Hindawi
Mathematical Problems in Engineering
Volume 2020, Article ID 5712461, 16 pages
https://doi.org/10.1155/2020/5712461

Research Article

Combining Imbalance Learning Strategy and Multiclassifier Estimator for Bug Report Classification

Shikai Guo,[1,2] **Siwen Wang**,[2] **Miaomiao Wei**,[2] **Rong Chen**,[1] **Chen Guo**,[2] **and Hui Li**[1]

[1]*The School of Marine Electrical Engineering, Dalian Maritime University, Dalian 116026, China*
[2]*The College of Information Science and Technology, Dalian Maritime University, Dalian 116026, China*

Correspondence should be addressed to Chen Guo; dmuguoc@126.com

Received 23 September 2019; Revised 29 October 2019; Accepted 8 November 2019; Published 17 February 2020

Guest Editor: Weifeng Pan

Since a large number of bug reports are submitted to the bug repository every day, efficiently assigning bug reports to the correct developer is a considerable challenge. Because of the large differences between the different components of different projects, the current bug classification mainly relies on the components of the bug report to dispatch bug reports to the designated developer or developer community. Unfortunately, the component information of the bug report is filled in by default according to the bug submitter and the result is often incorrect. Thus, an automatic technology that can identify high-impact bug reports can help developers to be aware of them early, rectify them quickly, and minimize the damages they cause. In this paper, we propose a method based on the combination of imbalanced learning strategies such as random undersampling (RUS), random oversampling (ROS), synthetic minority oversampling technique (SMOTE), and AdaCost algorithms with multiclass classification methods, OVO and OVA, to solve bug reports component classification problem. We investigate the effectiveness of different combinations, i.e., variants, each of which includes a specific imbalance learning strategy and a specific classification algorithm. We mainly perform an analytical study on five open bug repositories (Eclipse, Mozilla, GCC, OpenOffice, and NetBeans). The results show that different variants have different performance for bug reports component identification and the best performance variants are combined with the imbalanced learning strategy RUS and the OVA method based on the SVM classifier.

1. Introduction

There are many studies on bug report predictions that have been performed to help reduce software quality issues [1–5]. Software quality requires a great deal of effort in the testing and debugging process. However, in many cases, the developer's resources and time are limited, so many bugs accumulate and are not fixed in the bug repository.

Anvik et al. reported their personal communication with a Mozilla triager who explains, "everyday, almost 300 bugs appear that need triaging, which is far too much for only the Mozilla programmers to handle" [6]. Therefore, it is especially important to find an effective method for improving the efficiency of bug allocation and resolution. Many developer-recommended methods have been proposed to solve bug classification problems by recommending suitable developers for bug reports to improve the efficiency of bug fixes. Xie et al. proposed a developer-recommended method based on a topic model, using historical bug-solving records to build topic models, simulating developers' interest and expertise in bug-solving activities, and providing a helpful developer recommendation list for new bug reports [7]. On this basis, Xia et al. proposed a bug-based analysis between reports and a developer-based analysis for recommending a suitable developer list for new bug reports by calculating the relevance score [8]. Many researchers have proposed bug prediction techniques to prioritize software testing and debugging that can identify flawed components for developers and conduct considerable research on defect prediction. These techniques predict the allocation model based on features such as code lines, code complexity, and the number of modified files [9–11]. Yang et al. proposed using deep

learning techniques to predict changes in bug reports, extracting a set of expression features from initial variation features through deep confidence network algorithms, and constructing machine learning classifiers based on these expression features [12].

However, bug classification still has many problems and faces many challenges. Large-scale and low-quality bug data in bug repositories can prevent the usage of automatic bug classification techniques. Since software bug data are free-form text data, it is necessary to generate well-processed bug data to facilitate the application [13–15]. Tamrawi et al. proposed a caching model based on fuzzy sets and knowledge based on the professional repair of developers. The fuzzy set is defined as the relevant technical terms related to the bug report activities that developers have previously participated in. Developers are ranked by calculating the score between technical terms and bug reports [16]. Alenezi et al. proposed an efficient bug classification method based on the term selection method and a naive Bayesian classifier to construct a prediction model. This method improves the efficiency of bug classification by reducing the dimensions of the terms [17]. Some researchers have proposed automatic bug-dispatching techniques, such as support-vector machine (SVM) [18], *k*-nearest neighbor algorithm (KNN) [19], and naive Bayes multinomial (NBM) [20], to ensure that bug reports are assigned to the appropriate developers to improve the accuracy of bug allocation. Xia et al. used different combinations of imbalanced learning strategies and text classification algorithms to identify high-impact bug reports. The problem of class imbalance in bug reports is solved through imbalancing the processing of data.

Because of the large differences in component categories in different open source projects in the bug repository, it is clear from the analysis that managers rely on the component categories of bug reports to dispatch bug reports to a specific developer or developer group. The multiclass classification method OVO does not require retraining all classifiers when adding samples and only needs to retrain the classifiers associated with the added samples. The multiclass classification method OVA only needs to train the same number of classifiers, and the training time is relatively fast. In this paper, we propose an automatic bug reports component classification method that combines the imbalanced learning strategies such as random undersampling (RUS), random oversampling (ROS), synthetic minority oversampling technique (SMOTE), and AdaCost algorithms with the multiclass classification methods, OVO and OVA, to solve the automatic bug reports component classification problem. According to the mechanism of bug report classification, each bug report component is assigned to a specific developer or several developers. We recommend the appropriate developers to implement bug report classification through bug component classification. Since different classification algorithms have different sensitivities to different categories, we explore the effectiveness of different variants to find the optimal variant [21], i.e., each variant contains an imbalanced learning strategy (ILS) and a classification method. To verify the validity of our proposed model, we conduct experiments on five open source datasets: Mozilla, GCC, NetBeans, OpenOffice, and Eclipse.

The contributions of this article are as follows:

(i) We propose a new model that combines imbalanced learning strategies and multiclass classification methods to implement bug reports component classification

(ii) We improve bug reports component classification performance through different combinations of imbalanced learning strategies and classification algorithms, i.e., variants.

