2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC) | 978-1-6654-2463-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/COMPSAC51774.2021.00041

ROCT: Radius-based Class Overlap Cleaning Technique to Alleviate the Class Overlap Problem in Software Defect Prediction

Shuo Feng[1], Jacky Keung[1], Jie Liu[2], Yan Xiao[3]*, Xiao Yu[4], and Miao Zhang[1]
[1]Department of Computer Science, City University of Hong Kong, Hong Kong, China,
{shuofeng5-c,miazhang9-c}@my.cityu.edu.hk, Jacky.Keung@cityu.edu.hk
[2] School of Computer Science, Wuhan University, Wuhan, China, jieliu@whu.edu.cn
[3] School of Computing, National University of Singapore, Singapore, dcsxan@nus.edu.sg
[4] School of Computer Science and Technology, Wuhan University of Technology, Wuhan, China, xiaoyu@whut.edu.cn

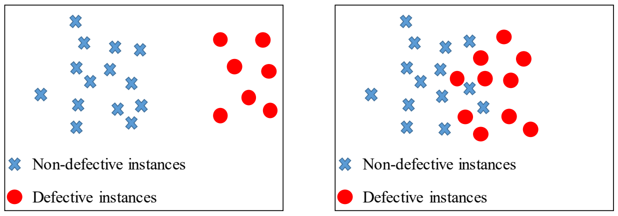
***Abstract*—The training data commonly used in software defect prediction (SDP) usually contains some instances that have similar values on features but are in different classes, which significantly degrades the performance of prediction models trained using these instances. This is referred to as the class overlap problem (COP). Previous studies have concluded that COP has a more negative impact on the performance of prediction models than the class imbalance problem (CIP). However, less research has been conducted on COP than CIP. Moreover, the performance of the existing class overlap cleaning techniques heavily relies on the settings of hyperparameters such as the value of K in the K-nearest neighbor algorithm or the K-means algorithm, but how to find those optimal hyperparameters is still a challenge. In this study, we propose a novel technique named the radius-based class overlap cleaning technique (ROCT) to better alleviate COP without tuning hyperparameters in SDP. The basic idea of ROCT is to take each instance as the center of a hypersphere and directly optimize the radius of the hypersphere. Then ROCT identifies those instances with the opposite label of the center instance as the overlapping instance and removes them. To investigate the performance of ROCT, we conduct the empirical experiment across 29 datasets collected from various software repositories on the K-nearest neighbor, random forest, logistic regression, and naive Bayes classifiers measured by AUC, $balance$, pd, and pf. The experimental results show that ROCT performs the best and significantly improves the performance of prediction models by as much as 15.2% and 29.9% in terms of AUC and $balance$ compared with the existing class overlap cleaning techniques. The superior performance of ROCT indicates that ROCT should be recommended as an efficient alternative to alleviate COP in SDP.**

***Index Terms*—Class Overlap, Class Imbalance, Data Preprocessing, Software Defect Prediction**

I. Introduction

Software defect prediction (SDP) plays an important role in software quality assurance [1], [2]. There are some practical issues, such as the class imbalance problem (CIP) [3], [4] and the class overlap problem (COP) [5], that significantly hinder the performance of SDP models. CIP and COP often occur simultaneously. Generally, there are more non-defective instances (i.e., the majority class instances) than defective ones (i.e., the minority class instances) in software defect datasets. The prediction model trained on these imbalanced datasets tends to focus on the non-defective instances and is more likely to predict the new software instances as non-defective, which is referred to as CIP. CIP degrades the performance of SDP models and thus makes the predicted results less reliable. However, some studies [6] have pointed out that CIP is not the main reason for the degradation in the performance of prediction models. In fact, they have concluded that only CIP does not necessarily degrade the performance of prediction models. For example, as shown in Fig. 1(a), it is not difficult for prediction models to separate the non-defective instances from the defective ones, even though the number of the non-defective instances far exceeds the number of the defective ones. Instead, these studies have concluded that COP takes the primary responsibility for the degradation in the performance of prediction models. COP refers to that the defective instances and the non-defective instances are mixed in the feature space, making prediction models difficult in separating the defective instances from the non-defective ones. As shown in Fig. 1(b), the number of the non-defective instances is equal to the number of the defective ones. However, it is still difficult for prediction models to make correct predictions for these instances because of COP. Based on the above conclusion, more effort should be paid to the research on COP. However, the fact is that more research is conducted on CIP than COP [7]. Besides, the performance of the existing class overlap cleaning techniques heavily depends on the settings of the hyperparameters, but how to find the optimal values of these hyperparameters is difficult.

In this study, we propose a novel technique named the radius-based class overlap cleaning technique (ROCT) to alleviate the aforementioned issues. ROCT takes each instance as the center of a hypersphere and finds the optimal radius of the hypersphere. Those instances with the opposite label of the center instance will be identified as the class overlapping instances, and will be removed. By removing the class overlapping instances, prediction models separate the defective instances from the non-defective ones more easily. The differential evolution algorithm [8] (DE) is employed to explore the optimal radius. We conduct experiments to



(a) CIP (b) COP

Fig. 1. The illustration of CIP and COP

compare the performance of ROCT with those of four existing class overlap cleaning techniques (i.e., Improved K-Means Clustering Cleaning Approach (IKMCCA) [7], Neighborhood Cleaning Learning (NCL) [9], Edited Nearest Neighbor (ENN) [10] and, Tomek-link (Tomek) [11]). We investigate the performances of ROCT and these techniques on four classifiers (i.e., K-nearest neighbor (K-NN), random forest (RF), logistic regression (LR), and naive Bayes (NB)) across 29 datasets collected from the AEEEM [12], NASA [13], PROMISE [14], ReLink [15], SOFTLAB [16] repositories in terms of four performance measures (i.e., the area under the ROC curve (AUC), $balance$, the probability of detection (pd), and the probability of false alarm (pf)). To further investigate whether there exists statistically significant difference between the performance of ROCT and those of the compared techniques, we employ the Wilcoxon signed-rank test [17] and Cliff's δ effect size [18]. The experimental results show that ROCT significantly improves the performance of prediction models compared with the existing class overlap cleaning techniques. Therefore, we recommend ROCT as an efficient algorithm to alleviate COP in SDP.

The remainder of this paper is organized as follows: Section II introduces the motivation of our work. Section III presents the background and the related work. Section IV details our proposed technique. In Section V, we introduce the experimental settings. The experimental results are presented in Section VI. We further analyze the threats to the validity of our work in Section VII. Finally, we conclude our work and introduce our future work in Section VIII.

