IAES International Journal of Artificial Intelligence (IJ-AI)
Vol. 13, No. 3, September 2024, pp. 2987~2998
ISSN: 2252-8938, DOI: 10.11591/ijai.v13.i3.pp2987-2998

A three-step combination strategy for addressing outliers and class imbalance in software defect prediction

Muhammad Rizky Pribadi[1,2], Hindriyanto Dwi Purnomo[2], Hendry[2]
[1]Department of Informatics, Faculty of Computer Sciences and Engineering, Universitas Multi Data Palembang, Palembang, Indonesia
[2]Department of Computer Science Doctoral, Faculty of Information Technology, Satya Wacana Christian University, Salatiga, Indonesia

Article Info

Article history:

Received Jan 15, 2024
Revised Feb 2, 2024
Accepted Feb 10, 2024

Keywords:

Classification
Imbalanced data
Outliers
Software defect prediction
Synthetic minority oversampling technique

ABSTRACT

Software defect prediction often involves datasets with imbalanced distributions where one or more classes are underrepresented, referred to as the minority class, while other classes are overrepresented, known as the majority class. This imbalance can hinder accurate predictions of the minority class, leading to misclassification. While the synthetic minority oversampling technique (SMOTE) is a widely used approach to address imbalanced learning data, it can inadvertently generate synthetic minority samples that resemble the majority class and are considered outliers. This study aims to enhance SMOTE by integrating it with an efficient algorithm designed to identify outliers among synthetic minority samples. The resulting method, called reduced outliers (RO)-SMOTE, is evaluated using an imbalanced dataset, and its performance is compared to that of SMOTE. RO-SMOTE first performs oversampling on the training data using SMOTE to balance the dataset. Next, it applies the mining outlier algorithm to detect and eliminate outliers. Finally, RO-SMOTE applies SMOTE again to rebalance the dataset before introducing it to the underlying classifier. The experimental results demonstrate that RO-SMOTE achieves higher accuracy, precision, recall, F1-score, and area under curve (AUC) values compared to SMOTE.

Corresponding Author:

Muhammad Rizky Pribadi
Department of Informatics, Faculty of Computer Sciences and Engineering
Universitas Multi Data Palembang
Palembang, Indonesia
Email: rizky@mdp.ac.id

1. INTRODUCTION

The widespread use of devices in our daily lives has heightened our dependence on various software systems [1]. Any disruption in the operation of software upon which we heavily rely can lead to severe consequences [2]. To ensure the flawless functioning of such systems, comprehensive testing procedures are essential. According to Sava [3], an expert in information technology (IT) security and services market analysis, global IT spending reached $3.8 trillion in 2020 and was expected to increase by approximately 5.1% to about $4.45 trillion in 2022 [3]. In the IT industry, approximately 35% of total development costs are allocated to quality control and testing [4].

One significant effort to reduce software testing costs is software defect prediction. It has become a practical approach to enhance the quality, efficiency, and cost-effectiveness of software testing by focusing on identifying defective software [5]. Recent research has extensively employed machine learning to develop models for defect prediction. Various methods have been proposed to address this task, relying on historical defect data, software metrics, and a chosen prediction algorithm. Techniques such as classification,

clustering, regression, and machine learning algorithms like artificial neural network, K-nearest neighbor (KNN), naive Bayes, decision tree, and ensemble methods have been utilized in this field [5]. The successful prediction of defective software allows for more effective resource allocation by prioritizing software that is predicted to contain defects [6].

However, datasets for software defect prediction often suffer from class imbalance, where very few defective software instances are available for comparison with non-defective ones [7]. To address this imbalance, software engineers can employ data-level techniques to manipulate training data and achieve a more balanced distribution [8]. These techniques include undersampling, which involves removing some data from the majority class, and oversampling, where new instances are added to the minority class by duplicating or creating new samples [9]. While undersampling poses the risk of losing important data, oversampling, particularly when the number of minority class instances is very small, can lead to a significant reduction in dataset size and potentially limit classifier performance [10]. One solution for handling imbalance issues using the oversampling method is the synthetic minority oversampling technique (SMOTE), which involves adding new instances through the KNN approach [11].

Previous studies have shown that SMOTE can be an effective method for addressing data imbalance in datasets. However, in the context of software defect prediction datasets, the issue goes beyond data imbalance; it also involves outliers [12]. While SMOTE is capable of mitigating class imbalance, it does not address the problem of outliers [13]. Research in software defect prediction is critical, as the quality of the data used directly impacts the model being developed. Software defect prediction datasets often contain noisy data with outliers and missing values, which can distort the results [14]. This issue is exemplified in a study [15] that utilized SMOTE to combat class imbalances in National Aeronautic and Space Administration (NASA) and least square support vector machine (LSSVM) datasets for classification. The study reported a low F1-score, such as the PC1 dataset achieving only a score of 0.2807, attributed to the substantial number of outliers in the training data after oversampling [16]. Supervised learning techniques, extensively employed in software defect prediction research, are highly sensitive to outliers, which can significantly reduce their accuracy [17].

Outliers, or anomalous values, refer to data points that significantly deviate from the norm in a dataset [18]. They are often considered unexpected values that fall outside a reasonable range, thereby affecting data analysis and the results of statistical tests [19]. Outliers can arise for various reasons, including measurement errors, data input errors, or very rare and unusual events that influence measurement outcomes [20]. For instance, if we have a dataset of human heights and one person's height deviates significantly from the rest, we classify that individual as an outlier. The presence of extreme outliers can substantially impact statistical measures like the mean and standard deviation, rendering these measures unrepresentative of the true data distribution [20]. Consequently, outliers are often identified and considered for removal before conducting statistical analysis. However, it's crucial to exercise caution when removing outliers, as doing so can alter the data distribution and affect the outcomes of statistical analysis [19].

This article introduces a novel model for addressing the class imbalance problem, termed reduced outliers (RO)-SMOTE. This innovative approach employs efficient algorithms to identify SMOTE-generated outliers and eliminate them from the training dataset. Following the removal of outliers, the dataset is resampled using the SMOTE method to achieve data balance. The primary contributions of this paper are:

- The introduction of a novel pre-processing technique for addressing data imbalance, RO-SMOTE, which leverages efficient algorithms to remove outliers after SMOTE oversampling.
- An evaluation of the proposed RO-SMOTE using various classifiers and cross-project datasets for software defect prediction.
- A comparative analysis of classifier performance when combined with RO-SMOTE against their performance when combined with SMOTE.

The remaining sections of this paper are structured as follows: section 2 provides a review of related work. In section 3, we introduce the proposed model, beginning with an overview of SMOTE and the efficient algorithm, followed by a detailed explanation of RO-SMOTE. Section 4 presents the experimental setup and discusses the obtained results. Finally, section 5 offers conclusions and recommendations for future research.