(iii) The validity of our proposed model is verified by experiments for the Mozilla, GCC, NetBeans, OpenOffice, and Eclipse projects, and the accuracy, precision, recall, and F1 are used as the evaluation metrics.

The remainder of this paper is organized as follows. The background knowledge and motivations are discussed in Section 2. The design of our approach is discussed in Section 3. The experimental design and results are presented in Section 4, and the conclusions are discussed in Section 5.

2. Background Knowledge and Motivations

2.1. Background Knowledge. Since the number of daily bugs is large and properly assigned and the human triager has difficulty grasping all the knowledge about bugs [22–25], it is time consuming and error prone for humans to manually classify bugs. Existing work uses text-based classification methods to assist in bug classification, for example, [26–29]. Existing work uses a text-based classification approach to assist in preventing misclassification in recommending the correct developer. In such an approach, the summary and description of the bug report are extracted as textual content and the developer who can fix the bug is marked as a label for classification. Then, the appropriate developer is predicted for the new bug report. Since the number of bug reports submitted to the bug repository is very large, during the bug classification process, developers resolve as many bug reports with a high degree of impact and severity as possible. Severity has become a key factor in determining the priority for bug fixes. A number of prediction methods for bug reporting severity labels have been proposed.

Xuan et al. used the modified REP algorithm and *K*-nearest neighbor algorithm to predict the severity of bug reports and fixer recommendations [30]. This method uses a topic model to find the topic to which each bug belongs, introduces topics to enhance similarity function REP, and uses a *K*-nearest neighbor algorithm to search historical bug reports similar to the new bug report. Based on features extracted from the nearest neighbor of the new bug severity prediction, fixer recommendations are realized. Zhang et al. proposed a new automated method, SEVERIS, which helps test engineers assign severity levels to bug reports [31]. Menzies and Marcus compared the text classification algorithms such as naive Bayes, naive Bayes multinomial, *K*-nearest neighbor, and support-vector machine to determine which particular algorithm is most suitable for bug reporting severity level prediction. The

results show that the naive Bayes multinomial algorithm has the highest classification accuracy [32]. Lamkanfi et al. proposed a new method of utilizing information retrieval by analyzing the severity label assigned by historical bug reports. Based on the document similarity function of BM25, the severity label is predicted for the new bug report [33]. Bug reports have serious imbalances. Tian et al. proposed a new sampling technique, CR-SMOTE, to enhance the classification of bug reports with real imbalanced severity distributions [34]. This method uses the RSMOTE sampling method combined with the ELM algorithm to achieve bug severity prediction. In subsequent research work, Chen proposed a fuzzy integral fusion multi-RSMOTE method to solve the problem of data distribution imbalance for the randomness problem of RSMOTE [35, 36]. In order to address the imbalance of the dataset, Guo et al. proposed an enhanced oversampling approach called CR-SMOTE to enhance the classification of bug reports with a realistically imbalanced severity distribution [37]. Pan et al. proposed an approach to empirically investigate the static and evolving topological properties enclosed in the weighted software networks by using weighted k-core decomposition [1]. In this study, Pan et al. explored the structural properties of the multilayer software network at the class level by progressively merging layers together, where each coupling type such as inheritance, implements, and method call defines a specific layer [38]. Jiang et al. proposed the ROSF method combining both information retrieval and supervised learning and recommend top-k code snippets for a given free-form query based on two stages [39, 40]. On this basis, Pan et al. proposed a novel approach to identify key class candidates in object-oriented software [41–43]. Chai et al. proposed an approach to cluster mashup services and determine the cluster number based on a genetic algorithm [44, 45]. To reduce the time developers spent analyzing bug reports, Jiang et al. used crowdsourced data to infer and summarized the valid attributes of bug reports [14]. To improve the quality of detection bug reports, Chen et al. proposed a new framework called the test report augmentation framework (TRAF) to help developers better understand and fix bug reports [14, 46–48].

2.2. Motivations. We find that actual data always contain noise and redundancy [49–51]. Noise data can mislead data analysis techniques, while redundant data can increase the cost of data processing [52]. In the bug repository, all bug reports are filled by developers in natural language. As the scale grows, low-quality bugs are accumulated in the bug repository. Such large-scale and low-quality bug data may undermine the effectiveness of bug fixes [53, 54]. Figure 1 shows the distribution of components in the top 10,000 bug reports for the five datasets. Because of the large number of component categories, it is difficult to achieve accurate bug report classification based on current classification methods. Table 1 shows that each developer performs an average of 2-3 components with a single component allocation. In Table 1, the second column represents an average of how each component can be solved by several developers. The third column represents how many components each developer can solve on average. Therefore, the rational allocation of components directly affects the classification efficiency of bug reports. The current component information is filled in by the bug reporter and is the default option and cannot be used directly. Therefore, focus should be on the allocation of the bug reports component.

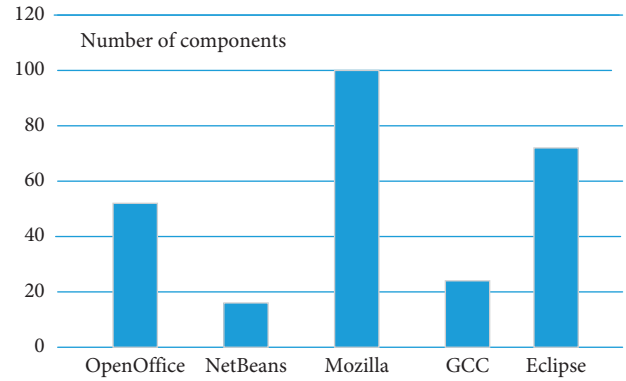

Figure 1: The number of different components in the top 10,000 bug reports for the OpenOffice, NetBeans, Mozilla, GCC, and Eclipse projects.

Table 1: Average of how each component is handled by each developer for the OpenOffice, NetBeans, Mozilla, GCC, and Eclipse projects.

