadi@ulm.ac.id.

Setyo Wahyu Saputro is a lecturer in Computer Science Department, Faculty of Mathematics and Natural Science, Lambung Mangkurat University in Banjarbaru. He received bachelor's degree also in Computer Science from Lambung Mangkurat Univesity, and received his master's degree in Informatics from STMIK Amikom University. His research interests include software engineering and artifial intelligence applications. He can be contacted at email: setyo.saputro@ulm.ac.id.