Information and Software Technology 139 (2021) 106662

Contents lists available at ScienceDirect

Information and Software Technology

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

Investigation on the stability of SMOTE-based oversampling techniques in software defect prediction

Shuo Feng [a], Jacky Keung [a], Xiao Yu [b,*], Yan Xiao [c], Miao Zhang [a]

[a] Department of Computer Science, City University of Hong Kong, Hong Kong, China
[b] School of Computer Science and Technology, Wuhan University of Technology, Wuhan, China
[c] School of Computing, National University of Singapore, Singapore

ARTICLE INFO

Keywords:
Software defect prediction
Class imbalance
Oversampling
SMOTE
Empirical Software Engineering

ABSTRACT

Context: In practice, software datasets tend to have more non-defective instances than defective ones, which is referred to as the class imbalance problem in software defect prediction (SDP). Synthetic Minority Oversampling TEchnique (SMOTE) and its variants alleviate the class imbalance problem by generating synthetic defective instances. SMOTE-based oversampling techniques were widely adopted as the baselines to compare with the newly proposed oversampling techniques in SDP. However, randomness is introduced during the procedure of SMOTE-based oversampling techniques. If the performance of SMOTE-based oversampling techniques is highly unstable, the conclusion drawn from the comparison between SMOTE-based oversampling techniques and the newly proposed techniques may be misleading and less convincing.
Objective: This paper aims to investigate the stability of SMOTE-based oversampling techniques. Moreover, a series of stable SMOTE-based oversampling techniques are proposed to improve the stability of SMOTE-based oversampling techniques.
Method: Stable SMOTE-based oversampling techniques reduce the randomness in each step of SMOTE-based oversampling techniques by selecting defective instances in turn, distance-based selection of K neighbor instances, and evenly distributed interpolation. Besides, we mathematically prove and also empirically investigate the stability of SMOTE-based and stable SMOTE-based oversampling techniques on four common classifiers across 26 datasets in terms of AUC, $balance$, and MCC.
Results: The analysis of SMOTE-based and stable SMOTE-based oversampling techniques shows that the performance of stable SMOTE-based oversampling techniques is more stable and better than that of SMOTE-based oversampling techniques. The difference between the worst and best performances of SMOTE-based oversampling techniques is up to 23.3%, 32.6%, and 204.2% in terms of AUC, $balance$, and MCC, respectively.
Conclusion: Stable SMOTE-based oversampling techniques should be considered as a drop-in replacement for SMOTE-based oversampling techniques.

1. Introduction

Software Defect Prediction (SDP) leverages historical data to train machine learning, data mining, or statistical models, and builds prediction models to predict whether an instance (e.g., a file, a module, or a class) introduced in the future is defective or non-defective [1–3]. The accurate prediction results can guide software testers to allocate the testing resource more efficiently [4,5]. However, due to the software quality assurance activity, defective instances are normally fewer than non-defective ones, which causes prediction models tending to be biased towards non-defective instances (i.e., majority class instances) and ignore defective ones (i.e., minority class instances). Consequently, the performance of prediction models is poor. This is referred to as the class imbalance problem in SDP.

SDP suffers from the class imbalance problem [6,7]. The prevalent technique to solve the class imbalance problem in SDP is to use the oversampling techniques. By adding instances into the minority class, these techniques introduce bias towards the minority class and balance the distribution of datasets. Nowadays, most oversampling techniques are synthetic-based. Among synthetic-based techniques, Synthetic Minority Oversampling TEchnique (SMOTE) [8] is the most popular one. Standard SMOTE works as follows. (1) The number of synthetic minority class instances that need to be generated is set up. (2) A minority class instance is selected randomly. (3) K-Nearest Neighbor (KNN)

* Corresponding author.
E-mail addresses: shuofeng5-c@my.cityu.edu.hk (S. Feng), jacky.keung@cityu.edu.hk (J. Keung), xiaoyu@whut.edu.cn (X. Yu), dcsxan@nus.edu.sg (Y. Xiao), miazhang9-c@my.cityu.edu.hk (M. Zhang).

https://doi.org/10.1016/j.infsof.2021.106662
Received 19 August 2020; Received in revised form 1 June 2021; Accepted 9 June 2021
Available online 17 June 2021

algorithm is employed to obtain the K nearest minority class neighbors of the selected instance, and one of these K instances is randomly chosen. (4) The minority class instance and its selected neighbor minority class instance are combined to generate a new synthetic instance by random interpolation. The procedures of 2 to 4 are iterative until the desired number of synthetic instances is reached. Based on SMOTE, several oversampling techniques are developed, such as ADAptive SYNthetic sampling approach (ADASYN) [9] and Borderline-SMOTE [10]. All these techniques generate synthetic instances like SMOTE and are referred to as SMOTE-based oversampling techniques in this study.

According to the aforementioned procedure of SMOTE-based oversampling techniques, randomness is introduced in (1) the initial selection of minority class instances used to generate synthetic instances, (2) the strategy about how to select the nearest neighbor minority class instances, and (3) the selection of the interpolation where the synthetic instances are generated. Therefore, when SMOTE-based oversampling techniques are applied to a dataset, the oversampled dataset varies each time, which may lead to a high variance in the performance of prediction models.

However, little work focused on evaluating the stability of SMOTE-based oversampling techniques. If the performance of SMOTE-based oversampling techniques is unstable, practitioners could not be confident about the datasets oversampled by SMOTE-based oversampling techniques. More importantly, as the most common oversampling techniques, SMOTE-based oversampling techniques were widely adopted by previous studies [9,11,12] as the baselines to compare with the newly proposed oversampling techniques. However, if the performance of SMOTE-based oversampling techniques is highly unstable, the conclusion drawn from the comparison between SMOTE-based oversampling techniques and the newly proposed technique is misleading and not convincing enough. To improve the stability of SMOTE-based oversampling techniques, we propose a series of stable SMOTE-based oversampling techniques aiming at reducing the randomness in each step of SMOTE-based oversampling techniques. Specifically, for a specific SMOTE-based technique, (1) if it randomly selects minority class instances with equal probability in the initial selection, our proposed technique turns it into selecting all minority class instances one by one. (2) If a certain SMOTE-based technique randomly selects the nearest neighbor minority class instances with equal probability, our proposed technique turns it into selecting the nearest neighbor minority class instances from distant to near one by one. We take this strategy based on the conclusion of Wong [13] and Bennin [14]. Selecting the distant neighbor minority class instances could increase the diversity of synthetic instances and better avoid the overgeneralization of prediction models. (3) If a certain SMOTE-based technique randomly selects the interpolation to generate synthetic instances, our proposed technique turns it into generating synthetic instances evenly distributed on the line between the two selected instances.

To investigate the stability of SMOTE-based and stable SMOTE-based oversampling techniques, we mathematically analyze the stability of SMOTE-based and stable SMOTE-based oversampling techniques, conduct empirical experiments to investigate whether the randomness introduced in SMOTE-based oversampling techniques hinders the performance of prediction models, and also statistically compare the performance of stable SMOTE-based oversampling techniques with that of SMOTE-based oversampling techniques in this study. We evaluate the performance of three common SMOTE-based oversampling techniques (i.e., SMOTE, Borderline-SMOTE, and ADASYN) and their corresponding stable techniques on four common classifiers (i.e., K-Nearest Neighbor (KNN), Support Vector Machine (SVM), Random Forest (RF), and Decision Tree (DT)). Based on a case study across 26 datasets collected from the PROMISE repository [15], we find that the difference between the best and worst performances of SMOTE-based oversampling techniques is up to 23.3% in terms of the Area Under the ROC Curve (AUC), 32.6% in terms of $balance$, and 204.2% in terms of the Matthews Correlation Coefficient (MCC). Based on the experimental results, we conclude that the performance of SMOTE-based oversampling techniques is unstable. In contrast, we mathematically prove that the variance of the synthetic instances generated by stable SMOTE-based oversampling techniques is smaller than that of SMOTE-based oversampling techniques. We also empirically prove that stable SMOTE-based oversampling techniques could gain more stable and better performance than SMOTE-based oversampling techniques. Therefore, stable SMOTE-based oversampling techniques should be considered as a drop-in replacement for SMOTE-based oversampling techniques.

We summarize the contributions of this paper as follows:

(1) We are the first to investigate the stability of SMOTE-based oversampling techniques in the area of SDP. Our experimental results show that the performance of SMOTE-based oversampling techniques is highly unstable, which could hurt the objectivity of the conclusion drawn from the study using SMOTE-based oversampling techniques as the baselines.

(2) We propose a series of stable SMOTE-based oversampling techniques improving the stability of SMOTE-based oversampling techniques. Our proposed techniques can be applied to any SMOTE-based oversampling technique.

(3) We mathematically and empirically prove that our proposed techniques significantly improve the stability of SMOTE-based oversampling techniques and also produce better results. Our proposed techniques should be considered as the alternative baseline techniques for SMOTE-based oversampling techniques.

(4) We open-source our replication package to make it easy for others to replicate our work and conduct further works. All data and source code in this study can be found at https://github.com/fengshuocn/stability-of-smote.

The remainder of this paper is organized as follows. Section 2 shows the related work and the background of our work. Section 3 details the algorithm of stable SMOTE-based oversampling techniques. We analyze the time complexity of stable SMOTE-based oversampling techniques in Section 4. Section 5 mathematically proves the stability of stable SMOTE-based oversampling techniques. Section 6 introduces the experimental settings, including the datasets, the classifiers, the evaluation measures, the statistical test, and the experimental procedure. In Section 7, we present and analyze the experimental results. Section 8 further discusses the generalizability of our conclusion. Section 9 lists the threats to validity. In Section 10, we conclude the paper and introduce the future work.

2. Related work and background

As the size of software applications grows, software testing becomes increasingly expensive and time-consuming. SDP uses statistical or machine learning classifiers to build defect prediction models. These prediction models will be used to conveniently identify software instances likely to be defective. Thus, practitioners can save limited testing resources and put more focus on these instances [16]. Prediction models are trained by available historical data and used to predict whether the instances introduced in the future are defective or not. The researches in SDP can be categorized into different types, such as modeling of prediction models [17–21], feature selection [22–25], and class imbalance learning [26–28].

The class imbalance problem is quite common in SDP. Naturally, defective instances only take a small portion of a dataset. The number of non-defective instances is larger than that of defective ones. Prediction models trained on the imbalanced data tend to put more focus on non-defective instances while ignoring defective instances. However, researchers and practitioners are more interested in defective instances. To alleviate the class imbalance problem, many techniques have been proposed. These techniques can be categorized as cost-sensitive learning, ensemble learning, and data resampling in general.

In SDP, misclassifying a defective instance leads to a more serious consequence than misclassifying a non-defective one. The idea behind cost-sensitive learning is that when classifiers wrongly predict an instance, more cost is assigned to the misclassification of a defective instance than a non-defective one. Cost-sensitive learning tries to minimize the overall cost, thus alleviates the class imbalance problem. Khan et al. [29] proposed a cost-sensitive deep neural network. This technique can automatically optimize the different costs for both the majority class and the minority class and could lead to a low overall cost. Zhang et al. [30] proposed an evolutionary cost-sensitive extreme learning machines that could properly define the cost matrix in cost-sensitive learning. These two techniques both outperform the baseline techniques.