Date Set	Developer/ per_component	Component/ per_developer
OpenOffice	19.42	3.29
NetBeans	19.56	2.84
Mozilla	16.59	2.52
GCC	29.39	2.86
Eclipse	6.79	1.24

Taking the OpenOffice dataset as an example, 27 components in the dataset are assigned to 1 to 5 developers, 3 components are assigned to 6 to 10 developers, 7 components are assigned to 11 to 20 developers, 6 components are assigned to 21–30 developers, and 10 components are assigned to more than 30 developers. Each component is assigned to approximately 20 developers. On average, each developer handles three components. Therefore, we can more accurately assign a more appropriate developer to the bug report by identifying the bug report component.

3. Methodology

In this section, we present the overall model for bug component allocation problems and detail the algorithms in the overall framework.

3.1. Overview. Inspired by the motivation in Section 2, we propose a new method by combining an imbalance learning strategy (ILS) with a multiclass classification method for bug

component classification problems. The model consists of the following three parts. (1) Data preprocessing: we use a text categorization technique to convert each bug report into a word vector based on the vector space model, which is mentioned in [34, 41]. (2) Imbalanced processing of data: because of the serious category imbalance in the dataset, we use four imbalanced learning strategies, RUS, ROS, SMOTE, and AdaCost, to process the data and obtain a balanced dataset to make the classification results more accurate. (3) Multiclass classification of data: we use the multiclass classification methods, OVO and OVA, for the balanced dataset to classify the bug component and solve the bug component classification problem. Since different classification algorithms are sensitive to different categories, we analyze the effectiveness of different variants on the classification of bug reports components, that is, an imbalanced learning strategy combined with a specific classification algorithm. Figure 2 shows the overall framework of our model.

3.2. Imbalanced Learning Strategy. Currently, the strategy for solving the problem of class imbalance of data is mainly divided into two directions. One direction starts with the data training set and reduces the class imbalance of the dataset by changing the sample distribution of the training set. The other direction starts with the learning algorithm according to the algorithm when solving the problem of class imbalance and modifies the algorithm to improve its efficiency. Many data sampling techniques have been introduced in the literature [30, 41, 55]. In our study, we choose RUS, ROS, and SMOTE methods based on changing datasets,and the AdaCost algorithm based on cost sensitivity.

3.2.1. Random Undersampling Method. The random undersampling (RUS) method directly undersamples most of the samples in the training set, that is, it removes some samples in the majority class so that the number of positive and negative examples is close and then learns.

That is, some samples are randomly selected from the majority class S_{maj} to form a sample set E, and then the sample set E is removed from S_{maj} to obtain a new dataset $S_{\text{new}} = S_{\text{maj}} - E$.

3.2.2. Random Oversampling Method. The random oversampling (ROS) method directly oversamples a small number of samples in the training set, that is, it increases some minority samples so that the number of positive and negative examples is close and then learns.

That is, some samples are randomly selected from the minority class S_{min}, and then the sample set E is generated by copying the selected samples, and they are added to S_{min} and the original dataset is expanded to obtain a new minority class set $S_{\text{new}} = S_{\text{min}} + E$.

3.2.3. Synthetic Minority Oversampling Technique. The basic idea of the SMOTE method is to randomly select a sample $\hat{x}_i$ from its nearest neighbors for each minority class sample x_i ($\hat{x}_i$ is a sample of minority classes) and then randomly select a point on the line between x_i and $\hat{x}_i$ as a newly synthesized sample of minority classes.

The specific method for synthesizing new minority samples by the SMOTE method is as follows:

(i) For each sample x_i in minority classes, the distance from x_i to all the samples in the sample set S_{min} of minority classes is calculated according to the Euclidean distance, and its k-nearest neighbors are obtained.

(ii) A sampling ratio is set according to the sample imbalance ratio to determine the sampling magnification N. For each minority sample x_i, several samples are selected randomly from its k-nearest neighbors, assuming that $\hat{x}_i$ is selected.

(iii) For each randomly selected nearest neighbor $\hat{x}_i$, a new sample is constructed with x_i according to the following formula:

$$x_{\text{new}} = x_i + \text{rand}(0,1) \times \left(\hat{x}_i - x_i\right). \tag{1}$$

3.2.4. AdaCost Algorithm. The AdaCost algorithm learns a classifier by iteration and updates the weight of the sample according to the performance of the current classifier. The weight update strategy greatly increases the weight of the costly misclassification sample, and the weight of the correct classification sample is appropriately reduced so that the weight reduction is relatively small. The overall idea is that the cost of the high sample weight is greatly reduced. The sample weights are updated according to the following formula [56]:

$$D_{t+1}(i) = \frac{D_t(i)\exp\left(-\alpha_t h_t(x_i) y_i \beta_i\right)}{Z_t}. \tag{2}$$

$\beta_+ = -0.5C_i + 0.5$, β_+ indicates the value of β in the case where the sample is correctly classified. $\beta_- = 0.5C_i + 0.5$, β_- indicates the value of β in the case where the sample is misclassified.

3.3. Multiclass Classification Method

3.3.1. OVO Method. Assuming there are m categories, the method creates a binomial classifier for the two categories and obtains $k = m * (\text{m} - 1)/2$ classifiers. When classifying new data, the k classifiers are sequentially used for classification. Each classification is equivalent to one vote, and each of the classification results is equivalent to which class is voted for. After classifying using all k classifiers, it is equivalent to k-th voting and the class with the most votes is selected as the final classification result. The following is a description of the structure of the algorithm.