II. Motivation

COP appears when there is a similar quantity of data from both classes in a region of the feature space, which makes prediction models hard to distinguish between the two classes and thus, the performance of prediction is poor [6]. Previous studies have shown that COP plays a more important role in negatively affecting the performance of prediction models than CIP. Some class overlap cleaning techniques are proposed to alleviate COP. Most of these techniques rely on certain algorithms, such as K-NN and K-means algorithms, to alleviate COP. For example, NCL [9] is a common technique to alleviate COP. It first finds the top K nearest neighbor instances for each minority class instance in a dataset. If there are majority class instances in these K instances, these majority class instances will be identified as the overlapping instances and removed. IKMCCA [7], the latest class overlap cleaning technique proposed to alleviate COP in SDP, employs the K-means algorithm to divide datasets into K clusters. Then the minority class instances in the clusters whose defect ratio p is below a predefined threshold will be identified as the overlapping instances and removed. Otherwise, the majority class instances in these clusters will be removed. However, how to decide the best values of the hyperparameters for these techniques is still a challenge. The author manually sets K as 3 in NCL. In IKMCCA, the author sets $K = n/m$, where n is the number of instances in a dataset, and m is manually set as 20. p is set as the original percentage of the minority class instances in the training datasets. However, there is a lack of rationales behind the authors' settings.

Essentially, the K value in NCL is used to aid in deciding the radius of the hypersphere whose center is each minority class instance. Then the majority class instances located in the hypersphere decided by the minority class instances are identified as the overlapping instances and should be removed. As shown in Fig. 2, for the hypersphere decided by the minority class instance A, the center is where the instance A is, and the radius varies along with different K values. For example, when K is set to be 3, the radius of the hypersphere decided by the instance A is the distance between the instances A and B, and the instance D will be identified as the overlapping instance and removed by NCL. If K is set to be 5, the radius is the distance between the instances A and C. In such a case, the instances C, D, and E will be removed. Moreover, the collected instances cannot fill up the feature space. The density of the collected instances varies in different areas of the feature space. For the same K value, the radius of the hypersphere in the low-density area is large, while the radius in the high-density area is small. This inconsistency may also negatively impact the performance of these techniques.

Therefore, we propose the radius-based class overlap cleaning technique (ROCT) to alleviate COP. ROCT takes each instance as the center of a hypersphere and directly optimizes the radius for the hypersphere. The instances with the opposite label in the hypersphere are the overlapping instances and should be removed. ROCT employs DE to optimize the radius and does not need to manually set any hyperparameter. The detail of ROCT is presented in Section IV.

III. Background and Related Work

A. Class overlap problem

COP usually causes a poor decision boundary and thus makes it difficult to construct effective prediction models with good ability to make accurate predictions. COP is usually related to the research on the data quality and the noise detection. Prati et al. [19] conducted an empirical study, and the experimental results show that CIP does not directly cause the degradation in the performance. Instead, the degradation in the performance is related to the degree of the class overlap.

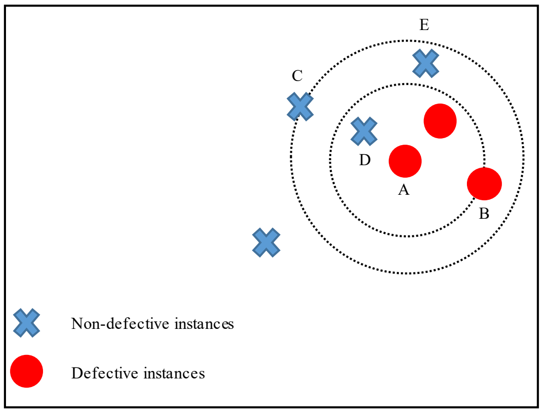


Fig. 2. The radius of the hypersphere decided by the number K

Kim et al. [20] proposed the closest list noise identification (CLNI) to detect and remove the noisy instances. CLNI first finds the top K-nearest neighbors of a target instance. Then based on the overlapping degree, CLNI decides whether the target instances are noise. The overlapping degree for the target instance is calculated as the ratio of the number of the minority class instances to the number of the majority class instances for the top K-nearest neighbors. Therefore, CLNI can be regarded as a technique to alleviate COP. NCL [9] is similar to CLNI. The difference between CLNI and NCL is that NCL directly removes all majority class instances that belong to the top K-nearest neighbors of a minority class instance. Tang et al. [21] proposed the K-means clustering cleaning approach (KMCCA). Initially, KMCCA is also used to detect the noisy instances like CLNI. KMCCA employs the K-means algorithm to divide a dataset into K clusters. Then the noise factor of an instance is calculated based on the cluster it belongs to as well as the distance between this instance and the center of its closet large cluster. Based on the noise factor of the instances for each cluster, KMCCA sorts them in the descending order and removes the top $p\%$ instances. Gong et al. [7] proposed IKMCCA. IKMCCA also employs the K-means algorithm to divide a dataset. Then, the minority class instances in a cluster are removed, if the percentage of the minority class instances is below a predefined threshold. Otherwise, the majority class instances are removed. The above-mentioned techniques all heavily rely on the setting of the hyperparameters. Inappropriate settings will significantly degrade their performances.

B. *Differential evolution*

DE, proposed by Storn et al. [8], is an evolutionary algorithm. DE is based on the mutation, crossover, and selection operations. DE initializes a group of candidate solutions which is the first generation. The next generation is mutated by first perturbing the first generation using a scaling factor F. Then, the crossover operation is applied to improve the diversity of the population based on the crossover rate CR. Next, the selection operation selects the best candidate as a new member of the next generation based on the fitness function. This process is iterated until the stop criterion of DE is met. Finally, the best candidate solution decided by the fitness function will be selected as the final solution.

IV. Methodology

In this section, we present the radius-based class overlap cleaning technique (ROCT). The core of ROCT is to find the optimal values of four parameters, which are r_{min}, t_{min}, r_{maj} and t_{maj}, respectively. The details of these four parameters are given as follows.

r_{min} is the radius of the minority class hypersphere whose center is each minority class instance. The number of the minority class hyperspheres is equal to the number of the minority class instances in a dataset. t_{min} is the threshold used to decide whether a hypersphere belongs to the minority class. If the ratio of the number of the minority class instances to the number of all instances in the hypersphere is beyond t_{min}, it indicates that this hypersphere belongs to the minority class, which means the majority class instances in this hypersphere are the overlapping instances and should be removed. r_{maj} and t_{maj} are similar to r_{min} and t_{min}. These two parameters are used to decide whether the minority class instances are the overlapping instances and should be removed or not.

It is notable that we discriminate between the minority class and majority class instances by separately exploring r_{min}, t_{min}, and r_{maj}, t_{maj}, which is because of CIP. If we treat the minority class and the majority class instances equally, and set the same radius and the same threshold of the hyperspheres for all instances, the hyperspheres will include more majority class instances than minority class instances, and the ratio of the number of the minority class instances to the number of all instances will always be low, whatever the center instance belongs to the majority class or the minority class. In fact, discriminating between the minority class and majority class instances is quite common in the area of CIP.