Ensemble learning combines different classifiers to enhance the performance of prediction models trained on imbalanced data. Based on the distribution of an ensemble classifier's margin, Feng et al. [31] proposed a novel ensemble algorithm that employs more low-margin examples than high-margin samples and could handle imbalanced data well. This algorithm also employs the undersampling technique to improve its performance. The comparison between the proposed algorithm and the state-of-the-art methods shows that the proposed algorithm is superior.

Data resampling techniques are more prevalent in SDP due to their easy employment and independence (i.e., they can be applied to any prediction model). As the most common oversampling techniques, SMOTE-based oversampling techniques were widely adopted as the baselines in many studies. SMOTE first randomly selects a minority class instance and one of its K nearest neighbor minority class instances. Then, a new synthetic minority class instance is generated by randomly selecting an interpolation between these two selected instances. The above process is iterated until the balance of data is achieved. It should be noted that the minority class instances used to generate synthetic instances are different at each running of SMOTE for a certain dataset. Based on SMOTE, Han et al. [10] proposed Borderline-SMOTE. SMOTE treats all minority class instances equally, and the probability of each instance used to generate synthetic instances is the same. Different from SMOTE, Borderline-SMOTE randomly selects those instances located at the decision boundary to generate synthetic instances. By doing this, the decision boundary between the minority class and the majority class can be strengthened. ADASYN, proposed by He et al. [9], is another variant of SMOTE. ADASYN also modifies the initial selection of SMOTE. ADASYN first calculates the number of synthetic instances that need to be generated for each minority class instance based on the level of difficulty of each minority class instance. The difficulty estimation is based on the ratio of instances belonging to the majority class in the neighborhood. Thus, it makes the initial selection of minority class instances remains the same for a certain dataset, which is different from SMOTE as well as Borderline-SMOTE. Nevertheless, the selection of the neighbor minority class instances and the interpolation are still random.

3. Methodology

The key idea of stable SMOTE-based oversampling techniques is to reduce or even eliminate the randomness introduced in each step of SMOTE-based oversampling techniques, thus reducing the variance of SMOTE-based oversampling techniques. The common steps shared by SMOTE-based oversampling techniques include (1) calculating the number of synthetic instances to be generated, (2) the initial selection of minority class instances used to generate synthetic instances, (3) the selection of the neighbor minority class instances, and (4) the selection of the interpolation where the new synthetic instances are generated. Randomness is introduced into SMOTE-based oversampling techniques through these steps. For instance, SMOTE introduces randomness in the second, third, fourth steps, and ADASYN only introduces randomness in the third and fourth steps. Stable SMOTE-based oversampling techniques also share the same steps but with some differences in details. Because stable SMOTE-based oversampling techniques share a similar procedure, we only present the algorithm and the pseudo-code (Algorithm 1) of stable SMOTE in the following. The capital letters represent quantities, and the lowercase letters represent the instances in datasets.

(1) Calculate the number of synthetic instances to be generated

The number N of synthetic instances that need to be generated is calculated as follows:

$$N = (M_{maj} - M_{min}) \times \beta, \tag{1}$$

where M_{maj} is the number of majority class instances, M_{min} is the number of minority class instances, and β can be set to any non-negative number. Normally, β is manually set up to 1, which means datasets will be fully balanced after being oversampled (Line 2 of Algorithm 1).

(2) Initialize the selection of minority class instances

In this step, we calculate R defined as:

$$R = N/M_{min}, \tag{2}$$

where R is the number of synthetic instances that need to be generated for each minority instance.

Then we compute R_k defined as:

$$R_k = floor(R/K), \tag{3}$$

where R_k is the number of synthetic instances that need to be generated for each neighbor minority class instance of a certain minority class instance x_i, K is the number of the nearest neighbor minority class instances which is set in advance, and $floor$ is the rounding down operation (Lines 3–4).

(3) Select the neighbor minority class instances

While $R_k > 1$, it indicates that all the K neighbor minority class instances of each minority class instance x_i will be selected to generate synthetic instances. We record each minority class instance x_i and its neighbor minority class instances $x_1, x_2, \dots, x_k$ as the pair (x_i, x_1), (x_i, x_2), …, (x_i, x_k), and the total number of all pairs are recorded as L. Then R_k is replaced by $R_k - 1$ until R_k is less than 1. It should be noted that there is no particular order for a pair (x_i, x_j). In other words, (x_i, x_j) and (x_j, x_i) are the same pair (Lines 5–12).

Next, stable SMOTE gets N_{new} by N subtracting L. R is also updated sequentially according to N_{new}. It is notable that after being updated, R is less than K. Therefore, it means that not all the K neighbor minority class instances of each minority class instance x_i will be used to generate synthetic instances in this step. Then R will be rounded down and used to select R neighbor minority class instances x_k of each minority class instance x_i. We here take the strategy that we select R neighbor minority class instances from distant to near for each x_i and recorded as (x_i, x_k). The reason we select the distant neighbor minority class instances is based on the conclusion of Wong [13] and Bennin [14] that SMOTE-based oversampling techniques may lead to the overgeneralization of prediction models because of selecting instances too close in distance. Selecting more distant instances could reduce the probability of overgeneralization (Lines 13–21).

Finally, N_{new} is updated by N subtracting L. After being updated, N_{new} is less than M_{min}, which means not all minority class instances will be selected to generate synthetic instances. Therefore, Stable SMOTE will randomly select N_{new} minority class instances x_i. Then for each selected x_i, its most distant neighbor minority class instance x_k will be selected, and the pair (x_i, x_k) will be recorded (Lines 22–27).

(4) Select the interpolation where the new synthetic instances are generated

For each pair (x_i, x_k), we calculate its counts as $N_{i,k}$. Then $N_{i,k}$ synthetic instances will be evenly distributed on the line between x_i and x_k (Lines 28–29).

According to the above description, the randomness is greatly reduced by stable SMOTE. Similar procedures can also be applied to

Algorithm 1 stable SMOTE

Input: Dataset *dataset* which includes M_{min} minority class instances and M_{maj} majority class instances, the ratio β controlling the final defect ratio and the number of the nearest neighbors K
Output: Balanced dataset *dataset*

```
1: Initialize array $arr$ for storing the pair of instances used to generate
   synthetic instances, and the length of $arr$ is recorded as $L$
2: $N = (M_{maj} - M_{min}) \times \beta$
3: $R = N/M_{min}$
4: $R_k = floor(R/K)$            ▷ $floor$ is the rounding down operation
5: while $R_k \geq 1$ do
6:     for $i = 1, 2..., M_{min}$ do
7:         for $k = 1, 2..., K$ do
8:             Insert $(x_i, x_k)$ into $arr$
9:         end for
10:    end for
11:    $R_k = R_k - 1$
12: end while
13: $N_{new} = N - L$
14: $R = N_{new}/M_{min}$
15: $R = floor(R)$
16: for $i = 1, 2..., M_{min}$ do
17:     for $j = 1, 2..., R$ do
18:         Select the neighbor minority class instance $x_k$ of $x_i$ from
            distant to near
19:         Insert $(x_i, x_k)$ into $arr$
20:     end for
21: end for
22: $N_{new} = N - L$
23: for $i = 1, 2..., N_{new}$ do
24:     Randomly select a minority class instance $x_i$
25:     Select the most distant neighbor minority class instance $x_k$ of $x_i$
26:     Insert $(x_i, x_k)$ into $arr$
27: end for
28: Count the number $N_{i,k}$ of the pair $(x_i, x_k)$ in $arr$
29: Generate $N_{i,k}$ synthetic instances on the line between $x_i$ and $x_k$
    evenly distributed.
```

Borderline-SMOTE, ADASYN, and other SMOTE-based oversampling techniques. For example, Borderline-SMOTE randomly selects the borderline minority class instances instead of all minority class instances with equal probability. For stable Borderline-SMOTE, most borderline instances will be selected a fixed number of times. For ADASYN, it selects the minority class instances based on their density distribution in the step of initializing the selection of minority class instances. Therefore, minority class instances will be selected a fixed number of times for ADASYN. Stable ADASYN takes the same strategy as ADASYN in this step. But for the rest of the steps, stable ADASYN will take the same strategy as stable SMOTE.

4. Time complexity of stable SMOTE

As can be seen in Algorithm 1, the time complexity of stable SMOTE is mostly governed by the following:

- Line 5 has the time complexity of $O(R_k)$
- Line 6 has the complexity of $O(M_{min})$
- Line 7 has the complexity of $O(K)$
- Line 16 has the complexity of $O(M_{min})$
- Line 17 has the complexity of $O(R)$
- Line 23 has the complexity of $O(N_{new})$
- Line 28 has the complexity of $O(N)$

Therefore, the time complexity of stable SMOTE is $O(R_k M_{min} K) + O(M_{min} R) + O(N_{new}) + O(N) = O((R_k K + R)M_{min}) + O(N_{new}) + O(N)$. The time complexity of SMOTE is $O(NM_{min})$. With the increase of the imbalanced level of data, R_k and N increase at the same speed. R is smaller than K. Therefore, the time complexity of stable SMOTE and SMOTE is in the same magnitude.

5. Theoretical analysis of stable SMOTE

In this section, we prove the stability of stable SMOTE mathematically. Blagus et al. [32] and Elreedy et al. [33] both provided some mathematical basis of SMOTE. We follow their format of the mathematical proof and refer the expected value as $E(\cdot)$, the variance as $var(\cdot)$, the covariance as $cov(,)$, the original instances as X_i, one of the K neighbor minority class instances as R_i, and the synthetic instances generated by SMOTE and stable SMOTE as S_i. Blagus and Elreedy have proved that the expected value of synthetic instances generated by SMOTE is the same as that of the original minority class instances (i.e., $E(S_i) = E(X_i)$). Blagus has also proved that the variance of SMOTE instances is smaller than that of the original minority class instances (i.e., $var(S_i^{SMOTE}) = \frac{2}{3}var(X_i)$). Based on their analysis, we further prove that the stability of stable SMOTE is superior to that of SMOTE.

