Applied Soft Computing 124 (2022) 109069

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Diversity based imbalance learning approach for software fault prediction using machine learning models

Pravali Manchala *, Manjubala Bisi

Department of Computer Science and Engineering, National Institute of Technology Warangal, India

ARTICLE INFO

Article history:
Received 11 October 2021
Received in revised form 17 May 2022
Accepted 24 May 2022
Available online 31 May 2022

Keywords:
Imbalance learning
Software fault prediction
Oversampling
Machine Learning Model
Deep Neural Network

ABSTRACT

The Software fault prediction (SFP) target is to distinguish between faulty and non-faulty modules. The prediction model's performance is vulnerable to the class imbalance issue in SFP. The existing oversampling approaches generate relatively identical synthetic data, which results in over-generalization and less diverse data. Moreover, many undesirable noisy modules are introduced while generating synthetic data. In this study, we propose the Weighted Average Centroid based Imbalance Learning Approach (WACIL), an effective synthetic over-sampling technique to mitigate the imbalance issue. The WACIL first finds borderline instances, then generates pseudo-data of them through a weighted average centroid concept and filters out inappropriate noise data through a filtration process. We conducted experiments on 24 PROMISE and NASA projects and compared them with some of the existing sampling approaches using K-Nearest Neighbors (KNN), Logistic Regression (LR), Naive Bayes (NB), Support Vector Machine (SVM), Decision Tree (DT) and Deep Neural Network (DNN) as classification models. WACIL achieves superior results in terms of Fall Out Rate (FOR), F-measure and Area Under Curve (AUC) and obtains comparable results in terms of Recall and G-mean compared to the competitive approaches. The statistical analysis indicates that WACIL's ability to outperform the other over-sampling techniques is significant under the statistical Wilcoxon signed rank test and matched pairs rank biserial correlation coefficient effect size. Hence, WACIL is advisable as a competent choice to deal with the imbalance issue in SFP.

Code metadata

Permanent link to reproducible Capsule: https://doi.org/10.24433/CO.6182805.v1

1. Introduction

Over the last few years, software has played a vital role in the human life cycle by increasing human dependency on software for various applications, like smart homes, manufacturing companies, clinical support units, air transport management, education systems, shopping, and a few more controlling systems. Commercial and government organizations also rely on software systems for their projects [1]. These days, software complexity is gradually increasing to carry through the demands of customers, while the occurrence of errors during the software development stage has increased. A Fault is a situation where a software product does not reach consumer demand because of errors that occur during software development, such types of faults may lower the software quality [2]. In software, the modules which contain defects or bugs (for example, logical, arithmetic, multi-threading, syntax, performance, interface defects, etc.) are classified as faulty modules, and the modules which do not contain any defects are classified as non-faulty modules. The fault management often cost about 80 percent of the total budget of the entire project [3]. Software testing crews spend more time on non-faulty modules to know whether they are faulty or non-faulty. Discovering and resolving those faulty modules is a time-consuming task [4]. In the early scenario, software developing organizations endeavored to provide superlative quality software cost-effectively by predicting software faulty modules in the early stage [4,5].

Currently, software fault prediction (SFP) is gaining more popularity. Research studies on prediction of software faults emphasized the eminence of many classification models, including Logistic Regression classifier [6,7], Support Vector Machine classifier [8], Random Forest classifier [7], K-Nearest Neighbors classifier [9], Bayesian Methods, Naive Bayes classifier [10,11],

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author.
E-mail addresses: mpravali@nitw.ac.in (P. Manchala), manjubalabisi@nitw.ac.in (M. Bisi).

https://doi.org/10.1016/j.asoc.2022.109069
1568-4946/

Decision Tree classifier [12], Artificial and Deep Neural Networks (ANN & DNN) [5,13–15].

There is an important problem associated with the prediction of software faults, that is the class imbalance issue [16,17]. In practice, this can be defined in a way where the ratio between the number of faulty (i.e. samples or modules) and non-faulty instances is very high in datasets used for SFP. The classifier is excessively trained on imbalanced datasets to recognize non-faulty instances rather than faulty instances. So, at the time of testing, the classifier yields results partial towards non-faulty instances, which means non-faulty instances are classified accurately and faulty instances can be misclassified as non-faulty [18]. The predictive performance of classifiers might be degraded by such type of skewed distribution. For these reasons, the fault prediction pays more attention to the class-imbalance issue and more research work has been raging on this issue. The imbalance learning algorithms are introduced to manage skewed distribution of faulty (minority) class and non-faulty (majority) class in datasets to maximize the performance of prediction models. The imbalance learning algorithms may modify the dataset before learning or may modify learning algorithm itself to reduce the skewed dataset impact on the overall performance. As per the studies, the imbalance learning strategies are mainly grouped into three prevalent methods: data level strategies, algorithm level strategies, and ensemble learning strategies.

(1) Data level strategies [19–29]: The first and most preferable approach, with its simplicity, is a variation of the re-sampling (i.e. over-sampling or under-sampling) approach, whereby an imbalanced dataset with a skewed class distribution is transformed into a balanced dataset by wisely offering pseudo-instances (i.e. synthetic or artificial instances) into the faulty class. A huge amount of research work has been raging on this approach, such as Random Under Sampling (RUS) [9,30], Random Over Sampling (ROS) [19], Synthetic Minority Over-sampling Technique (SMOTE) [20], Borderline-Synthetic Minority Over-sampling Technique (BSMOTE) [21], to name a few. These data level approaches produce efficient outcomes with simple implementation, but the accurate outcome depends on the chosen problem and the classification algorithm.

(2) Algorithm level strategies [31–34]: These approaches directly alter the classification mechanism to cope with imbalanced data, such as cost-sensitive learning techniques and one-class learning techniques, to name a few. We are not sure to use appropriate algorithm on the selected dataset, so choosing an algorithm level approach varies based on the dataset.

(3) Ensemble learning strategies [35–40]: Another recommended approach outperforms the existing classifiers by working with a combination of multiple classifiers, namely bagging, boosting, SMOTEBoost, RAMOBoost, and AdaBoost.NC, to name a few. Combination of multiple classifiers means being able to learn and combine the strength of individual classifier.

1.1. Our contribution

Our contribution in this paper is to present a class-imbalance approach to reduce the percentage of class-imbalance between faulty and non-faulty instances and to maximize the performance of the constructed predictive model.

The proposed Weighted Average Centroid based Imbalance Learning (WACIL) approach is accustomed to build balanced datasets from imbalanced datasets through the introduction of pseudo-instances into the faulty class. Our proposed model is compared with six competitive approaches, namely ROS, SMOTE, BSMOTE, MAHAKIL, SOTB (Semi-supervised Oversampling Approach based on Trigonal Barycenter Theory) and Not over sampled data (NOS).

In order to maximize the strength of the faulty class, first of all, we perform filtered pseudo-instance generation with WACIL. Afterwards, KNN, LR, NB, SVM, DT and DNN classifiers are constructed on generated pseudo-data and measured the performance. The performance measures considered in this work are FOR, Recall, F-measure, AUC and G-mean. We conducted empirical experimentation to systemically demonstrate the performance of WACIL in comparison with competitive approaches using 24 imbalanced datasets from PROMISE and NASA projects. The empirical experimentation results evidence that WACIL enhances the diverse nature of the pseudo-instances (i.e., WACIL achieves lower FOR) and achieves better results with respect to overall performance assessment measures. In particular, for most of the projects the performance of WACIL with LR and DNN is better than other classifiers.

This paper is organized as follows. The motivation for our work is explained in Section 2. Related work is reviewed in Section 3. Section 4 describes our proposed approach. Experimental setup is explained in Section 5, experimental results are demonstrated in Section 6 and discussion of those results are presented in Section 7. Section 8 discusses the threats to the validity of our approach. The conclusion and future work suggestions are discussed in Section 9.

2. Motivation

The prominent methods are available to tackle class imbalance issues, including resampling strategies (data level), ensemble learning strategies and cost-sensitive learning strategies (algorithm level). However, our research uncovers various flaws and inadequacies in existing methodologies that could arise in a variety of situations. Throughout this section, we provide a thorough report of them.

The ROS is the earliest oversampling technique, which replicate faulty class instances and introduces them into the same class to construct a balanced training dataset [41]. The classifier constructed based on this redundant training data can produce good performance on training data and poor performance on test data (over fitting) [42].

SMOTE is one of the notorious synthetic sample generation algorithms. Unlike ROS, it introduces synthetic data into the faulty class [20]. Nevertheless, as per Benin et al. [24] the diversity of generated synthetic data is significantly not enough for classifiers to reduce biased results, and this algorithm increases the count of false positives. The reason is that the algorithm considers K-closest instances of each faulty instance to generate synthetic data, which might result in synthetic data being introduced into a very closed cluster of existing instances. It might reduce the diverse nature of synthetic instances and overfit the model. In addition, some of the non-faulty instances are misclassified as faulty. Faulty instances near the decision boundary between faulty and non-faulty classes are called borderline faulty instances. They might get some of the non-faulty instances in the K-closet instances set, so the generated synthetic data obtains similar features to those closest non-faulty instances. This may result in misclassification of borderline non-faulty instances as faulty class that can increase the false positive rate.

BSMOTE follows a similar procedure to SMOTE to generate synthetic data. In addition, it strengthens the border by considering borderline faulty class instances to generate synthetic data to overcome the problem of faulty instance misclassification (and to increase the recognition rate of faulty instances) [21]. However, as per Barua et al. [23], the improper way of borderline instance selection, BSMOTE generates inappropriate instances and raises the number of false positives.

Unlike SMOTE based sampling approaches, MAHAKIL [24] uses unique and dissimilar instances of faulty class to generate synthetic data. As a result, generated synthetic data does not form sub clusters inside the faulty class, but the generated instances may cross the faulty class boundary. These are known as noisy instances, which may impact the performance of the classification model. Furthermore, as per Liu [25], diversity of the synthetic faulty instances is not rich enough to reduce false positive outcomes.

The SOTB approach generates non-intersecting triangles, then computes synthetic data from those triangles. In the early iterations, the diverse nature of synthetic instances is assured, but non-intersecting triangles become very close in later rounds. The generated synthetic data may form sub clusters and become non-diverse in nature. The approach may produce a few undesirable instances during the process. Those noisy instances may affect the classification model's performance.

With a view to conquering the aforementioned disadvantages, we have proposed an oversampling approach called the WACIL method. WACIL can improve the diversity of generated data and works in the following way: (1) Our algorithm does not oversample the entire faulty class, instead it oversamples only borderline instances that can reduce overfitting. While collecting borderline instances, it considers cosine similarity rather than Euclidean distance because instances in close distance may not have similar kinds of features. (2) Our algorithm does not consider the K-closest instance set, instead it considers three partitions and shuffles them in all iterations and finds a weighted average of them to achieve diversity. (3) And our proposed algorithm employs a filtration process in order to exclude inappropriately generated instances, which reduces the number of false positives. The application of our proposed work is to convert the imbalanced dataset to a balanced dataset (for example, medical datasets, DNA microarray data and machinery faulty datasets). The proposed work can be applied to identify the software faulty modules in the early phases of software development process which can reduce testing time, control unwanted resource consumption in the software testing phase and reduce the effort required for software testing. Overall, it lowers the total budget required for software development.

3. Related work

A comprehensive survey work is presented over imbalance learning in SFP and other fault prediction models.

The class imbalance issue tackle by 3 approaches as explained in Section 1. Among them, we focused more on sampling strategies, especially over-sampling techniques. The cost-sensitive approach works by assigning miss-classification costs [32,43]. However, it is not clear how much it will cost. A lot of research has been done in the field of SFP through ensemble learning approaches [37,39,40,44]. But these methods consume more time for the construction of fault prediction models because they have to combine multiple models. However, sampling approaches also have problems associated with them, but we are trying to overcome them by introducing a noise filtered sampling technique.

To resolve imbalance issues, the data-level strategies resample the actual dataset and construct synthetic information to alter the faulty class distribution (over-sampling) or else exclude data from non-faulty classes (under-sampling) to prepare a balanced dataset. The data-level strategies have a benefit over other two strategies that they do not alter the classifier. For these reasons, we focused our research to data-level strategies.

RUS [9,30] eliminates instances from the non-faulty class of the datasets at random. However, the instances excluded from the non-faulty class might hold essential data for classification models; the usage of under-sampling is least frequent in SFP. ROS [19] randomly selects instances from faulty classes and reintroduces them into the same class, so that it can maximize the strength of the class. However, ROS does not provide any additional data to the predictive model, which may result in severe overgeneralization of the model. Chawla et al. [20] introduced SMOTE algorithm on the basis of ROS concept. It consists of two steps. In the first step, each faulty instance of a faulty class finds their K-closest neighbors, then it randomly selects one of the closest neighbors and computes a random synthetic instance with the selected neighbor in the second step, as in Eq. (1).

$$Inc_{new} = Inc_j + rand(0, 1) * (Inc_{jneighbor} - inc_j) \quad (1)$$

where, Inc_j is a randomly chosen original faulty instance, $Inc_{jneighbor}$ is one of the neighbors of Inc_j and Inc_{new} represents the generated synthetic instance. SMOTE yields more diversified instances than ROS since it creates synthetic instances by integrating them rather than duplicating them. During the past two decades a significant number of researchers have been motivated by the conception of SMOTE, particularly, Borderline-SMOTE [21], Adaptive Synthetic Sampling (ADASYN) approach [22], Majority Weighted Minority Oversampling Technique (MWMOTE) [23], Geometric SMOTE [45], Stable SMOTE [46] and SMOTE based on furthest neighbor algorithm (SMOTEFUNA) [47], to name a few. In general, most classifiers first learn the decision boundary between two classes in the training phase. The instances far away from the decision boundary train very well, but the instances very near the boundary misclassify the most. Han et al. [21] motivated by this reason, suggested Borderline-SMOTE; it selects near border faulty instances and then performs synthetic data generation on those near border samples as like SMOTE to maximize the recognition rate of near border faulty instances.

Unlike SMOTE Bennin et al. [24] suggested an oversampling technique called MAHAKIL. In the first stage, it calculates the Mahalanobis distance between the mean vector and all other instances and arranges them in ascending order according to their Mahalanobis distance. In the second stage, it makes two partitions of the data and generate synthetic data according to the chromosomal inheritance theory concept, it calculates the average of two equally indexed samples from each partition and considers them as synthetic data. The synthetic data generated by MAHAKIL has better diversity than the SMOTE based approaches and this can overcome the problem of overgeneralization.