2. RELATED WORK

A successful strategy for handling imbalanced dataset classification involves adopting a different perspective by treating the data as anomalies or noise. This is necessary because the initial dataset may contain irregularities, and some resampling methods, like SMOTE, may inadvertently introduce more noise, worsening the imbalance problem. High levels of noise within the training dataset can adversely impact classifier performance, a perspective that several studies have addressed [16].

Asniar *et al.* [21] proposed SMOTE-local outlier factor (LOF), which combined SMOTE for oversampling with LOF for noise detection in three real-world datasets characterized by low imbalance ratios. However, its performance across various classifiers, datasets with different imbalance ratios, and dimensions remains uncertain. Additionally, LOF is computationally intensive compared to other techniques. Some researchers [22], [23] employed the interquartile range (IQR) for outlier detection in their oversampling method. Initially, they detected outliers within the imbalanced dataset using the IQR algorithm and then oversampled these identified outliers with replacement to integrate them into the original dataset. Finally, they applied SMOTE to balance the dataset. The IQR, however, does not account for variations in data distribution as it relies solely on the two data points, namely Q1 and Q3, to gauge data spread. Consequently, the IQR may not accurately represent outliers, especially in datasets with diverse distributions [24].

Another technique, the outlier detection-based oversampling technique (ODBOT) [25], was introduced for imbalance datasets in multi-class scenarios. It clusters data using weight-based bat algorithm with k-means (WBBA-KM) to identify dissimilarities between minority and majority classes and then identifies outliers in the minority class by comparing the relationship of differences between the cluster centers in the minority and majority classes. Samples are generated according to the best minority cluster boundary. However, the use of the WBBA-KM algorithm comes with high complexity, primarily due to the time required to construct the hierarchical tree from large datasets [26]. Another study introduced the selective oversampling approach (SOA) [27], which identifies outliers in minor classes using a one-class support vector machine (OCSVM) before removing them and oversampling the data with SMOTE and adaptive synthetic sampling (ADASYN). However, OCSVM's accuracy may be compromised when dealing with limited training data, and it can be time-consuming when applied to large datasets [28].

Furthermore, SMOTE-iterative-partitioning filter (IPF) [28] combines oversampling and noise-filtering methods. The process commences by augmenting the training data through SMOTE, followed by the removal of noise and boundary samples using the IPF. Although this method was tested on real-world datasets featuring up to 19 attributes and a minimum imbalance ratio of 35:301, its effectiveness in scenarios with lower imbalance ratios or greater dimensions has not been investigated. It is important to highlight that the evaluation focused solely on the C4.5 classifier, utilizing the area under curve (AUC) metric, and did not extend the analysis to include other classifiers or metrics.

Puri and Gupta [29] proposed a hybrid model named K-means-SMOTE–edited nearest neighbors (ENN), which integrates bagging, K-means clustering, ENN, and adaptive boosting (AdBoost). In this approach, the model begins by clustering the training data within individual subsamples through K-means clustering [29]. Subsequently, it employs SMOTE to oversample data within each cluster and removes noise using ENN. The study evaluated this approach using real-world datasets with a maximum of 9 attributes, and it didn't assess the model's performance on datasfets with higher dimensions, primarily due to significant computational time requirements, which placed it last among the fifteen other approaches it was compared to.

In addition, there's another proposal called noise-adaptive synthetic oversampling technique (NASOTECH) [30]. NASOTECH makes use of the noise-adaptive synthetic oversampling (NASO) rule, which involves applying the KNN method to calculate the total distance from each sample in the minority class to its Kth nearest neighbors. These cumulative distances are then utilized to determine the noise ratio for each minority class sample, guiding the generation of synthetic samples. The evaluation of this approach was conducted exclusively with a single classifier, namely the support vector machine (SVM), using real-world datasets with a maximum of 294 features. However, its performance with other classifiers employing different operational principles has not been explored.

These research findings illustrate that integrating SMOTE with outlier cleaning methods can improve the performance of diverse machine learning classifiers. This enhancement is achieved by furnishing the classifiers with balanced training datasets while minimizing noise. Nevertheless, a few of these studies have only applied their techniques to a restricted range of classifiers, leaving their effectiveness unexplored with other classifier types. Furthermore, some studies have neglected the evaluation of crucial performance metrics. Furthermore, previous research has primarily concentrated on datasets with a restricted number of attributes. Moreover, none of these studies have tackled datasets with highly severe imbalance ratios.

3. METHOD

This section presents the proposed methodology, which combines SMOTE and efficient algorithms to address the outlier issue in imbalanced data. Subsection 3.1 describes SMOTE [11] as the basis of the proposed method. While subsection 3.2 discusses the utilization of efficient algorithms [21] to identify the outliers generated by SMOTE.

3.1. Synthetic minority oversampling technique

The SMOTE was initially introduced by Chawla *et al.* [11]. This method has gained popularity for addressing class imbalance issues. To tackle class imbalance, the SMOTE technique augments the underrepresented class with synthetic data, which is generated within the minority class using the KNN method [5]. SMOTE has been widely adopted by researchers and practitioners, particularly in the context of software defect prediction, as demonstrated some references [5], [13], [15], [31], [32]. An illustration of the SMOTE method is presented in Figure 1.

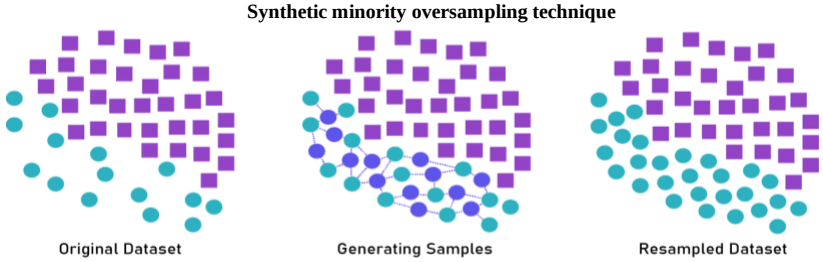

Figure 1. Illustration of SMOTE

SMOTE has two hyperparameters: N, representing the percentage of oversampling for minority classes (non-defect), and k, which signifies the number of nearest neighbors used to generate synthetic data. There is no definitive rule for determining the values of k and N. The formula used to generate a new instance is found in (1). In this way, SMOTE helps optimize the class distribution in the dataset, enhancing the machine learning model's ability to recognize and predict the minority class more accurately.