To find the optimal values of the four parameters, we employ DE [8]. The range of r_{min} and r_{maj} explored by DE is set from 0 to dis_{max}, where dis_{max} is the largest distance between any two instances in a dataset. The range of t_{min} and t_{maj} is set from 0 to 1. In this study, we set the fitness function to explore the maximum AUC value of prediction models. For the hyperparameter settings of DE, there is no uniform rule. We set the population size N as ten times the dimensionality D of instances by convention. As for the scaling factor F and the crossover rate CR, we have tried various combinations. Experimentally, we choose the combination of F equalling 0.3 and CR equalling 0.9, which minimizes the execution time of ROCT while achieving a similar performance compared with the other settings. The number of the generation G is set to be ten, because the convergence was achieved on all classifiers across all selected datasets in no more than ten generations. We present the details of the settings of DE in Table I.

TABLE I
HYPERPARAMETER CONFIGURATIONS OF DE

Hyperparameters	Values
The population size, N	10 * D
The number of generations, G	10
The mutation constant, F	0.3
The crossover rate, CR	0.9
The range of r_{min}	(0, dis_{max})
The range of r_{maj}	(0, dis_{max})
The range of t_{min}	(0, 1)
The range of t_{maj}	(0, 1)

Algorithm 1 ROCT algorithm

Input: Dataset N including the minority class instances N_{min} and the majority class instances N_{maj}
Output: Dataset N removing the overlapping instances

```
1: apply DE to $N$ to obtain $r_{min}$, $t_{min}$, $r_{maj}$ and $t_{maj}$
2: for each instance $X_i$ in $N$ do
3:     calculate the distance $d_i$ between $X_i$ and all the other instances in $N$
4:     if $X_i$ in $N_{min}$ then
5:         select $X$ whose $d_i < r_{min}$ from $N$
6:         calculate the ratio $t$ of the minority class instances in $X$
7:         if $t > t_{min}$ then
8:             remove the majority class instances from $N$
9:         end if
10:    else
11:        select $X$ whose $d_i < r_{maj}$ from $N$
12:        calculate the ratio $t$ of the majority class instances in $X$
13:        if $t > t_{maj}$ then
14:            remove the minority class instances from $N$
15:        end if
16:    end if
17: end for
18: return Dataset $N$ removing the overlapping instances
```

Algorithm 1 shows the pseudo-code of ROCT. First, ROCT obtains the optimal values of r_{min}, t_{min}, r_{maj}, and t_{maj} by applying DE (Line 1). Then, for each instance X_i, ROCT calculates the distance between it and all the other instances (Lines 2 to 3). If X_i belongs to the minority class, r_{min} and t_{min} are applied to decide whether the majority class instances in the hypersphere decided by X_i are the overlapping instances and should be removed or not (Lines 4 to 9). Otherwise, r_{maj} and t_{maj} are applied (Lines 10 to 16). Finally, the dataset removing the overlapping instances is returned (Line 18).

V. EXPERIMENTAL DESIGN

This section details the experimental design, including the baseline techniques, the adopted datasets, the classifiers, the performance measures, the statistical test, as well as the experimental procedure.

A. Baselines

The class overlap cleaning techniques we adopt as the baselines are IKMCCA [7], NCL [9], ENN [10], and Tomek [11]. We give a brief introduction to these techniques as below:
IKMCCA. IKMCCA is the latest class overlap cleaning technique proposed to alleviate COP in SDP. It first employs the K-means algorithm to cluster datasets into K clusters. Then for each cluster, IKMCCA calculates the percentage of the minority class instances. If the percentage is lower than a predefined threshold, the corresponding cluster is identified as the majority class cluster, and the minority class instances in this cluster are removed. Otherwise, the majority class instances are removed.
NCL. The main idea of NCL is to locate the overlapping instances that have contradictory nearest neighbor instances. Specifically, for each minority class instance, NCL calculates its top K nearest neighbor instances and removes the majority class instances from the top K nearest neighbor instances if these K instances contain the majority class instances. NCL only removes the majority class instances to avoid worsening CIP.
ENN. For each instance, ENN finds its K nearest neighbors. If this instance does not agree with the majority of its K nearest neighbors, in other words, if this instance is misclassified by the K-NN algorithm, it will be removed. ENN edits out noisy instances as well as close border cases, leaving smoother decision boundaries.
Tomek. If one minority class instance and one majority class instance are each other's nearest neighbor, these two instances are one tomek link. Then there are two ways to handle Tomek link. One way is to remove all tomek links in a dataset. The other way is to remove only the majority class instance in each tomek link.

B. Datasets

In this study, we adopt 29 datasets collected from the AEEEM [12], NASA [13], PROMISE [14], ReLink [15], and SOFTLAB [16] repositories to conduct the experiment. These datasets were widely adopted in previous studies [7], [22]. The projects from PROMISE are multi-version, and we only use the first version of each project as Gong did [7]. Table II presents the details of the adopted datasets.

C. Classifiers

We select four classifiers to build prediction models. These classifiers are the K-nearest neighbor (K-NN), random forest (RF), logistic regression (LR), and naive Bayes (NB). These classifiers were widely adopted by previous studies in SDP. To make others replicate our work conveniently and avoid reinventing the wheel, we adopt the Sklearn package [23] for our experiment. The hyperparameters of these classifiers are all set to be the default values, because we focus on the performance of different techniques instead of tuning the hyperparameters of the classifiers.

D. Performance Measures

In SDP, the performance measures that are not significantly affected by CIP are preferable. In this study, we adopt AUC, *balance*, *pd*, and pf to measure the performance of prediction models. AUC [24]–[26] is the abbreviation of the area under the ROC curve, which is widely adopted to measure the overall performance of prediction models in SDP. *balance*, *pd*, and pf [27]–[29] are computed using the results from a confusion matrix (Table III). In SDP, defective instances are considered as positive instances and non-defective instances as negative instances. Then the outcomes of a prediction model can be categorized into four types:

TABLE II
DESCRIPTION OF 29 DATASETS

Group	Dataset	Language	Granularity	Number of metrics	# Instances	% Defect Ratio
NASA	CM1	C	Function	37	327	12.84
	MW1				253	10.67
	PC1				705	8.65
	PC3				1077	12.44
	PC4				1458	12.21
SOFTLAB	AR1			29	121	7.44
	AR3				163	12.70
	AR4				107	18.69
	AR5				36	22.22
	AR6				101	14.85
AEEEM	Equinox Framework	Java	Class	61	324	39.81
	Eclipse JDT core				997	20.66
	Apache Luence				691	9.26
	Mylyn				1862	13.16
	Eclipse PDE UI				1497	13.96
PROMISE	ant1.3			20	125	16.00
	camel1.0				339	3.83
	ivy1.1				111	56.76
	jedit3.2				272	33.09
	log4j1.0				135	25.19
	lucene2.0				195	46.67
	poi1.5				237	59.49
	synapse1.0				157	10.19
	velocity1.4				196	75.00
	xalan2.4				723	15.21
	xercesinit				162	47.53
ReLink	Apache HTTP Server		File	26	194	50.52
	OpenIntents Safe				56	39.29
	ZXing				399	29.57

- True Positive (TP): the number of correctly predicted positive instances
- True Negative(TN): the number of correctly predicted negative instances
- False Positive(FP): the number of instances which are actually negative instances but wrongly predicted as positive instances
- False Negative(FN): the number of instances which are actually positive instances but wrongly predicted as negative instances

TABLE III
A CONFUSION MATRIX

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

A confusion matrix is composed of these four categorized outcomes. $balance$, pd, and pf are calculated based on the confusion matrix, and the mathematical definitions are given below:

$$pd = \frac{TP}{TP + FN}, \quad (1)$$

$$pf = \frac{FP}{TN + FP}, \quad (2)$$

$$balance = 1 - \frac{\sqrt{(0 - pf)^2 + (1 - pd)^2}}{\sqrt{2}}. \quad (3)$$

The higher values of AUC, $balance$, pd, and the lower values of pf represent a better performance.