Liu et al. [25] presented SOTB, it uses the concept of trigonal barycenter theory to generate synthetic instances. First, it finds the Mahalanobis distance between each instance and the mean of them, and then finds the farthest instance from the mean. Later, it finds Mahalanobis between that farthest instance and the other instances. Furthermore, it arranges instances in ascending order according to their distance. Then it forms non-intersecting triangles and takes an average of them as a synthetic data. The process goes on until the ratio becomes equal between the two classes. This way, it generates more diverse samples than the MAHAKIL.

In recent studies, Zhang et al. [26] presented an improved MAHAKIL based on K-means clustering to maximize the faulty class accuracy. Feng et al. [27], presented the Complexity-based OverSampling Technique (COSTE). COSTE uses differential evolution to assign complexity to each feature of instances, and then all the instances are arranged in ascending order based on the complexity measure. A further synthesizing step chooses the adjacent instances of the arranged set and takes the average of them to produce synthetic data for further computation.

Some other approaches to SFP have been studied. Saez et al. [48] presented SMOTE-Iterative Partitioning Filter (SMOTE-IPF), the oversampling process is same as the SMOTE. Later, it excludes the noisy and borderline samples from oversampled data.

Lina Gong et al. [49], presented the Cluster-based Over-sampling with noise filtering (KMFOS) technique to mitigate imbalanced class issues along with a noise filter. Firstly, the K-means algorithm is utilized to generate clusters. Then, from those clusters it generates synthetic data. Finally, KMFOS introduced Closest List Noise Identification (CLNI) techniques to filter out noise instances.

Faruk et al. [50] proposed a different classification model for SFP. It is a combination of an Artificial bee colony (ABC) algorithm and ANN model. ABC was utilized to optimize the weights of the ANN model while training, and the cost was assigned to misclassified classes. At present, deep learning is gaining popularity and recognition from SFP. Zhao et al. [13] introduced a novel method for assessing code functional similarity called DeepSim. They created a semantic representation by encoding a matrix based on data and code control flow, then employed a DNN model to extract features from the matrix and perform fault prediction classification. Qiao et al. analyzed the performance of imbalanced software datasets through deep learning. They employed two variants of deep learning models (multi layer perceptron and convolution neural networks) over four NASA datasets. Through manipulation of parameters, they observed better performance with the ReLU activation function [14].

4. Methodology

4.1. Overview of WACIL

Motivated by the issues discussed in the preceding section, as a part of this research, we devised a sampling approach, namely the Weighted Average Centroid based Imbalance Learning approach (WACIL). The proposed oversampling algorithm comprises two primary stages: the first stage is the selection of hard-to-classify (borderline) instances from the faulty class (minority class). The second stage is the introduction of filtered pseudo-instances into the faulty class. The two stages are repeated until the training data becomes balanced. The comprehensive description of two stages is listed in the following subsections.

4.2. Extraction of borderline instances

Many traditional classifiers strive to extract and learn decision boundary in order to achieve better performance. Because instances which are away from the decision borderline might classify accurately, on the contrary, instances that are near and on the decision borderline have a higher likelihood of being misclassified. We can call them hard-to-classify instances. Therefore, it is important to generate pseudo-instances of hard-to-classify faulty instances to build a better classifier.

In our proposed approach (Shown in Algorithm 1), we designed a way to extract important hard-to-classify borderline instances of faulty class and construct the borderline faulty instance set, FS_{binc}.

Suppose that the whole training set is T, the faulty class set is FS and the non-faulty class set is NFS, and the whole process of extracting borderline instances in set FS_{binc} (steps 1–13 of Algorithm 1) can be described as follows:

We have experimented with projects of high imbalance and moderate imbalance rates from the PROMISE and NASA datasets for different K (closest neighbors) values. The experimental results indicate that K = 5 produces better performance for both highly and moderately imbalanced projects, so we consider K = 5 for all the experiments.

1. The foremost step in our approach is the identification of the closest neighbors of each and every instance CN(Inc_i) of faulty modules in FS and NFS, differently, according to the cosine similarity measure by considering K = 5. For each instance (Inc_i), computes the sum of the distances of the closest neighbors from NFS and FS and store those distances in sets D_{NFS} and D_{FS}, respectively. For Inc_i, if sum of the nearest neighbors distance ($dist_{Inci}$) $\in D_{FS}$ is greater than $dist_{Inci} \in D_{NFS}$ then append Inc_i to a new temporary set (FS_{tmp}), else the instance considered as a noise instance and not added to FS_{tmp}.
2. The further step is, for each $Inc_i \in FS_{tmp}$, identifying the closest neighbors CN (Inc_i) in the total original training dataset according to the cosine similarity measure by considering K = 5. Make a set of union of all the closest neighbors from NFS in NFS_{tmp}, for which Inc_i the closest neighbors from NFS are greater than or equal to three. Here, three or more out of the five closest neighbors are from the non-faulty class (NFS) which share more boundaries with the non-faulty class. Therefore, we have considered the count to be greater than or equal to three.
3. Now the conclusive move of extraction of borderline instances, for each $Inc_i \in NFS_{tmp}$, identifies the closest neighbors CN(Inc_i) in FS according to the cosine similarity measure by considering K = 5. Take a set of union of all the closest neighbors in FS_{binc}. The borderline instances in set FS_{binc} are used to generate filtered pseudo-instances of the faulty class.

4.3. Weighted average centroid based pseudo-instance generation

The aim of this stage is to balance the dataset through diverse pseudo-instances introduction in faulty class. Once the borderline instances of faulty modules have been extracted, Mahalanobis distance computation and the filtered pseudo-instance generation are employed. This stage explains how the pseudo faulty instances are generated and how many should be generated to make dataset balance.

4.3.1. Mahalanobis distance computation

The first and foremost step is determining the distance of extracted borderline instances of faulty class from their mean. The Euclidean distance has been a chart-topping measure for decades [51], for determining the distance between two points, but it is diplomatic towards highly correlated features and units of features. For example, if only one feature of two instances holds more difference, then the Euclidean distance between them is very high, irrespective of other less difference features. This can misdirect the results. For this reason, Mahalanobis distance (MD) considered in this study as a distance measure. In the Algorithm 1, steps 17 to 19 interpret mathematical computation of Mahalanobis distance.

MD can be defined as a distance measure between point x_i and Distribution μ. MD is calculated as follows in Eq. (2),

$$\mathrm{MD}(x_i, \mu) = \sqrt{((x_i - \mu)^T \times COV^{-1} \times (x_i - \mu))} \quad (2)$$

where $X = \{x1, x2, x3, \ldots, xn\}_{n\times m}$, COV^{-1} is the inverse covariance matrix and $\mu = \{\mu 1, \mu 2, \mu 3 \ldots, \mu m\}_{m\times 1}$. μ is the mean distribution of all features, $\mu 1$ is mean of first feature, $\mu 2$ is of second feature, …., and μm is of mth feature, respectively. The 'm' represents the total number of features, 'n' represents the total number of instances in dataset and x_i is an instance having m- features.

for example, consider below given Table 1 as example dataset $X_{4\times 3}$.

The mean distribution of all features $\mu = \{66, 642.5, 44\}_{3\times 1}$

The covariance matrix of the dataset is $COV = \begin{bmatrix} 52.67 & -90 & -46.33 \\ -90 & 2158.33 & 80 \\ -46.33 & 80 & 71.33 \end{bmatrix}_{3\times 3}$

The inverse covariance matrix is $COV^{-1} = \begin{bmatrix} 4.57200858e{-}02 & 8.40694623e{-}04 & 2.87538561e{-}02 \\ 8.40694623e{-}04 & 4.98874111e{-}04 & -1.34263269e{-}05 \\ 2.87538561e{-}02 & -1.34263269e{-}05 & 3.27103193e{-}02 \end{bmatrix}_{3\times 3}$

Table 1
Example dataset.

S. no	feature1	feature2	feature3
1	66	660	36
2	69	690	40
3	56	640	55
4	73	580	39

Consider one instance $y = \{63, 700, 37\}$ to find Mahalanobis distance from mean. According to Eq. (2) $y - \mu = \{-3, 57.5, -7\}_{3\times1}$

$$\mathrm{MD}(y, \mu) = \sqrt{((y-\mu)^T \times COV^{-1} \times (y-\mu))} = 2.14$$

A few datasets produce singular covariance matrices. We cannot generate the inverse of singular covariance matrices. In such circumstances, we computed Moore–Penrose inverse/generalized inverse [52] of a covariance matrix for Mahalanobis distance calculation.

Algorithm 1 WACIL Algorithm

Require: Imbalanced training dataset (**T**) and K
Ensure: Balanced training dataset (**T′**)
1: Make a partition of **T** in to faulty class set (**FS**) and non-faulty class set (**NFS**)
2: For each faulty instance $Inc_i \in$ **FS**, compute K closest instances in (**FS**) and store the sum of distance of all K neighbors in $\mathbf{D_{FS}}$ according to cosine similarity measure.
3: In similar way, for each faulty instance $Inc_i \in$ **FS**, compute K closest instances in **NFS** and store the sum of distance of all K neighbors in $\mathbf{D_{NFS}}$ according to cosine similarity measure.
4: Initialize an empty set FS_{tmp}
5: **if** $dist_{Inci} \in \mathbf{D_{FS}} > dist_{Inci} \in \mathbf{D_{NFS}}$ **then**
6: FS_{tmp}.append(Inc_i)
7: **end if**
8: Initialize an empty set NFS_{tmp}
9: For each instance $Inc_i \in FS_{tmp}$, compute K closest instances ($CN(Inc_i)$ in **T**
10: **if** count of $CN(Inc_i) \in$ **NFS** ≥ 3 **then**
11: Take union of those closest instances belongs to **NFS** in NFS_{tmp}
12: **end if**
13: For each instance $Inc_i \in NFS_{tmp}$, compute K closest instances in **FS** , then take the union of all these closest instances in FS_{binc}
14: Compute the number of Pseudo-instances (N) to be generated, $N = Nn - Np$
15: Initialize N_{cnt}, to keep track of generated pseudo-instances
16: **while** $N_{cnt} < N$ **do**
17: Compute Mahalanobis distance (MD) between $\mathbf{FS}_{binc}$ and mean μ using Eq. (2)
18: Arrange $\mathbf{FS}_{binc}$ in ascending order with respect to its Mahalanobis distance
19: Initialize an empty set Inc_{pseudo}
20: Initialize n with $arraylength(\mathbf{FS}_{binc})$
21: Create three partitions from ($\mathbf{FS}_{binc}$)
22: Compute each partition length, j=n/3
23: Case_1 = $\{\mathbf{FS}_{binc(1)}, \mathbf{FS}_{binc(2)}, \ldots \mathbf{FS}_{binc(j)}\}$, Case_2 = $\{\mathbf{FS}_{binc(j+1)}, \mathbf{FS}_{binc(j+2)}, \ldots \mathbf{FS}_{binc(2j)}\}$ and Case_3 = $\{\mathbf{FS}_{binc(2j+1)}, \mathbf{FS}_{binc(2j+2)}, \ldots \mathbf{FS}_{binc(3j)}\}$
24: **for** $i = 1, 2 \ldots\ldots j$ **do**
25: Inc1=Case_1[i], Inc2=Case_2[i] and Inc3=Case_3[i]
26: **if** $N_{cnt} < N$ **then**
27: $S \leftarrow min(Inc1, Inc2, Inc3) + rand(0, 1) \times (max(Inc1, Inc2, Inc3) - min(Inc1, Inc2, Inc3))$
28: Compute K closest instance of S in **T**
29: Calculate the number of non-faulty instances ($n1$) and the number of faulty instances ($n0$) in K-closest neighbors
30: **if** $n0 > n1$ and $n0 \neq K$ **then**
31: $Inc_{pseudo} \leftarrow Inc_{pseudo} + S$
32: $N_{cnt} \leftarrow N_{cnt} + 1$
33: **end if**
34: **end if**
35: **end for**
36: $\mathbf{FS}_{binc} = \mathbf{FS}_{binc} + Inc_{pseudo}$
37: **end while**
38: $\mathbf{T}' = Union(NFS, FS, FS_{binc})$;
39: return **T′**;

4.3.2. Filtered pseudo-instance generation

Suppose the count of faulty modules is Np and non-faulty modules is Nn, So the number of faulty pseudo instances that have to be generated is N = Nn−Np. The set of borderline instances are arranged in ascending order with respect to their Mahalanobis distance and then divided into three equal cases or sets, followed by pseudo-instances generated. Particularly, for few datasets the number of faulty modules that occur close to the boundary is extremely low, so the algorithm functions appropriately through three partitions. Instances are sorted in ascending order, so Case_1 contains samples which are too close to the mean distribution, Case_2 contains fair distanced samples and Case_3 contains far distanced samples.

After partition, in order to cope with the problem of nearest sample synthesis, we shuffle the instances of all three cases. The following step is the tagging process: In all three cases, instances are tagged with the same labels, sequentially. Case_2 and Case_3 pursue a consistent pattern of Case_1, which signifies case 2 and 3 instances tagged with similar labels as Case_1.

Tag_set = $\{tag_1, tag_2, \ldots tag_j\}$, this Tag set is same for all three cases for all instances, sequentially. Where j = n/3 and n is the total number of instances.

The generated faulty modules from these three partitions are filtered and includes in the training set. For one combination select instances with tag1 from all cases (inc_1, inc_{j+1} and inc_{2j+1}), for the next combination select instances with tag2 from all cases and repeat similar type combination selection for the remaining instances. It takes the weighted average of the selected combination of all three instances.

Suppose, Inc1∈Case_1, Inc2∈Case_2 and Inc3∈Case_3 having same tag.