$$x' = x + rand(0,1) * |x - x_k| \quad (1)$$

Algorithm SMOTE(T,N,k)
Input: Number of minority class samples T; Amount of SMOTE N%; Number of nearest neighbors k
Output: (N/100) * T synthetic minority class samples

```
(* If N is less than 100% randomize the minority class samples as only a random percent of them will be SMOTEd.*
if N < 100
   then Randomize the T minority class samples
      T = (N/100) * T
      N = 100
endif
N = (int)(N/100) ( * The amount of SMOTE is assumed to be in integral multiples of 100.*)
k = Number of nearest neighbors
numattrs = Number of attributes
Samples[][]: array for original minority class samples
newindex: keeps a count number of synthetic samples generated,
initialized to 0
Synthetic[][]: array for synthetic for synthetic samples
(*Compute k nearest neighbors for each minority class sample only.*)
for i ← 1 to T
      Compute k nearest neighbors for i, and save the indices in the nnarray
      Populate(N, i, nnarray)
endfor
Populate(N, i, nnarray) (* Function to generate the synthetic samples. *)
while N ≠ 0
      Choose a random number between 1 and k, call it nn.
      This step chooses one of the k nearest neighbors of the k
      nearest neighbors of i.
      for attr ← 1 to numattrs
            Compute: dif = Sample[nnarray[nn]][attr] – Sample[i][attr]
            Compute: gap = Random number between 0 and 1
            Synthetic[newindex][attr] = Sample[i][attr] + gap * dif
      endfor
      Newindex++
```

25. N = N – 1
26. endwhile
27. **return** (* End of Populate *)
End of Pseudo-Code

3.2. Mining outliers

Outliers are data points or observations that significantly deviate from the expected or typical data points within a dataset [33]. When the number of outliers increases in an imbalanced dataset, it can lead to a decline in the performance of the classification algorithm due to reduced model accuracy [25]. In this study, the identified outliers are primarily derived from synthetic minority class samples created by SMOTE.

This study aims to detect the noise introduced by SMOTE using efficient algorithms, which can effectively identify outliers based on a point's distance from its KNN [20]. Another method used for outlier detection is the LOF. LOF calculates the distance between each data point and other data points within a specific range to assess the degree of abnormality for each data point in the dataset [34]. However, the LOF algorithm has some limitations, such as longer execution times and sensitivity to the minimum point value. The third method used for outlier detection is density-based spatial clustering of applications with noise (DBSCAN) [35], but it is susceptible to high-dimensional datasets, making it challenging to distinguish between outlier points and dense points.

The current study seeks to identify outliers generated by SMOTE using an approach recommended by Ramaswamy *et al.* [20]. This method offers a more efficient strategy for outlier identification, assigning distinct outlier levels to each object. It proposes a formulation for recognizing distance-based outliers by computing the distance between a point and its k-th nearest neighbor. The ranking of each point is determined by its distance from the k-th nearest neighbor, and the top n points in this ranking are designated as outliers. Parameters for the number of neighbors (k) and the number of outliers (n) allow for customization. This approach is grounded in a straightforward and intuitive distance-based outlier criterion, as articulated by Knorr and Ng: 'A point p within a dataset is considered an outlier based on two parameters, k and d, if no more than k points in the dataset are found at a distance of d or less from p'.

This algorithm introduces a new boolean attribute called 'outlier' to the provided ExampleSet. If the 'outlier' attribute has a true value, it signifies that the corresponding example is an outlier, while a false value indicates that the example is not an outlier. The 'outlier' attribute is set to true for a total of n examples (where n is determined by the number of outliers parameter). This operator supports various distance functions, and you can specify your desired distance function by setting it through the distance function parameter.

Figure 2 illustrates the RO-SMOTE algorithm, which combines SMOTE with the mining outliers algorithm [20] to obtain training data with minimal outliers before applying it to the learning algorithm. The core concept involves using SMOTE on imbalanced training data to balance it. However, the results of balanced SMOTE often contain outliers [36], [37], which can mislead the classifier. Therefore, in the subsequent step, the mining outliers algorithm [20] is used to remove the detected outliers, resulting in clean but imbalanced data that is then integrated with the training data. To prepare this data for classifier use, SMOTE is reapplied to balance it and produce clean, balanced data. Figure 2 offers an overview of this process.

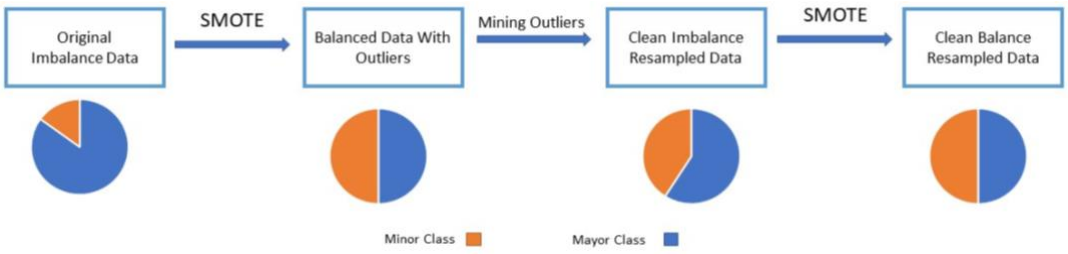

Figure 2. An overview of the RO-SMOTE mode, illustrating the steps and both the minor and major classes involved in each step

Figure 3 outlines the specific steps of the proposed process. The process initiated by normalizing the dataset through Z-score normalization [37]. Subsequently, the normalized dataset underwent resampling using SMOTE to create balanced data. The outcomes fof data resampling were further refined using an efficient algorithm since the newly balanced data still contained outliers.

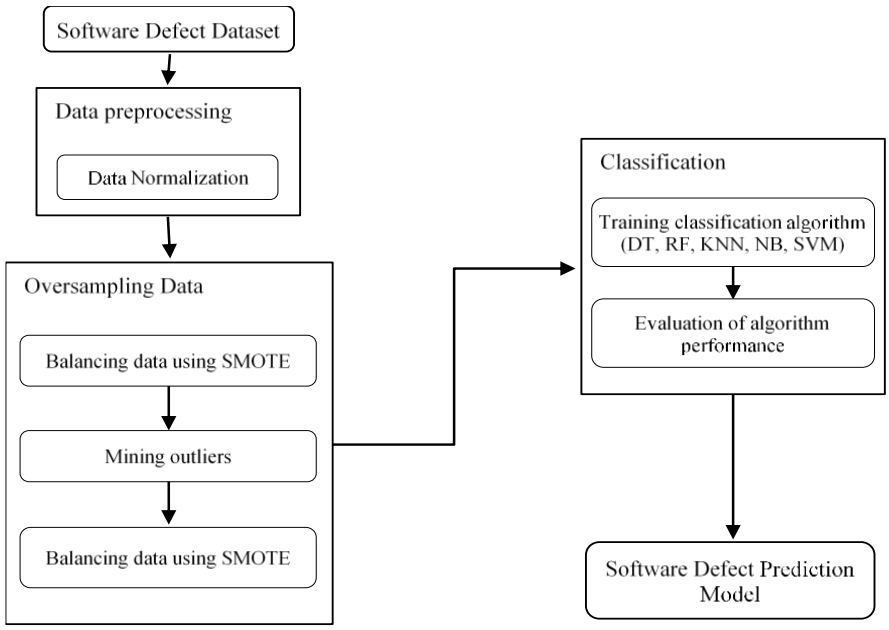

Figure 3. Flowchart of research process

4. RESULTS AND DISCUSSION

In this section, we discuss the evaluation of RO-SMOTE, which includes assessing its performance with three different classification algorithms: KNN, SVM, and neural network (NN). We conducted this evaluation on datasets from 11 cross-projects for software defect prediction, each with varying imbalance ratios. The evaluation employed multiple metrics, such as balanced accuracy [38], precision, recall, F1-score, and AUC.