For stable SMOTE, a new synthetic instance is obtained according to the following expression:

$$S_i = X_i + w(R_i - X_i), \tag{4}$$

where $w = \frac{l}{r+1}$, $l = 1, 2, \ldots, r$. r is the number of times that the pair of (X_i, R_i) is used to generate synthetic instances.

$$E(S_i) = E(X_i) - E(w)E(X_i) + E(w)E(R_i). \tag{5}$$

It is easy to prove that $E(w) = \frac{1}{2}$, and we obtain

$$E(S_i) = \frac{1}{2}(E(X_i) + E(R_i)). \tag{6}$$

According to Blagus's and Elreedy's conclusion that $E(X_i) = E(R_i)$, we prove that $E(S_i) = E(X_i)$, which means the expected value of the synthetic instances generated by stable SMOTE equals that of the original minority class instances as well as that of the synthetic instances generated by SMOTE. Then, we calculate the variance of S_i as follows:

$$var(S_i) = E(S_i^2) - E(S_i)^2, \tag{7}$$

$$\begin{aligned} E(S_i^2) &= E(X_i^2 + 2wX_i \cdot (R_i - X_i) \\ &\quad + w^2(R_i^2 - 2R_i \cdot X_i + X_i^2)), \end{aligned} \tag{8}$$

$$E(w) = \frac{1}{2}, \tag{9}$$

and

$$\begin{aligned} E(w^2) &= \sum_1^r (\frac{l^2}{(r+1)^2}) \\ &= \frac{(2r+1)}{6(r+1)} = a. \end{aligned} \tag{10}$$

Therefore,

$$E(S_i^2) = aE(X_i^2) + aE(R_i^2) + (1-2a)E(X_i \cdot R_i), \tag{11}$$

$$\begin{aligned} var(S_i) &= aE(X_i^2) + aE(R_i^2) + (1-2a)E(X_i \cdot R_i) \\ &\quad - \frac{1}{4}E(R_i)^2 - \frac{1}{4}E(X_i)^2 - \frac{1}{2}E(X_i)E(R_i) \\ &= a(var(X_i)) + a(var(R_i)) + (1-2a)cov(X_i, R_i) \\ &\quad + (a - \frac{1}{4})E(X_i)^2 + (a - \frac{1}{4})E(R_i)^2 \\ &\quad + (\frac{1}{2} - 2a)E(X_i)E(R_i), \end{aligned} \tag{12}$$

because $E(X_i) = E(R_i)$, the expression can be further simplified as follow:

$$var(S_i) = a(var(X_i)) + a(var(R_i)) + (1-2a)cov(X_i, R_i), \tag{13}$$

Table 1
Description of 26 imbalanced datasets collected from the PROMISE repository.

Projects	# Instances	% Defect ratio
ant-1.3	125	16
ant-1.4	178	22.5
ant-1.5	293	10.9
ant-1.6	351	26.2
ant-1.7	745	22.3
camel-1.0	339	3.8
camel-1.2	608	35.5
camel-1.4	872	16.6
camel-1.6	965	19.5
ivy-1.4	241	6.6
ivy-2.0	352	11.4
jedit-3.2	272	33.1
jedit-4.0	306	24.5
jedit-4.1	312	25.3
jedit-4.2	367	13.1
jedit-4.3	492	2.2
log4j-1.0	135	25.2
log4j-1.1	109	33.9
poi-2.0	314	11.8
synapse-1.0	157	10.2
synapse-1.1	222	27.0
synapse-1.2	256	33.6
velocity-1.6	229	34.1
xalan-2.4	723	15.2
xerces-1.2	440	16.1
xerces-1.3	453	15.2

According to [33], X_i and R_i are independent and therefore, $cov(X_i, R_i) = 0$. We obtain

$$var(S_i) = 2a(var(X_i)) = \frac{(2r+1)var(X_i)}{3(r+1)}, \tag{14}$$

$$var(S_i) = \frac{2var(X_i)}{3} - \frac{var(X_i)}{3(r+1)} < var(S_i^{SMOTE}). \tag{15}$$

We prove that the variance of the synthetic instances generated by stable SMOTE is smaller than that of SMOTE.

6. Experimental settings

This section introduces the details about the experiments, including the comparison techniques, the datasets, the selected classifiers, the performance measures, the statistical test, and also the experimental procedure.

6.1. Implementation setting

In this study, we investigate three SMOTE-based techniques and their corresponding stable techniques. Specifically, SMOTE, stable SMOTE, Borderline-SMOTE, stable Borderline-SMOTE, ADASYN, and stable ADASYN are adopted. The common hyperparameters of these techniques are the number of the nearest neighbor minority class instances K, the distance metric, and the final defect ratio of the datasets after being processed by the techniques. We set K as 5, select the Euclidean distance as the distance metric, and set the final defect ratio as 0.5, all of which follow the default settings of these techniques.

6.2. Datasets

In this study, we adopt 26 imbalanced datasets of 10 projects collected from the PROMISE repository [15]. The PROMISE repository is maintained by Jureczko and Madeyski, and has been widely used in many previous studies. All the selected datasets are open source, making it easy for others to compare and replicate our work. The defect ratio of the 26 selected is less than 40% to make sure oversampling techniques work properly. The information about these datasets is listed in Table 1. There are 20 metrics in each dataset measuring the complexity of an instance (e.g., Lines of Code, Weighted Methods per Class). All the values of these metrics are numeric. Besides these metrics, there is an additional metric to indicate whether the current instance is defective or non-defective. If the instance is defective, this metric will be 1. Otherwise, the metric will be labeled as 0. Table 2 details these metrics.

Table 2
Description of the metrics.

Abbreviation	Description
WMC	Weighted methods per class
DIT	Depth of Inheritance Tree
NOC	Number of Children
CBO	Coupling between object classes
RFC	Response for a Class
LCOM	Lack of cohesion in methods
CA	afferent couplings
CE	efferent couplings
NPM	Number of Public Methods
LCOM3	Lack of cohesion in methods, different from LCOM
LOC	Lines of Code
DAM	Data Access Metric
MOA	Measure of Aggregation
MFA	Measure of Functional Abstraction
CAM	Cohesion Among Methods of Class
IC	Inheritance Coupling
CBM	Coupling Between Methods
AMC	Average Method Complexity
MAX(CC)	Maximum value of CC methods of the investigated class
AVG(CC)	Arithmetic mean of the CC value in the investigated class

Table 3
Hyperparameter configuration.

Classifier	Hyperparameters
DT	critera = {gini, entropy}, depth = {30, 50, 60, 100}, leaf = {2, 3, 5, 10}
KNN	K = {1, 3, 5, 7, 9, 11}
RF	mtry = {10, 50, 100, 200, 500, 1000}
SVM	c = {0.01, 0.1, 1, 10, 100, 1000}

Table 4
Confusion matrix.

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

6.3. Classifiers

In this study, we aim at investigating the stability of SMOTE-based oversampling techniques instead of deciding which classifier is the best for building prediction models. Therefore, we adopt four common classifiers (i.e., K-Nearest Neighbor (KNN), Support Vector Machine (SVM), Random Forest (RF), and Decision Tree (DT)). We choose these four classifiers because they were adopted by many previous studies [11,34] in SDP and performed quite well. To avoid reinventing the wheel, we implemented these classifiers using the Sklearn package [35] in Python. The hyperparameters of these classifiers are optimized by the 5-fold grid search. The hyperparameter configuration is presented in Table 3.

6.4. Evaluation measures for imbalanced datasets

Because of the class imbalance problem in SDP, some performance measures such as accuracy are not appropriate to measure the performance of defect prediction models. The performance measures that are not significantly impacted by the class imbalance problem are adopted

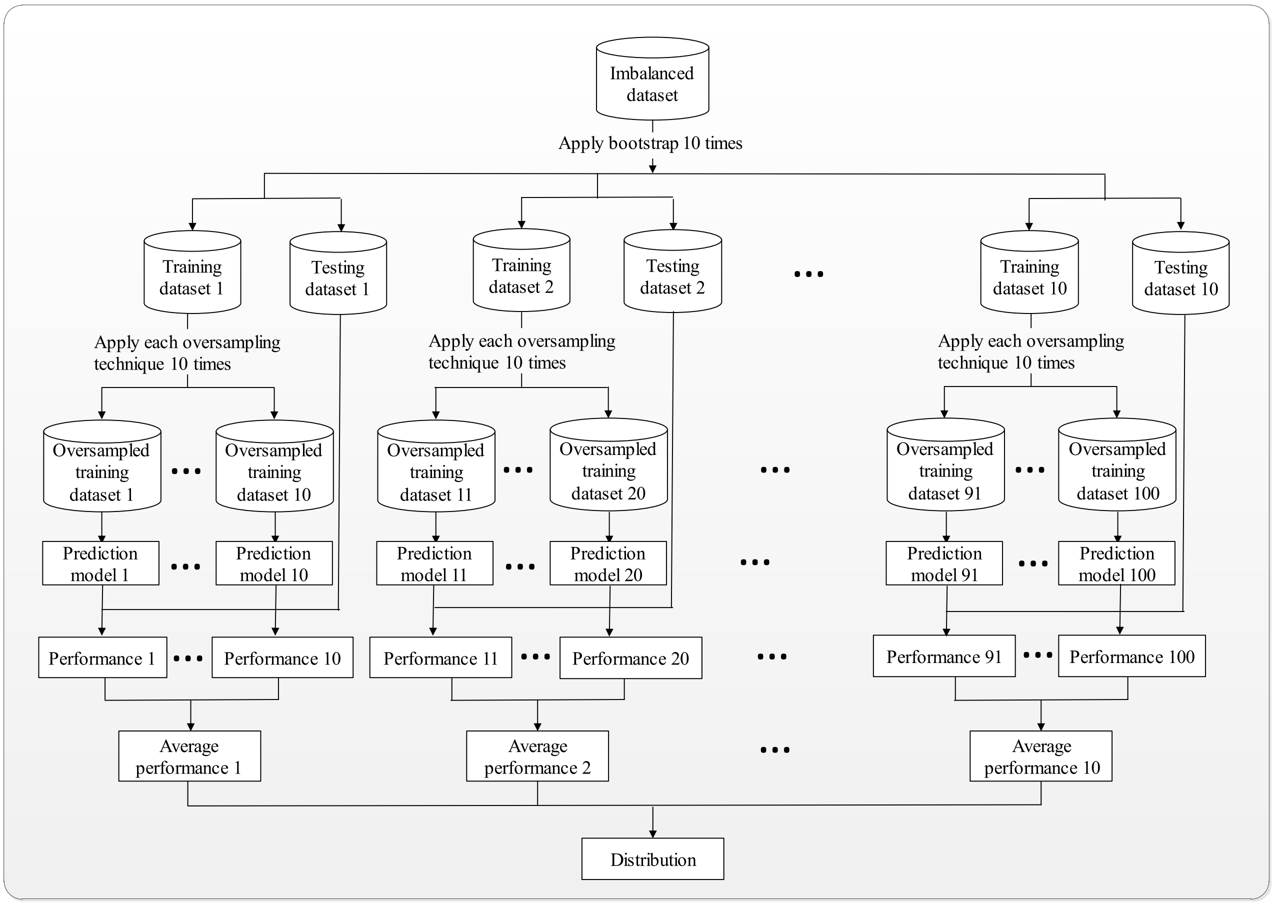


Fig. 1. Experimental procedure.

in SDP. In this study, we choose AUC, *balance*, and MCC to measure the performance. AUC is calculated from the Receiver Operating Characteristics (ROC) curve, and it is able to handle the trade-off between the true and false positive. *balance* is another common performance measure that finds the balance between *pd* (probability of detection) and *pf* (probability of false alarm). We also adopt MCC, which has been confirmed as the most suitable performance measure for the presence of imbalanced data in the recent study [36]. These performance measures were widely adopted by the previous studies in SDP [11, 34,37]. Their performances are stable and satisfactory. Adopting these common performance measures could improve the generalizability of our conclusion, and it is also easy for others to replicate our work. These performance measures are computed based on the outcomes of the confusion matrix (Table 4). Generally, the minority class instances are regarded as positive and the majority class instances as negative. True Positive (TP) represents the positive instances that are correctly classified. False Positive (FP) represents the negative instances that are misclassified as positive. True Negative (TN) represents negative instances that are correctly classified, and False Negative (FN) represents the positive instances that are misclassified as negative. The mathematical definitions of *balance* and MCC are given below:

$$pd = \frac{TP}{TP + FN} \tag{16}$$

$$pf = \frac{FP}{TN + FP} \tag{17}$$

$$balance = 1 - \frac{\sqrt{(0 - pf)^2 + (1 - pd)^2}}{\sqrt{2}}, \tag{18}$$

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{19}$$

Higher values of AUC, *balance*, and MCC represent the better overall performance of prediction models.