Inc1 = $\{inc_{11}, inc_{12}, \ldots inc_{1m}\}$
Inc2 = $\{inc_{21}, inc_{22}, \ldots inc_{2m}\}$
Inc3 = $\{inc_{31}, inc_{32}, \ldots inc_{3m}\}$. Where 'm' represents the number of features present in training dataset. Unlike existing techniques, the minimum and maximum values belong to different instances, thus the generated pseudo faulty module resides in between them and shares similar features with all of these modules. The pseudo instances are generated using the following equations:

$$\begin{aligned} Min &= [Minimum(Inc1, Inc2, Inc3)]_{1\times m} \\ Max &= [Maximum(Inc1, Inc2, Inc3)]_{1\times m} \\ rand &= [random(0, 1)]_{1\times m} \\ New\_inc &= Min + rand(0, 1) \times (Max - Min) \end{aligned} \tag{3}$$

As we know, after generating artificial faulty modules, there could be some undesirable or redundant instances generated during implementation. These instances can influence the performance of the predictive model. They often occur in the space of the non-faulty class, share many more boundaries with the non-faulty class or are completely redundant with the actual faulty module. In order to find out these undesirable instances, a cosine similarity measure is employed. Firstly, we apply cosine similarity to get the closest neighbors of a pseudo-instance by considering K = 5. Then, take a different count of the nearest neighbors for faulty modules and non-faulty modules, respectively. If the number of faulty modules less than the number of non-faulty modules, then that faulty instance is considered as an undesirable instance. Else the number of faulty modules is equal to K, then that pseudo-instance is considered as an redundant instance. These instances should be excluded from the pseudo-instance set.

Repeat the above explained two stage process until the generated filtered pseudo-instances count equals N, which means the ratio of faulty and non-faulty modules becomes 1:1. Through these two stages, the training dataset is constructed. For example,

Table 2
Summary of PROMISE and NASA project datasets.

Project	Datasets	#Total instances	# faulty instances (%fi)	#Metrics	Project	Datasets	#Total instances	# faulty instances (%fi)	#Metrics
PROMISE	ant-1.7	745	166(22.28%)	20	PROMISE	xalan-2.4	723	110(15.21%)	20
	arc	234	27(11.54%)	20		xerces-1.3	453	69(15.23%)	20
	camel-1.6	965	188(19.48%)	20	NASA	CM1	327	42(12.84%)	37
	ivy-2.0	352	40(11.36%)	20		MW1	253	27(10.67%)	37
	jedit-4.2	367	48(13.10%)	20		KC3	194	36(18.56%)	39
	log4j-1.0	135	34(25.19%)	20		MC1	1988	46(2.31%)	38
	lucene-2.0	195	91(46.67%)	20		MC2	125	44(35.20%)	39
	poi-2.0	314	37(11.78%)	20		PC1	705	61(8.65%)	37
	redaktor	176	27(15.34%)	20		PC2	745	16(2.15%)	36
	synapse-1.2	256	86(13.85%)	20		PC3	1077	134(12.44%)	37
	tomcat	858	77(8.97%)	20		PC4	1287	177(13.75%)	37
	velocity-1.6	229	78(34.06%)	20		PC5	1711	471(27.53%)	38

assume a dataset with Nn = 680, Np = 70, and N becomes 610 (680 − 70 = 610). Our proposed algorithm iterates (steps 16–37 of Algorithm 1) until the number of pseudo-faulty class samples reaches 610. At the completion of all iterations, the ratio of both classes turns to 1:1 and the dataset becomes balanced. Further, prediction models built on this training dataset can learn parameters equally well with both classes, do not give biassed results, and do not over generalize the model.

5. Experimental setup

In order to show the performance of our proposed approach, we performed experimentation and answered the following three research questions:

RQ1: Does WACIL contribute to the diverse nature of generated pseudo-instances of faulty class data?

RQ2: How effectively does WACIL tackle the class imbalance problem?

RQ3: How does WACIL's overall performance compare with the state-of-the-art strategies?

In this Section, we reported the overall summary of the datasets and a detailed description of the model performance assessment measures used to evaluate our simulation work. Next, a brief explanation of baseline methods, classifier selection, experimental framework design and statistical comparison measures of WACIL with baseline methods are shown.

5.1. Experimental objects

With the aim of evaluating our proposed method, we experimented with a total of 24 different datasets, among them, 14 extracted from PROMISE projects and 10 from NASA projects [53], respectively. The detailed information of datasets are shown in Table 2.

All the datasets are not balanced. That means there is a big difference between the percentage of faulty instances and non-faulty instances in each dataset. The PROMISE project datasets imbalance rate varies from 8.97% to 46.67% and the NASA project datasets varies from 2.15% to 35.2%. These differences are clearly shown in the Table 2 in terms of number and percentage of faulty instances (%fi).

5.2. Performance assessment measures

Our task is binary classification of modules, which means predicting whether a module is faulty or not. Table 3 illustrates the confusion matrix for binary classification task. In the datasets, faulty modules are labeled as positive and non-faulty modules as negative. Observe the measures TP, FP, TN and FN. The TP cell contains true positive results; it can be defined as the number of faulty modules identified as faulty. The TN cell contains true negative results; it can be defined as the number of non-faulty modules identified as non-faulty. The FP cell contains false positive results; it can be defined as the number of non-faulty modules identified as faulty modules. The FN cell contains false negative results; it can be defined as the number of faulty modules identified as non-faulty modules.

Table 3
Confusion matrix.

	Predicted: Positive (faulty)	Predicted: Negative (non-faulty)
Actual: Positive (faulty)	TP	TN
Actual: Negative (non-faulty)	FP	FN

The performance assessment measures considered in our work are defined as follows:

$$Recall = \frac{TP}{TP + TN} \tag{4}$$

$$FOR = \frac{FP}{FP + TN} \tag{5}$$

$$F - measure = \frac{2TP}{2TP + FP + FN} \tag{6}$$

$$G - mean = \sqrt{\frac{TP}{TP + FP} * \frac{TP}{TP + FN}} \tag{7}$$

In our simulation work, our experimental SFP datasets are imbalanced in nature. Hence, we should go with multiple assessment measures.

Fall Out Rate (FOR), also called False Positive Rate (FPR), is a single class measure [24,54]. It can be defined in a way where FOR is the measure of modules belonging to a non-faulty class that are incorrectly classified as faulty class modules.

Recall, also called True Positive Rate (TPR), is also a single class measure. It can be defined in a way where recall is the measure of modules belonging to a faulty class are correctly classified as faulty class modules.

Unlike FOR and Recall, the micro averaged F-measure is used to measure the overall performance [28,54]. It is a harmonic mean of two assessment measures, recall and precision.

Here, we considered two more measures to assess the overall performance of prediction models; those are the AUC score and the G-mean score. AUC is a balanced measure [55,56]. It is a trade-off between TPR and FPR. The G-mean incorporates the classifier's accuracy into faulty and non-faulty class modules [57]. A high value of F-measure along with high AUC score indicates that the comprehensive performance of the predictive model is good and high G-mean also a good indication of model's comprehensive performance.

5.3. Baseline methods

In order to answer the research questions RQ1, RQ2 and RQ3, we validated the outcome of WACIL and related it with ROS, SMOTE, BSMOTE, MAHAKIL and SOTB, and also with the Not Oversampled (NOS) original dataset. A brief overview of these techniques is as follows:

ROS: It is a basic imbalance learning approach, simply replicate the instance of faulty class to maximize the class instances distribution, but the predictive model may face the problem of severe over-generalization [19].

SMOTE: It is the most conventional imbalance learning algorithm proposed by Chawla et al. [20]. The main motive is the formation of synthetic instances within the faulty class. The first step is the generation of the k-closest neighbors of a faulty instance, and the next step is the generation of a random instance between that particular faulty instance and its nearest neighbors.

Borderline-SMOTE: It is a modified version of SMOTE proposed by Han et al. [21]. BSMOTE divides the faulty class into three categories such as safe, danger and noisy. The instances which are on the border of both classes are misclassified most of the time, which means they are harder to classify compared to other instances. These instances are stored in the danger set and on this set the SMOTE technique is applied to generate synthetic instances.

MAHAKIL: It employs Mahalanobis distance to find the order of instances in faulty class [24]. Later, it divides the ordered set into two partitions, then according to chromosomal inheritance theory concept these two partition samples generates synthetic data. The same procedure is followed by using previously generated synthetic data to produce some more synthetic instances until the required ratio reached between faulty and non-faulty classes.

SOTB: It uses the concept of trigonal barycenter theory to generate synthetic instances of faulty class [25]. First, it finds the Mahalanobis distance between farthest instance and rest of the instances. Later, arranges instances in ascending order according to their distance and calculates synthetic data from non-intersecting triangles of arranged set. Next steps utilizes previously generated instances to generate some more synthetic data. The process goes on until the ratio becomes equal between the two classes.

5.4. Classifier selection

In our experimental work, we have considered six classifiers to assess the performance of models. Among them, five are conventional machine learning models, namely KNN [9], LR [6,7], NB [10, 11], SVM [8] and DT [12], and the other model is DNN [14,15]. These classifiers are implemented using scikit-learn package in Python.

The fully connected DNN model is implemented in Python using Keras and it uses Tensorflow as the back end. The DNN is modeled in the following manner: The number of input neurons varies on the basis of the number of features of each dataset (i.e., 20 to 38). The number of hidden layers are three and the number of neurons in a particular hidden layer is chosen over a set (6, 8, 12 and 16) on the basis of dataset performance requirement. Finally, the output layer contains only one neuron because our task is binary classification. The conventional way of optimizing parameters is called *Grid search* or a *parameter sweep*. It is one of the best search for hyper-parameter tuning. In order to achieve the best performance, the following batch size, # training epochs, activation function, optimizer, learning rate, and dropout rate are tuned through grid search:

Batch size: [80, 100, 120, 160, 180, 200],
Training epochs: [10, 15,20,25,30,35,40],
Activation function: [relu, tanh, softmax, sigmoid, linear],
Optimizer: [RMSprop, SGD, adam, Nadam, Adamax],
Learning rate: [0.001, 0.01, 0.1, 0.2, 0.3] and
Dropout rate: [0.0, 0.1, 0.2, 0.3, 0.4, 0.5]

5.5. Experimental design

To acquire extensive knowledge and also to notice the effectiveness of the proposed model, we conducted experiments on 24 diverse imbalanced datasets selected from PROMISE and NASA projects. Conventional classifiers like KNN, LR, NB, SVM and DT, along with DNN, are used as classification models. Due to the fact that selected datasets are imbalanced in nature, the proposed imbalance learning model makes the dataset into a balanced one by introducing pseudo-instances into the faulty class to make the ratio 1:1 with the non-faulty class, then observe the outcome and relate it to diverse sampling techniques namely ROS, SMOTE, BSMOTE, MAHAKIL, SOTB and also with the not oversampled dataset. ROS, SMOTE and BSMOTE are implemented using the imblearn library available in Python. MAHAKIL and SOTB are implemented as per the algorithm given in [24,25]. The experimental framework is depicted in the Fig. 1. According to empirical research [24,25,58], assigning 30%–20% of the dataset to testing and the remaining 70%–80% to training yields the greatest outcomes. We have experimented with both splits, 70/30% (training set/testing set) and 80/20% (training set/testing set) to access the performance of the model. The experimental results indicate that a 70% training set and a 30% testing set deliver the best performance. In order to evaluate the data accurately, we split the data into 70% training and 30% testing sets. The oversampling techniques were performed on the training set, so that the training process happens equally well on both classes. After that, on the training set, we perform K-Fold (10-Fold) cross-validation to conquer issues such as selection bias or overfitting, as well as to provide insight into how the model can extrapolate to an unknown dataset. To reduce the impact of randomness, the average performance measures of 10 iterations are noted as the final measures.

Based on the experimental FOR outcome, we can answer RQ1 whether WACIL can diversely spread the generated pseudo-instances or not. To answer RQ2, we compared all the competitive imbalance learning strategies with WACIL over recall and FOR. Lower FOR and higher Recall outcomes indicate how effectively imbalanced data is handled. To answer RQ3, we compared all the competitive imbalance learning strategies with WACIL over F-measure, AUC and G-mean. The higher the values, better the overall performance of a particular approach. If WACIL achieves the better results over all these measures, we can conclude the superiority of our approach.

5.6. Statistical measures

In contemplation of measuring the statistical performance differences, we have elected the Wilcoxon Signed Rank (WSR) test [47,59]. On the basis of statistical analysis, the WSR test does not expect the analyzed results to follow a known distribution.

For example, assume there are n predicted outputs of one of the performance metrics of two approaches. Let us denote $P = \{p1, p2, p3, \ldots, pn\}$, $Q = \{q1, q2, q3, \ldots, qn\}$, with respectively each method and difference values as $Dif = \{p1-q1, p2-q2, p3-q3, \ldots, pn-qn\}$. The Dif ranked according to its absolute differences (rank will be assigned to $abs(Dif)$), and zero differences are excluded for further analysis. Let W_+ is the representation of sum of positive ranks, W_- is the representation of the sum of negative

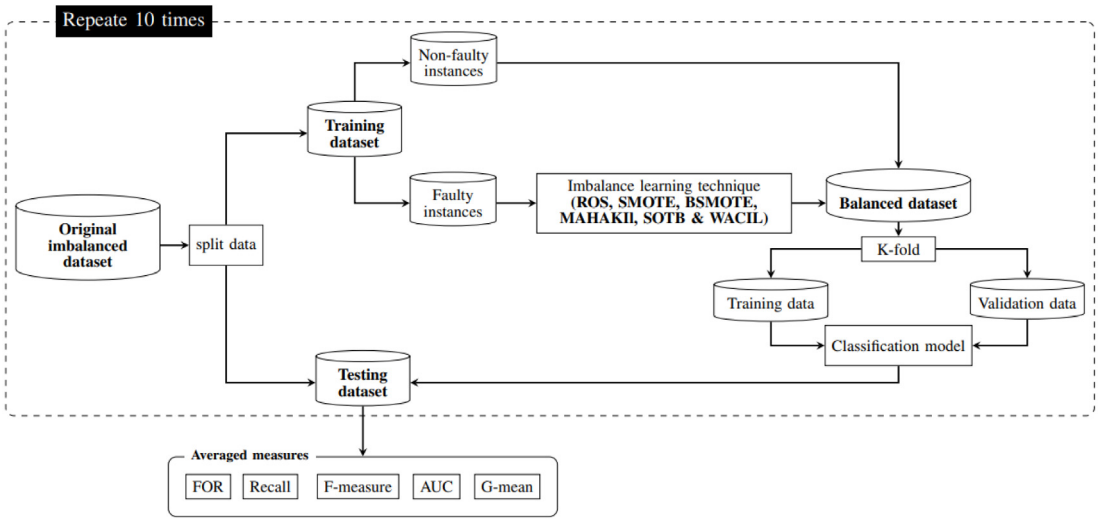

Fig. 1. Experimental framework.

ranks, and now 'n' is the count of non-zero differences. W_+ and W_- are represented as below in Eqs. (8) and (9),

$$W_+ = \sum_{i=0}^{n} \text{Rank}(\text{Abs}(Dif_i)), \quad \text{where } Dif > 0 \tag{8}$$

$$W_- = \sum_{i=0}^{n} \text{Rank}(Abs(Dif_i)), \quad \text{where } Dif < 0 \tag{9}$$

The WSR is used for statistical hypothesis testing using *P*-value. Assumption of the null hypothesis (H0) is the results obtained from the two approaches are statistically relative (similar). At a confidence level of 95% WSR test assumes H0 is true. If the *P*-value is greater than the significance level α ($\alpha = 0.05$), then we fail to reject H0, such as observed statistical evidence is inadequate to reject it, else there is a considerable difference in rejecting the default hypothesis H0. Nevertheless, the *P*-value does not intimate about the strength of the relationship. The matched-pairs rank biserial correlation coefficient (W_c) can be used to measure effect-size for the WSR test [60–62]. The Eq. (10) is used to represent W_c:

$$\mathrm{W}_c = \frac{4\left|T - \left(\frac{W_+ + W_-}{2}\right)\right|}{n(n+1)} \tag{10}$$

where, T is the minimum of W_+ and W_- and n is the count of non-zero difference samples. The effect-size can be rendered using three labels [47], small ($\mathrm{W}_c \leq 0.1$), medium ($0.1 < \mathrm{W}_c < 0.5$) and large ($\mathrm{W}_c \geq 0.5$).