4.1. Datasets

This study utilized datasets from 11 cross-projects for software prediction sourced from NASA and promise projects. To distribute the dataset, we applied K-fold cross validation (K=10). In this study, one fold was reserved for testing data, while the remaining nine folds were allocated for training data in each dataset. Table 1 provides details regarding the number of features, samples, and imbalance ratios for each dataset. These datasets include attributes like static software metrics that help characterize the presence of defects in the software.

Table 1. Properties of datasets in 11 projects

Datasets	Project	#instances	#Attributes	#defects	%defect
NASA MDP	CM1	327	38	42	12.84
	KC1	2109	22	326	15.4
	PC1	679	38	25	10
PROMISE	Ant-1.3	125	21	20	16
	Ant-1.7	745	21	166	22.28
	Camel-1.6	965	21	188	19.48
	Ivy-2.0	352	21	40	11.36
	Jedit-4.3	492	21	11	2.24
	Poi-3.0	442	21	281	63.57
	Synapse-1.2	256	21	86	33.59
	Velocity-1.6	229	21	78	34.06

4.2. Results discussion for classifier

In this section, we delve into the results of implementing RO-SMOTE with KNN, SVM, and NN algorithms. The hyperparameters for these classifiers have been optimized using 10-fold grid search, and the hyperparameter configurations are outlined in Table 2. The inclusion of three distinct algorithms in this paper serves the purpose of evaluating the compatibility of RO-SMOTE with various algorithms. RO-SMOTE's performance was then compared with that of SMOTE to assess the improvement in classifier algorithm

performance. The performance of each classifier was evaluated based on balanced accuracy, F1-score, precision, recall, and AUC across all 11 datasets. The results for each classifier are presented in individual tables. Balanced accuracy score represents an enhanced version of the standard accuracy metric, specifically tailored to enhance performance on datasets with imbalanced class distributions. It accomplishes this by calculating the average accuracy for each class individually, as opposed to the aggregation employed in standard accuracy calculations [38]. The F1-score, on the other hand, is the harmonic mean of precision and recall from a classification model. In the context of imbalanced datasets, the F1-score serves as a more reliable alternative to the standard accuracy metric. It provides a more accurate assessment of the model's performance, especially when prioritizing the evaluation of minority or critical classes [39].

Table 2. Hyperparameter configuration

Classifier	Hyperparameters
SMOTE	$k = 2$
NN	training cycles = 200, learning rate= 0.01
KNN	k = 5
SVM (linear)	Kernel cache = 200
SVM (radial)	Kernel gamma = 1.0
SVM (poly)	Kernel degree = 2.0

Table 3 compares the performance of SMOTE and RO-SMOTE in the NN classifier. From this table, the performance of RO-SMOTE was superior to SMOTE in terms of balanced accuracy metrics (0.42%-5.2%), F1-score (1.18%-4.38%), and AUC (0.50%-5.70%). In terms of precision, RO-SMOTE outperformed SMOTE in six datasets (0.62%-7.3%), while SMOTE excelled in five datasets (0.34%-3.89%). Regarding recall, RO-SMOTE was superior in 11 datasets (0.26%-12.29%), while SMOTE only excelled in 2 datasets (1.62%-2.17%).

Table 3. Results summary of NN classifier

Dataset	Model	Metrics				
		Balanced accuracy (%)	Precision (%)	Recall (%)	F1-score(%)	AUC (%)
CM1	SMOTE	80.63	79.15	84.19	81.59	0.861
	RO-SMOTE	83.06	83.64	82.57	83.10	0.888
KC1	SMOTE	73.39	71.94	77.29	74.51	0.818
	RO-SMOTE	75.11	70.87	86.60	77.54	0.832
PC1	SMOTE	82.75	78.47	91.12	84.32	0.887
	RO-SMOTE	86.06	81.72	93.33	87.13	0.914
Ant-1.3	SMOTE	86.19	85.75	88.73	87.21	0.889
	RO-SMOTE	91.39	93.05	89.67	91.32	0.946
Ant-1.7	SMOTE	77.72	78.98	75.67	77.28	0.838
	RO-SMOTE	79.15	78.64	81.05	79.82	0.854
Camel-1.6	SMOTE	74.38	71.97	81.04	76.23	0.810
	RO-SMOTE	75.03	73.39	78.87	76.03	0.820
Ivy-2.0	SMOTE	83.49	88.47	77.83	82.80	0.910
	RO-SMOTE	85.93	85.59	86.79	86.18	0.924
Jedit-4.3	SMOTE	94.18	93.16	95.42	94.27	0.975
	RO-SMOTE	94.60	93.78	95.68	95.68	0.980
Poi-3.0	SMOTE	79.19	78.91	80.86	79.87	0.844
	RO-SMOTE	79.85	76.56	86.11	81.05	0.807
Synapse-1.2	SMOTE	79.12	79.45	79.41	79.42	0.846
	RO-SMOTE	81.52	80.56	84.17	82.32	0.875
Velocity-1.6	SMOTE	77.13	82.00	72.21	76.79	0.841
	RO-SMOTE	80.27	78.11	84.50	81.17	0.832

Table 4 compares the performance of SMOTE and RO-SMOTE in the KNN classifier. The experimental results demonstrated that RO-SMOTE outperformed SMOTE in terms of balanced accuracy in nine datasets (0.06%-7.81%), while SMOTE excelled in two datasets (0.48%-4.46%). In terms of precision, RO-SMOTE excelled in six datasets (0.21%-15.43%), while SMOTE excelled in two datasets (0.19%-10.47%). RO-SMOTE outperformed in recall in seven datasets (0.12%-14.73%), while SMOTE outperformed in four datasets (0.31%-17.97%). Furthermore, for the results of the F1-score, RO-SMOTE produced better results than SMOTE in eight datasets (0.11%-5.93%), while SMOTE excelled in three datasets (0.4%-3.69%). For AUC, RO-SMOTE excelled in eight datasets (0.20%-2.80%), while SMOTE excelled in three datasets (0.10%-2.20%).