In line 1, y is initialized to null. Lines 2–4 indicate that a classifier is designed between any two samples and classifiers need to be designed. Lines 5–7 indicate that the classification results are obtained. Lines 8–13 represent the voting strategy, and if the sample belongs to the class, one is added. Line 14

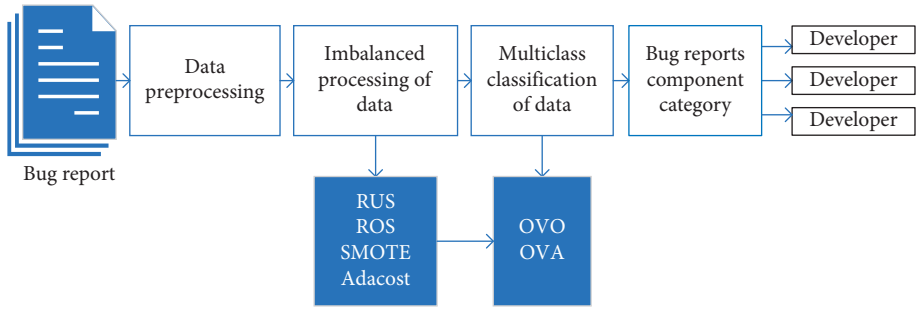

Figure 2: The overall framework of our model.

indicates that the class with the most votes lasting is the class of the unknown sample. Line 17 indicates that the class score with the most votes is returned.

3.3.2. OVA Method. Assuming there are n categories, the method establishes n binomial classifiers; each classifier classifies one of the categories and the remaining categories. When making predictions, we use the n binomial classifiers to classify the data and obtain the probability that the data belong to the current class, and one of the categories with the highest probability is selected as the final prediction result. The following is a description of the structure of the algorithm.

In line 1, y is initialized to null. Lines 2–4 indicate that there are n groups of classifications, i.e., n classifiers. Lines 5–11 indicate that each group of classification results $h_\theta^{(i)}(x)$ is obtained, and the value with the highest probability is selected as the prediction result. Line 12 indicates that the maximum value of the classification value is returned.

4. Experimental Design

4.1. Experimental Datasets. To demonstrate the effectiveness of the proposed approach, we carry out a series of experiments on bug repositories of five large open source projects, namely, Mozilla [57], GCC [58], NetBeans [59], OpenOffice, [60] and Eclipse [61]. By analyzing the components of the bug datasets in these five open source projects, we found that there were category imbalances in all five projects. Table 2 shows the different components and their numbers of bug reports in these five large open source projects. From Table 2, we found that the maximum number of community components reached 10,473, while the number of incubator components was only 19 for the Eclipse project.

Therefore, we need to reduce the datasets; we read the first 1,000 rows of data in five open source projects, and Table 3 shows the word frequency in the five datasets and the total number of bug reports for the original dataset. To more accurately identify the category of bug reports, we removed the number of bug report columns with word frequencies less than 5 and bug reports with a category of less than 50. Table 4 shows the categories of the processed datasets and the number of columns after word frequency reduction.

4.2. Evaluation Metrics. We use accuracy, precision, recall, and F1 as our evaluation metrics. These metrics are commonly used measures for evaluating classification performance [62].

The number of true positives (TP) is the number instances that are correctly divided into positive cases, that is, the number of instances that are actually positive examples and are classified into positive examples by the classifier. The number of false positives (FP) is the number of instances that are incorrectly divided into positive examples, that is, the number of instances that are actually negative but are classified as positive by the classifier. The number of false negatives (FN) is the number of instances that are incorrectly divided into negative examples, that is, the number of instances that are actually positive but are classified as negative by the classifier. The number of true negatives (TN) is the number of instances that are correctly divided into negative examples, that is, the number of instances that are actually negative and are classified as negative by the classifier. Based on the values of TP, FP, FN, and TN, the accuracy, precision, recall, and F1 are calculated as follows.

4.2.1. Accuracy. Accuracy is the number of correctly classified samples divided by the total number of samples. Generally, the higher the correct rate, the better the classifier. We formally define the accuracy as follows:

$$\text{Accurancy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{FP} + \text{TN}}. \tag{3}$$

4.2.2. Precision. Precision is the ratio that is actually divided into positive examples. We formally define the precision as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{4}$$

4.2.3. Recall. Recall is the proportion of positive cases that are classified as positive examples. Mathematically, recall is defined as

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{5}$$

Input:
class y, *sample* i, *sample* j, *learning algorithm* H *number of training* T
Output:
class_score
(1) $y \longleftarrow \varnothing$.
(2) **For** i from 1 to m do
(3) **For** j from 2 to m do
(4) **Create a binomial classifier with samples of class i and class j, and for all samples, obtain classifiers**
(5) **Train the classifier to obtain the classification result.**
(6) $h_\theta^{(i)}(x) = H(y = i \mid x; \theta)$
(7) $h_\theta^{(j)}(x) = H(y = j \mid x; \theta)$
(8) Voting strategy:
(9) **if** $h_\theta^{(i)}(x) > h_\theta^{(j)}(x)$
(10) y_i increase one vote.
(11) **Else**
(12) y_j increase one vote.
(13) **End if**
(14) $class\_score = \max_{i=1,\ldots,m}\left\{\sum_{1\le j\ne i\le m,t=1} T_{y_i}, \sum_{1\le j\ne i\le m,t=1} T_{y_i}\right\}$
(15) **End for**
(16) **End for**
(17) Return *class_score*

Algorithm 1: OVO method.

Input:
class y, *sample* i, *sample* j, *learning algorithm* H
Output:
H(x)
(1) $y \longleftarrow \varnothing$.
(2) **For** all y from 1 to n do
(3) Choosing one class and lumping all the others into a single second class, obtain n classifiers.
(4) Repeat the previous step.
(5) Train n classifiers $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y = i$.
(6) $h_\theta^{(1)}(x) = H(y = 1 \mid x; \theta)$
(7) $h_\theta^{(2,3,\ldots,n)}(x) = H(y = 2, 3, \ldots, n \mid x; \theta)$
(8) Record the probability value of the sample belonging to the current class.
(9) Repeat steps 6, 7, and 8 until all the n classifiers have been trained, and separately record the probability values of each sample belonging to the current class.
(10) Pick the class i that maximizes.
(11) **End for**
(12) Return $H(x) = \max_i(h_\theta^{(i)}(x))$

Algorithm 2: OVA method.

Table 2: Distribution of bug reports components in five open source projects.