Furthermore, win-draw-loss statistics [63] are adopted to compare the overall performance of each predictive model with all other predictive models (classification models * imbalance learning approaches-1), with respect to all performance assessment measures. Win-draw-loss statistics utilize the WSR test to compare predictive models and increment counters according to *P*-value. If *P*-value is greater than the α then both models win counter increases by one, else win counter increases for greater rank sum model and loss counter increases for lesser rank sum model. The rank sum is the sum of ranks of particular model.

6. Experimental results

6.1. Overall results

In this section, we have reported the experimental results to answer the research questions. We have computed WACIL's performance and compare it to competitive approaches. Figs. 2 and 3 represent the results of PROMISE and NASA projects respectively for individual classifiers over all the measures in box plots. Furthermore, Fig. 4 shows the distribution of corresponding average (average of all classifier results) results across 24 datasets. The little red circle in each box represents the mean value of each method. According to results we can conclude the following:

(1) Across all datasets and classification models, NOS and WACIL outperform all the other methods through the best FOR values with mean of 0.088 and 0.131, respectively. While ROS (0.198), SMOTE (0.206) and BSMOTE (0.199) offer the worse FOR values. MAHAKIL (0.163) and SOTB (0.162) produce moderately better results than those methods.

(2) In terms of Recall, WACIL with DNN and DT produce better recall values than other sampling approaches. Although the average recall values of WACIL (0.48) are not always the best, our approach can archive a similar distribution of Recall results to MAHAKIL (0.483) and SOTB (0.489). While NOS (0.286) produces the lowest recall values, BSMOTE (0.528) and SMOTE (0.541) produce the best.

(3) For almost all datasets and all the classification models, NOS and WACIL outperform all the competitive approaches over F-measure with highest means of 0.806 and 0.811, respectively. While ROS (0.758), SMOTE (0.752) and BSMOTE (0.760) produce the worse outcomes. MAHAKIL (0.781) and SOTB (0.785) produce moderately better values.

(4) For the vast majority of datasets, the WACIL technique outperforms the other compared approaches in terms of AUC with average of 0.747. This implies that the model's ability to discriminate between faulty and non-faulty classes is better, while other models mean results are 0.711, 0.728, 0,726, 0.726, 0.729, 0.721 for NOS, ROS, SMOTE, BSMOTE, MAHAKIL and SOTB, respectively.

(5) WACIL outperforms all the competitive approaches over PROMISE datasets with respect to G-mean outcome. However, over NASA datasets, WACIL with DNN and DT gives comparable results to the competitive methods. On KNN, LR and NB, SMOTE performs better, and on SVM, ROS produces moderately

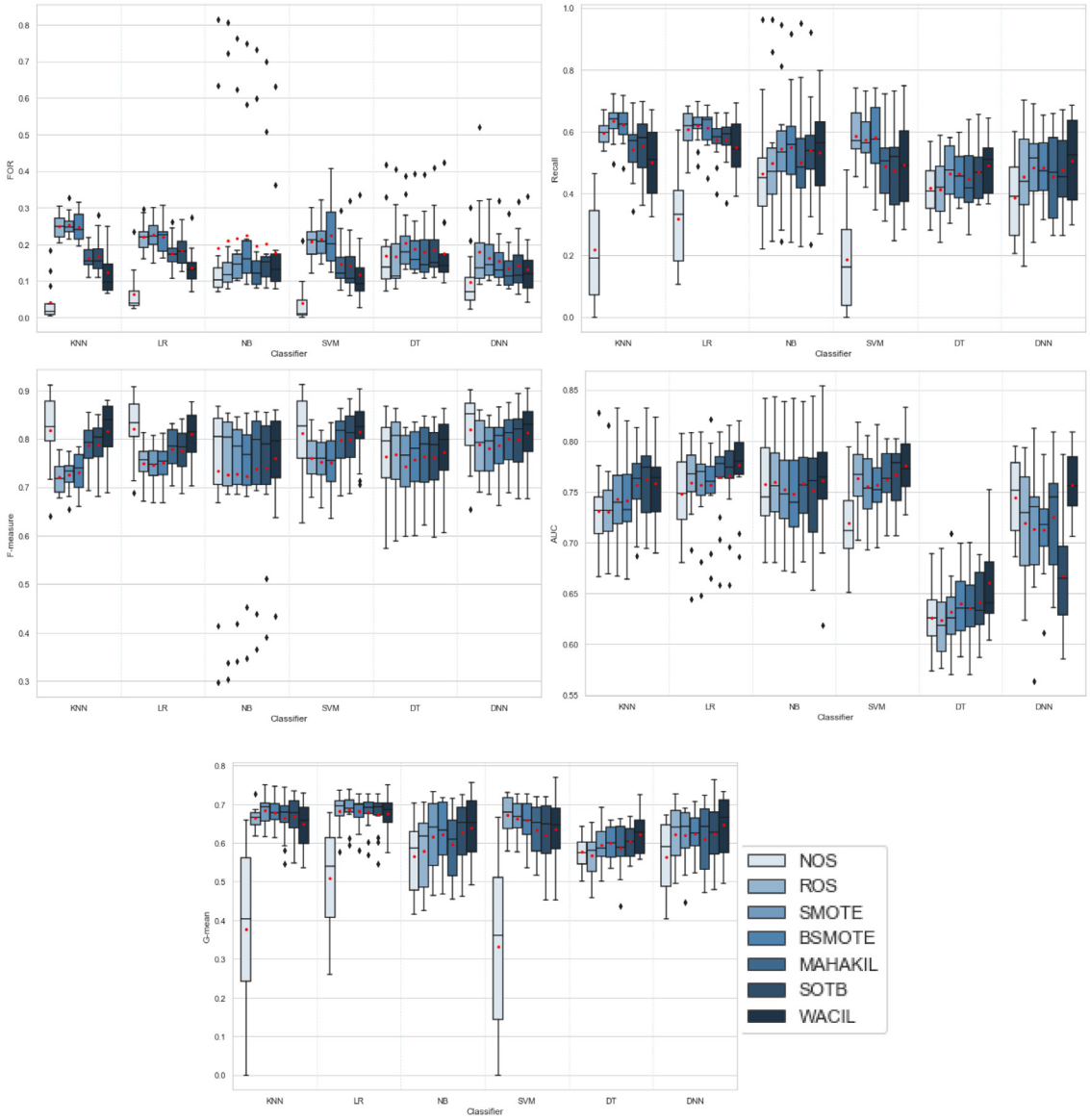

Fig. 2. Boxplots of six imbalance learning techniques and NOS on six classifiers over 14 PROMISE project datasets in terms of FOR, Recall, F-measure, AUC and G-mean.

better G-mean values. On the average results of both project datasets WACIL (0.609) gives comparable results to other models. While other models mean results are 0.436, 0.622, 0.634, 0.627, 0.611, 0.617 for NOS, ROS, SMOTE, BSMOTE, MAHAKIL and SOTB, respectively.

As per Benin [24], the diverse nature of generated pseudo-instances can be defined by FOR values. The lower the FOR, the greater the diversity. We found that the WACIL approach achieves a lower FOR than the other oversampling techniques. As a result, we can answer the **RQ1**, yes, WACIL contribute towards diversity.

The predictive model built on the SMOTE and BSMOTE generated training datasets resulted in higher mean Recall and higher mean FOR. Furthermore, MAHAKIL and SOTB are better alternatives to ROS, SMOTE, and BSMOTE to generate diverse instances. Over DT and DNN, WACIL results has higher mean Recall over all the datasets than those of other sampling approaches. The other classification models with WACIL gets comparable results over PROMISE datasets and fewer steps away from the competitive approaches over NASA datasets. The fundamental reason is that various data has various distributions, and it is possible that the data distribution is only acceptable for certain classifiers. As a result, we can respond to **RQ2**. WACIL tackle imbalanced data by introducing pseudo-instance with lower FOR values and with comparable Recall values.

In accordance with FOR and Recall, one cannot justify particular approach being superior to others. Therefore, to assess the comprehensive performance, we consider F-measure, AUC and G-mean. Greater these values, the better the overall performance. Therefore, we compared these measures between different imbalance learning techniques to answer **RQ3**. Figs. 2, 3 and 4 demonstrate that WACIL outperforms all other approaches in terms of AUC and in terms of F-measure, outperforms all other approaches except NOS. In terms of G-mean, WACIL gives a decent output. Hence, we can conclude that the comprehensive performance of WACIL is superior to other methods.

6.2. Statistical performance comparison

To further examine the outcomes as well as to determine if there is any statistically significant difference between the proposed approach and the other methods, a WSR test is applied on the WACIL outcomes versus each other method.

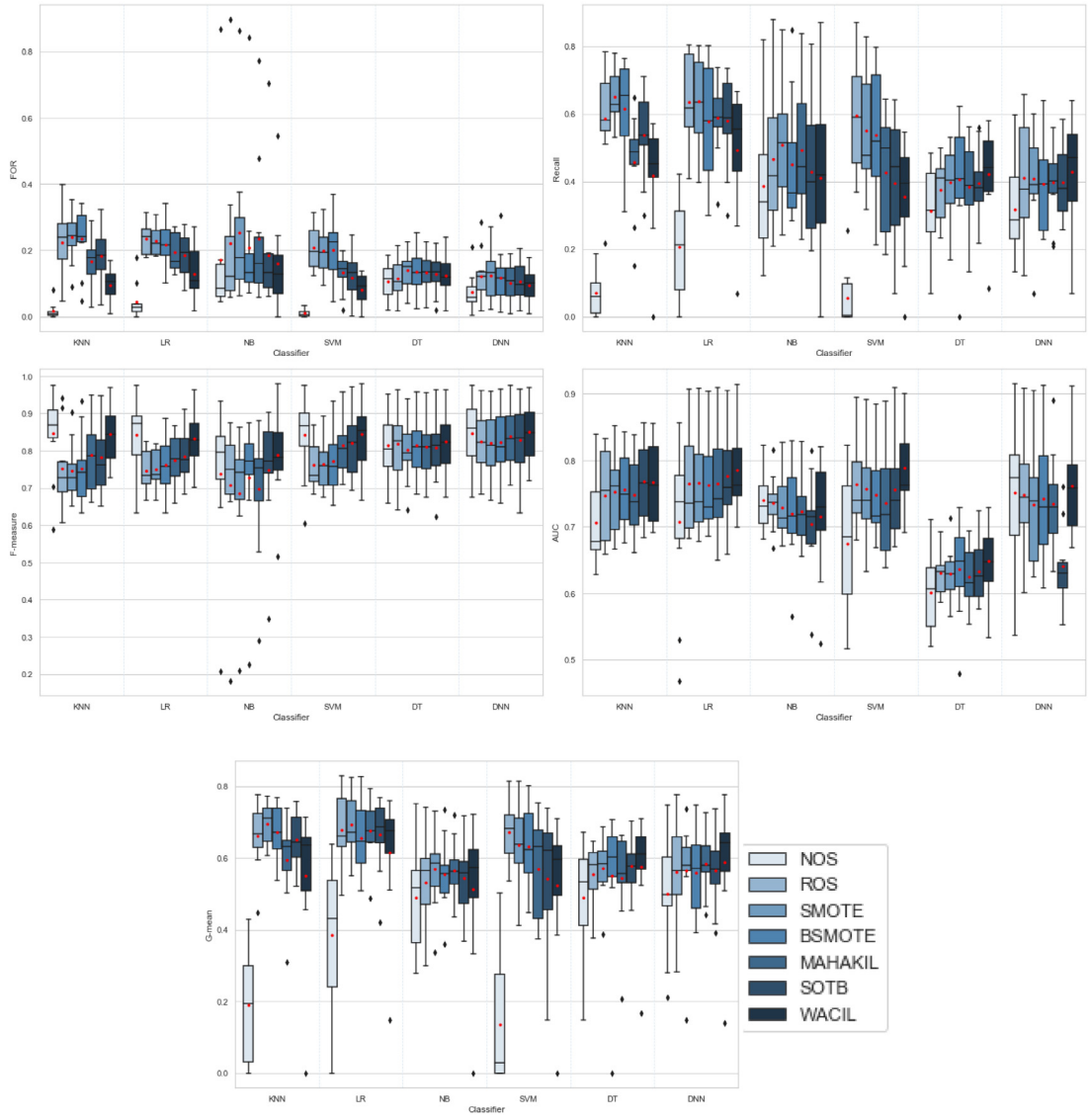

Fig. 3. Boxplots of six imbalance learning techniques and NOS on six classifiers over 10 NASA project datasets in terms of FOR, Recall, F-measure, AUC and G-mean.