E. Experimental Procedure

Fig. 3 shows the flow of the experiment. We first apply the log-transformation to the datasets as Gong did [7]. Then we employ the 5-fold cross-validation with the stratification method to divide the datasets into five folds. The stratification method is to keep the ratio of the minority class instances to the majority class instances in each fold remaining the same as the original dataset. We use the four folds as the training data and apply the compared techniques to the training data. The left one fold is used as the testing data.Then this procedure will be iterated five times to ensure all five folds are used as both the training and the testing data. For ROCT, we also use the 5-fold cross-validation with the stratification method to divide the datasets into five parent folds. We further divide the four parent folds into five sub-folds. Then we take four sub-folds as the training data and apply DE to these four sub-folds. The left sub-fold is used to decide the optimal values of the parameters of ROCT. This procedure is also repeated five times to ensure all sub-folds are used for both training and testing. Once the optimal values are found, ROCT is applied to the four parent folds, and the left parent fold is used to validate the performance of ROCT. We also repeat this procedure five times. For each technique, we iterate the above process ten times to reduce the variance and bias. After ten iterations, we record the average values of AUC, $balance$, pd, and pf. For

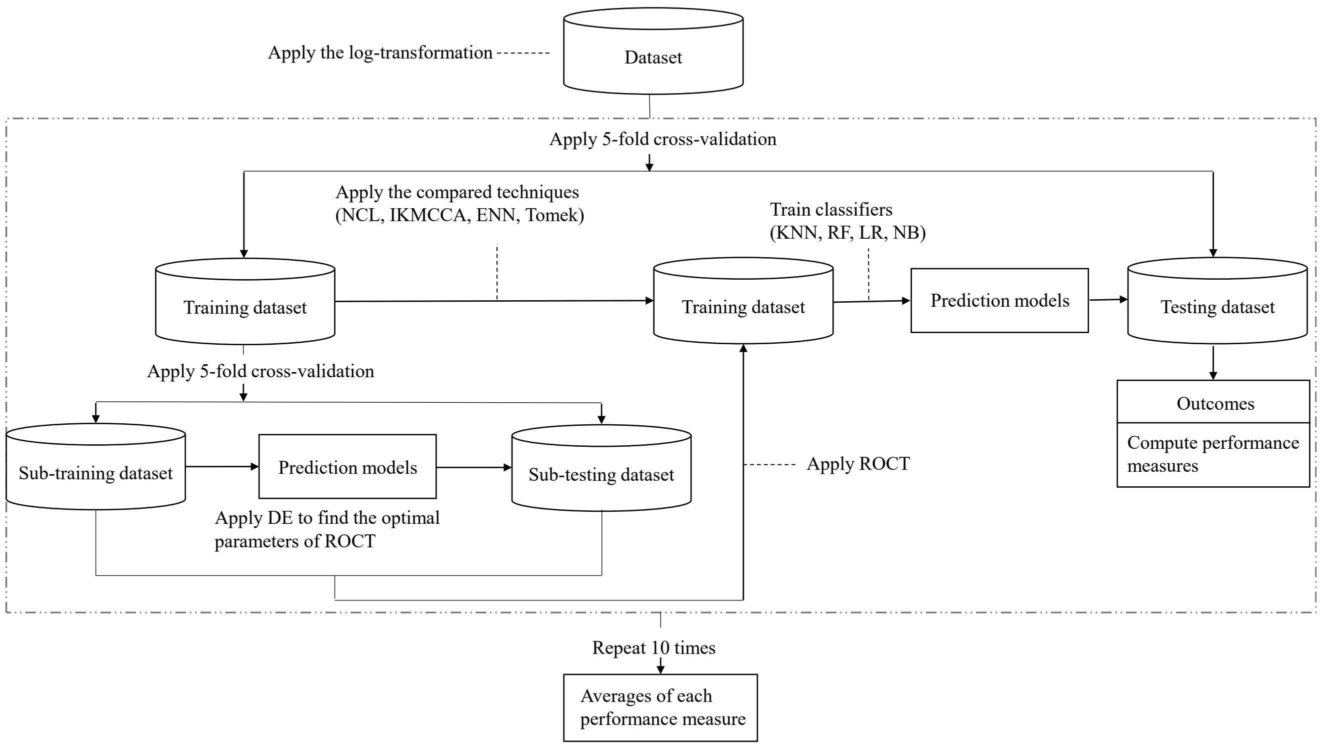

Apply the log-transformation
Dataset
Apply 5-fold cross-validation
Apply the compared techniques (NCL, IKMCCA, ENN, Tomek)
Train classifiers (KNN, RF, LR, NB)
Training dataset
Training dataset
Prediction models
Testing dataset
Apply 5-fold cross-validation
Outcomes
Compute performance measures
Sub-training dataset
Prediction models
Sub-testing dataset
Apply ROCT
Apply DE to find the optimal parameters of ROCT
Repeat 10 times
Averages of each performance measure

Fig. 3. The experimental framework

the hyperparameters of the compared techniques, we follow the authors' settings.

F. Statistical Test

To analyze whether there exists statistically significant difference between the performance of ROCT and those of the compared techniques, we adopt the Wilcoxon signed-rank test (Wilcoxon) [30]. Wilcoxon is a non-parametric test and takes the null hypothesis that data are paired and come from the same distribution. The alternative hypothesis is that data come from a different distribution. We employ Wilcoxon at the 95% confidence level. If the p-value is less than 0.05, the null hypothesis is rejected. Otherwise, the null hypothesis cannot be rejected. Furthermore, to quantify the difference between the performances of ROCT and other techniques, the effect size is computed (i.e., Cliff's δ). We interpret the effect size 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 $\delta > 0.474$) as Kampenes [31] did.

We also adopt the win-draw-loss strategy to show a detailed performance of each technique across every single dataset in terms of AUC.

VI. Experimental Results

In this section, we present the performance of ROCT and the baseline techniques in terms of pd, pf, $balance$, and AUC. We further present the performance of each technique across every single dataset in terms of AUC. To ease the demonstration, we refer IKMCCA as IKM in this section.