Table 4. Results summary of KNN classifier

Dataset	Model	Metrics				
		Balanced accuracy (%)	Precision (%)	Recall (%)	F1-score(%)	AUC (%)
CM1	SMOTE	88.09	82.33	97.32	89.20	0.942
	RO-SMOTE	88.75	97.76	79.35	87.60	0.954
KC1	SMOTE	85.45	81.21	92.32	86.41	0.928
	RO-SMOTE	85.51	81.02	92.81	86.52	0.935
PC1	SMOTE	93.17	89.20	98.35	93.55	0.972
	RO-SMOTE	92.69	88.72	98.04	93.15	0.971
Ant-1.3	SMOTE	85.24	80.62	95.18	87.30	0.928
	RO-SMOTE	93.05	94.49	92.00	93.23	0.956
Ant-1.7	SMOTE	84.63	79.66	93.44	86.00	0.920
	RO-SMOTE	85.80	82.22	91.79	86.74	0.920
Camel-1.6	SMOTE	80.05	74.06	92.53	82.27	0.880
	RO-SMOTE	79.95	73.74	93.40	82.41	0.889
Ivy-2.0	SMOTE	90.39	96.06	84.43	89.87	0.946
	RO-SMOTE	85.93	85.59	86.79	86.18	0.924
Jedit-4.3	SMOTE	94.08	91.47	97.30	94.29	0.984
	RO-SMOTE	95.05	93.13	97.42	95.23	0.986
Poi-3.0	SMOTE	82.20	81.51	84.72	83.08	0.886
	RO-SMOTE	83.33	81.72	86.04	83.82	0.914
Synapse-1.2	SMOTE	80.88	85.61	74.71	79.79	0.902
	RO-SMOTE	82.27	84.68	81.08	82.84	0.899
Velocity-1.6	SMOTE	77.49	83.60	68.75	75.45	0.872
	RO-SMOTE	80.41	78.97	83.48	81.16	0.854

Table 5 presents a comparison of the experimental results of SMOTE and RO-SMOTE in the SVM classifier on the linear kernel. For balanced accuracy, RO-SMOTE provided better results than SMOTE in eight datasets (0.18%-6.94%), while SMOTE excelled in two datasets (0.26%-0.28%). In terms of precision, RO-SMOTE excelled in eight datasets (0.15%-10.42%), while SMOTE excelled in three datasets (0.23%-1.73%). As for recall, RO-SMOTE excelled in 10 datasets (0.19%-9.02%), while SMOTE excelled in only one dataset (0.82%). For the F1-Score, RO-SMOTE produced better results in 10 datasets (0.17%-8.06%), while SMOTE was only superior in one dataset (0.57%). In terms of AUC value, RO-SMOTE outperformed SMOTE in eight datasets (0.40%-10.90%), while SMOTE excelled in three datasets (0.40%-0.70%).

Table 5. Results summary of SVM (linear) classifier

Dataset	Model	Metrics				
		Balanced accuracy (%)	Precision (%)	Recall (%)	F1-score(%)	AUC (%)
CM1	SMOTE	79.29	79.09	79.98	79.53	0.858
	RO-SMOTE	80.86	80.44	81.43	80.93	0.873
KC1	SMOTE	72.46	72.93	71.51	72.21	0.815
	RO-SMOTE	72.64	73.08	71.70	72.38	0.811
PC1	SMOTE	80.48	77.77	85.85	81.61	0.852
	RO-SMOTE	81.40	77.23	89.31	82.83	0.856
Ant-1.3	SMOTE	82.38	79.11	89.45	83.96	0.867
	RO-SMOTE	91.57	89.53	95.00	92.18	0.976
Ant-1.7	SMOTE	76.08	80.09	69.75	74.56	0.827
	RO-SMOTE	75.80	79.86	68.93	73.99	0.820
Camel-1.6	SMOTE	65.64	67.90	59.60	63.48	0.719
	RO-SMOTE	66.36	68.29	61.35	64.63	0.733
Ivy-2.0	SMOTE	81.26	80.04	83.68	81.82	0.871
	RO-SMOTE	83.44	81.25	87.82	84.41	0.889
Jedit-4.3	SMOTE	83.07	84.76	80.73	82.70	0.906
	RO-SMOTE	86.13	87.56	84.51	86.01	0.925
Poi-3.0	SMOTE	78.47	76.73	82.18	79.36	0.850
	RO-SMOTE	78.21	75.00	85.40	79.86	0.844
Synapse-1.2	SMOTE	75.29	73.71	78.82	76.18	0.802
	RO-SMOTE	75.29	72.74	84.33	78.11	0.808
Velocity-1.6	SMOTE	73.52	74.89	70.79	72.78	0.800
	RO-SMOTE	80.46	81.90	79.81	80.84	0.845

Table 6 compares the performance of SMOTE and RO-SMOTE in the Radial kernel SVM classifier. The experimental results show that, in terms of balanced accuracy, RO-SMOTE was superior in eight datasets (0.16%-3.05%), while SMOTE excelled in three datasets (0.42%-0.61%). For precision, RO-SMOTE excelled in six datasets (0.42%-5.36%), while SMOTE excelled in five datasets (1.10%-4.32%). In terms of recall, RO-SMOTE outperformed in seven datasets (1.07%-9.21%), while SMOTE excelled in four datasets (0.83%-4.06%). Furthermore, in terms of F1-score, RO-SMOTE produced better results than

SMOTE in seven datasets (0.12%-3.73%), while SMOTE excelled in four datasets (0.16%-1.28%). Finally, for AUC, RO-SMOTE excelled in eight datasets (0.10%-2.80%), while SMOTE excelled in three datasets (0.30%-1.20%).

Table 6. Results summary of SVM (radial) classifier

Dataset	Metrics					
	Model	Balanced accuracy (%)	Precision (%)	Recall (%)	F1-score(%)	AUC (%)
CM1	SMOTE	88.31	90.10	86.44	88.23	0.948
	RO-SMOTE	89.21	92.32	85.61	88.84	0.945
KC1	SMOTE	79.25	77.26	83.01	80.03	0.872
	RO-SMOTE	78.64	78.55	78.95	78.75	0.860
PC1	SMOTE	91.57	89.62	94.09	91.80	0.969
	RO-SMOTE	94.60	93.09	96.46	94.75	0.974
Ant-1.3	SMOTE	89.52	96.09	82.91	89.01	0.947
	RO-SMOTE	91.57	93.76	89.00	91.32	0.948
Ant-1.7	SMOTE	86.27	82.72	92.38	87.28	0.939
	RO-SMOTE	86.43	81.62	94.29	87.50	0.950
Camel-1.6	SMOTE	82.37	83.98	80.19	82.04	0.898
	RO-SMOTE	81.86	82.50	81.26	81.88	0.886
Ivy-2.0	SMOTE	88.79	88.62	89.45	89.03	0.944
	RO-SMOTE	89.37	91.57	86.86	89.15	0.950
Jedit-4.3	SMOTE	93.97	94.78	93.13	93.95	0.980
	RO-SMOTE	93.55	92.51	94.82	93.65	0.968
Poi-3.0	SMOTE	83.98	86.37	80.79	83.49	0.906
	RO-SMOTE	85.91	91.73	79.09	84.94	0.924
Synapse-1.2	SMOTE	81.76	89.25	72.94	80.27	0.892
	RO-SMOTE	84.81	89.67	79.00	84.00	0.920
Velocity-1.6	SMOTE	80.45	80.46	82.71	81.57	0.902
	RO-SMOTE	81.11	76.14	91.92	83.29	0.915