Tables 4 and 5 show the WSR test results over AUC for WACIL vs. SMOTE and WACIL vs. SOTB on the LR classification model, respectively. Where ranks are assigned to absolute differences, the sign of differences is multiplied to ranks and computed into positive and negative rank sums via Eqs. (8) and (9). The samples which produce zero difference are not considered for further ranking process. And also, the effect size is computed on the basis of these results via Eq. (10). The WSR test P-value [59,64], computed for WACIL vs SMOTE (see Table 4) is 0.002670949, less than 0.05, so we have evidence to reject the null hypothesis at a significance of 0.05. Both WACIL and SMOTE are statistically different and one can conclude that WACIL can statistically outperform SMOTE ($W_+ = 219$, $W_- = 34$) or we can reject the null hypothesis based on the critical value of the WSR test [40,65], if T is less than the critical value. Here, T is minimum of 219 and 34, that is, 34 is less than the critical value 65.

The Table 5 shows results for WACIL vs. SOTB on AUC values. From which, one can see that P-value (0.072686595) >0.05 and T is a minimum of 195.5 and 80.5, that is, 80.5 is not less than the critical value 73, so we do not have any evidence to reject the null hypothesis at a significance of 0.05. That means both WACIL and SOTB have a statistically identical distribution of results.

In order to answer research questions, we statistically analyze each performance measure of all competitive approaches over all classification models. Tables 6, 7 and 8 summarize the WSR test results of WACIL with other competitive approaches. The tabular results compare methods with respect to P-value, effect size and rank sums. For each performance measure, it shows the P-value (<0.05 or >0.05), the effect size (W_c) and the positive rank sum (W_+)/negative rank sum (W_-). The positive rank sum is of WACIL and the negative rank sum is of competitive approaches. In the Tables, the bold part P-value $<$**0.05** represents WACIL is superior to competitive approaches, and the Italian font bold part P-value $>$***0.05*** represents WACIL is statistically identical to competitive approaches.

In order to answer research question **RQ1**, we compared the FOR results of our proposed approach with the competitive approaches. In Tables 6, 7 and 8, the first row of all the tables shows the comparison of FOR values. We can notice the following points:

- WACIL with KNN, LR, and SVM, statistically outperforms all the compared sampling approaches with high effect sizes except NOS.

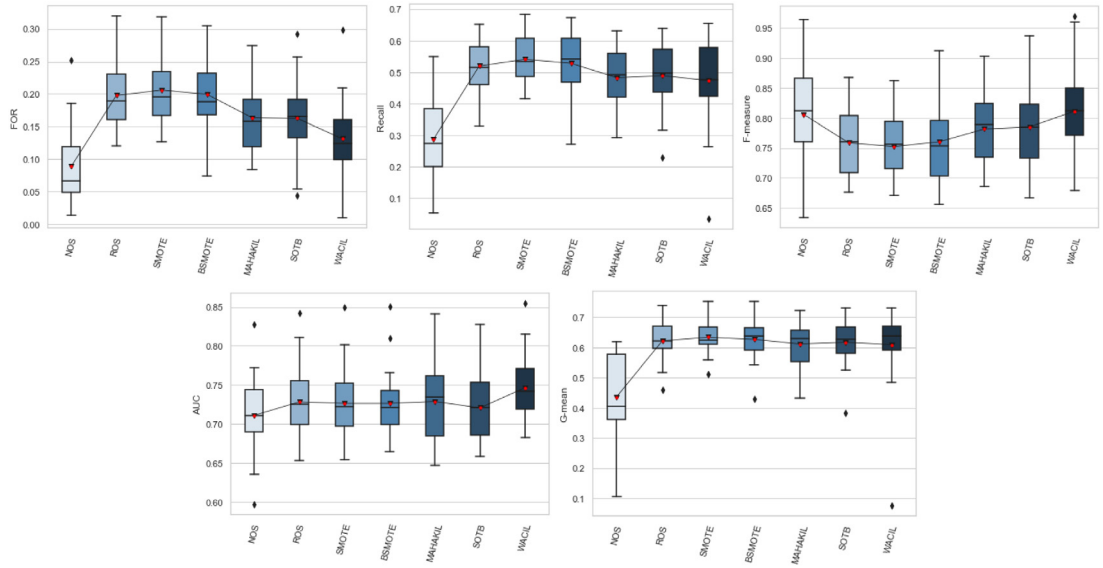

Fig. 4. Boxplots of six imbalance learning techniques and NOS on averaged results over 24 PROMISE and NASA datasets in terms of FOR, Recall, F-measure, AUC and G-mean.

Table 4
WSR test of AUC score between WACIL and SMOTE on LR classifier.

Datasets	WACIL	SMOTE	Abs Dif	Sign of Dif	Ranks	Signed ranks
Ant-1.7	0.820	0.802	0.018	1	8	8
Arc	0.765	0.735	0.030	1	13.5	13.5
Camel-1.6	0.709	0.681	0.028	1	12	12
Ivy-2.0	0.779	0.742	0.037	1	16	16
Jedit-4.2	0.816	0.809	0.007	1	3	3
Log4j-1.0	0.814	0.772	0.042	1	18	18
Lucene-2.0	0.766	0.766	0	–	–	–
Poi-2.0	0.686	0.648	0.038	1	17	17
Redaktor	0.791	0.774	0.017	1	7	7
Synapse-1.2	0.770	0.768	0.002	1	1	1
Tomcat	0.80	0.780	0.02	1	9	9
Velocity-1.6	0.794	0.773	0.021	1	10.5	10.5
Xalan-2.4	0.781	0.760	0.021	1	10.5	10.5
Xerces-1.3	0.778	0.778	0	–	–	–
CM1	0.814	0.718	0.096	1	22	22
MW1	0.758	0.704	0.054	1	20	20
KC3	0.699	0.683	0.016	1	6	6
MC1	0.751	0.802	0.051	-1	19	-19
MC2	0.745	0.737	0.008	1	4	4
PC1	0.843	0.875	0.032	-1	15	-15
PC2	0.741	0.678	0.063	1	21	21
PC3	0.82	0.811	0.009	1	5	5
PC4	0.914	0.908	0.006	1	2	2
PC5	0.768	0.738	0.030	1	13.5	13.5

Non-zero differences $(n) = 22$, $W_+ = 219$, $W_- = 34$

Table 5
WSR test of AUC score between WACIL and SOTB on LR classifier.

Datasets	WACIL	SMOTE	Abs Dif	Sign of Dif	Ranks	Signed ranks
Ant-1.7	0.820	0.803	0.017	1	15	15
Arc	0.765	0.743	0.022	1	16	16
Camel-1.6	0.709	0.696	0.013	1	11.5	11.5
Ivy-2.0	0.779	0.779	0	–	–	–
Jedit-4.2	0.816	0.815	0.001	1	1.5	1.5
Log4j-1.0	0.814	0.764	0.050	1	20	20
Lucene-2.0	0.766	0.763	0.003	1	3	3
Poi-2.0	0.686	0.658	0.028	1	17	17
Redaktor	0.791	0.799	0.008	-1	6	-6
Synapse-1.2	0.770	0.769	0.001	1	1.5	1.5
Tomcat	0.80	0.787	0.013	1	11.5	11.5
Velocity-1.6	0.794	0.778	0.016	1	14	14
Xalan-2.4	0.781	0.771	0.010	1	9	9
Xerces-1.3	0.778	0.791	0.013	-1	11.5	-11.5
CM1	0.814	0.737	0.077	1	22	22
MW1	0.758	0.765	0.007	-1	5	-5
KC3	0.699	0.712	0.013	-1	11.5	-11.5
MC1	0.751	0.812	0.061	-1	21	-21
MC2	0.745	0.754	0.009	-1	7.5	-7.5
PC1	0.843	0.873	0.03	-1	18	-18
PC2	0.741	0.658	0.083	1	23	23
PC3	0.820	0.815	0.005	1	4	4
PC4	0.914	0.905	0.009	1	7.5	7.5
PC5	0.768	0.732	0.036	1	19	19

Non-zero differences $(n) = 23$, $W_+ = 195.5$, $W_- = 80.5$

- WACIL with NB, statistically outperforms ROS, SMOTE, and BSMOTE and for NOS, MAHAKIL, and SOTB rank sum is less than our approach but P-value is >0.05, so it is statistically proven our approach has an identical distribution of results with those models on the NB classifier.
- WACIL with DT, statistically outperforms SMOTE and BSMOTE. For NOS, ROS, MAHAKIL, and SOTB, the P-value is >0.05, so it is statistically proven that our approach has an identical distribution of results with them on the DT classifier.
- WACIL with DNN, statistically outperforms ROS, SMOTE, BSMOTE and SOTB except NOS and MAHAKIL (>0.05), which has an identical distribution as our proposed approach.
- Table 8 demonstrates that on average our method outperforms all the other sampling approaches. The original data gives lower FOR results than any of the sampling techniques.

In reference to the aforementioned observations, we can conclude that, in terms of FOR, WACIL is superior to competitive approaches except NOS. As per Benin. [24], it is statically proven that the FOR results are good enough to say WACIL contribute towards the diversity of generated pseudo-instances.

In order to answer research question **RQ2**, we compared the recall results of our proposed approach with the competitive approaches.

- The statistical performance of WACIL with KNN and LR is poor in terms of recall score; it outperforms only NOS.

Table 6
Statistical comparison of WACIL with other competitive techniques in terms of FOR, Recall, F-measure, AUC and G-mean using KNN, LR and NB.

K-Nearest neighbor							
Measure	Measure	NOS	ROS	SMOTE	BSMOTE	MAHAKIL	SOTB
FOR	*P*-value	1.82E−05(< 0.05)	2.07E−05 (<**0.05**)	1.82E−05 (<**0.05**)	1.82E−05 (<**0.05**)	0.000255 (<**0.05**)	1.82E−05 (<**0.05**)
	W_c	1 (0/300)	0.993 (299/1)	1 (300/0)	1 (300/0)	0.857 (278.5/21.5)	1 (300/0)
Recall	*P*-value	2.70E−05 (<**0.05**)	7.14E−05 (<0.05)	1.82E−05 (<0.05)	2.07E−05 (<0.05)	0.01263 (<0.05)	2.35E−05 (<0.05)
	W_c	1 (276/0)	0.927 (11/289)	1 (0/300/)	0.993 (1/299)	0.591 (56.5/219.5)	0.987 (2/298)
F-measure	*P*-value	0.4489411 (>***0.05***)	1.82E−05 (<**0.05**)	1.82E−05 (<**0.05**)	1.82E−05 (<**0.05**)	0.000162 (<**0.05**)	3.98E−05 (<**0.05**)
	W_c	0.173 (17/124)	1 (300/0)	1 (300/0)	1 (299/1)	0.899 (262/14)	1 (253/0)
AUC	*P*-value	3.64E−05 (<**0.05**)	4.02E−05 (<**0.05**)	0.004673 (<**0.05**)	0.001549 (<**0.05**)	0.051986 (>***0.05***)	0.338372 (>***0.05***)
	W_c	0.963 (294.5/5.5)	0.978 (273/0)	0.678 (231.5/44.5)	0.771 (224/29)	0.453 (218/82)	0.223 (116.5/183.5)
G-mean	*P*-value	3.09E−05 (<**0.05**)	2.88E−02 (<0.05)	0.0018439 (<0.05)	0.001755 (<0.05)	0.061933 (>***0.05***)	0.001673 (<0.05)
	W_c	0.993 (275/1)	0.513 (73/227)	0.723 (41.5/258.5)	0.73 (40.5/259.5)	0.455 (69/184)	0.733 (40/260)
Logistic regression							
Measure	Measure	NOS	ROS	SMOTE	BSMOTE	MAHAKIL	SOTB
FOR	P-value	2.07056E−05 (<0.05)	2.66915E−05 (<**0.05**)	3.8838E−05 (<**0.05**)	2.07056E−05 (<**0.05**)	0.00013647 (<**0.05**)	9.05972E−05 (<**0.05**)
	W_c	0.993 (1/299)	0.98 (297/3)	0.96 (294/6)	0.993 (299/1)	0.89 (283.5/16.5)	0.917 (287.5/12.5)
Recall	*P*-value	1.82153E−05 (<**0.05**)	0.001636198 (<0.05)	0.000828696 (<0.05)	0.001016031 (<0.05)	0.02986533 (<0.05)	0.015576468 (<0.05)
	W_c	1 (300/0)	0.767 (29.5/223.5)	0.777 (33.5/266.5)	0.767 (35/265)	0.507 (74/226)	0.589 (52/201)
F-measure	*P*-value	0.259005754 (>***0.05***)	1.81974E−05 (<**0.05**)	1.82153E−05 (<**0.05**)	1.82153E−05 (<**0.05**)	4.38232E−05 (<**0.05**)	3.02696E−05 (<**0.05**)
	W_c	0.267 (110/190)	1 (300/0)	1 (300/0)	1 (300/0)	0.953 (293/7)	0.977 (296.5/3.5)
AUC	*P*-value	0.000787036 (<**0.05**)	0.003100486 (<**0.05**)	0.002670949 (<**0.05**)	0.001242669 (<**0.05**)	0.008504642 (<**0.05**)	0.072686595 (>***0.05***)
	W_c	0.787 (268/32)	0.69 (253.5/46.5)	0.731 (219/34)	0.753 (263/37)	0.627 (224.5/51.5)	0.417 (195.5/80.5)
G-mean	*P*-value	1.82153E−05 (<**0.05**)	0.201451172 (>***0.05***)	0.170197795 (>***0.05***)	0.265059089 (>***0.05***)	0.391341771 (>***0.05***)	0.423687147 (>***0.05***)
	W_c	1 (300/0)	0.308 (95.5/180.5)	0.317 (102.5/197.5)	0.263 (110.5/189.5)	0.203 (119.5/180.5)	0.19 (121.5/178.5)
Naive Bayes							
Measure	Measure	NOS	ROS	SMOTE	BSMOTE	MAHAKIL	SOTB
FOR	P-value	0.3155264 (>***0.05***)	0.897692 (>***0.05***)	0.0177021 (<**0.05**)	0.02062 (<**0.05**)	0.737954 (>***0.05***)	0.077703 (>***0.05***)
	W_c	0.239 (105/171)	0.03 (145.5/154.5)	0.553 (233/67)	0.54 (231/69)	0.08 (149/127)	0.42 (196/80)
Recall	*P*-value	0.0396624 (<**0.05**)	0.241427 (>***0.05***)	0.4662411 (>***0.05***)	0.475051 (>***0.05***)	0.345754 (>***0.05***)	0.753304 (>***0.05***)
	W_c	0.48 (222/78)	0.273 (191/109)	0.17 (124.5/175.5)	0.167 (125/175)	0.22 (183/117)	0.07 (139.5/160.5)
F-measure	*P*-value	0.1102107 (>***0.05***)	0.103404 (>***0.05***)	0.0018226 (<**0.05**)	0.003903 (<**0.05**)	0.124506 (>***0.05***)	0.003404 (<**0.05**)
	W_c	0.38 (190.5/85.5)	0.38 (207/93)	0.739 (240/36)	0.667 (250/50)	0.37 (189/87)	0.687 (253/47)
AUC	*P*-value	0.4661956 (>***0.05***)	0.587209 (>***0.05***)	0.2415541 (>***0.05***)	0.280234 (>***0.05***)	0.235469 (>***0.05***)	0.042491 (<**0.05**)
	W_c	0.17 (175.5/124.5)	0.123 (168.5/131.5)	0.279 (176.5/99.5)	0.257 (173.5/102.5)	0.286 (177/98.5)	0.473 (221/79)
G-mean	*P*-value	0.0078806 (<**0.05**)	0.014566 (<**0.05**)	0.3896018 (>***0.05***)	0.345754 (>***0.05***)	0.041046 (<**0.05**)	0.715111 (>***0.05***)
	W_c	0.617 (242.5/57.5)	0.57 (235.5/64.5)	0.209 (153/100)	0.22 (183/117)	0.477 (221.5/78.5)	0.087 (150/126)

- Over NB, our approach is statistically superior to NOS and statistically has an identical distribution to all other competitive approaches.
- Over SVM, our approach statistically outperforms NOS, and has identical distribution results as MAHAKIL and SOTB. ROS, SMOTE and BSMOTE are better than WACIL.
- Over DT, our approach statistically outperforms NOS, ROS, BSMOTE and MAHAKIL, and has an identical distribution of results to SMOTE and SOTB.
- Over DNN, our approach statistically outperforms NOS, MAHAKIL and SOTB, and has identical distribution results as ROS, SMOTE and BSMOTE.
- Table 8 demonstrates that on average our method outperforms NOS, statistically identical to MAHAKIL and SOTB and unable to outperform the rest of the competitive approaches.