TABLE IV
THE PERFORMANCE OF ROCT, NCL, IKM, ENN, AND TOMEK ON THE SELECTED CLASSIFIERS ACROSS 29 DATASETS IN TERMS OF pd

	ROCT	NCL	IKM	ENN	Tomek
K-NN	**0.777**	0.561	0.596	0.539	0.433
p-value		$<$ **.05**	$<$ **.05**	$<$ **.05**	$<$ **.05**
Cliff's δ		0.577	0.655	0.598	0.781
RF	**0.735**	0.538	0.591	0.557	0.391
p-value		$<$ **.05**	$<$ **.05**	$<$ **.05**	$<$ **.05**
Cliff's δ		0.498	0.503	0.397	0.746
LR	**0.768**	0.534	0.593	0.559	0.411
p-value		$<$ **.05**	$<$ **.05**	$<$ **.05**	$<$ **.05**
Cliff's δ		0.570	0.560	0.534	0.738
NB	**0.712**	0.680	0.649	0.682	0.653
p-value		$>$ .05	$<$ **.05**	$>$ .05	$<$ **.05**
Cliff's δ		0.059	0.224	0.033	0.210

We first present the pd values of each technique on the four classifiers. It can be seen that ROCT performs the best compared with the baselines in terms of pd from Table IV. Based on the conclusion of Chen [9], an effective class overlap technique should increase the pd values of prediction models. ROCT obtains the highest pd values on all classifiers. Moreover, the differences in the pd values between ROCT and the compared techniques are statistically significant on most classifiers. On the K-NN, RF, and LR classifiers, the effect sizes are all practical. The higher pd values of ROCT reflect a better ability to find defects. Compared with ROCT, the lower pd values of the baselines indicate a poor ability to find defects, which shows the inefficiency of these techniques.

TABLE V
THE PERFORMANCE OF ROCT, NCL, IKM, ENN, AND TOMEK ON THE SELECTED CLASSIFIERS ACROSS 29 DATASETS IN TERMS OF pf

	ROCT	NCL	IKM	ENN	Tomek
K-NN	0.299	0.204	0.236	0.194	**0.148**
p-value		$<$ **.05**	$<$ **.05**	$<$ **.05**	$<$ **.05**
Cliff's δ		0.465	0.265	0.520	0.587
RF	0.276	0.181	0.230	0.192	**0.120**
p-value		$<$ **.05**	$>$ .05	$<$ **.05**	$<$ **.05**
Cliff's δ		0.522	0.227	0.451	0.703
LR	0.293	0.198	0.234	0.203	**0.131**
p-value		$<$ **.05**	$<$ **.05**	$<$ **.05**	$<$ **.05**
Cliff's δ		0.505	0.284	0.484	0.641
NB	0.275	0.296	0.285	0.304	**0.268**
p-value		$>$ .05	$>$ .05	$>$ .05	$>$ .05
Cliff's δ		0.140	0.115	0.170	0.046

There is a positive correlation between pd and pf. Usually, a high pd value follows a high pf value. From Table V, it can be seen that there is no significant difference between the performance of ROCT and those of the compared techniques on the NB classifiers. On the other classifiers, the compared techniques obtain lower pf values. Tomek performs the best in terms of pf. It significantly outperforms ROCT on most classifiers with practical effect sizes. However, this is at the expense of degrading the pd values, which means it is difficult for Tomek to find defects in a dataset, and thus less practical.

TABLE VI
THE PERFORMANCE OF ROCT, NCL, IKM, ENN, AND TOMEK ON THE SELECTED CLASSIFIERS ACROSS 29 DATASETS IN TERMS OF $balance$

	ROCT	NCL	IKM	ENN	Tomek
K-NN	**0.708**	0.613	0.630	0.601	0.549
p-value		$<$ **.05**	$<$ **.05**	$<$ **.05**	$<$ **.05**
Cliff's δ		0.574	0.508	0.605	0.722
RF	**0.693**	0.609	0.629	0.612	0.537
p-value		$<$ **.05**	$<$ **.05**	$<$ **.05**	$<$ **.05**
Cliff's δ		0.442	0.436	0.424	0.646
LR	**0.700**	0.592	0.626	0.604	0.539
p-value		$<$ **.05**	$<$ **.05**	$<$ **.05**	$<$ **.05**
Cliff's δ		0.620	0.508	0.574	0.715
NB	**0.686**	0.657	0.647	0.653	0.655
p-value		$<$ **.05**	$<$ **.05**	$<$ **.05**	$<$ **.05**
Cliff's δ		0.282	0.282	0.270	0.275

We then use $balance$ and AUC to measure the overall performance of ROCT and the baseline techniques. Table VI presents the $balance$ values of each technique together with the p-values and the effect sizes. We can see that the performance of ROCT is superior in terms of $balance$. ROCT gains the highest $balance$ values on all four classifiers and significantly outperforms all the compared techniques. Moreover, the effect sizes between ROCT and the other techniques reach medium, or even large on the K-NN. RF and LR classifiers. On the NB classifier, the effect size between ROCT and the compared techniques is small. Compared with the second-best performing technique on each classifier, the improvement brought by ROCT is up to 12.4%, 10.2%, 11.8%, and 4.4% in terms of $balance$, respectively. Compared with the worst performing technique on each classifier, the improvement brought by ROCT is up to 29.0%, 29.1%, 29.9%, and 6.0% in terms of $balance$, respectively.

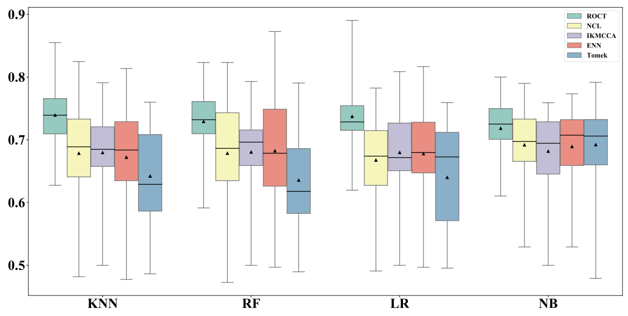


Fig. 4. The boxplots of the performance of ROCT and the class overlap cleaning techniques in terms of AUC

TABLE VII
THE PERFORMANCE OF ROCT, NCL, IKM, ENN, AND TOMEK ON THE SELECTED CLASSIFIERS ACROSS 29 DATASETS IN TERMS OF AUC