Projects	The largest number of component names	Maximum number of components	The smallest number of component names	Minimum number of components	Total number of bug reports (row ∗ column)
Mozilla	Cloud services	4243	Composer	15	(18793 ∗ 35365)
GCC	gcc	13254	Classpath	710	(13964 ∗ 52405)
NetBeans	Java	11593	Groovy	341	(19451 ∗ 40445)
OpenOffice	Impress	2209	Xml	1	(23481 ∗ 50359)
Eclipse	Community	10473	Incubator	19	(40938 ∗ 66029)

Table 3: Word frequency distribution of five datasets and total number of bug reports.

Project	Number of columns with word frequency 1	Number of columns with word frequency 2	Number of columns with word frequency 3	Number of columns with word frequency 4	Number of columns with word frequency below 5	Number of columns with word frequency greater than or equal to 5	The total number of training sets for bug reports
Mozilla	14822	2714	1310	786	19632	4406	(9048 ∗ 24038)
GCC	12438	2627	1287	816	17168	4638	(4851 ∗ 21806)
NetBeans	14121	3507	1616	874	20118	6780	(8538 ∗ 26898)
OpenOffice	14714	2887	1278	832	19711	4591	(9922 ∗ 24302)
Eclipse	15032	3232	1532	853	20649	4854	(9992 ∗ 25503)

Table 4: Category distribution of processed datasets and the number of columns after word frequency reduction.

Project	Maximum number of categories for the bug component	Minimum number of categories for the bug component	Number of columns with word frequency greater than 5
Mozilla	1089	52	4406
GCC	958	86	4638
NetBeans	2170	73	6780
OpenOffice	2770	82	4591
Eclipse	1064	52	4854

Table 5: Experimental results of the OVO and OVA methods based on the NBM, KNN, and SVM classifiers for the Mozilla dataset.

Project	Classifier	Evaluation metrics	OVO method	OVA method
Mozilla	NBM	Accuracy	0.5066	0.5066
		Precision	0.5336	0.5336
		Recall	0.5066	0.5066
		F1	0.5074	0.5074
	KNN	Accuracy	Memory overflow	0.2674
		Precision	Memory overflow	0.3785
		Recall	Memory overflow	0.2674
		F1	Memory overflow	0.2814
	SVM	Accuracy	0.3569	0.5928
		Precision	0.704	0.6015
		Recall	0.3569	0.5928
		F1	0.3626	0.5787

Table 6: Experimental results of the OVO and OVA methods based on the NBM, KNN, and SVM classifiers for the GCC dataset.

Project	Classifier	Evaluation metrics	OVO method	OVA method
GCC	NBM	Accuracy	0.6343	0.6179
		Precision	0.6462	0.6385
		Recall	0.6343	0.6179
		F1	0.6389	0.6207
	KNN	Accuracy	0.4356	0.4356
		Precision	0.461	0.4417
		Recall	0.4356	0.4356
		F1	0.3817	0.4239
	SVM	Accuracy	0.5705	0.6539
		Precision	0.6723	0.6685
		Recall	0.5705	0.6539
		F1	0.4977	0.6153

Table 7: Experimental results of the OVO and OVA methods based on the NBM, KNN, and SVM classifiers for the Eclipse dataset.

Project	Classifier	Evaluation metrics	OVO method	OVA method
Eclipse	NBM	Accuracy	0.7332	0.7044
		Precision	0.7706	0.7532
		Recall	0.7332	0.7044
		F1	0.7429	0.7185
	KNN	Accuracy	0.4947	0.5292
		Precision	0.6949	0.6382
		Recall	0.4947	0.5292
		F1	0.5045	0.5462
	SVM	Accuracy	0.6417	0.7892
		Precision	0.7836	0.8033
		Recall	0.6417	0.7892
		F1	0.616	0.7781

Table 8: Experimental results of the OVO and OVA methods based on the NBM, KNN, and SVM classifiers for the OpenOffice dataset.

Project	Classifier	Evaluation metrics	OVO method	OVA method
OpenOffice	NBM	Accuracy	0.425	0.407
		Precision	0.5173	0.5019
		Recall	0.425	0.407
		F1	0.4438	0.4226
	KNN	Accuracy	0.3598	0.3747
		Precision	0.4967	0.3996
		Recall	0.3598	0.3747
		F1	0.3584	0.37
	SVM	Accuracy	0.4255	0.5205
		Precision	0.6025	0.5321
		Recall	0.4255	0.5205
		F1	0.3497	0.4964

Table 9: Experimental results of the OVO and OVA methods based on the NBM, KNN, and SVM classifiers for the NetBeans dataset.

Project	Classifier	Evaluation metrics	OVO method	OVA method
NetBeans	NBM	Accuracy	0.5024	0.4941
		Precision	0.5341	0.5117
		Recall	0.5024	0.4941
		F1	0.5097	0.4973
	KNN	Accuracy	0.3939	0.354
		Precision	0.4594	0.4077
		Recall	0.3939	0.354
		F1	0.3834	0.3508
	SVM	Accuracy	0.4614	0.616
		Precision	0.6537	0.626
		Recall	0.4614	0.616
		F1	0.4432	0.6051

4.2.4. F-Measure. *F*-measure is also known as *F*-score. When the precision (P) and recall (R) indicators sometimes appear contradictory, you need to use the indicator to consider both of them. *F*-measure is the weighted harmonic average of precision and recall.

$$F - \text{measure} = \frac{(\alpha^2 + 1) * P * R}{\alpha^2 * (P + R)}. \quad (6)$$

When $a = 1$, it is the most common F1, i.e.,

$$F1 = \frac{2 * P * R}{P + R}. \quad (7)$$

5. Experimental Results

In this section, the experimental results are discussed in relation to the specific research questions.