In reference to the aforementioned observations of FOR and recall, we can conclude that in terms of FOR, our approach is superior to all the compared sampling techniques except NOS. In terms of recall, our approach is superior to NOS, MAHAKIL, and SOTB except ROS, SMOTE, and BSMOTE on average. So, it is statically proven that WACIL produces lower FOR values with the comparable recall values on average.

In order to answer research question Q3, we compared the F-measure, AUC and G-mean results of our proposed approach with the competitive approaches.

- In terms of F-measure, our approach outperforms all the competitive sampling approaches and is statistically identical to NOS over KNN, LR, SVM, and DNN. WACIL with NB, outperforms SMOTE, BSMOTE and SOTB and is statistically identical to NOS, ROS and MAHAKIL. WACIL with DT outperforms NOS, SMOTE, BSMOTE, MAHAKIL and SOTB and is statistically identical to ROS.
- In terms of AUC, our approach outperforms all the competitive approaches over SVM and DT. WACIL with KNN outperforms NOS, ROS, SMOTE, and BSMOTE and is statistically identical to MAHAKIL and SOTB. WACIL outperforms SOTB and is statistically identical to the rest of the methods over NB. WACIL is statistically identical to SOTB and outperforms the rest of the methods over LR. WACIL is statistically identical to NOS and outperforms the rest of the methods over DNN.
- In terms of G-mean, WACIL outperforms all the competitive approaches over DT and DNN. WACIL is superior to NOS, statistically identical to MAHAKIL over KNN. WACIL outperforms NOS and is statistically identical to the rest of the competitive approaches over LR. WACIL outperforms NOS, ROS, MAHAKIL and SOTB and is statistically identical to SMOTE and BSMOTE over NB. WACIL outperforms NOS, statistically identical to MAHAKIL and SOTB over SVM.
- Table 8 demonstrates that on average WACIL outperform all the other competitive approaches in terms of AUC. In terms of F-measure, WACIL outperform all the sampling

Table 7
Statistical comparison of WACIL with other competitive techniques in terms of FOR, Recall, F-measure, AUC and G-mean using SVM, DT and DNN.

Support vector machine							
Measure	Measure	NOS	ROS	SMOTE	BSMOTE	MAHAKIL	SOTB
FOR	P-value W_c	2.70E−05 (<0.05) 1 (0/276)	2.67E−05 (<**0.05**) 0.98 (297/3)	2.07E−05 (<**0.05**) 0.993 (299/1)	2.07E−05 (<**0.05**) 0.993 (299/1)	0.000162 (<**0.05**) 0.88 (282/18)	9.07E−05 (<**0.05**) 0.913 (287/13)
Recall	*P*-value W_c	2.70E−05 (<**0.05**) 1 (276/0)	5.61E−05 (<0.05) 0.94 (9/291)	0.0001288 (<0.05) 0.893 (16/284)	0.000285 (<0.05) 0.847 (23/277)	0.24719 (>***0.05***) 0.27 (109.5/190.5)	0.692535 (>***0.05***) 0.094 (125/151)
F-measure	*P*-value W_c	0.7494278 ((>***0.05***)) 0.072 (148/128)	9.61E−05 (<**0.05**) 0.91 (286.5/13.5)	3.52E−05 (<**0.05**) 0.986 (274/2)	2.67E−05 (<**0.05**) 0.98 (297/3)	0.002828 (<**0.05**) 0.703 255.5/44.5	0.002038 (<**0.05**) 0.747 (221/32)
AUC	*P*-value W_c	1.82E−05 (<**0.05**) 1 (300/0)	0.008574 (<**0.05**) 0.613 (242/58)	0.0031035 (<**0.05**) 0.693 (254/46)	0.000914 (<**0.05**) 0.786 (246.5/29.5)	0.001591 (<**0.05**) 0.737 (260.5/39.5)	0.007039 (<**0.05**) 0.66 (210/43)
G-mean	*P*-value W_c	2.70E−05 (<**0.05**) 1 (276/0)	0.000262 (<0.05) 0.87 (18/258)	0.0042747 (<0.05) 0.667 (50/250)	0.005581 (<0.05) 0.647 (53/247)	0.539006 (>***0.05***) 0.147 (128/172)	0.951492 (>***0.05***) 0.004 (137.5/138.5)
Decision tree							
Measure	Measure	NOS	ROS	SMOTE	BSMOTE	MAHAKIL	SOTB
FOR	P-value W_c	0.0151325 (<0.05) 0.567 (65/235)	0.116083 (>***0.05***) 0.363 (95.5/204.5)	0.0005456 (<0.05) 0.803 (270.5/29.5)	0.006191 (<0.05) 0.656 (228.5/47.5)	0.119399 (>***0.05***) 0.363 (204.5/95.5)	0.068017 (>***0.05***) 0.431 (197.5/78.5)
Recall	P-value W_c	2.67E−05 (<**0.05**) 0.98 (297/3)	0.000747 (<**0.05**) 0.787 (268/32)	0.0537719 (>***0.05***) 0.45 (217.5/82.5)	0.016384 (<**0.05**) 0.56 (234/66)	0.010128 (<**0.05**) 0.6 (240/60)	0.071861 ((>***0.05***)) 0.42 (213/87)
F-measure	*P*-value W_c	0.0225179 (<**0.05**) 0.551 (214/62)	0.136001 (>***0.05***) 0.355 (187/89)	7.12E−05 (<**0.05**) 0.93 (289.5/10.5)	0.006784 (<**0.05**) 0.641 (226.5/49.5)	0.002865 (<**0.05**) 0.714 (236.5/39.5)	0.008135 (<**0.05**) 0.634 (225.5/50.5)
AUC	*P*-value W_c	3.09E−05 (<**0.05**) 0.993 (275/1)	0.000748 (<**0.05**) 0.783 (267.5/32.5)	0.0018439 (<**0.05**) 0.727 (259/41)	0.004668 (<**0.05**) 0.674 (231/45)	0.001017 (<**0.05**) 0.767 (265/35)	0.016265 (<**0.05**) 0.576 (217.5/58.5)
G-mean	*P*-value W_c	1.82E−05 (<**0.05**) 1 (300/0)	0.000919 (<**0.05**) 0.773 (266/34)	0.032107 (<**0.05**) 0.5 (225/75)	0.005579 (<**0.05**) 0.643 (246.5/53.5)	0.005642 (<**0.05**) 0.659 (229/47)	0.072654 (>***0.05***) 0.435 (198/78)
Deep neural network							
Measure	Measure	NOS	ROS	SMOTE	BSMOTE	MAHAKIL	SOTB
FOR	*P*-value W_c	0.0011827 (<**0.05**) 0.757 (263.5/36.5)	0.001242 (<**0.05**) 0.77 (265.5/34.5)	0.0006383 (<**0.05**) 0.797 (269.5/30.5)	0.005579 (<**0.05**) 0.653 (248/52)	0.587171 (>***0.05***) 0.13 (169.5/130.5)	0.04297 (<**0.05**) 0.482 (204.5/71.5)
Recall	*P*-value W_c	3.43E−05 (<**0.05**) 0.967 (295/5)	0.161492 (>***0.05***) 0.327 (199/101)	0.2123927 (>***0.05***) 0.297 (179/97)	0.05742 (>***0.05***) 0.443 (216.5/83.5)	0.003252 <**0.05** 0.683 (252.5/47.5)	0.030814 <**0.05** 0.518 (209.5/66.5)
F-measure	*P*-value W_c	0.3141425 (>***0.05***) 0.245 (95.5/157.5)	0.000127 (<**0.05**) 0.913 (264/12)	6.32E−05 (<**0.05**) 0.933 (290/10)	2.66E−05 (<**0.05**) 0.98 (297/3)	0.010979 (<**0.05**) 0.6 (240/60)	0.000122 (<**0.05**) 0.957 (226/5)
AUC	*P*-value W_c	0.1102725 (>***0.05***) 0.377 (190/86)	0.002455 (<**0.05**) 0.703 (255.5/44.5)	3.88E−05 (<**0.05**) 0.963 (294.5/5.5)	0.000828 (<**0.05**) 0.783 (267.5/32.5)	0.000965 (<**0.05**) 0.786 (246.5/29.5)	1.82E−05 (<**0.05**) 1 (300/0)
G-mean	*P*-value W_c	5.61E−05 (<**0.05**) 0.94 (219/9)	0.013755 (<**0.05**) 0.583 (218.5/57.5)	2.64E−02 (<**0.05**) 0.525 (210.5/65.5)	0.007871 (<**0.05**) 0.617 (242.5/57.5)	0.004273 (<**0.05**) 0.667 (250/50)	1.06E−02 (<**0.05**) 0.609 (222/54)

Table 8
Statistical comparison of WACIL with other competitive techniques in terms of average FOR, Recall, F-measure, AUC and G-mean.

Measure	Measure	NOS	ROS	SMOTE	BSMOTE	MAHAKIL	SOTB
FOR	P-value W_c	0.000162316 (<0.05) 0.88 (18/282)	3.02413E−05 (<**0.05**) 0.97 (295.5/4.5)	2.07056E−05 (<**0.05**) 0.993 (299/1)	2.07056E−05 (<**0.05**) 0.993 (299/1)	0.000374651 (<**0.05**) 0.827 (274/26)	2.6987E−05 (<**0.05**) 0.977 (296.5/3.5)
Recall	P-value W_c	2.07056E−05 (<**0.05**) 0.993 (299/1)	0.021434925 (<0.05) 0.537 (69.5/230.5)	0.000776477(<0.05) 0.801 (27.5/248.5)	0.00057523 (<0.05) 0.803 (29.5/270.5)	0.897634428 (>***0.05***) 0.03 (154.5/145.5)	0.174603819 (>***0.05***) 0.317 (102.5/197.5)
F-measure	P-value W_c	0.447004889 (>***0.05***) 0.178 (162.5/113.5)	2.6987E−05 (<**0.05**) 1 (276/0)	1.82153E−05 (<**0.05**) 1 (300/0)	1.81795E−05 (<**0.05**) 1 (300/0)	6.32333E−05 (<**0.05**) 0.933 (290/10)	3.9959E−05 (<**0.05**) 1 (253/0)
AUC	P-value W_c	1.8019E−05 (<**0.05**) 1 (300/0)	0.000312397 (<**0.05**) 0.859 (256.5/19.5)	3.42058E−05 (<**0.05**) 0.967 (295/5)	2.6987E−05 (<**0.05**) 1 (276/0)	8.01794E−05 (<**0.05**) 0.92 (288/12)	1.81259E−05 (<**0.05**) 1 (300/0)
G-mean	P-value W_c	2.07056E−05 <**0.05** 0.993 (299/1)	0.310397062 (>***0.05***) 0.237 (185.5/114.5)	0.317261121 (>***0.05***) 0.23 (115.5/184.5)	0.303656438 (>***0.05***) 0.24 (114/186)	0.010614157 (<**0.05**) 0.601 (221/55)	0.315330456 (>***0.05***) 0.239 (221/55)

approaches and statically identical to NOS. In terms of G-mean, WACIL outperform NOS and MAHAKIL, statistically identical to rest of the methods.

We can conclude that in terms of F-measure and AUC, WACIL is superior to almost all competitive approaches on average and in terms of G-mean, most of the time, WACIL is superior to all methods over DT and DNN classifiers. For the rest of the classifiers, WACIL outperform NOS, MAHAKIL and SOTB and are identical to ROS, SMOTE and BSMOTE on average. Hence, it is statically proven that the comprehensive performance of WACIL is superior to the competitive approaches.

The Table 9 reports the comparison results of win-draw-loss over each predictive model (classification model + imbalance learning techniques). We have computed the total number of wins, draws, and losses for each model over all the five performance measures. For each performance measure, each single predictive model computes win-draw-loss statistics against 41 other predictive models (6 classification models * 7 imbalance learning techniques - 1). The ranks are assigned to predictive models on the basis of wins-draws. The predictive models which have the highest wins-draws are assigned higher ranks (starts from rank 1). In Table 9, gray part cells in columns represent top 7 predictive models with respect to individual measure. WACIL ranked according to individual performance measure in the following way, FOR of WACIL ranked 6, 12, 18, 5, 21 and 7. The

Table 9
Statistical comparison of all the predictive models over all performance measures. W-Wins, D-Draws and L-Losses.