6.5. Statistical test

We apply the Scott–Knott Effect Size Difference test (the Scott–Knott ESD test) [37] to make multiple comparisons among the selected oversampling techniques. The Scott–Knott ESD test uses a hierarchical clustering algorithm to divide the set of treatment means into statistically different groups like the Scott–Knott test. The Scott–Knott test assumes that the data should be normally distributed, and the groups created by the Scott–Knott test may be trivially different from one another. The Scott–Knott ESD test log-transforms the data and considers the effect size to solve these limitations.

The Wilcoxon signed-rank test (Wilcoxon) is used to make a pairwise comparison between samples. We also adopt it to check whether the variances of AUC, *balance*, and MCC of each SMOTE-based oversampling technique is statistically significantly different from that of its corresponding stable SMOTE-based oversampling technique. Wilcoxon is a non-parametric test that takes the null hypothesis that two techniques are not significantly different.

In this study, we employ the two tests both at 95% confidence level.

The effect size (i.e., Cliff's δ) between the variance of each SMOTE-based oversampling technique and that of its corresponding stable SMOTE-based oversampling technique is also computed to ascertain the practical significance of the experimental results. By convention, the effect size is interpreted as negligible ($0 <$ Cliff's $\delta < 0.147$), small ($0.147 <$ Cliff's $\delta < 0.33$), medium ($0.33 <$ Cliff's $\delta < 0.474$) or large (Cliff's $\delta > 0.474$) [11].

We also adopt the win-draw-loss strategy to investigate the performance of each technique across every single dataset.

6.6. Experimental procedure

In this study, we first apply the min–max normalization method to the studied datasets to adjust all the features into the same range from 0 to 1. Next, we validate the performances of different oversampling techniques by adopting the out-of-sample bootstrap validation technique to divide the selected datasets into the training and testing data. The reason that we adopt the out-of-sample bootstrap is that the training and testing data generated by the out-of-sample bootstrap can balance the bias and variance well [37]. Specifically, we apply the out-of-sample bootstrap to each selected dataset ten times and get ten different pairs of the training and testing data. Then for each pair, we apply each of SMOTE-based and stable SMOTE-based oversampling techniques ten times only to the training data and keep the testing data unchanged. We terminate the oversampling process when the number of minority class instances and that of majority class instances are equal according to Ahmad Abu's conclusion [38] that oversampling techniques perform the best when the number is equal. After that, we will get ten groups of training data, each of which includes ten oversampled training data. Then we use the $10 \times 10 = 100$ oversampled training data to build $10 \times 10 = 100$ prediction models for each dataset, and we get $26 \times 10 \times 10 = 2600$ outcomes in total. First, to compare the overall performance of each technique, we calculate the average performance of the 100 prediction models for each dataset in terms of AUC, $balance$, and MCC. Because we adopt 26 datasets in this paper, we obtain 26 values in terms of each performance measure. We take these values as a distribution and use the Scott–Knott ESD test to divide each distribution of AUC, $balance$, and MCC of all studied techniques into groups with a significant difference. Then we calculate the best, upper quartile, median, lower quartile, and worst performances of each technique in terms of AUC, $balance$, MCC, and divide them into different groups with a significant difference using the Scott–Knott ESD test just like the way obtaining the average performance. Next, we calculate the variance of the average performance of each technique in terms of AUC, $balance$, and MCC. Then we use Wilcoxon as well as Cliff's δ to compare the variance of each SMOTE-based oversampling technique against that of its corresponding stable SMOTE-based oversampling technique. Fig. 1 shows the whole workflow of the experiment.

7. Experimental results and analysis

In this section, we first compare the overall performance of SMOTE-based with stable SMOTE-based oversampling techniques in terms of AUC, $balance$, and MCC using the Scott–Knott ESD test. Then we present the best, upper quartile, median, lower quartile, worst, and average performances of each oversampling technique in terms of AUC, $balance$, and MCC. We also compare the variance of the performance of each oversampling technique with regard to AUC, $balance$, and MCC. To ease the demonstration of experimental results, we refer to stable SMOTE as S-SMOTE, Borderline-SMOTE as Borderline, stable Borderline-SMOTE as S-Borderline, and stable ADASYN as S-ADASYN in this section.

7.1. The overall performance

Fig. 2 presents the Scott–Knott ESD test ranking of each oversampling technique in terms of AUC, $balance$, and MCC across all studied datasets on the four selected classifiers. The leftmost boxplot represents the technique performing the best, while the rightmost boxplot represents the worst one. None represents the classifiers trained using the original data without any processing. First, it can be seen that the data processed by the oversampling techniques indeed significantly improves the performance of prediction models compared with the original data. Besides, we can see that there is no significant difference between each stable SMOTE-based oversampling technique and its corresponding SMOTE-based oversampling technique in terms of AUC, $balance$, and MCC, except that S-ADASYN significantly outperforms

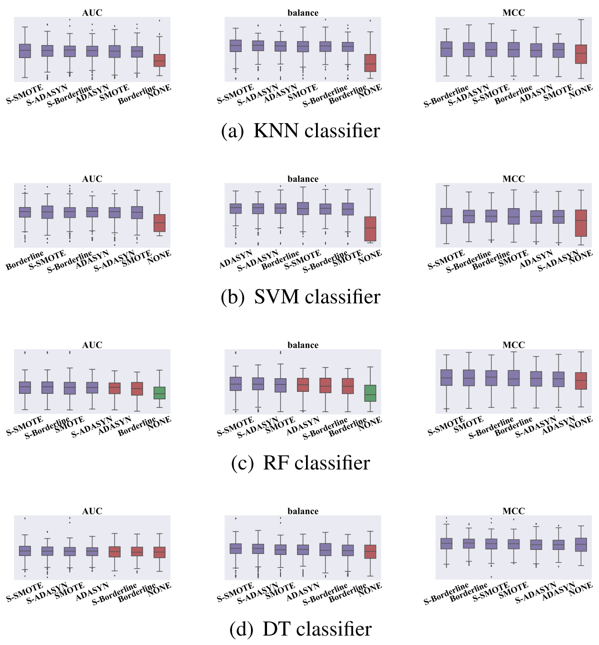


Fig. 2. The Scott–Knott ESD test ranking of AUC, $balance$, and MCC values of each oversampling technique across 26 datasets (purple>red>green). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

ADASYN in terms of AUC and $balance$ on the RF classifier. Although there is no significant difference, it can be seen that each stable SMOTE-based oversampling technique outperforms its corresponding technique on most classifiers. From Fig. 2, we can see that the overall performance of our proposed techniques is quite satisfactory compared with SMOTE-based oversampling techniques.

7.2. The best, upper quartile, median, lower quartile, worst, and average performances

Then we compare the best, upper quartile, median, lower quartile, worst, and average performances of each oversampling technique in terms of AUC, $balance$, and MCC. If the best, upper quartile, median, lower quartile, worst, and average performances of a certain SMOTE-based oversampling technique are divided into more groups than that of its corresponding stable SMOTE-based oversampling technique by the Scott–Knott ESD test, it indicates that the latter technique is more stable than the former one.

Fig. 3 shows that the best, upper quartile, median, lower quartile, worst, and average performances of each SMOTE-based oversampling technique are divided into more groups than that of its corresponding stable technique on the KNN and SVM classifiers in terms of AUC. It should be noted that because of the initial selection of instances in ADASYN and S-ADASYN, S-ADASYN generates exactly the same synthetic instances for a certain dataset. Therefore, the performances of S-ADASYN remain the same on the KNN and SVM classifiers. However, on the RF and DT classifiers, the performances of all oversampling techniques are all divided into five groups in terms of AUC. This is probably because of the randomness of the classifiers counteracting the positive effect brought by stable SMOTE-based oversampling techniques. Fig. 4 shows that with regard to $balance$, the performances of each SMOTE-based oversampling technique are divided into more groups by the Scott–Knott ESD test than that of its corresponding stable technique on the KNN, SVM, and DT classifiers. On the RF classifier, the performances of S-Borderline are divided into four groups, while those of Borderline are in five groups in terms of $balance$. In terms of MCC, it can be seen that the stability of each stable SMOTE-based oversampling

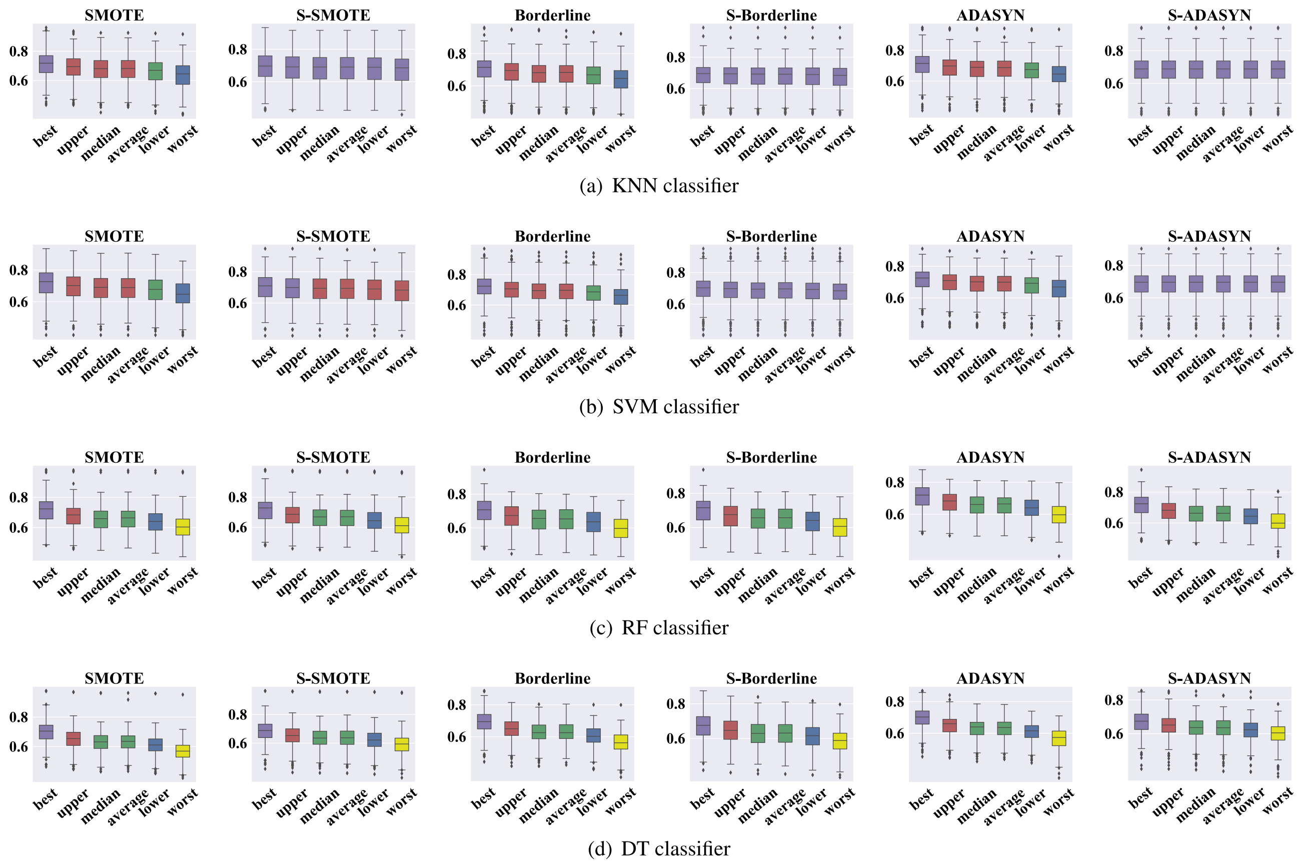


Fig. 3. The Scott–Knott ESD test ranking of the best, upper quartile, median, average, lower quartile, and worst performances of each oversampling technique in terms of AUC across 26 datasets (purple>red>green>blue>yellow). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

technique consistently outperforms that of its corresponding technique from Fig. 5.