Table 7 compares the performance of SMOTE and RO-SMOTE in the SVM kernel poly classifier. In this table, the performance of RO-SMOTE was superior to SMOTE in the balanced accuracy metric (0.01%-17.60%). Regarding precision, RO-SMOTE was better in five datasets (1.62%-22.13%), while SMOTE excelled in six datasets (0.08%-30.66%). In terms of recall, RO-SMOTE excelled in six datasets (4.76%-82.37%), while SMOTE only excelled in five datasets (0.10%-7.09%). For the F1-score, RO-SMOTE measurements produced better results in nine datasets (0.39%-43.83%), while SMOTE was only superior in two datasets (0.24%-3.14%). Finally, in the AUC value, RO-SMOTE outperformed SMOTE in 10 datasets (3.00%-27.30%), while SMOTE excelled in one dataset (6.30%).

Table 7. Results summary of SVM (poly) classifier

Dataset	Metrics					
	Model	Balanced accuracy (%)	Precision (%)	Recall (%)	F1-score(%)	AUC (%)
CM1	SMOTE	57.13	87.05	16.71	28.04	0.502
	RO-SMOTE	61.14	56.39	99.08	71.87	0.775
KC1	SMOTE	61.81	81.04	39.82	53.40	0.756
	RO-SMOTE	64.35	83.54	35.94	50.26	0.793
PC1	SMOTE	64.78	85.43	35.57	50.23	0.655
	RO-SMOTE	65.55	81.35	40.33	53.93	0.769
Ant-1.3	SMOTE	67.14	61.74	96.36	75.26	0.763
	RO-SMOTE	84.74	83.87	89.27	86.49	0.915
Ant-1.7	SMOTE	69.17	74.21	61.23	67.10	0.788
	RO-SMOTE	71.96	68.32	82.32	74.67	0.813
Camel-1.6	SMOTE	66.15	61.74	85.85	71.83	0.728
	RO-SMOTE	68.21	64.33	82.32	72.22	0.761
Ivy-2.0	SMOTE	72.10	90.39	49.63	64.08	0.872
	RO-SMOTE	77.87	92.01	61.15	73.47	0.898
Jedit-4.3	SMOTE	78.48	71.06	97.08	82.06	0.925
	RO-SMOTE	86.02	79.69	96.98	87.49	0.955
Poi-3.0	SMOTE	67.80	71.13	66.58	68.78	0.799
	RO-SMOTE	74.19	68.37	90.44	77.87	0.736
Synapse-1.2	SMOTE	69.41	63.15	95.29	75.96	0.783
	RO-SMOTE	69.42	63.07	94.71	75.72	0.813
Velocity-1.6	SMOTE	63.53	83.52	33.83	48.15	0.719
	RO-SMOTE	70.62	72.08	73.82	72.94	0.800

After conducting experiments on all selected datasets and classifiers, involving 11 datasets and 5 classifiers, a total of 110 experiments were performed. The results demonstrate that the proposed RO-SMOTE approach can significantly enhance classification performance in predicting software defects. According to the experimental results, RO-SMOTE outperformed SMOTE 47 times, while SMOTE outperformed RO-SMOTE only 8 times in terms of balanced accuracy measurements. Furthermore, RO-SMOTE excelled 36 times in precision measurements, 40 times in recall, 45 times in F1-score, and 42 times in the AUC value. The average F1-score values in Figure 4 indicate that RO-SMOTE's F1-score is consistently higher, highlighting a balance between the model's ability to identify true positives (precision) and its ability to detect all true positive cases (recall). RO-SMOTE improves the performance of all the classifiers used when compared to using SMOTE. RO-SMOTE enhances the performance of NN by 9.44%, KNN by 8%, linear SVM by 10%, radial SVM by 8%, and poly SVM by 14%.

In the NN classifier, RO-SMOTE consistently improves balanced accuracy, F1-score, and AUC across all 11 datasets. Therefore, selecting the right parameters for the NN classifier can notably enhance the performance of the resulting software defect prediction model. When it comes to SVM with three different kernels-linear, radial, and poly, the radial kernel achieved the highest balanced accuracy at 94.60% in the PC1 dataset. This demonstrates that the radial kernel is the most suitable for classifying software defect predictions using SVM. In contrast, using the poly kernel in SVM resulted in a decrease in classification performance compared to the radial kernel. For instance, in the PC1 dataset, the balanced accuracy decreased by 29.05%. However, implementing RO-SMOTE on the poly kernel of SVM led to a significant increase in recall compared to SMOTE, as seen in the CM1 dataset, where the recall increased by 82.37%. This, in turn, boosted the F1-score to 71.87%, representing a 43.83% increase. A higher F1-score value indicates a balance between the model's precision and recall.

Moreover, the outcomes of these experiments suggest that RO-SMOTE outperforms SMOTE in both balanced accuracy and F1-score across all the datasets examined. In instances where datasets have a larger volume of data and more significant imbalance, as observed in the Jedit dataset, the RO-SMOTE method exhibits superior balanced accuracy compared to SMOTE. Conversely, for datasets containing fewer data samples, such as synapse-1.2 and velocity-1.6, SMOTE consistently achieves substantially better balanced accuracy, especially when combined with different classifiers. This poses a challenge for future research, as it will require testing the proposed method on alternative datasets with varying combinations of data sample quantities and imbalance ratios.

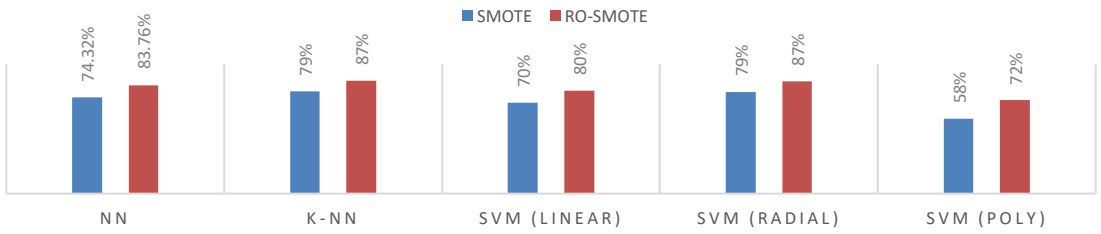

Figure 4. Comparison of average performance F1-scores between SMOTE and RO-SMOTE

4.3. Comparison with the current state of art

In this section, we assess the effectiveness of RO-SMOTE in comparison to the current state-of-the-art method, SMOTE-LOF using Ant-1.3 dataset [21]. KNN classifiers have been used for this comparisson. The metrics considered for comparison include Accuracy, F1-score, and AUC. The classifications undergo training and evaluation through 10-fold cross-validation. Only the most optimal results for the SMOTE-LOF model are included in the analysis. In the case of the Ant-1.3 dataset, the findings demonstrated that the KNN classifier, when assessed based on Accuracy, AUC and F1-score metrics with RO-SMOTE, surpassed the performance of SMOTE-LOF by 7.81%, 0.028 and 5.93%, respectively.