Model	FOR					Recall					F-measure					AUC					G-mean					Total				
	W	L	D	W-L	Rank	W	L	D	W-L	Rank	W	L	D	W-L	Rank	W	L	D	W-L	Rank	W	L	D	W-L	Rank	W	L	D	W-L	Rank
KNN+WACIL	29	5	7	24	6	8	14	19	-6	23	34	0	7	34	2.5	18	2	21	16	10	11	11	19	0	16	100	32	73	68	4
KNN+SOTB	9	17	15	-8	27.5	23	7	11	16	12	9	12	20	-3	19.5	21	2	18	19	7	29	2	10	27	10	91	40	74	51	7
KNN+MAHAKIL	10	14	17	-4	23	21	12	8	9	14	9	12	20	-3	19.5	14	7	20	7	20	25	10	6	15	14	79	55	71	24	13
KNN+BSMOTE	0	33	8	-33	41	36	0	5	36	2	0	30	11	-30	40	10	10	21	0	25.5	30	0	11	30	5.5	76	73	56	3	19
KNN+SMOTE	0	34	7	-34	42	37	0	4	37	1	0	32	9	-32	42	12	11	18	1	23.5	33	0	8	33	2.5	82	77	46	5	18
KNN+ROS	0	30	11	-30	39.5	30	2	9	28	6	0	31	10	-31	41	9	17	15	-8	29	27	3	11	24	11	66	83	56	-17	28
KNN+NOS	40	0	1	40	1.5	1	40	0	-39	41	33	0	8	33	5.5	8	23	10	-15	33	1	40	0	-39	41	83	103	19	-20	29
LR+WACIL	18	7	16	11	12	22	9	10	13	13	30	0	11	30	7.5	39	0	2	39	1	29	0	12	29	7	138	16	51	122	1
LR+SOTB	9	22	10	-13	32.5	29	5	7	24	8	9	15	17	-6	23	29	0	12	29	3	29	1	11	28	8.5	105	43	57	62	5
LR+MAHAKIL	9	20	12	-11	30.5	28	5	8	23	9	9	15	17	-6	23	22	1	18	21	4.5	29	1	11	28	8.5	97	42	66	55	6
LR+BSMOTE	2	29	10	-27	37	32	0	9	32	5	3	26	12	-23	36	18	2	21	16	10	31	0	10	31	4	86	57	62	29	10
LR+SMOTE	1	30	10	-29	38	34	0	7	34	4	0	26	15	-26	38	20	3	18	17	8	34	0	7	34	1	89	59	57	30	9
LR+ROS	0	30	11	-30	39.5	35	0	6	35	3	0	27	14	-27	39	18	2	21	16	10	33	0	8	33	2.5	86	59	60	27	11.5
LR+NOS	39	2	0	37	3	2	39	0	-37	40	34	0	7	34	2.5	8	3	30	5	21	2	39	0	-37	40	85	83	37	2	20.5
NB+WACIL	11	5	25	6	18	7	9	25	-2	18.5	11	6	24	5	15	10	2	29	8	18.5	7	14	20	-7	23.5	46	36	123	10	17
NB+SOTB	9	5	27	4	20	7	9	25	-2	18.5	3	11	27	-8	27.5	9	12	20	-3	27	7	14	20	-7	23.5	35	51	119	-16	26.5
NB+MAHAKIL	1	6	34	-5	25	6	8	27	-2	18.5	1	9	31	-8	27.5	10	6	25	4	22	5	17	19	-12	32	23	46	136	-23	30
NB+BSMOTE	0	9	32	-9	29	7	7	27	0	16	0	16	25	-16	32	9	9	23	0	25.5	5	12	24	-7	23.5	21	53	131	-32	34
NB+SMOTE	0	11	30	-11	30.5	11	4	26	7	15	0	15	26	-15	31	10	9	22	1	23.5	6	12	23	-6	18.5	27	51	127	-24	31
NB+ROS	1	5	35	-4	23	5	9	27	-4	21	0	9	32	-9	29.5	12	3	26	9	17	4	21	16	-17	34	22	47	136	-25	32
NB+NOS	15	3	23	12	10.5	3	19	19	-16	36	0	9	32	-9	29.5	11	3	27	8	18.5	3	31	7	-28	39	32	65	108	-33	35
SVM+WACIL	35	4	2	31	5	4	17	20	-13	34.5	33	0	8	33	5.5	37	0	4	37	2	7	16	18	-9	28.5	116	37	52	79	2
SVM+SOTB	18	6	17	12	10.5	5	15	21	-10	31.5	20	6	15	14	11	23	2	16	21	4.5	5	15	21	-10	30.5	71	44	90	27	11.5
SVM+MAHAKIL	16	8	17	8	15.5	8	14	19	-6	23	19	8	14	11	12	14	4	23	10	16	8	14	19	-6	18.5	65	48	92	17	15.5
SVM+BSMOTE	2	28	11	-26	35.5	23	3	15	20	10	3	26	12	-23	36	14	3	24	11	15	24	5	12	19	13	66	65	74	1	22
SVM+SMOTE	3	29	9	-26	35.5	23	6	12	17	11	3	26	12	-23	36	16	2	23	14	12.5	25	5	11	20	12	70	68	67	2	20.5
SVM+ROS	4	28	9	-24	34	28	1	12	27	7	4	22	15	-18	33.5	22	2	17	20	6	30	0	11	30	5.5	88	53	64	35	8
SVM+NOS	40	0	1	40	1.5	0	41	0	-41	42	30	0	11	30	7.5	8	27	6	-19	34	0	41	0	-41	42	78	109	18	-31	33
DT+WACIL	13	13	15	0	21	8	14	19	-6	23	14	10	17	4	16.5	6	34	1	-28	35	10	14	17	-4	17	51	85	69	-34	36
DT+SOTB	12	16	13	-4	23	6	16	19	-10	31.5	9	16	16	-7	25.5	1	36	4	-35	38.5	7	16	18	-9	28.5	35	100	70	-65	37
DT+MAHAKIL	11	17	13	-6	26	5	18	18	-13	34.5	10	16	15	-6	23	1	36	4	-35	38.5	4	23	14	-19	35	31	110	64	-79	39.5
DT+BSMOTE	10	18	13	-8	27.5	6	15	20	-9	29.5	9	16	16	-7	25.5	2	35	4	-33	37	6	16	19	-10	30.5	33	100	72	-67	38
DT+SMOTE	8	21	12	-13	32.5	6	14	21	-8	26.5	3	21	17	-18	33.5	0	36	5	-36	40	4	17	20	-13	33	21	109	75	-88	41
DT+ROS	17	9	15	8	15.5	3	29	9	-26	37	9	11	21	-2	18	0	37	4	-37	41	3	25	13	-22	36	32	111	62	-79	39.5
DT+NOS	18	8	15	10	13.5	3	35	5	-32	38.5	9	14	18	-5	21	0	39	2	-39	42	3	29	9	-26	37	33	125	49	-92	42
DNN+WACIL	27	5	9	22	7	12	14	15	-2	18.5	34	0	7	34	2.5	16	2	23	14	12.5	20	10	11	10	15	109	31	65	78	3
DNN+SOTB	22	7	12	15	9	6	15	20	-9	29.5	26	6	9	20	10	5	34	2	-29	36	8	15	18	-7	23.5	67	77	61	-10	24.5
DNN+MAHAKIL	24	5	12	19	8	6	17	18	-11	33	29	4	8	25	9	8	17	16	-9	30	8	15	18	-7	23.5	75	58	72	17	15.5
DNN+BSMOTE	19	9	13	10	13.5	6	14	21	-8	26.5	17	10	14	7	14	8	20	13	-12	32	8	15	18	-7	23.5	58	68	79	-10	24.5
DNN+SMOTE	15	10	16	5	19	6	14	21	-8	26.5	15	11	15	4	16.5	8	18	15	-10	31	8	15	18	-7	23.5	52	68	85	-16	26.5
DNN+ROS	16	9	16	7	17	6	14	21	-8	26.5	19	10	12	9	13	8	15	18	-7	28	8	15	18	-7	23.5	57	63	85	-6	23
DNN+NOS	37	3	1	34	4	3	35	3	-32	38.5	34	0	7	34	2.5	15	3	23	12	14	3	30	8	-27	38	92	71	42	21	14

recall of WACIL ranked 23, 13, 18.5, 34.5, 23 and 18.5. The F-measure of WACIL ranked 2.5, 7.5, 15, 5.5, 16.5 and 2.5. The AUC of WACIL ranked 10, 1, 18.5, 2, 35 and 12.5 and the G-mean of WACIL ranked 16, 7, 23.5, 28.5, 17 and 15 on KNN, LR, NB, SVM, DT and DNN, respectively. The total (all performance measures combined) win-draw-losses are reported in the last column of the Table 9. On average WACIL ranked 4, 1, 17, 2, 36 and 3 over KN, LR, NB, SVM, DT and DNN, respectively. From the aforementioned observations, we can conclude that the classifiers LR and DNN with WACIL performs the best against all other combinations. WACIL with SVM performed the best over FOR, F-measure and AUC. WACIL with NB performed the best over all measures except G-mean. All the sampling techniques in combination with DT work in a poor way. Overall, the total results ranks show that WACIL clearly outperforms against all other predictive models with 1, 2, 3 and 4 highest ranks.

7. Discussion

7.1. Why does our proposed WACIL perform better than other sampling techniques?

Our proposed approach is more effective than the other compared approaches. This is because, unlike other approaches, before synthetic sample generation, WACIL wisely selects border samples to maximize the recognition rate of faulty modules, considers the average centroid as synthetic samples to increase diversity in generated pseudo data and filters the pseudo data to reduce the percentage of misclassification. The effectiveness of WACIL is demonstrated through the empirical results in Section 6.

From the standpoint of architecture, most of the software fault prediction datasets comprise fewer faulty modules and a large percentage of non-faulty modules, which results in extremely poor performance on faulty modules. This can be overwhelmed by compared oversampling approaches by introducing new synthetic information into the faulty module class. While a few synthetic instances overlapped with the non-faulty class, because SMOTE based approaches use the nearest neighbors of instances to generate synthetic information, sometimes these may cross the boundary of the faulty class and overlap with the non-faulty class. On the contrary, MAHAKIL and SOTB make sure that the synthetic data does not cross the boundary of a faulty class, but the problem is that the diversity of generated samples is poor.

The modules which are close to the boundary are misclassified most frequently [21,23]. WACIL wisely selects border instances which are used to generate pseudo-instances. The selected instance is neither near nor too far, and to mitigate the limitation of closet sample selection, it divides the pool of borderline faulty class instances into three clusters (cases) according to Mahalanobis distance and shuffles them. WACIL generates pseudo-instances by considering the weighted average of the minimum and maximum values of those cluster instances. So, the generated pseudo data distributes in a wide range and is rich in diversity. Furthermore, generated pseudo data is filtered to exclude out-of-boundary information, ensuring that the distribution of pseudo

data is managed to remain within the boundary of the faulty class, and WACIL does not consider closest modules to generate pseudo-instances. So, it avoids falling into the trap of sub-cluster enhancement. All these challenges are wisely handled by WACIL, resulting in the better overall performance in terms of faulty and non-faulty module prediction.

7.2. Does filtration of generated pseudo-instance affect the WACIL performance?

During the generation of pseudo-instances, there could be some noisy modules generated. In order to maximize the performance of the prediction model, they have to be excluded from the set of pseudo-instances. In the filtration step, we excluded instances which had a greater number of closest neighbors from the non-faulty class than the faulty class. When a classification model is trained on pseudo-instances which have a large number of closest neighbors from the non-faulty class, it misclassifies non-faulty modules as faulty modules during testing. As a result, excluding them reduces the number of false positives and increases the true negatives, which can improve the FPR and TNR of the predictive model (Discussed in Section 6). In the filtration step, we excluded instances which have all K closet neighbors from the faulty class; this can reduce the predictive model's over-generalization over the faulty module class.

7.3. What effect does the choice of classifiers have on WACIL's performance?

As mentioned in Section 5.4, we have chosen a total of six classifiers as prediction models, including KNN, LR, NB, SVM, DT, and DNN. We have noticed the influence of these classifiers on WACIL and other compared approaches over 24 datasets. We have analyzed classifier performance using various performance measures, including FOR, recall, F-measure, AUC, and G-mean. From the Table 9, on the basis of ranks, one can conclude that WACIL with KNN, NB, and SVM provide the best results, despite showing a slight difference over a few measures and it statistically achieves superior rankings with LR and DNN classifiers in comparison to other classifiers over all measures. Thus, LR and DNN are the supremely appropriate classifiers to build a model on WACIL generated training datasets.

8. Threats to validity

The subsequent subsections addresses the ascertained potential threats to our experimental determinations.

8.1. Construct validity

In order to assess the predictive model, we have carefully selected five performance assessment measures. FOR and recall are single-class measures and overall performance assessment measures including micro averaged F-measure, AUC and G-mean. These measures have been utilized in the SFP area for decades. Different assessment measures cause different results, which could lead to construction threats in this study. Nonetheless, some other assessing measures, such as G-measure, balance and Matthews's correlation coefficient (MCC), are also beneficial to explore. We make an attempt to include them in future works.

8.2. External validity

Our study utilizes a total of publicly available 24 datasets from PROMISE and NASA projects, and these datasets have been utilized extensively in SFP for decades. The considered datasets are diverse in number of features, instances and the percentage of faulty modules, thus it might be helpful to acquire the generalization of our determinations over divergent circumstances. Although our research is based on publicly available data, commercial projects contain different features that may threaten the external validity of our study. Besides, a detailed description of our proposed algorithm is given in Section 4. Thus, one can implement it through the steps outlined in the algorithm to validate other commercial software projects with different features. Further, as a means of classification, we adopted five traditional machine learning models and Deep Neural Network as a part of this study. We compared our results with five sampling techniques. However, there are many other sampling techniques to compare, as well as numerous classification models to train, which we intend to incorporate into future studies.

8.3. Internal validity

Our proposed WACIL generates pseudo-instances from borderline instances. The main assumption of WACIL is that those borderline instances forms three clusters (cases). The datasets we utilized in our work yield a minimum of three border instances. Datasets that do not produce at least three border instances may jeopardize our study's internal validity. We utilize inbuilt library functions for some of the compared approaches (ROS, SMOTE and BSMOTE). The MAHAKIL and SOTB, we precisely followed the corresponding algorithm steps given in the research papers [24,25]. The optimal hyper-parameter selection of the DNN classifier is done by grid search, which may give biased results with different datasets at different times. This could be an internal threat to our studies.