	ROCT	NCL	IKM	ENN	Tomek
K-NN	**0.739**	0.679	0.680	0.672	0.642
p-value		$<$ **.05**	$<$ **.05**	$<$ **.05**	$<$ **.05**
Cliff's δ		0.486	0.539	0.517	0.684
RF	**0.729**	0.679	0.681	0.682	0.636
p-value		$<$ **.05**	$<$ **.05**	$<$ **.05**	$<$ **.05**
Cliff's δ		0.361	0.448	0.365	0.636
LR	**0.737**	0.668	0.680	0.678	0.640
p-value		$<$ **.05**	$<$ **.05**	$<$ **.05**	$<$ **.05**
Cliff's δ		0.591	0.532	0.517	0.707
NB	**0.718**	0.692	0.682	0.689	0.692
p-value		$<$ **.05**	$<$ **.05**	$<$ **.05**	$<$ **.05**
Cliff's δ		0.282	0.351	0.279	0.258

Table VII presents the AUC values of the class overlap cleaning techniques across 29 datasets. It can be seen that ROCT achieves the highest AUC values and significantly outperforms the other techniques on all classifiers. Furthermore, ROCT significantly outperforms the compared techniques with small, medium or even large effect sizes. Compared with the second-best performing technique on each classifier, the improvement brought by ROCT is up to 8.7%, 6.9%, 8.4%, and 3.8% in terms of AUC, respectively. Compared with the worst performing technique on each classifier, the improvement brought by ROCT is up to 15.1%, 14.6%, 15.2%, and 5.3% in terms of AUC.

Fig. 4 shows the boxplot of AUC values of the techniques across 29 datasets on the four classifiers. The black triangle in the boxplot represents the mean value, and the black line is the median value. Except that ENN obtains the highest maximum AUC value on the RF classifier, it can be seen that ROCT obtains the highest maximum, mean, median, and minimum AUC values on all classifiers. Considering AUC can handle the trade-off between pd and pf well, higher AUC values indicate a better overall performance of prediction models. Therefore, from Fig 4, we can see that the overall performance of ROCT is superior to those of the compared techniques.

We further take the win-draw-loss strategy to show the detailed performance of ROCT across every single dataset in terms of AUC, because AUC is one of the common performance measure in SDP. W/D/L in Tables VIII, IX, X, and XI represents ROCT performs better than, the same as or worse than the corresponding baseline in terms of AUC across each dataset. It can be seen that no baseline obtains more than seven wins against ROCT across 29 datasets on all classifiers. Notably, we observe that the baseline techniques obtain 0.5 in terms of AUC on some datasets such as camel1.0 and AR5. When prediction models obtain AUC value equalling 0.5, it means the predicted results of the prediction models are not better than a random guess. This shows that these datasets are difficult for prediction models to make correct predictions, even if they are processed by some baseline techniques. On the contrary, ROCT obtains much higher AUC values than these baselines on these datasets, which shows the superiority of ROCT and indicates that ROCT can better process those difficult datasets than the previous techniques.

TABLE VIII
AUC VALUES ON THE KNN CLASSIFIER ACROSS 29 DATASETS

	ROCT	NCL	IKM	ENN	Tomek
EQ	0.755	0.746	0.749	0.738	0.682
JDT	0.775	0.765	0.752	0.760	0.737
LC	0.714	0.659	0.666	0.635	0.586
ML	0.727	0.691	0.694	0.686	0.613
PDE	0.695	0.650	0.658	0.635	0.579
CM1	0.694	0.565	0.592	0.578	0.529
MW1	0.726	0.751	0.706	0.729	0.584
PC1	0.741	0.595	0.685	0.587	0.583
PC3	0.766	0.689	0.728	0.655	0.559
PC4	0.817	0.825	0.785	0.814	0.693
ant1.3	0.787	0.718	0.704	0.718	0.632
camel1.0	0.759	0.500	0.569	0.500	0.500
ivy1.1	0.697	0.613	0.641	0.650	0.693
jedit3.2	0.767	0.752	0.738	0.719	0.743
log4j1.0	0.741	0.733	0.714	0.754	0.724
lucene2.0	0.704	0.641	0.680	0.647	0.629
poi1.5	0.726	0.716	0.680	0.684	0.744
synapse1.0	0.750	0.694	0.746	0.727	0.589
velocity1.4	0.709	0.685	0.705	0.695	0.699
xalan2.4	0.739	0.654	0.669	0.658	0.613
xercesinit	0.772	0.715	0.668	0.736	0.739
Apache	0.731	0.651	0.697	0.594	0.678
Safe	0.736	0.696	0.721	0.711	0.708
ZXing	0.628	0.622	0.607	0.618	0.590
AR1	0.665	0.482	0.582	0.477	0.486
AR3	0.855	0.791	0.791	0.741	0.741
AR4	0.755	0.621	0.674	0.644	0.611
AR5	0.830	0.807	0.500	0.807	0.760
AR6	0.683	0.651	0.609	0.600	0.598
W/D/L		27/0/2	29/0/0	27/0/2	28/0/1

We further present the boxplot of r_{min} and r_{maj} explored by DE on the 29 datasets. Fig. 5 presents the boxplots of the optimal values of t_{min} and t_{maj} on the 29 datasets. It can be seen that t_{min} values are generally smaller than t_{maj} values, which fits our intuition because of CIP, and it also indicates the necessity of discriminating between the minority class and majority class instances.

According to the experimental results, we conclude that

TABLE IX
AUC VALUES ON THE RF CLASSIFIER ACROSS 29 DATASETS

	ROCT	NCL	IKM	ENN	Tomek
EQ	0.778	0.743	0.768	0.754	0.766
JDT	0.778	0.762	0.756	0.788	0.722
LC	0.729	0.664	0.669	0.614	0.610
ML	0.733	0.698	0.703	0.681	0.618
PDE	0.694	0.660	0.659	0.646	0.604
CM1	0.675	0.576	0.599	0.576	0.490
MW1	0.713	0.740	0.706	0.711	0.618
PC1	0.758	0.611	0.674	0.626	0.602
PC3	0.768	0.653	0.730	0.659	0.565
PC4	0.803	0.761	0.793	0.783	0.686
ant1.3	0.737	0.754	0.670	0.749	0.567
camel1.0	0.655	0.494	0.582	0.497	0.495
ivy1.1	0.710	0.686	0.650	0.596	0.685
jedit3.2	0.769	0.796	0.744	0.777	0.765
log4j1.0	0.742	0.722	0.714	0.723	0.659
lucene2.0	0.678	0.599	0.696	0.614	0.634
poi1.5	0.711	0.707	0.677	0.685	0.728
synapse1.0	0.700	0.628	0.679	0.684	0.556
velocity1.4	0.761	0.794	0.739	0.773	0.791
xalan2.4	0.736	0.643	0.706	0.678	0.571
xercesinit	0.757	0.686	0.711	0.723	0.746
Apache	0.732	0.656	0.704	0.569	0.645
Safe	0.726	0.707	0.716	0.679	0.673
ZXing	0.646	0.635	0.590	0.650	0.598
AR1	0.712	0.473	0.561	0.564	0.491
AR3	0.818	0.773	0.750	0.873	0.723
AR4	0.715	0.698	0.674	0.675	0.659
AR5	0.823	0.823	0.500	0.807	0.590
AR6	0.591	0.534	0.614	0.640	0.583
W/D/L		24/1/4	27/0/2	22/0/7	27/0/2