5.1. RQ1: Which Classification Method Has Better Classification Effect Based on NBM, KNN, and SVM Classifiers' OVO and OVA Methods? To answer this question, we use the OVO and OVA multiclass classification methods based on NBM, KNN, and SVM classifiers, which together contain six variants (i.e., OVO method-based NBM classifier, OVO method-based KNN classifier, OVO method-based SVM classifier, OVA method-based NBM classifier, OVA method-based KNN classifier, and OVA method-based SVM classifier) and record the experimental results. We use accuracy, precision, recall, and F1 as evaluation criteria. Tables 5–9 show our experimental results for the Mozilla, GCC, NetBeans, OpenOffice, and Eclipse datasets.

From Tables 5–9, it can be seen that the OVA multiclass classification method based on the SVM classifier improves the effect most obviously for the five datasets. The experimental results of the OVA method based on the SVM classifier improved by 0.5928, 0.6015, 0.5928, and 0.5787, respectively, for the Mozilla dataset. The experimental results of the OVA method based on the SVM classifier improved by 0.6539, 0.6685, 0.6539, and 0.6153, respectively, for the GCC dataset. The experimental results of the OVA method based on the SVM classifier improved by 0.7892, 0.8033, 0.7892, and 0.7781, respectively, for the Eclipse dataset. The experimental results of the OVA method based on the SVM classifier improved by 0.5205, 0.5321, 0.5205, and 0.4964, respectively, for the OpenOffice dataset. The experimental results of the OVA method based on the SVM classifier improved by 0.6160, 0.6260, 0.6160, and 0.6051, respectively, for the NetBeans dataset. Therefore, for the Mozilla, GCC, NetBeans, OpenOffice, and Eclipse datasets, the OVA method based on the SVM classifier has greater efficiency in solving bug reports component classification problems.

5.2. RQ2: What Is the Impact of Imbalanced Learning Strategies on the Multiclass Classification OVO Method in Solving Bug Reports Component Allocation Problems? Specifically, the question explores whether an imbalanced learning strategy has an impact on the OVO classification method. To answer this question, we use the imbalanced learning strategies RUS, ROS, SMOTE, and AdaCost algorithms to process the Mozilla, GCC, Eclipse, OpenOffice, and NetBeans datasets and then use the multiclass classification method OVO based on SVM, KNN, and NBM classifiers to

Table 10: The results of our classification using the ILS (RUS, ROS, SMOTE, and AdaCost) and OVO method based on SVM, KNN, and NBM classifiers for the Mozilla dataset.

Project	ILS	Classifier	Evaluation metrics	OVO method
Mozilla	RUS	SVM	Accuracy	0.2
			Precision	0.525
			Recall	0.2
			F1	0.2294
		KNN	Accuracy	0.0918
			Precision	0.2724
			Recall	0.0918
			F1	0.1042
		NBM	Accuracy	0.4408
			Precision	0.4611
			Recall	0.4408
			F1	0.4372
	ROS	SVM	Accuracy	0.9288
			Precision	0.9415
			Recall	0.9288
			F1	0.9316
		NBM	Accuracy	0.894
			Precision	0.893
			Recall	0.894
			F1	0.8906
	SMOTE	SVM	Accuracy	0.3541
			Precision	0.5533
			Recall	0.3541
			F1	0.3817
		NBM	Accuracy	0.4745
			Precision	0.4907
			Recall	0.4745
			F1	0.4759
	AdaCost	SVM	Accuracy	0.5392
			Precision	0.6477
			Recall	0.5392
			F1	0.534

Table 11: The results of our classification using the ILS (RUS, ROS, SMOTE, and AdaCost) and OVO method based on SVM, KNN, and NBM classifiers for the GCC dataset.

Project	ILS	Classifier	Evaluation metrics	OVO method
GCC	RUS	SVM	Accuracy	0.2905
			Precision	0.4873
			Recall	0.2905
			F1	0.265
		KNN	Accuracy	0.1623
			Precision	0.3543
			Recall	0.1623
			F1	0.0903
		NBM	Accuracy	0.5213
			Precision	0.5024
			Recall	0.5213
			F1	0.5023
	ROS	SVM	Accuracy	0.9016
			Precision	0.9035
			Recall	0.9016
			F1	0.9009
		NBM	Accuracy	0.8601
			Precision	0.8589
			Recall	0.8601
			F1	0.8589
	SMOTE	SVM	Accuracy	0.5036
			Precision	0.5702
			Recall	0.5036
			F1	0.4864
		NBM	Accuracy	0.6271
			Precision	0.6244
			Recall	0.6271
			F1	0.622
	AdaCost	SVM	Accuracy	0.7167
			Precision	0.7127
			Recall	0.7167
			F1	0.682

solve the problem of bug reports component classification. We combine ILS with OVO method based on SVM, KNN, and NBM classifiers and an imbalanced learning strategy combined with a classification method for the five datasets. We use accuracy, precision, recall, and F1 as evaluation criteria.

In addition, first, when we use RUS combined with the OVO method based on SVM, KNN, and NBM classifiers, it is found that RUS combined with the OVO method based on the KNN classifier is the least effective compared with other combinations. Therefore, we use RUS and SMOTE combined with the OVO method based on SVM and NBM classifiers to carry out experiments for the five datasets. According to RQ1, we learned that the OVO and OVA methods based on the SVM classifier work best before the data are imbalanced. Therefore, we only use the algorithm-based AdaCost to combine the OVO method based on the SVM classifier to experiment and observe the experimental results. Tables 10–14 show the results of our experiments using ILS combined with the OVO method for the Mozilla, GCC, NetBeans, OpenOffice, and Eclipse datasets.