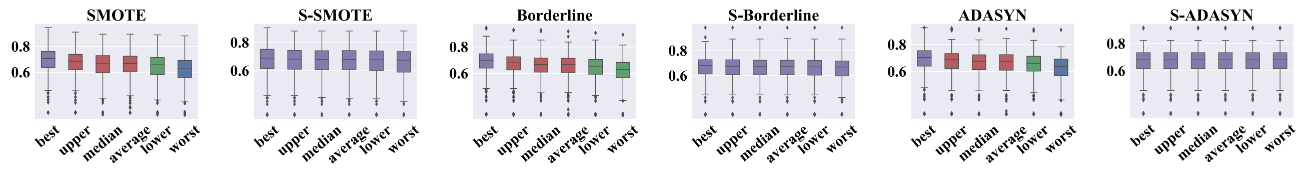
It is notable that the difference between the best and worst performances of SMOTE is up to 13.1%, 10.5%, 19.4%, and 23.3% on the KNN, SVM, RF, and DT classifiers, while that of S-SMOTE is only 2.5%, 3.3%, 17.1%, and 15.9% in terms of AUC. With regard to *balance*, the difference is up to 13.7%, 12.3%, 28.2%, and 32.6% for SMOTE on the KNN, SVM, RF, and DT classifiers. Considering MCC, the difference is as large as 59.5%, 47.2%, 112.9%, and 204.2%, respectively. The performance of Borderline and ADASYN is similar to that of SMOTE.

7.3. The variance

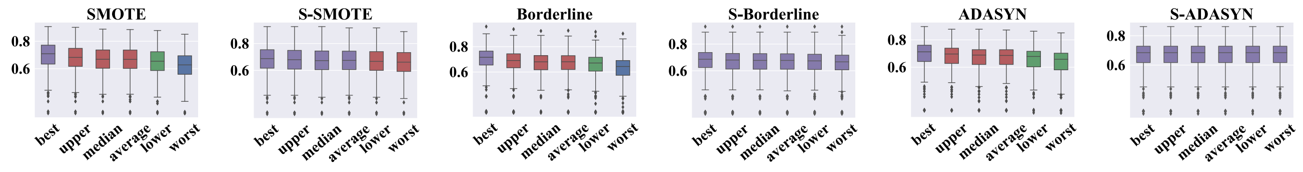
To further explore the stability of SMOTE-based and stable SMOTE-based oversampling techniques, we take the win-draw-loss strategy to compare the stability of stable SMOTE-based oversampling techniques against SMOTE-based oversampling techniques across every single dataset on the four classifiers in terms of MCC, because MCC is the most suitable performance measure in the presence of imbalanced datasets. Tables 5, 6, 7, and 8 present the values and the variances of MCC of each technique across every single dataset on each classifier. W/D/L in these tables represents that each stable SMOTE-based oversampling technique performs better than, the same as, or worse than its corresponding technique in terms of the values and variances of MCC. *p*-value indicates whether there exists a significant difference between each stable SMOTE-based oversampling technique and its corresponding SMOTE-based oversampling technique. The effect size is also computed.

From these tables, we can observe that stable SMOTE-based oversampling techniques consistently obtain positive win-loss values in terms of the variances as well as the values of MCC. Specifically, the variances of stable SMOTE-based oversampling techniques on the KNN, SVM, and DT classifiers are smaller than their corresponding techniques across every single dataset. The statistical test also confirms the superiority of stable SMOTE-based oversampling techniques. On the KNN classifier, S-SMOTE, S-Borderline, and S-ADASYN significantly outperform SMOTE, Borderline, and ADASYN in terms of both the variances and values of MCC. The effect sizes also achieve large. A similar trend can also be observed on the SVM and DT classifiers. Because of the randomness of the RF classifier, stable SMOTE-based oversampling techniques fail to outperform their corresponding techniques across every single dataset. However, they still obtain much smaller variances on most datasets.

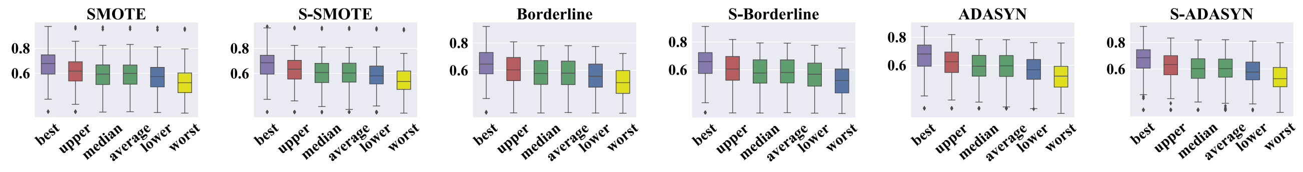
Based on the above experimental results, we can see that the performance of SMOTE-based oversampling techniques is very unstable. The difference between the best and worst performances of SMOTE-based oversampling techniques can be as much as 23.3%, 32.6%, and 204.2% in terms of AUC, *balance*, and MCC. With such high variance, practitioners will have less confidence in the results produced by SMOTE-based oversampling techniques. Moreover, as common baselines, the instability of SMOTE-based oversampling techniques will make the conclusion drawn from the comparison between SMOTE-based oversampling techniques and the newly proposed techniques less convincing. Meanwhile, the performances of stable SMOTE-based oversampling techniques are better and much more stable than those of SMOTE-based oversampling techniques, especially S-ADASYN, which generates fixed data for a particular dataset. Therefore, we conclude that stable SMOTE-based oversampling techniques are superior to SMOTE-based oversampling techniques and should be considered as the effective alternatives for SMOTE-based oversampling techniques.



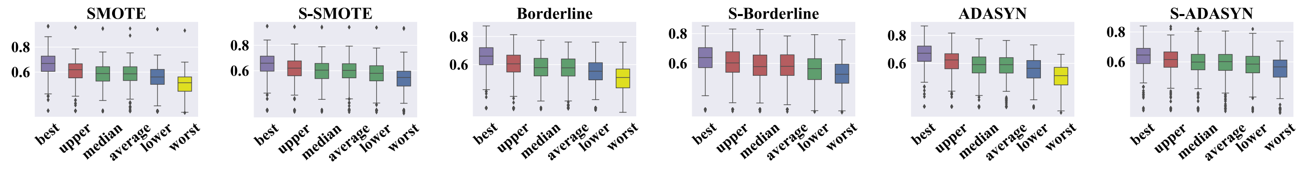
(a) KNN classifier



(b) SVM classifier



(c) RF classifier

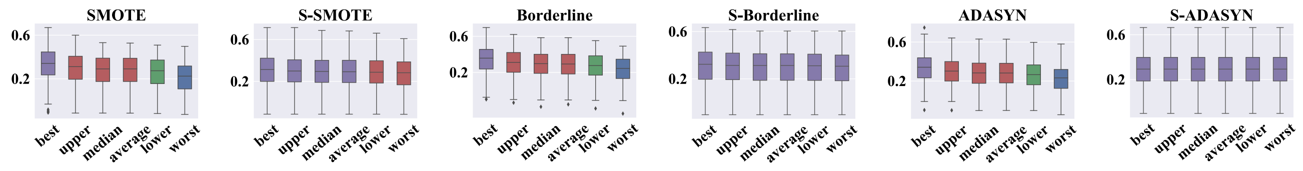


(d) DT classifier

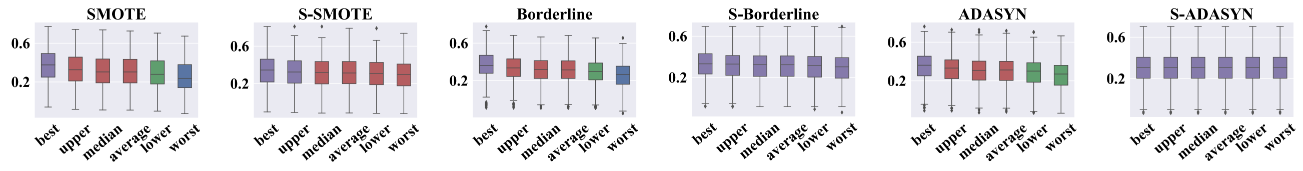
Fig. 4. The Scott–Knott ESD test ranking of the best, upper quartile, median, average, lower quartile, and worst performances of each oversampling technique in terms of *balance* across 26 datasets (purple>red>green>blue>yellow). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 5
The performance of SMOTE-based and stable SMOTE-based oversampling techniques on the KNN classifier in terms of MCC across 26 datasets.