TABLE X
AUC VALUES ON THE LR CLASSIFIER ACROSS 29 DATASETS

	ROCT	NCL	IKM	ENN	Tomek
EQ	0.754	0.747	0.746	0.728	0.712
JDT	0.779	0.762	0.765	0.767	0.727
LC	0.742	0.674	0.695	0.683	0.631
ML	0.719	0.667	0.687	0.680	0.611
PDE	0.699	0.655	0.669	0.647	0.601
CM1	0.697	0.622	0.644	0.684	0.531
MW1	0.723	0.682	0.730	0.693	0.558
PC1	0.762	0.608	0.668	0.606	0.555
PC3	0.751	0.648	0.727	0.655	0.560
PC4	0.837	0.783	0.809	0.813	0.737
ant1.3	0.749	0.627	0.673	0.713	0.596
camel1.0	0.741	0.497	0.624	0.497	0.498
ivy1.1	0.715	0.676	0.636	0.665	0.685
jedit3.2	0.774	0.756	0.725	0.741	0.759
log4j1.0	0.753	0.747	0.687	0.749	0.720
lucene2.0	0.707	0.608	0.651	0.605	0.692
poi1.5	0.680	0.674	0.668	0.641	0.677
synapse1.0	0.786	0.602	0.656	0.572	0.523
velocity1.4	0.725	0.715	0.733	0.738	0.722
xalan2.4	0.726	0.659	0.672	0.655	0.587
xercesinit	0.738	0.653	0.663	0.655	0.722
Apache	0.718	0.699	0.660	0.694	0.717
Safe	0.729	0.700	0.738	0.729	0.682
ZXing	0.620	0.581	0.592	0.611	0.571
AR1	0.676	0.491	0.584	0.591	0.495
AR3	0.795	0.673	0.805	0.655	0.682
AR4	0.716	0.722	0.624	0.725	0.673
AR5	0.890	0.757	0.500	0.817	0.690
AR6	0.684	0.686	0.688	0.651	0.649
W/D/L		27/0/2	24/0/5	26/1/2	29/0/0

TABLE XI
AUC VALUES ON THE NB CLASSIFIER ACROSS 29 DATASETS

	ROCT	NCL	IKM	ENN	Tomek
EQ	0.776	0.736	0.756	0.737	0.696
JDT	0.756	0.745	0.739	0.746	0.743
LC	0.743	0.732	0.729	0.736	0.742
ML	0.670	0.638	0.645	0.633	0.639
PDE	0.698	0.674	0.679	0.674	0.660
CM1	0.646	0.645	0.626	0.635	0.632
MW1	0.704	0.693	0.701	0.698	0.696
PC1	0.729	0.680	0.686	0.682	0.688
PC3	0.731	0.710	0.721	0.713	0.706
PC4	0.740	0.733	0.729	0.732	0.732
ant1.3	0.788	0.763	0.758	0.768	0.792
camel1.0	0.715	0.711	0.710	0.710	0.724
ivy1.1	0.679	0.633	0.637	0.627	0.682
jedit3.2	0.755	0.761	0.741	0.764	0.741
log4j1.0	0.781	0.750	0.759	0.743	0.762
lucene2.0	0.701	0.670	0.695	0.653	0.683
poi1.5	0.708	0.666	0.669	0.677	0.656
synapse1.0	0.734	0.688	0.736	0.681	0.688
velocity1.4	0.709	0.697	0.589	0.707	0.718
xalan2.4	0.722	0.713	0.709	0.712	0.713
xercesinit	0.734	0.701	0.673	0.709	0.726
Apache	0.750	0.747	0.704	0.732	0.709
Safe	0.718	0.693	0.726	0.693	0.736
ZXing	0.610	0.582	0.574	0.574	0.572
AR1	0.617	0.529	0.671	0.529	0.479
AR3	0.800	0.659	0.668	0.659	0.659
AR4	0.725	0.707	0.628	0.725	0.711
AR5	0.760	0.790	0.500	0.773	0.757
AR6	0.636	0.618	0.618	0.567	0.624
W/D/L		27/0/2	26/0/3	27/1/1	24/0/5

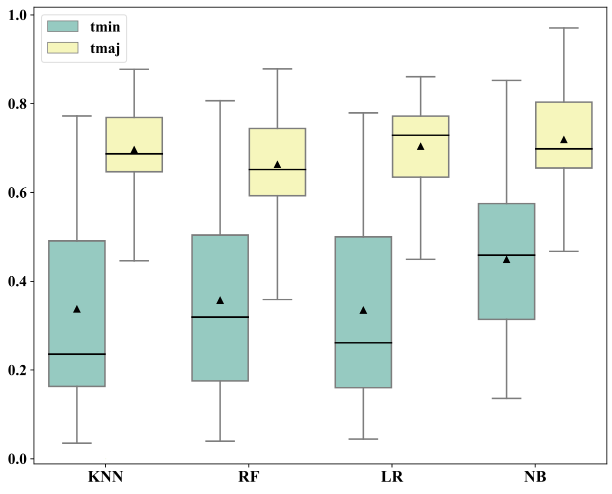


Fig. 5. The boxplots of t_{min} and t_{maj} of ROCT on the 29 datasets

ROCT achieves a significantly better overall performance compared with the existing class overlap cleaning techniques in terms of AUC and $balance$. ROCT also obtains significantly higher pd values. With respect to the pf values, Tomek performs the best. However, this is at the expense of degrading the ability of prediction models to find defects. Therefore, ROCT should be considered as an effective technique to alleviate COP in SDP.

VII. THREATS TO VALIDITY

In this study, 29 datasets collected from various repositories are adopted. These datasets are measured by various metrics. However, there are still some other datasets measured by other metrics. It will become a threat to the validity of our conclusion if our results fail to generalize to other datasets or other types of metrics. However, these datasets and these metrics were widely adopted in previous studies. Their performances are stable and satisfactory. Four common classifiers are selected, and the hyperparameters of these classifiers are set to be the default values. The conclusion may vary on the other classifiers or with different hyperparameter settings. We detail the settings of the experiment. Therefore, it would be easy for others to replicate our work by adopting different classifiers or hyperparameter settings on any available dataset. During the experiment, biases and variances may be introduced. To minimize biases and variances, the 5-fold cross-validation method is adopted, and the experiment is repeated ten times. We adopt four performance measures, one of which is threshold-independent (i.e., AUC), and the other three are threshold-dependent (i.e., $balance$, pd, and pf). There are also many other performance measures such as F-measure or G-mean. If different performance measures are adopted, the conclusion may be different. We plan to adopt more performance measures to validate the performance of ROCT in the future.