From Tables 10–14, it can be seen that the combination of the imbalanced leaning strategy RUS and the OVO method based on SVM and NBM classifiers has higher efficiency in solving the bug reports component classification problem compared to the OVO method based on the KNN classifier for the Mozilla, GCC, NetBeans, OpenOffice, and Eclipse datasets. The combination of imbalanced leaning strategy ROS and the OVO method based on SVM and NBM classifiers has the largest improvement in solving the bug reports component classification problem compared with other combinations for the Mozilla, GCC, NetBeans, OpenOffice, and Eclipse datasets. From Table 10, for the Mozilla dataset, the combination of the ROS and OVO method based on the SVM classifier has the greatest improvement, increasing by 0.9288, 0.9415, 0.9288, and 0.9316, respectively. A combination of the ROS and the OVO method based on the NBM classifier also greatly improved compared with other combinations, increasing by 0.8940, 0.8930, 0.8940, and 0.8906, respectively. The combination of the RUS and OVO method based on the KNN classifier had the lowest improvement, 0.0918, 0.2724, 0.0918, and 0.1042, respectively. The combination of the AdaCost and OVO method based on the SVM classifier had higher improvement compared with the combination of the SMOTE and OVO methods based on the SVM and NBM classifiers,

Table 12: The results of our classification using the ILS (RUS, ROS, SMOTE, and AdaCost) and OVO method based on SVM, KNN, and NBM classifiers for the Eclipse dataset.

Project	ILS	Classifier	Evaluation metrics	OVO method
Eclipse	RUS	SVM	Accuracy	0.1923
			Precision	0.572
			Recall	0.1923
			F1	0.2094
		KNN	Accuracy	0.1713
			Precision	0.4548
			Recall	0.1713
			F1	0.1713
		NBM	Accuracy	0.5734
			Precision	0.6175
			Recall	0.5734
			F1	0.5719
	ROS	SVM	Accuracy	0.9288
			Precision	0.9415
			Recall	0.9288
			F1	0.9316
		NBM	Accuracy	0.917
			Precision	0.9181
			Recall	0.917
			F1	0.9154
	SMOTE	SVM	Accuracy	0.5842
			Precision	0.6729
			Recall	0.5842
			F1	0.5919
		NBM	Accuracy	0.7144
			Precision	0.7288
			Recall	0.7144
			F1	0.7147
	AdaCost	SVM	Accuracy	0.6689
			Precision	0.7966
			Recall	0.6689
			F1	0.6652

Table 13: The results of our classification using the ILS (RUS, ROS, SMOTE, and AdaCost) and OVO method based on SVM, KNN, and NBM classifiers for the OpenOffice dataset.

Project	ILS	Classifier	Evaluation metrics	OVO method
OpenOffice	RUS	SVM	Accuracy	0.1538
			Precision	0.3992
			Recall	0.1538
			F1	0.1635
		KNN	Accuracy	0.1052
			Precision	0.1896
			Recall	0.1052
			F1	0.0958
		NBM	Accuracy	0.4736
			Precision	0.5044
			Recall	0.4736
			F1	0.475
	ROS	SVM	Accuracy	0.9204
			Precision	0.9346
			Recall	0.9204
			F1	0.9225
		NBM	Accuracy	0.8489
			Precision	0.8496
			Recall	0.8489
			F1	0.8437
	SMOTE	SVM	Accuracy	0.4075
			Precision	0.4693
			Recall	0.4075
			F1	0.3844
		NBM	Accuracy	0.4455
			Precision	0.469
			Recall	0.4455
			F1	0.4515
	AdaCost	SVM	Accuracy	0.4856
			Precision	0.528
			Recall	0.4856
			F1	0.4439

increasing by 0.5392, 0.6477, 0.5392, and 0.5340, respectively.

From Table 11, for the GCC dataset, the combination of the ROS and OVO method based on the SVM classifier had the largest improvement, increasing by 0.9016, 0.9035, 0.9016, and 0.9009, respectively. A combination of the ROS and OVO method based on the NBM classifier also greatly improved compared with other combinations, increasing by 0.8601, 0.8589, 0.8601, and 0.8589, respectively. The combination of the RUS and OVO method based on the KNN classifier has the lowest improvement, 0.1623, 0.3543, 0.1623, and 0.0903, respectively. The combination of the AdaCost and OVO method based on the SVM classifier had higher improvement compared to the combination of the SMOTE and OVO method, which increased by 0.7167, 0.7127, 0.7167, and 0.6820, respectively. The combination of the SMOTE and OVO method based on the NBM classifier was more efficient than the combination of the SMOTE and OVO method based on the SVM classifier, with elevations of 0.6271, 0.6244, 0.6271, and 0.6220, respectively. From Table 12, for the Eclipse dataset, the combination of the ROS and OVO method based on the SVM classifier had the greatest improvement, increasing by 0.9288, 0.9415, 0.9288, and 0.9316, respectively. A combination of the ROS and OVO method based on the NBM classifier also greatly improved compared with other combinations, increasing by 0.9170, 0.9181, 0.9170, and 0.9154, respectively. The combination of the RUS and OVO method based on the KNN classifier had the lowest improvement, 0.1713, 0.4548, 0.1713, and 0.1713, respectively. The combination of the SMOTE and OVO method based on the NBM classifier had higher improvement compared with the combination of the SMOTE, AdaCost, and OVO method based on the SVM classifier, which increased by 0.7144, 0.7288, 0.7144, and 0.7147, respectively. From Table 13, for the OpenOffice dataset, the combination of the ROS and OVO method based on the SVM classifier had the greatest improvement, increasing by 0.9204, 0.9346, 0.9204, and 0.9225, respectively. A combination of the ROS and OVO method based on the NBM classifier also greatly improved compared with other combinations, increasing by 0.8489, 0.8496, 0.8489, and 0.8437, respectively. The combination of the RUS and OVO method based on the KNN classifier had the lowest improvement, 0.1052, 0.1896, 0.1052, and 0.0958,

Table 14: The results of our classification using the ILS (RUS, ROS, SMOTE, and AdaCost) and OVO method based on SVM, KNN, and NBM classifiers for the NetBeans dataset.