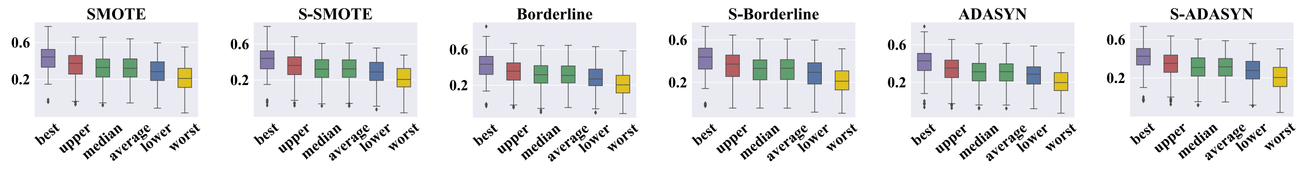
	SMOTE		S-SMOTE		Borderline		S-Borderline		ADASYN		S-ADASYN	
	MCC	var	MCC	var	MCC	var	MCC	var	MCC	var	MCC	var
ant-1.3	0.208	0.062	0.188	0.009	0.313	0.058	0.335	0.006	0.266	0.046	0.286	0.000
ant-1.4	0.145	0.067	0.143	0.026	0.097	0.063	0.121	0.011	0.087	0.066	0.116	0.000
ant-1.5	0.298	0.036	0.288	0.007	0.294	0.031	0.307	0.001	0.285	0.036	0.275	0.000
ant-1.6	0.415	0.035	0.438	0.012	0.416	0.028	0.445	0.006	0.403	0.030	0.426	0.000
ant-1.7	0.347	0.023	0.366	0.009	0.361	0.022	0.367	0.004	0.360	0.020	0.365	0.000
camel-1.0	0.058	0.041	0.083	0.000	0.046	0.011	0.036	0.000	0.111	0.029	0.109	0.000
camel-1.2	0.157	0.036	0.180	0.017	0.180	0.031	0.196	0.015	0.163	0.025	0.164	0.000
camel-1.4	0.148	0.026	0.178	0.007	0.150	0.025	0.150	0.005	0.140	0.019	0.155	0.000
camel-1.6	0.216	0.028	0.218	0.005	0.215	0.025	0.213	0.007	0.239	0.023	0.232	0.000
ivy-1.4	0.030	0.015	0.034	0.001	0.011	0.021	0.019	0.000	0.024	0.024	0.034	0.000
ivy-2.0	0.211	0.039	0.234	0.006	0.241	0.028	0.246	0.003	0.230	0.025	0.233	0.000
jedit-3.2	0.462	0.034	0.458	0.012	0.471	0.036	0.471	0.013	0.396	0.033	0.425	0.000
jedit-4.0	0.433	0.039	0.450	0.012	0.422	0.031	0.457	0.008	0.433	0.030	0.456	0.000
jedit-4.1	0.431	0.034	0.453	0.013	0.414	0.028	0.432	0.009	0.398	0.031	0.433	0.000
jedit-4.2	0.299	0.028	0.323	0.007	0.339	0.021	0.346	0.001	0.281	0.020	0.296	0.000
jedit-4.3	0.156	0.047	0.140	0.009	0.108	0.049	0.049	0.009	0.160	0.046	0.154	0.000
log4j-1.0	0.373	0.055	0.440	0.022	0.375	0.056	0.443	0.012	0.365	0.037	0.385	0.000
log4j-1.1	0.424	0.065	0.455	0.041	0.444	0.063	0.483	0.009	0.464	0.057	0.509	0.000
poi-2.0	0.167	0.034	0.173	0.004	0.250	0.030	0.283	0.003	0.175	0.041	0.192	0.000
synapse-1.0	0.248	0.034	0.271	0.002	0.286	0.036	0.296	0.000	0.278	0.032	0.317	0.000
synapse-1.1	0.309	0.048	0.321	0.017	0.354	0.039	0.345	0.009	0.335	0.046	0.331	0.000
synapse-1.2	0.374	0.041	0.365	0.018	0.350	0.037	0.371	0.004	0.408	0.043	0.385	0.000
velocity-1.6	0.328	0.048	0.342	0.022	0.353	0.049	0.364	0.017	0.313	0.045	0.339	0.000
xalan-2.4	0.290	0.021	0.303	0.006	0.284	0.020	0.276	0.003	0.249	0.020	0.267	0.000
xerces-1.2	0.213	0.032	0.207	0.006	0.239	0.034	0.233	0.006	0.227	0.031	0.215	0.000
xerces-1.3	0.393	0.029	0.405	0.004	0.401	0.023	0.401	0.006	0.402	0.023	0.413	0.000
average	0.274	0.038	0.287	0.011	0.285	0.034	0.294	0.006	0.277	0.034	0.289	0.000
W/D/L	7/0/19	0/0/26	19/0/7	26/0/0	6/3/17	0/0/26	17/3/6	26/0/0	7/0/19	0/0/26	19/0/7	26/0/0
p-value			< .05	< .05			< .05	< .05			< .05	< .05
Cliff's δ			0.062	0.917			0.056	0.988			0.059	1.000



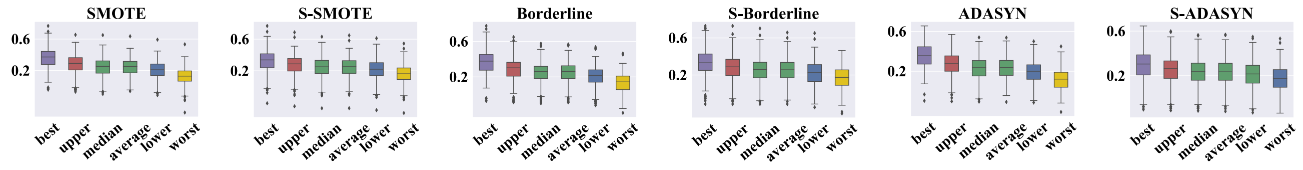
(a) KNN classifier



(b) SVM classifier



(c) RF classifier



(d) DT classifier

Fig. 5. The Scott–Knott ESD test ranking of the best, upper quartile, median, average, lower quartile, and worst performances of each oversampling technique in terms of MCC across 26 datasets (purple>red>green>blue>yellow). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 6
The performance of SMOTE-based and stable SMOTE-based oversampling techniques on the SVM classifier in terms of MCC across 26 datasets.

	SMOTE		S-SMOTE		Borderline		S-Borderline		ADASYN		S-ADASYN	
	MCC	var	MCC	var	MCC	var	MCC	var	MCC	var	MCC	var
ant-1.3	0.336	0.056	0.334	0.015	0.380	0.052	0.410	0.006	0.291	0.042	0.283	0.000
ant-1.4	0.244	0.060	0.280	0.023	0.087	0.077	0.128	0.015	0.140	0.043	0.117	0.000
ant-1.5	0.253	0.036	0.246	0.007	0.357	0.037	0.341	0.007	0.293	0.038	0.281	0.000
ant-1.6	0.479	0.026	0.496	0.013	0.423	0.031	0.423	0.012	0.437	0.017	0.437	0.000
ant-1.7	0.424	0.021	0.420	0.010	0.379	0.018	0.395	0.006	0.414	0.018	0.406	0.000
camel-1.0	0.039	0.012	0.042	0.000	0.152	0.042	0.127	0.000	0.060	0.012	0.064	0.000
camel-1.2	0.213	0.031	0.227	0.019	0.222	0.035	0.218	0.021	0.217	0.026	0.224	0.000
camel-1.4	0.225	0.024	0.226	0.008	0.260	0.021	0.249	0.010	0.249	0.022	0.248	0.000
camel-1.6	0.229	0.022	0.229	0.009	0.252	0.020	0.251	0.008	0.283	0.016	0.285	0.000
ivy-1.4	0.079	0.036	0.083	0.002	0.084	0.044	0.098	0.001	0.125	0.044	0.106	0.000
ivy-2.0	0.301	0.051	0.329	0.012	0.290	0.031	0.269	0.001	0.296	0.026	0.305	0.000
jedit-3.2	0.542	0.034	0.554	0.015	0.523	0.029	0.512	0.011	0.526	0.029	0.511	0.000
jedit-4.0	0.385	0.041	0.388	0.017	0.371	0.035	0.364	0.008	0.308	0.028	0.327	0.000
jedit-4.1	0.495	0.037	0.519	0.012	0.390	0.030	0.402	0.010	0.463	0.026	0.498	0.000
jedit-4.2	0.351	0.024	0.360	0.006	0.369	0.036	0.395	0.012	0.342	0.023	0.338	0.000
jedit-4.3	0.056	0.016	0.064	0.008	0.053	0.009	0.078	0.000	0.118	0.024	0.144	0.000
log4j-1.0	0.525	0.047	0.551	0.020	0.377	0.062	0.369	0.016	0.489	0.047	0.483	0.000
log4j-1.1	0.465	0.057	0.496	0.021	0.538	0.058	0.523	0.019	0.560	0.038	0.555	0.000
poi-2.0	0.206	0.041	0.214	0.012	0.249	0.028	0.254	0.003	0.167	0.009	0.178	0.000
synapse-1.0	0.373	0.036	0.372	0.006	0.435	0.031	0.425	0.000	0.328	0.025	0.325	0.000
synapse-1.1	0.356	0.047	0.357	0.022	0.342	0.046	0.341	0.018	0.363	0.033	0.358	0.000
synapse-1.2	0.451	0.035	0.434	0.020	0.390	0.043	0.366	0.017	0.409	0.034	0.392	0.000
velocity-1.6	0.303	0.046	0.316	0.018	0.414	0.036	0.401	0.024	0.387	0.043	0.404	0.000
xalan-2.4	0.268	0.023	0.273	0.009	0.280	0.021	0.291	0.010	0.253	0.024	0.240	0.000
xerces-1.2	0.023	0.039	0.039	0.014	0.073	0.041	0.073	0.014	0.059	0.026	0.060	0.000
xerces-1.3	0.314	0.029	0.302	0.010	0.356	0.026	0.356	0.004	0.321	0.019	0.322	0.000
average	0.305	0.036	0.314	0.013	0.309	0.036	0.310	0.010	0.304	0.028	0.304	0.000
W/D/L	6/1/19	0/0/26	19/1/6	26/0/0	14/3/9	0/0/26	9/3/14	26/0/0	14/1/11	0/0/26	11/1/14	26/0/0
p-value			< .05	< .05			> .05	< .05			> .05	< .05
Cliff's δ			0.056	0.917			0.003	0.932			0.012	1.000

Table 7
The performance of SMOTE-based and stable SMOTE-based oversampling techniques on the RF classifier in terms of MCC across 26 datasets.

	SMOTE		S-SMOTE		Borderline		S-Borderline		ADASYN		S-ADASYN	
	MCC	var	MCC	var	MCC	var	MCC	var	MCC	var	MCC	var
ant-1.3	0.278	0.138	0.286	0.125	0.330	0.118	0.348	0.112	0.209	0.131	0.218	0.125
ant-1.4	0.196	0.097	0.183	0.085	0.207	0.094	0.232	0.094	0.211	0.104	0.204	0.092
ant-1.5	0.284	0.089	0.287	0.097	0.301	0.081	0.313	0.080	0.316	0.063	0.301	0.062
ant-1.6	0.452	0.050	0.445	0.051	0.444	0.065	0.442	0.056	0.431	0.050	0.431	0.046
ant-1.7	0.428	0.039	0.432	0.037	0.408	0.043	0.405	0.037	0.403	0.037	0.406	0.044
camel-1.0	0.204	0.108	0.214	0.083	0.107	0.074	0.106	0.061	0.116	0.107	0.110	0.094
camel-1.2	0.220	0.052	0.219	0.048	0.195	0.052	0.187	0.052	0.213	0.053	0.214	0.051
camel-1.4	0.214	0.043	0.220	0.046	0.234	0.043	0.229	0.038	0.216	0.051	0.218	0.047
camel-1.6	0.205	0.043	0.219	0.035	0.210	0.041	0.214	0.034	0.216	0.043	0.228	0.041
ivy-1.4	0.085	0.085	0.101	0.075	0.049	0.104	0.033	0.084	0.033	0.073	0.043	0.068
ivy-2.0	0.249	0.078	0.254	0.085	0.241	0.063	0.256	0.075	0.287	0.078	0.288	0.064
jedit-3.2	0.472	0.057	0.488	0.053	0.504	0.049	0.506	0.046	0.458	0.056	0.478	0.052
jedit-4.0	0.379	0.050	0.346	0.049	0.382	0.071	0.363	0.063	0.400	0.051	0.381	0.053
jedit-4.1	0.445	0.051	0.455	0.055	0.461	0.052	0.471	0.047	0.479	0.057	0.476	0.052
jedit-4.2	0.359	0.071	0.380	0.061	0.379	0.050	0.386	0.055	0.339	0.063	0.352	0.056
jedit-4.3	0.260	0.028	0.252	0.025	0.170	0.024	0.137	0.034	0.169	0.030	0.168	0.025
log4j-1.0	0.411	0.085	0.428	0.084	0.374	0.087	0.398	0.069	0.378	0.091	0.408	0.076
log4j-1.1	0.463	0.086	0.474	0.075	0.468	0.083	0.481	0.083	0.452	0.082	0.451	0.079
poi-2.0	0.254	0.084	0.275	0.080	0.270	0.072	0.269	0.073	0.283	0.072	0.285	0.076
synapse-1.0	0.245	0.094	0.225	0.091	0.207	0.112	0.250	0.116	0.218	0.080	0.231	0.087
synapse-1.1	0.419	0.074	0.423	0.065	0.388	0.075	0.410	0.056	0.391	0.073	0.410	0.067
synapse-1.2	0.448	0.065	0.434	0.056	0.424	0.056	0.417	0.063	0.400	0.054	0.408	0.062
velocity-1.6	0.361	0.062	0.354	0.062	0.336	0.073	0.335	0.063	0.344	0.066	0.341	0.067
xalan-2.4	0.263	0.046	0.259	0.050	0.253	0.047	0.259	0.053	0.266	0.050	0.265	0.055
xerces-1.2	0.298	0.057	0.291	0.051	0.293	0.052	0.279	0.050	0.302	0.057	0.306	0.052
xerces-1.3	0.365	0.057	0.354	0.056	0.380	0.064	0.401	0.054	0.377	0.059	0.354	0.057
average	0.318	0.069	0.319	0.064	0.308	0.067	0.313	0.063	0.304	0.067	0.307	0.063
W/D/L	11/0/15	6/1/19	15/0/11	19/1/6	12/0/14	7/3/16	14/0/12	16/3/7	10/1/15	7/0/19	15/1/10	19/0/7
p-value			> .05	< .05			> .05	> .05			> .05	> .05
Cliff's δ			0.015	0.080			0.044	0.092			0.044	0.012