8.4. Conclusion validity

The experiments are repeated 10 times and consider the average performance measures to avoid biased results, and the results of WACIL are compared to those of other sampling approaches. Furthermore, we have used WSR test for win-draw-loss analysis and WSR with the matched pairs rank biserial correlation coefficient method for effect size computation. As a result, the conclusion validity threat to our research has a limited impact.

9. Conclusion and future work

SFP confronts a challenge, that is class imbalance issue. Many prominent over-sampling approaches generated synthetic data which is not diverse in nature. Simultaneously, they ignored noisy data, which raises the misclassification rate of classifiers. Exploiting these issues, we demonstrated the imbalanced learning criteria through the proposed WACIL approach, it introduces diverse filtered pseudo-instances around the border of the faulty class. WACIL reduces false positives and increases the diversity of newly generated pseudo-instances. We evaluated our proposed approach based on empirical results, by comparing it with not oversampled data and the five oversampling approaches, namely ROS, SMOTE, BSMOTE, MAHAKIL and SOTB over 24 imbalanced PROMISE and NASA projects, using six classification models, namely KNN, LR, NB, SVM, DT and DNN and five performance measures, namely FOR, recall, F-measure, AUC and G-mean. The experimental outcomes and statistical measures show that the performance of our method is superior to compared approaches.

In particular, WACIL outperforms all other oversampling methods in terms of FOR, F-measure, and AUC, while yielding comparable results in terms of recall and G-mean. WACIL with LR and DNN outperforms all the other predictive models over all the performance measures. Whereas the rest of the classifiers combined with WACIL produce better and comparable results over a few measures.

In future studies, we aspire to employ our proposed approach to other software defect datasets as well as to some other commercial projects to evaluate their performance. We will also consider comparing the WACIL's performance to some more complex imbalance learning approaches with some more additional classifiers over other measures. And we also plan to extend our work to cross project defect prediction (CPDP) along with various feature selection techniques to generate related synthetic training data for CPDP with required features.

CRediT authorship contribution statement

Pravali Manchala: Conceptualization, Methodology/Study design, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Writing – review & editing, Visualization. **Manjubala Bisi:** Conceptualization, Methodology/Study design, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing – review & editing, Visualization, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] M. Riaz, E. Mendes, E. Tempero, A systematic review of software maintainability prediction and metrics, in: 2009 3rd International Symposium on Empirical Software Engineering and Measurement, IEEE, 2009, pp. 367–377.

[2] M.R. Lyu, et al., Handbook of Software Reliability Engineering, Vol. 222, IEEE Computer Society Press CA, 1996.

[3] S. Planning, The Economic Impacts of Inadequate Infrastructure for Software Testing, National Institute of Standards and Technology, 2002.

[4] N. Shrikanth, S. Majumder, T. Menzies, Early life cycle software defect prediction. Why? How? in: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), IEEE, 2021, pp. 448–459.

[5] C. Manjula, L. Florence, Deep neural network based hybrid approach for software defect prediction using software metrics, Cluster Comput. 22 (4) (2019) 9847–9863.

[6] Y. Dong, H. Guo, W. Zhi, M. Fan, Class imbalance oriented logistic regression, in: 2014 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, IEEE, 2014, pp. 187–192.

[7] D. Muchlinski, D. Siroky, J. He, M. Kocher, Comparing random forest with logistic regression for predicting class-imbalanced civil war onset data, Polit. Anal. 24 (1) (2016) 87–103.

[8] D. Gray, D. Bowes, N. Davey, Y. Sun, B. Christianson, Using the support vector machine as a classification method for software defect prediction with static code metrics, in: International Conference on Engineering Applications of Neural Networks, Springer, 2009, pp. 223–234.

[9] M. Beckmann, N.F. Ebecken, B.S.P. de Lima, et al., A KNN undersampling approach for data balancing, J. Intell. Learn. Syst. Appl. 7 (04) (2015) 104.

[10] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, IEEE Trans. Softw. Eng. 33 (1) (2006) 2–13.

[11] B. Turhan, A. Bener, Analysis of Naive Bayes' assumptions on software fault data: An empirical study, Data Knowl. Eng. 68 (2) (2009) 278–290.

[12] W. Liu, S. Chawla, D.A. Cieslak, N.V. Chawla, A robust decision tree algorithm for imbalanced data sets, in: Proceedings of the 2010 SIAM International Conference on Data Mining, SIAM, 2010, pp. 766–777.

[13] G. Zhao, J. Huang, Deepsim: deep learning code functional similarity, in: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2018, pp. 141–151.

[14] L. Qiao, X. Li, Q. Umer, P. Guo, Deep learning based software defect prediction, Neurocomputing 385 (2020) 100–110.

[15] O. Al Qasem, M. Akour, M. Alenezi, The influence of deep learning algorithms factors in software fault prediction, IEEE Access 8 (2020) 63945–63960.

[16] B. Krawczyk, Learning from imbalanced data: open challenges and future directions, Prog. Artif. Intell. 5 (4) (2016) 221–232.

[17] C. Tantithamthavorn, A.E. Hassan, K. Matsumoto, The impact of class rebalancing techniques on the performance and interpretation of defect prediction models, IEEE Trans. Softw. Eng. 46 (11) (2018) 1200–1219.

[18] N.V. Chawla, N. Japkowicz, A. Kotcz, Special issue on learning from imbalanced data sets, ACM SIGKDD Explor. Newsl. 6 (1) (2004) 1–6.

[19] N. Junsomboon, T. Phienthrakul, Combining over-sampling and under-sampling techniques for imbalance dataset, in: Proceedings of the 9th International Conference on Machine Learning and Computing, 2017, pp. 243–247.

[20] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, SMOTE: synthetic minority over-sampling technique, J. Artificial Intelligence Res. 16 (2002) 321–357.

[21] H. Han, W.-Y. Wang, B.-H. Mao, Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning, in: International Conference on Intelligent Computing, Springer, 2005, pp. 878–887.

[22] H. He, Y. Bai, E.A. Garcia, S. Li, ADASYN: Adaptive synthetic sampling approach for imbalanced learning, in: 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), IEEE, 2008, pp. 1322–1328.

[23] S. Barua, M.M. Islam, X. Yao, K. Murase, MWMOTE–majority weighted minority oversampling technique for imbalanced data set learning, IEEE Trans. Knowl. Data Eng. 26 (2) (2012) 405–425.

[24] K.E. Bennin, J. Keung, P. Phannachitta, A. Monden, S. Mensah, Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction, IEEE Trans. Softw. Eng. 44 (6) (2017) 534–550.

[25] D. Liu, S. Qiao, N. Han, T. Wu, R. Mao, Y. Zhang, C.-A. Yuan, Y. Xiao, SOTB: semi-supervised oversampling approach based on trigonal barycenter theory, IEEE Access 8 (2020) 50180–50189.

[26] Y. Zhang, T. Zuo, L. Fang, J. Li, Z. Xing, An improved MAHAKIL oversampling method for imbalanced dataset classification, IEEE Access 9 (2020) 16030–16040.

[27] S. Feng, J. Keung, X. Yu, Y. Xiao, K.E. Bennin, M.A. Kabir, M. Zhang, COSTE: Complexity-based OverSampling technique to alleviate the class imbalance problem in software defect prediction, Inf. Softw. Technol. 129 (2021) 106432.

[28] L. Gong, S. Jiang, L. Bo, L. Jiang, J. Qian, A novel class-imbalance learning approach for both within-project and cross-project defect prediction, IEEE Trans. Reliab. 69 (1) (2019) 40–54.

[29] Z. Sun, J. Zhang, H. Sun, X. Zhu, Collaborative filtering based recommendation of sampling methods for software defect prediction, Appl. Soft Comput. 90 (2020) 106163.

[30] X.-Y. Liu, J. Wu, Z.-H. Zhou, Exploratory undersampling for class-imbalance learning, IEEE Trans. Syst. Man Cybern. B 39 (2) (2008) 539–550.

[31] B. Zadrozny, J. Langford, N. Abe, Cost-sensitive learning by cost-proportionate example weighting, in: Third IEEE International Conference on Data Mining, IEEE, 2003, pp. 435–442.

[32] M. Liu, L. Miao, D. Zhang, Two-stage cost-sensitive learning for software defect prediction, IEEE Trans. Reliab. 63 (2) (2014) 676–686.

[33] M.J. Siers, M.Z. Islam, Software defect prediction using a cost sensitive decision forest and voting, and a potential solution to the class imbalance problem, Inf. Syst. 51 (2015) 62–71.

[34] K.H. Kim, S.Y. Sohn, Hybrid neural network with cost-sensitive support vector machine for class-imbalanced multimodal data, Neural Netw. 130 (2020) 176–184.

[35] Z. Sun, Q. Song, X. Zhu, Using coding-based ensemble learning to improve software defect prediction, IEEE Trans. Syst. Man Cybern. C 42 (6) (2012) 1806–1817.

[36] H. Tong, B. Liu, S. Wang, Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning, Inf. Softw. Technol. 96 (2018) 94–111.

[37] S.K. Pandey, R.B. Mishra, A.K. Tripathi, BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques, Expert Syst. Appl. 144 (2020) 113085.
[38] Z. Yuan, X. Chen, Z. Cui, Y. Mu, ALTRA: Cross-project software defect prediction via active learning and tradaboost, IEEE Access 8 (2020) 30037–30049.
[39] N.V. Chawla, A. Lazarevic, L.O. Hall, K.W. Bowyer, SMOTEBoost: Improving prediction of the minority class in boosting, in: European Conference on Principles of Data Mining and Knowledge Discovery, Springer, 2003, pp. 107–119.
[40] S. Chen, H. He, E.A. Garcia, RAMOBoost: Ranked minority oversampling in boosting, IEEE Trans. Neural Netw. 21 (10) (2010) 1624–1642.
[41] Y. Kamei, A. Monden, S. Matsumoto, T. Kakimoto, K.-i. Matsumoto, The effects of over and under sampling on fault-prone module detection, in: First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007), IEEE, 2007, pp. 196–204.
[42] V. García, J.S. Sánchez, R.A. Mollineda, On the effectiveness of preprocessing methods when dealing with different levels of class imbalance, Knowl.-Based Syst. 25 (1) (2012) 13–21.
[43] T.M. Khoshgoftaar, E.B. Allen, W.D. Jones, J.P. Hudepohl, Data mining for predictors of software quality, Int. J. Softw. Eng. Knowl. Eng. 9 (05) (1999) 547–563.
[44] R. Bryll, R. Gutierrez-Osuna, F. Quek, Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets, Pattern Recognit. 36 (6) (2003) 1291–1302.
[45] G. Douzas, F. Bacao, Geometric SMOTE a geometrically enhanced drop-in replacement for SMOTE, Inform. Sci. 501 (2019) 118–135.
[46] S. Feng, J. Keung, X. Yu, Y. Xiao, M. Zhang, Investigation on the stability of SMOTE-based oversampling techniques in software defect prediction, Inf. Softw. Technol. (2021) 106662.
[47] A.S. Tarawneh, A.B. Hassanat, K. Almohammadi, D. Chetverikov, C. Bellinger, Smotefuna: Synthetic minority over-sampling technique based on furthest neighbour algorithm, IEEE Access 8 (2020) 59069–59082.
[48] J.A. Sáez, J. Luengo, J. Stefanowski, F. Herrera, SMOTE–IPF: Addressing the noisy and borderline examples problem in imbalanced classification by a re-sampling method with filtering, Inform. Sci. 291 (2015) 184–203.
[49] L. Gong, S. Jiang, L. Jiang, Tackling class imbalance problem in software defect prediction through cluster-based over-sampling with filtering, IEEE Access 7 (2019) 145725–145737.
[50] O.F. Arar, K. Ayan, Software defect prediction using cost-sensitive neural network, Appl. Soft Comput. 33 (2015) 263–277.
[51] S.P. Patel, S.H. Upadhyay, Euclidean distance based feature ranking and subset selection for bearing fault diagnosis, Expert Syst. Appl. 154 (2020) 113400.
[52] D. Pappas, K. Kiriakopoulos, G. Kaimakamis, Optimal portfolio selection with singular covariance matrix, in: International Mathematical Forum, Vol. 5, 2010, pp. 2305–2318.
[53] T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic, Y. Jiang, Implications of ceiling effects in defect predictors, in: Proceedings of the 4th International Workshop on Predictor Models in Software Engineering, 2008, pp. 47–54.
[54] H. He, E.A. Garcia, Learning from imbalanced data, IEEE Trans. Knowl. Data Eng. 21 (9) (2009) 1263–1284.
[55] T. Fawcett, ROC graphs: Notes and practical considerations for researchers, Mach. Learn. 31 (1) (2004) 1–38.
[56] A.P. Bradley, The use of the area under the ROC curve in the evaluation of machine learning algorithms, Pattern Recognit. 30 (7) (1997) 1145–1159.
[57] M. Kubat, S. Matwin, et al., Addressing the curse of imbalanced training sets: one-sided selection, in: Icml, Vol. 97, Citeseer, 1997 pp. 179–186.
[58] A. Gholamy, V. Kreinovich, O. Kosheleva, Why 70/30 or 80/20 relation between training and testing sets: A pedagogical explanation, 2018.
[59] F. Wilcoxon, Individual comparisons by ranking methods, Biom. Bull. 1 (6) (1945) 80–83.
[60] B.M. King, P.J. Rosopa, E.W. Minium, Statistical Reasoning in the Behavioral Sciences, John Wiley & Sons, 2018.
[61] D.S. Kerby, The simple difference formula: An approach to teaching nonparametric correlation, Compr. Psychol. 3 (2014) 11–IT.
[62] M. Tomczak, E. Tomczak, The need to report effect size estimates revisited. An overview of some recommended measures of effect size, Trends Sport Sci. 1 (21) (2014) 19–25.
[63] M. Azzeh, Y. Elsheikh, M. Alseid, An optimized analogy-based project effort estimation, 2017, arXiv preprint arXiv:1703.04563.
[64] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
[65] Critical Value Table of Wilcoxon Signed-Ranks Test [Online]. Available: https://www.real-statistics.com/statistics-tables/wilcoxon-signed-ranks-table/.