5. CONCLUSION

This paper introduces a model for classifying imbalanced datasets in software defect prediction. The model employs three steps: first, oversampling the training data using SMOTE, second, removing noise from the resulting oversampled data using the efficient algorithm, and finally, combining the cleaned data with the original training data and rebalancing using SMOTE. Evaluation experiments were conducted on 11 datasets

for software defect prediction, utilizing five different classifiers and five evaluation metrics. The results indicate that all classifiers demonstrated improved performance when the proposed RO-SMOTE model was implemented, as opposed to using SMOTE alone. The extent of performance improvement varied across datasets, classifiers, and metrics. NN classification consistently delivered strong performance, while other classifiers showed high performance when combined with RO-SMOTE across various metrics for one or more datasets. Compared to SMOTE, RO-SMOTE demonstrated performance improvements of up to 7.8% in balanced accuracy, 22.13% in precision, 82.37% in recall, 43.83% in F1 metrics, and an increase of 27.30% in AUC. In the future, our research will expand to include multiclass imbalanced datasets. Additionally, we plan to leverage deep learning models, such as AutoML and other deep learning approaches, for noise detection and removal. These models will be combined with various resampling techniques. Moreover, we intend to utilize deep learning models for data resampling to address the limitations of existing techniques.

REFERENCES

[1] Y. Yang and M. Han, "Impact of mobile device usage and temporal distance on consumer post-consumption evaluations: evidence from TripAdvisor," *Electronic Commerce Research and Applications*, vol. 56, 2022, doi: 10.1016/j.elerap.2022.101208.

[2] B. Kocaman, S. Gelper, and F. Langerak, "Till the cloud do us part: technological disruption and brand retention in the enterprise software industry," *International Journal of Research in Marketing*, vol. 40, no. 2, pp. 316–341, 2023, doi: 10.1016/j.ijresmar.2022.11.001.

[3] J. A. Sava, "Information technology (IT) spending forecast worldwide from 2012 to 2023, by segment," *Statista*, 2023. Accessed: May 10, 2023. [Online]. Available: https://www.statista.com/Statistics/268938/Global-It-Spending-By-Segment/

[4] V. K. Kulamala, L. Kumar, and D. P. Mohapatra, "Software fault prediction using LSSVM with different kernel functions," *Arabian Journal for Science and Engineering*, vol. 46, no. 9, pp. 8655–8664, 2021, doi: 10.1007/s13369-021-05643-2.

[5] R. B. Bahaweres, F. Agustian, I. Hermadi, A. I. Suroso, and Y. Arkeman, "Software defect prediction using neural network based smote," in *International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, IEEE, Oct. 2020, pp. 71–76. doi: 10.23919/EECSI50503.2020.9251874.

[6] 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.

[7] A. Iqbal *et al.*, "Performance analysis of machine learning techniques on software defect prediction using NASA datasets," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 5, pp. 300–308, 2019, doi: 10.14569/ijacsa.2019.0100538.

[8] G. Douzas, F. Bacao, and F. Last, "Improving imbalanced learning through a heuristic oversampling method based on k-means and SMOTE," *Information Sciences*, vol. 465, pp. 1–20, 2018, doi: 10.1016/j.ins.2018.06.056.

[9] G. Douzas and F. Bacao, "Effective data generation for imbalanced learning using conditional generative adversarial networks," *Expert Systems with Applications*, vol. 91, pp. 464–471, 2018, doi: 10.1016/j.eswa.2017.09.030.

[10] I. K. Timotius and S.-G. Miaou, "Arithmetic means of accuracies: a classifier performance measurement for imbalanced data set," in *2010 International Conference on Audio, Language and Image Processing*, 2010, pp. 1244–1251. doi: 10.1109/ICALIP.2010.5685124.

[11] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, Jun. 2002, doi: 10.1613/jair.953.

[12] J. Pachouly, S. Ahirrao, K. Kotecha, G. Selvachandran, and A. Abraham, "A systematic literature review on software defect prediction using artificial intelligence: datasets, data validation methods, approaches, and tools," *Engineering Applications of Artificial Intelligence*, vol. 111, 2022, doi: 10.1016/j.engappai.2022.104773.

[13] S. Feng, J. Keung, X. Yu, Y. Xiao, and M. Zhang, "Investigation on the stability of SMOTE-based oversampling techniques in software defect prediction," *Information and Software Technology*, vol. 139, 2021, doi: 10.1016/j.infsof.2021.106662.

[14] D. Bowes, T. Hall, and J. Petrić, "Software defect prediction: do different classifiers find the same defects?," *Software Quality Journal*, vol. 26, no. 2, pp. 525–552, Jun. 2018, doi: 10.1007/s11219-016-9353-3.

[15] T. F. Husin, M. R. Pribadi, and Yohannes, "Implementation of LSSVM in classification of software defect prediction data with feature selection," in *2022 9th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, IEEE, Oct. 2022, pp. 126–131. doi: 10.23919/EECSI56542.2022.9946611.

[16] P. Vuttipittayamongkol, E. Elyan, and A. Petrovski, "On the class overlap problem in imbalanced data classification," *Knowledge-Based Systems*, vol. 212, Jan. 2021, doi: 10.1016/j.knosys.2020.106631.

[17] T. Sharma, A. Jatain, S. Bhaskar, and K. Pabreja, "Ensemble machine learning paradigms in software defect prediction," *Procedia Computer Science*, vol. 218, pp. 199–209, 2023, doi: 10.1016/j.procs.2023.01.002.

[18] A. Kumar, A. Kumar, A. K. Bashir, M. Rashid, V. D. A. Kumar, and R. Kharel, "Distance based pattern driven mining for outlier detection in high dimensional big dataset," *ACM Transactions on Management Information Systems*, vol. 13, no. 1, pp. 1–17, Mar. 2022, doi: 10.1145/3469891.

[19] I. Souiden, M. N. Omri, and Z. Brahmi, "A survey of outlier detection in high dimensional data streams," *Computer Science Review*, vol. 44, May 2022, doi: 10.1016/j.cosrev.2022.100463.

[20] S. Ramaswamy, R. Rastogi, and K. Shim, "Efficient algorithms for mining outliers from large data sets," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, New York, USA: ACM, 2000, pp. 427–438. doi: 10.1145/342009.335437.

[21] Asniar, N. U. Maulidevi, and K. Surendro, "SMOTE-LOF for noise identification in imbalanced data classification," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 6, pp. 3413–3423, 2022, doi: 10.1016/j.jksuci.2021.01.014.