Table 8
The performance of SMOTE-based and stable SMOTE-based oversampling techniques on the DT classifier in terms of MCC across 26 datasets.

	SMOTE		S-SMOTE		Borderline		S-Borderline		ADASYN		S-ADASYN	
	MCC	var	MCC	var	MCC	var	MCC	var	MCC	var	MCC	var
ant-1.3	0.172	0.132	0.149	0.087	0.173	0.117	0.123	0.078	0.204	0.135	0.230	0.087
ant-1.4	0.141	0.096	0.153	0.066	0.169	0.090	0.206	0.065	0.141	0.095	0.205	0.049
ant-1.5	0.224	0.097	0.216	0.055	0.189	0.084	0.183	0.044	0.190	0.089	0.214	0.039
ant-1.6	0.330	0.063	0.325	0.043	0.337	0.069	0.349	0.058	0.354	0.061	0.333	0.037
ant-1.7	0.296	0.048	0.302	0.040	0.296	0.051	0.288	0.040	0.297	0.062	0.291	0.024
camel-1.0	0.035	0.071	0.046	0.048	0.098	0.065	0.068	0.015	0.084	0.067	0.088	0.039
camel-1.2	0.178	0.054	0.188	0.048	0.171	0.052	0.180	0.047	0.158	0.054	0.174	0.025
camel-1.4	0.175	0.045	0.170	0.031	0.162	0.049	0.183	0.034	0.173	0.047	0.166	0.025
camel-1.6	0.182	0.043	0.183	0.030	0.176	0.044	0.198	0.031	0.156	0.044	0.147	0.020
ivy-1.4	0.026	0.090	0.046	0.063	0.040	0.089	0.052	0.069	0.007	0.077	0.010	0.049
ivy-2.0	0.208	0.090	0.207	0.065	0.216	0.087	0.197	0.059	0.217	0.082	0.184	0.047
jedit-3.2	0.361	0.069	0.364	0.052	0.346	0.078	0.344	0.046	0.354	0.076	0.367	0.036
jedit-4.0	0.335	0.067	0.326	0.049	0.317	0.062	0.307	0.053	0.307	0.077	0.325	0.039
jedit-4.1	0.328	0.063	0.320	0.056	0.358	0.068	0.365	0.054	0.332	0.068	0.308	0.041
jedit-4.2	0.280	0.067	0.288	0.046	0.289	0.075	0.299	0.045	0.243	0.067	0.215	0.040
jedit-4.3	0.120	0.041	0.129	0.033	0.133	0.062	0.150	0.046	0.153	0.052	0.150	0.040
log4j-1.0	0.317	0.101	0.309	0.059	0.283	0.108	0.268	0.076	0.306	0.108	0.322	0.063
log4j-1.1	0.453	0.098	0.433	0.073	0.447	0.097	0.467	0.089	0.330	0.098	0.351	0.061
poi-2.0	0.182	0.092	0.211	0.049	0.283	0.089	0.271	0.041	0.248	0.077	0.255	0.043
synapse-1.0	0.212	0.093	0.229	0.080	0.174	0.089	0.202	0.080	0.171	0.095	0.193	0.046
synapse-1.1	0.310	0.081	0.322	0.065	0.325	0.069	0.326	0.056	0.296	0.077	0.318	0.044
synapse-1.2	0.295	0.066	0.273	0.049	0.315	0.070	0.289	0.061	0.312	0.078	0.312	0.040
velocity-1.6	0.277	0.068	0.279	0.054	0.288	0.073	0.285	0.049	0.308	0.076	0.262	0.040
xalan-2.4	0.215	0.048	0.205	0.043	0.221	0.060	0.235	0.039	0.199	0.054	0.220	0.027
xerces-1.2	0.270	0.060	0.315	0.035	0.265	0.057	0.279	0.037	0.229	0.053	0.191	0.029
xerces-1.3	0.285	0.054	0.290	0.040	0.306	0.056	0.323	0.040	0.278	0.060	0.244	0.038
average	0.239	0.073	0.241	0.052	0.245	0.073	0.248	0.052	0.233	0.074	0.233	0.041
W/D/L	11/0/15	0/0/26	15/0/11	26/0/0	11/0/15	0/0/26	15/0/11	26/0/0	11/1/14	0/0/26	14/1/11	26/0/0
p-value			> .05	< .05			> .05	< .05			> .05	< .05
Cliff's δ			0.021	0.556			0.036	0.607			0.030	0.870

Table 9
The performance of SMOTE and stable SMOTE in terms of MCC across 26 datasets under the balanced ratio of 0.4.

KNN	SMOTE	S-SMOTE	p-value	Cliff's δ
MCC	0.289	**0.301**	< .05	0.059
variance	0.043	**0.018**	< .05	0.840
SVM				
MCC	0.306	**0.312**	< .05	0.041
variance	0.037	**0.018**	< .05	0.737
RF				
MCC	0.306	**0.311**	> .05	0.024
variance	0.070	**0.067**	> .05	0.092
DT				
MCC	0.231	**0.239**	> .05	0.041
variance	0.073	**0.056**	< .05	0.503

Bold indicates better values.

Table 10
The performance of SMOTE and stable SMOTE in terms of MCC across 26 datasets under the balanced ratio of 0.6.

KNN	SMOTE	S-SMOTE	p-value	Cliff's δ
MCC	0.277	**0.291**	< .05	0.080
variance	0.034	**0.008**	< .05	0.976
SVM				
MCC	0.281	**0.285**	> .05	0.030
variance	0.031	**0.011**	< .05	0.923
RF				
MCC	0.307	**0.312**	< .05	0.036
variance	0.062	**0.059**	> .05	0.044
DT				
MCC	0.236	**0.241**	> .05	0.033
variance	0.075	**0.051**	< .05	0.598

Bold indicates better values.

Table 11
The performance of SMOTE and stable SMOTE in terms of MCC across 26 datasets under the setting of K equaling 3.

KNN	SMOTE	S-SMOTE	p-value	Cliff's δ
MCC	0.273	**0.287**	< .05	0.074
variance	0.036	**0.011**	< .05	0.891
SVM				
MCC	0.268	**0.268**	> .05	0.021
variance	0.031	**0.010**	< .05	0.805
RF				
MCC	0.315	**0.317**	> .05	0.012
variance	0.068	**0.066**	> .05	0.041
DT				
MCC	**0.237**	0.235	> .05	0.021
variance	0.072	**0.052**	< .05	0.633

Bold indicates better values.

Table 12
The performance of SMOTE and stable SMOTE in terms of MCC across 26 datasets under the setting of K equaling 7.

KNN	SMOTE	S-SMOTE	p-value	Cliff's δ
MCC	0.284	**0.296**	< .05	0.041
variance	0.037	**0.010**	< .05	0.997
SVM				
MCC	0.267	**0.268**	> .05	0.009
variance	0.029	**0.011**	< .05	0.757
RF				
MCC	0.315	**0.315**	> .05	0.006
variance	0.067	**0.061**	< .05	0.175
DT				
MCC	0.238	**0.239**	> .05	0.047
variance	0.073	**0.050**	< .05	0.598

Bold indicates better values.

8. Discussion

In SMOTE-based oversampling techniques, the default balanced ratio of the number of minority class instances to that of all instances is set to 0.5. We adopt this default value in the above experiments. To generalize our conclusion, we adopt different defect ratios to re-conduct the experiments. If the experimental results are consistent with the above experimental results, our conclusion would be more general. In this section, we only present the values and the variances of MCC for SMOTE and S-SMOTE on the selected four classifiers under the balanced ratio of 0.4 and 0.6, respectively. From Tables 9 and 10, it can be seen that the performance of stable SMOTE consistently outperforms that of SMOTE in terms of MCC under different balanced ratios. Moreover, the variance of stable SMOTE is significantly smaller than that of SMOTE with practical effect sizes in terms of MCC. Therefore, our conclusion is consistent under different balanced ratios.

Besides, the default K value is 5 for SMOTE-based oversampling techniques. This value is adopted in the above experiments. Agrawal et al. [39] have argued that different K values affect the performance of SMOTE-based oversampling techniques. Therefore, we explore different K values to consolidate our conclusion. Specifically, we investigate the performance of SMOTE and stable SMOTE under the setting of K equaling 3 and 7. From Tables 11 and 12, we observe that our conclusion that stable SMOTE performs more stable than SMOTE is consistent under different K values.

Furthermore, we obtain

$$var(S_i) = \frac{2var(X_i)}{3} - \frac{var(X_i)}{3(r+1)} < var(S_i^{SMOTE}), \tag{20}$$

in Section 5. It is clear that with the increase of r, the variance of SMOTE becomes closer to that of stable SMOTE, but the variance of SMOTE cannot become smaller than that of stable SMOTE. Our work is based on Blagus's and Elreedy's work. They all assume that the generated instances follow the uniform distribution. There are some extremes that may generate synthetic instances with smaller variances than stable SMOTE. For example, if the synthetic instances are generated around the same point between minority class instances, the variance of the generated instances will be smaller than that of the instances generated by stable SMOTE. However, the probability of this case happening is extremely low, which is confirmed by our experiments. Besides, if the synthetic instances are generated around the same point, they will be too similar and lack diversity, which will lead to the overgeneralization of prediction models. In fact, the way that stable SMOTE generates synthetic instances is to improve the stability of SMOTE-based oversampling technique while remaining the diversity of the generated synthetic instances.