VIII. CONCLUSION AND FUTURE WORK

Previous studies have pointed out that COP plays a more important role in affecting the performance of prediction models than CIP. However, few works have been done to alleviate COP compared with CIP. Moreover, the performance of the existing class overlap cleaning techniques is heavily dependent on the proper settings of the hyperparameters. To alleviate these issues, we propose a novel class overlap cleaning technique-ROCT, which gets rid of the difficulty in deciding the optimal hyperparameters. ROCT directly identifies the class overlapping area by looking for the optimal radius of the hypersphere decided by each instance. The instances with the opposite label of the center instance in these hyperspheres are identified as the overlapping instances and removed. Empirical experiments are conducted to validate the performance of ROCT. The experimental results show that ROCT significantly improves the overall performance of prediction models compared with the existing class overlap cleaning techniques in terms of AUC and $balance$. Based on the experimental results, we recommend applying ROCT to alleviate COP in SDP, thereby improving the performance of prediction models.

For future study, we plan to generalize ROCT to more datasets with different metrics on more classifiers, and validate its performance using more performance measures. We also plan to conduct an empirical study to investigate the performance of the existing class overlap cleaning techniques. Moreover, we plan to propose a technique that can alleviate COP and CIP simultaneously.

IX. 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. Yu, J. Liu, J. W. Keung, Q. Li, K. E. Bennin, Z. Xu, J. Wang, and X. Cui, "Improving ranking-oriented defect prediction using a cost-sensitive ranking svm," *IEEE Transactions on Reliability*, vol. 69, no. 1, pp. 139–153, 2019.

[2] X. Yu, M. Wu, Y. Jian, K. E. Bennin, M. Fu, and C. Ma, "Cross-company defect prediction via semi-supervised clustering-based data filtering and mstra-based transfer learning," *Soft Comput.*, vol. 22, no. 10, pp. 3461–3472, 2018.

[3] S. Feng, J. Keung, X. Yu, Y. Xiao, K. E. Bennin, M. A. Kabir, and M. Zhang, "Coste: Complexity-based oversampling technique to alleviate the class imbalance problem in software defect prediction," *Information and Software Technology*, p. 106432, 2020.

[4] X. Yu, J. Liu, Z. Yang, X. Jia, Q. Ling, and S. Ye, "Learning from imbalanced data for predicting the number of software defects," in *28th IEEE International Symposium on Software Reliability Engineering, ISSRE 2017, Toulouse, France, October 23-26, 2017*, pp. 78–89, IEEE Computer Society, 2017.

[5] B. Das, N. C. Krishnan, and D. J. Cook, "Handling class overlap and imbalance to detect prompt situations in smart homes," in *2013 IEEE 13th International Conference on Data Mining Workshops*, pp. 266–273, IEEE, 2013.

[6] V. López, A. Fernández, S. García, V. Palade, and F. Herrera, "An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics," *Information sciences*, vol. 250, pp. 113–141, 2013.

[7] L. Gong, S. Jiang, R. Wang, and L. Jiang, "Empirical evaluation of the impact of class overlap on software defect prediction," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 698–709, IEEE, 2019.

[8] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[9] L. Chen, B. Fang, Z. Shang, and Y. Tang, "Tackling class overlap and imbalance problems in software defect prediction," *Software Quality Journal*, vol. 26, no. 1, pp. 97–125, 2018.

[10] D. L. Wilson, "Asymptotic properties of nearest neighbor rules using edited data," *IEEE Transactions on Systems, Man, and Cybernetics*, no. 3, pp. 408–421, 1972.

[11] D. Devi, B. Purkayastha, *et al.*, "Redundancy-driven modified tomek-link based undersampling: a solution to class imbalance," *Pattern Recognition Letters*, vol. 93, pp. 3–12, 2017.

[12] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, no. 4-5, pp. 531–577, 2012.

[13] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the nasa software defect datasets," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1208–1215, 2013.

[14] J. S. Shirabad and T. J. Menzies, "The promise repository of software engineering databases," *School of Information Technology and Engineering, University of Ottawa, Canada*, vol. 24, 2005.

[15] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "Relink: recovering links between bugs and changes," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pp. 15–25, 2011.

[16] G. Blanchard and R. Loubere, "High-order conservative remapping with a posteriori mood stabilization on polygonal meshes," 2016.

[17] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in statistics*, pp. 196–202, Springer, 1992.

[18] G. Macbeth, E. Razumiejczyk, and R. D. Ledesma, "Cliff's delta calculator: A non-parametric effect size program for two groups of observations," *Universitas Psychologica*, vol. 10, no. 2, pp. 545–555, 2011.

[19] R. C. Prati, G. E. Batista, and M. C. Monard, "Class imbalances versus class overlapping: an analysis of a learning system behavior," in *Mexican international conference on artificial intelligence*, pp. 312–321, Springer, 2004.

[20] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," in *2011 33rd International Conference on Software Engineering (ICSE)*, pp. 481–490, IEEE, 2011.

[21] W. Tang and T. M. Khoshgoftaar, "Noise identification with the k-means algorithm," in *16th IEEE International Conference on Tools with Artificial Intelligence*, pp. 373–378, IEEE, 2004.

[22] K. E. Bennin, J. Keung, P. Phannachitta, A. Monden, and S. Mensah, "Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction," *IEEE Transactions on Software Engineering*, vol. 44, no. 6, pp. 534–550, 2017.

[23] 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," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[24] A. Saifudin and Y. Yulianti, "Dimensional reduction on cross project defect prediction," in *Journal of Physics: Conference Series*, vol. 1477, p. 032011, IOP Publishing, 2020.

[25] H. Tong, B. Liu, and S. Wang, "Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning," *Information and Software Technology*, vol. 96, pp. 94–111, 2018.

[26] Y. Shao, B. Liu, S. Wang, and G. Li, "A novel software defect prediction based on atomic class-association rule mining," *Expert Systems with Applications*, vol. 114, pp. 237–254, 2018.

[27] K. E. Bennin, J. W. Keung, and A. Monden, "On the relative value of data resampling approaches for software defect prediction," *Empirical Software Engineering*, vol. 24, no. 2, pp. 602–636, 2019.

[28] K. E. Bennin, J. Keung, and A. Monden, "Impact of the distribution parameter of data sampling approaches on software defect prediction models," in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, pp. 630–635, IEEE, 2017.

[29] X. Cai, Y. Niu, S. Geng, J. Zhang, Z. Cui, J. Li, and J. Chen, "An under-sampled software defect prediction method based on hybrid multi-objective cuckoo search," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 5, p. e5478, 2020.

[30] D. Rey and M. Neuhäuser, "Wilcoxon-signed-rank test," in *International encyclopedia of statistical science*, pp. 1658–1659, Springer, Berlin, Heidelberg, 2011.

[31] V. B. Kampenes, T. Dybå, J. E. Hannay, and D. I. Sjøberg, "A systematic review of effect size in software engineering experiments," *Information and Software Technology*, vol. 49, no. 11-12, pp. 1073–1086, 2007.