[22] N. Nnamoko and I. Korkontzelos, "Efficient treatment of outliers and class imbalance for diabetes prediction," *Artificial Intelligence in Medicine*, vol. 104, 2020, doi: 10.1016/j.artmed.2020.101815.

[23] A. Banerjee, K. Ghosh, S. Chatterjee, and D. Sen, "FOFO: fused oversampling framework by addressing outliers," in *2021 International Conference on Emerging Smart Computing and Informatics (ESCI)*, 2021, pp. 238–242. doi: 10.1109/ESCI50559.2021.9397056.

[24] B. Dastjerdy, A. Saeidi, and S. Heidarzadeh, "Review of applicable outlier detection methods to treat geomechanical data," *Geotechnics*, vol. 3, no. 2, pp. 375–396, May 2023, doi: 10.3390/geotechnics3020022.

[25] M. H. Ibrahim, "ODBOT: Outlier detection-based oversampling technique for imbalanced datasets learning," *Neural Computing and Applications*, vol. 33, no. 22, pp. 15781–15806, 2021, doi: 10.1007/s00521-021-06198-x.

[26] M. H. Ibrahim, "WBBA-KM: a hybrid weight-based bat algorithm with k-means algorithm for cluster analysis," *Journal of Politeknik Dergisi*, vol. 25, no. 1, pp. 65–73, Mar. 2022, doi: 10.2339/politeknik.689384.

[27] P. Gnip, L. Vokorokos, and P. Drotár, "Selective oversampling approach for strongly imbalanced data," *PeerJ Computer Science*, vol. 7, Jun. 2021, doi: 10.7717/peerj-cs.604.
[28] N. Usman, E. Utami, and A. D. Hartanto, "Comparative analysis of elliptic envelope, isolation forest, one-class SVM, and local outlier factor in detecting earthquakes with status anomaly using outlier," in *2023 International Conference on Computer Science, Information Technology and Engineering (ICCoSITE)*, IEEE, Feb. 2023, pp. 673–678. doi: 10.1109/ICCoSITE57641.2023.10127748.
[29] A. Puri and M. K. Gupta, "Knowledge discovery from noisy imbalanced and incomplete binary class data," *Expert Systems with Applications*, vol. 181, Nov. 2021, doi: 10.1016/j.eswa.2021.115179.
[30] M. T. Vo, T. Nguyen, H. A. Vo, and T. Le, "Noise-adaptive synthetic oversampling technique," *Applied Intelligence*, vol. 51, no. 11, pp. 7827–7836, Nov. 2021, doi: 10.1007/s10489-021-02341-2.
[31] C. Pak, T. T. Wang, and X. H. Su, "An empirical study on software defect prediction using over-sampling by SMOTE," *International Journal of Software Engineering and Knowledge Engineering*, vol. 28, no. 6, pp. 811–830, 2018, doi: 10.1142/S0218194018500237.
[32] H. Alsawalqah, H. Faris, I. Aljarah, L. Alnemer, and N. Alhindawi, "Hybrid SMOTE-ensemble approach for software defect prediction," in *Advances in Intelligent Systems and Computing*, vol. 575, Springer, 2017, pp. 355–366. doi: 10.1007/978-3-319-57141-6_39.
[33] B. Chander and G. Kumaravelan, "Outlier detection strategies for WSNs: a survey," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 8, pp. 5684–5707, Sep. 2022, doi: 10.1016/j.jksuci.2021.02.012.
[34] O. Alghushairy, R. Alsini, T. Soule, and X. Ma, "A review of local outlier factor algorithms for outlier detection in big data streams," *Big Data and Cognitive Computing*, vol. 5, no. 1, pp. 1–24, 2021, doi: 10.3390/bdcc5010001.
[35] A. Arafa, N. El-Fishawy, M. Badawy, and M. Radad, "RN-SMOTE: Reduced noise SMOTE based on DBSCAN for enhancing imbalanced data classification," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 8, pp. 5059–5074, 2022, doi: 10.1016/j.jksuci.2022.06.005.
[36] L. Chen, B. Fang, Z. Shang, and Y. Tang, "Tackling class overlap and imbalance problems in software defect prediction," *Software Quality Journal*, vol. 26, no. 1, pp. 97–125, Mar. 2018, doi: 10.1007/s11219-016-9342-6.
[37] Henderi, T. Wahyuningsih, and E. Rahwanto "Comparison of min-max normalization and Z-score normalization in the k-nearest neighbor (KNN) algorithm to test the accuracy of types of breast cancer," *IJIIS: International Journal of Informatics and Information Systems*, vol. 4, no. 1, pp. 13–20, Mar. 2021, doi: 10.47738/ijiis.v4i1.73.
[38] K. H. Brodersen, C. S. Ong, K. E. Stephan, and J. M. Buhmann, "The balanced accuracy and its posterior distribution," in *2010 20th International Conference on Pattern Recognition*, IEEE, Aug. 2010, pp. 3121–3124. doi: 10.1109/ICPR.2010.764.
[39] 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, doi: 10.1186/s12864-019-6413-7.

BIOGRAPHIES OF AUTHORS

Muhammad Rizky Pribadi is a doctoral student at Satya Wacara Christian University, and lecturer at Universitas Multi Data Palembang, Indonesia. He got masater from Universitas Indonesia in 2014 and graduate from Universitas Multi Data Palembang in 2011, both computer science. His research interest inculude machine learning, information retrieval, text mining, and computer vision. He can be contacted at email: rizky@mdp.ac.id.

Hindriyanto Dwi Purnomo is a professor at Department of Information Technology, Satya Wacana Christian University, Indonesia. He received his Bachelor degree in Engineering Physics, from Gadjah Mada University, Indonesia, in 2005, Magister of Information Technology from The University of Melbourne, Australia in 2009 and Doctor of Philosophy in Industrial and System Engineering from Chung Yuan Christian University, Taiwan, in 2013. He was a visiting scholar at Sophia University, Japan, in 2022. He has received several recognitions and awards for his academic achievement. He also has published many articles in reputable journals, conferences and books. His research interests are in the field of metaheuristics, soft computing, machine learning, and deep learning. He can be contacted at email: hindriyanto.purnomo@uksw.edu.

Hendry received a B.S. degree in Information Technology from Technology School of Surabaya, in 2005, an M.S. degree in Information Technology from 10 November Institute of Technology, Surabaya, in 2009, and a Ph.D. degree in Information Management from Chaoyang University of Technology in 2018. From 2012-2014 he was the Director of the Business and Technology Incubator at Satya Wacana Christian University. He is now a lecturer in Faculty of Information Technology, Satya Wacana Christian University, Central Java, Indonesia. His research interests include domain ontology, recommendation systems, knowledge engineering, and applications of artificial intelligence. He can be contacted at email: hendry@uksw.edu.