9. Threats to validity

In this section, we discuss the threats to the external, internal, and construct validity of our study.

9.1. External validity

External validity is the ability that could generate results outside the specifications of the current study [40]. To ensure the external study, we select 26 widely adopted datasets from the PROMISE repository. A large number of studies were conducted on these datasets, and all these datasets performed well. Besides, the defect metrics considered in this study may be a threat to our study. We only adopt the static code metrics. Thus, we cannot claim that we could generalize our conclusion to other types of metrics. However, the static code metrics were also widely adopted in many previous studies. Besides, the static code metrics are easy to collect and can be easily leveraged to replicate our work. For the choice of the classifiers, we only adopt the KNN, SVM, RF, and DT classifiers in this study. These four classifiers are common choices to build the prediction models in SDP, and their performances are satisfactory. However, there are many other available techniques, and we plan to investigate more prediction models in the future.

9.2. Internal validity

Internal validity is determined by how well a study can rule out alternative explanations for its findings. In this study, we select three commonly-used SMOTE-based oversampling techniques (i.e., SMOTE, Borderline-SMOTE, and ADASYN). These oversampling techniques are commonly used as the baseline techniques in many studies. However, there are some other SMOTE-based oversampling techniques, such as Safe-level-SMOTE [41] and AHC [42]. We intend to investigate more SMOTE-based oversampling techniques in the future study.

9.3. Construct validity

Construct validity is the degree to which a test measures what it claims, or purports, to be measuring. In this study, three common performance measures (i.e., AUC, $balance$, and MCC) are adopted to measure the performances of different oversampling techniques. These three performance measures are adopted widely in many previous studies to measure overall performance. Nonetheless, many other performance measures such as F-measure or G-measure are also worth investigating. We plan to explore more performance measures in our future work.

10. Conclusion and future work

SMOTE-based oversampling techniques are widely adopted in the area of SDP to alleviate the class imbalance problem. When SMOTE-based oversampling techniques are employed, randomness is introduced. However, as common baseline oversampling techniques, if the performance of SMOTE-based oversampling techniques is unstable, the conclusion drawn by comparing the performance of SMOTE-based oversampling techniques with that of newly proposed techniques is suspicious and less convincing.

In this study, we empirically investigate the performance of SMOTE-based oversampling techniques. We find that the performance of SMOTE-based oversampling techniques is highly unstable, and their instability negatively impacts the performance of prediction models in terms of AUC, $balance$, and MCC on the KNN, SVM, RF, and DT classifiers. To improve the stability of SMOTE-based oversampling techniques, we propose a series of stable SMOTE-based oversampling techniques. We mathematically and empirically validate that our proposed techniques produce more stable and better results than SMOTE-based oversampling techniques. Therefore, we recommend stable SMOTE-based oversampling techniques to be adopted as an effective alternative for SMOTE-based oversampling techniques.

In our future work, we plan to adopt more software datasets with different types of metrics, more classifiers, and more performance measures to generalize our conclusion. In addition, we also intend to apply stable SMOTE-based oversampling techniques to more SMOTE-based oversampling techniques such as AHC or Safe-level-SMOTE to investigate whether the performance of stable SMOTE-based oversampling techniques could still be stable and could also produce comparable or even better results. Moreover, a detailed investigation on which factor is the most influential to the instability of SMOTE-based oversampling techniques will be carried.

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.

Acknowledgments

This work is supported in part by the General Research Fund of the Research Grants Council of Hong Kong (No. 11208017) and the research funds of City University of Hong Kong (No. 7005028 and 7005217), and the Research Support Fund by Intel (No. 9220097), and funding supports from other industry partners (No. 9678149, 9440227, 9229029, 9440180, and 9220103).

References

[1] X. Chen, Y. Mu, K. Liu, Z. Cui, C. Ni, Revisiting heterogeneous defect prediction methods: How far are we?, Inf. Softw. Technol. 130 (2021) 106441.
[2] W. Li, W. Zhang, X. Jia, Z. Huang, Effort-aware semi-supervised just-in-time defect prediction, Inf. Softw. Technol. 126 (2020) 106364.
[3] A. Rahman, L. Williams, Source code properties of defective infrastructure as code scripts, Inf. Softw. Technol. 112 (2019) 148–163.
[4] N. Li, M. Shepperd, Y. Guo, A systematic review of unsupervised learning techniques for software defect prediction, Inf. Softw. Technol. 122 (2020) 106287.
[5] X. Chen, D. Zhang, Y. Zhao, Z. Cui, C. Ni, Software defect number prediction: Unsupervised vs supervised methods, Inf. Softw. Technol. 106 (2019) 161–181.
[6] F. Provost, Machine learning from imbalanced data sets 101, in: Proceedings of the AAAI'2000 Workshop on Imbalanced Data Sets, Vol. 68, AAAI Press, 2000, pp. 1–3.
[7] C. Pak, T.T. Wang, X.H. Su, An empirical study on software defect prediction using over-sampling by SMOTE, Int. J. Softw. Eng. Knowl. Eng. 28 (06) (2018) 811–830.
[8] 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.
[9] 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.
[10] 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.
[11] 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.
[12] 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.
[13] G.Y. Wong, F.H. Leung, S.H. Ling, A novel evolutionary preprocessing method based on over-sampling and under-sampling for imbalanced datasets, in: Iecon 2013-39th Annual Conference of the Ieee Industrial Electronics Society, IEEE, 2013, pp. 2354–2359.
[14] K.E. Bennin, J. Keung, A. Monden, P. Phannachitta, S. Mensah, The significant effects of data sampling approaches on software defect prioritization and classification, in: 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM, IEEE, 2017, pp. 364–373.
[15] G. Boetticher, The PROMISE repository of empirical software engineering data, 2007, http://promisedata.org/repository.
[16] F. Zhang, A.E. Hassan, S. McIntosh, Y. Zou, The use of summation to aggregate software metrics hinders the performance of defect prediction models, IEEE Trans. Softw. Eng. 43 (5) (2016) 476–491.
[17] T. Zhou, X. Sun, X. Xia, B. Li, X. Chen, Improving defect prediction with deep forest, Inf. Softw. Technol. 114 (2019) 204–216.
[18] S. Wang, T. Liu, J. Nam, L. Tan, Deep semantic feature learning for software defect prediction, IEEE Trans. Softw. Eng. (2018).
[19] Ö.F. Arar, K. Ayan, A feature dependent Naive Bayes approach and its application to the software defect prediction problem, Appl. Soft Comput. 59 (2017) 197–209.
[20] M. Ochodek, M. Staron, W. Meding, Simsax: A measure of project similarity based on symbolic approximation method and software defect inflow, Inf. Softw. Technol. 115 (2019) 131–147.
[21] C. Liu, D. Yang, X. Xia, M. Yan, X. Zhang, A two-phase transfer learning model for cross-project defect prediction, Inf. Softw. Technol. 107 (2019) 125–136.
[22] C. Ni, W.S. Liu, X. Chen, Q. Gu, D.X. Chen, Q.G. Huang, A cluster based feature selection method for cross-project software defect prediction, J. Comput. Sci. Tech. 32 (6) (2017) 1090–1107.
[23] Q. Yu, S.j. Jiang, R.c. Wang, H.y. Wang, A feature selection approach based on a similarity measure for software defect prediction, Front. Inf. Technol. Electron. Eng. 18 (11) (2017) 1744–1753.
[24] Z. Xu, J. Liu, X. Luo, Z. Yang, Y. Zhang, P. Yuan, Y. Tang, T. Zhang, Software defect prediction based on kernel PCA and weighted extreme learning machine, Inf. Softw. Technol. 106 (2019) 182–200.

[25] T. Shippey, D. Bowes, T. Hall, Automatically identifying code features for software defect prediction: Using ast n-grams, Inf. Softw. Technol. 106 (2019) 142–160.

[26] 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) (2020) 40–54.

[27] P.R. Bal, S. Kumar, WR–ELM: Weighted regularization extreme learning machine for imbalance learning in software fault prediction, IEEE Trans. Reliab. 69 (4) (2020) 1355–1375.

[28] X. Yu, J. Liu, J.W. Keung, Q. Li, K.E. Bennin, Z. Xu, J. Wang, X. Cui, Improving ranking-oriented defect prediction using a cost-sensitive ranking SVM, IEEE Trans. Reliab. 69 (1) (2020) 139–153.

[29] S.H. Khan, M. Hayat, M. Bennamoun, F.A. Sohel, R. Togneri, Cost-sensitive learning of deep feature representations from imbalanced data, IEEE Trans. Neural Netw. Learn. Syst. 29 (8) (2017) 3573–3587.

[30] L. Zhang, D. Zhang, Evolutionary cost-sensitive extreme learning machine, IEEE Trans. Neural Netw. Learn. Syst. 28 (12) (2016) 3045–3060.

[31] W. Feng, W. Huang, J. Ren, Class imbalance ensemble learning based on the margin theory, Appl. Sci. 8 (5) (2018) 815.

[32] R. Blagus, L. Lusa, SMOTE for high-dimensional class-imbalanced data, BMC Bioinformatics 14 (2013) 106, http://dx.doi.org/10.1186/1471-2105-14-106.

[33] D. Elreedy, A. Atiya, A comprehensive analysis of synthetic minority oversampling technique (SMOTE) for handling class imbalance, Inform. Sci. 505 (2019) http://dx.doi.org/10.1016/j.ins.2019.07.070.

[34] K.E. Bennin, J.W. Keung, A. Monden, On the relative value of data resampling approaches for software defect prediction, Empir. Softw. Eng. 24 (2) (2019) 602–636.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in python, J. Mach. Learn. Res. 12 (2011) 2825–2830.

[36] Q. Song, Y. Guo, M. Shepperd, A comprehensive investigation of the role of imbalanced learning for software defect prediction, IEEE Trans. Softw. Eng. 45 (12) (2018) 1253–1269.

[37] C. Tantithamthavorn, S. McIntosh, A.E. Hassan, K. Matsumoto, An empirical comparison of model validation techniques for defect prediction models, IEEE Trans. Softw. Eng. 43 (1) (2016) 1–18.

[38] A.A. Shanab, T.M. Khoshgoftaar, R. Wald, A. Napolitano, Impact of noise and data sampling on stability of feature ranking techniques for biological datasets, in: 2012 IEEE 13th International Conference on Information Reuse Integration, IRI, 2012, pp. 415–422, http://dx.doi.org/10.1109/IRI.2012.6303039.

[39] A. Agrawal, T. Menzies, Is ‘‘better data’’ better than ‘‘better data miners’’?, in: 2018 IEEE/ACM 40th International Conference on Software Engineering, ICSE, IEEE, 2018, pp. 1050–1061.

[40] J. Keung, E. Kocaguneli, T. Menzies, Finding conclusion stability for selecting the best effort predictor in software effort estimation, Autom. Softw. Eng. 20 (4) (2013) 543–567.

[41] C. Bunkhumpornpat, K. Sinapiromsaran, C. Lursinsap, Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2009, pp. 475–482.

[42] G. Cohen, M. Hilario, H. Sax, S. Hugonnet, A. Geissbuhler, Learning from imbalanced data in surveillance of nosocomial infection, Artif. Intell. Med. 37 (1) (2006) 7–18.