Project	ILS	Classifier	Evaluation metrics	OVO method
NetBeans	RUS	SVM	Accuracy	0.1729
			Precision	0.2464
			Recall	0.1729
			F1	0.1484
		KNN	Accuracy	0.1518
			Precision	0.3168
			Recall	0.1518
			F1	0.1242
		NBM	Accuracy	0.4008
			Precision	0.4098
			Recall	0.4008
			F1	0.3966
	ROS	SVM	Accuracy	0.9043
			Precision	0.9125
			Recall	0.9043
			F1	0.9058
		NBM	Accuracy	0.8244
			Precision	0.8204
			Recall	0.8244
			F1	0.8175
	SMOTE	SVM	Accuracy	0.4432
			Precision	0.54
			Recall	0.4432
			F1	0.4366
		NBM	Accuracy	0.5019
			Precision	0.5089
			Recall	0.5019
			F1	0.5028
	AdaCost	SVM	Accuracy	0.5645
			Precision	0.6295
			Recall	0.5645
			F1	0.5584

Table 15: The results of our classification using the ILS (RUS, ROS, SMOTE, and AdaCost) and OVA method based on SVM, KNN, and NBM classifiers for the Mozilla dataset.

Project	ILS	Classifier	Evaluation metrics	OVA method
Mozilla	RUS	SVM	Accuracy	0.4612
			Precision	0.493
			Recall	0.4612
			F1	0.4503
		KNN	Accuracy	0.1346
			Precision	0.3396
			Recall	0.1346
			F1	0.156
		NBM	Accuracy	0.3693
			Precision	0.3876
			Recall	0.3693
			F1	0.3612
	ROS	SVM	Accuracy	0.946
			Precision	0.9483
			Recall	0.946
			F1	0.945
		NBM	Accuracy	0.7424
			Precision	0.7683
			Recall	0.7424
			F1	0.742
	SMOTE	SVM	Accuracy	0.4834
			Precision	0.5162
			Recall	0.4834
			F1	0.4901
		NBM	Accuracy	0.4237
			Precision	0.4424
			Recall	0.4237
			F1	0.4188
	AdaCost	SVM	Accuracy	0.4701
			Precision	0.58
			Recall	0.4701
			F1	0.4549

respectively. The combination of the AdaCost and OVO method based on the SVM classifier had higher improvement compared with the combination of the SMOTE and OVO method based on the SVM and NBM classifiers, increasing by 0.4856, 0.5280, 0.4856, and 0.4439, respectively. From Table 14, for the NetBeans dataset, the combination of the ROS and OVO method based on the SVM classifier had the greatest improvement, increasing by 0.9043, 0.9125, 0.9043, and 0.9058, respectively. A combination of the ROS and OVO method based on the NBM classifier also greatly improved compared with other combinations, increasing by 0.8244, 0.8204, 0.8244, and 0.8175, respectively. The combination of the RUS and OVO method based on the KNN classifier had the lowest improvement, 0.1518, 0.3168, 0.1518, and 0.1242, respectively. The combination of the AdaCost and OVO method based on the SVM classifier had higher improvement compared with the combination of the SMOTE and OVO method based on the SVM and NBM classifiers, which increased by 0.5645, 0.6295, 0.5645, and 0.5584, respectively.

Therefore, the combination of the imbalanced leaning strategy RUS and the OVO method based on SVM and NBM classifiers has higher efficiency in solving the bug reports component classification problem compared with other combinations for the Mozilla, GCC, NetBeans, OpenOffice, and Eclipse datasets.

5.3. RQ3: How Much Improvement Does the Classification of Bug Reports Component Have in Combination with the Imbalanced Learning Strategies and the OVA Method? To answer this question, we use the imbalanced learning strategies RUSUS, ROS, SMOTE, and AdaCost algorithms to process the Mozilla, GCC, Eclipse, OpenOffice, and NetBeans datasets and then use the multiclass classification method OVA based on SVM, KNN, and NBM classifiers to solve the problem of bug reports component classification. We combine ILS with OVA method based on SVM, KNN, and NBM classifiers and an imbalanced learning strategy with a classification method for the five datasets. Then, we use accuracy, precision, recall, and F1 as evaluation criteria and we build the classifier with reference to the combination of RQ2. Tables 15–19 show the results of our experiments using ILS combined with the OVA method for the Mozilla, GCC, NetBeans, OpenOffice, and Eclipse datasets.

Table 16: The results of our classification using the ILS (RUS, ROS, SMOTE, and AdaCost) and OVA method based on SVM, KNN, and NBM classifiers for the GCC dataset.

Project	ILS	Classifier	Evaluation metrics	OVA method
GCC	RUS	SVM	Accuracy	0.5042
			Precision	0.4818
			Recall	0.5042
			F1	0.4784
		KNN	Accuracy	0.2222
			Precision	0.4332
			Recall	0.2222
			F1	0.1807
		NBM	Accuracy	0.4786
			Precision	0.4687
			Recall	0.4786
			F1	0.4668
	ROS	SVM	Accuracy	0.919
			Precision	0.9198
			Recall	0.919
			F1	0.9185
		NBM	Accuracy	0.826
			Precision	0.832
			Recall	0.826
			F1	0.8235
	SMOTE	SVM	Accuracy	0.5777
			Precision	0.6044
			Recall	0.5777
			F1	0.5556
		NBM	Accuracy	0.6096
			Precision	0.6186
			Recall	0.6096
			F1	0.6114
	AdaCost	SVM	Accuracy	0.7116
			Precision	0.695
			Recall	0.7116
			F1	0.6899

Table 17: The results of our classification using the ILS (RUS, ROS, SMOTE, and AdaCost) and OVA method based on SVM, KNN, and NBM classifiers for the Eclipse dataset.

Project	ILS	Classifier	Evaluation metrics	OVA method
Eclipse	RUS	SVM	Accuracy	0.5909
			Precision	0.6701
			Recall	0.5909
			F1	0.5906
		KNN	Accuracy	0.2762
			Precision	0.6386
			Recall	0.2762
			F1	0.2967
		NBM	Accuracy	0.5419
			Precision	0.6012
			Recall	0.5419
			F1	0.5404
	ROS	SVM	Accuracy	0.946
			Precision	0.9483
			Recall	0.946
			F1	0.945
		NBM	Accuracy	0.8705
			Precision	0.8745
			Recall	0.8705
			F1	0.8694
	SMOTE	SVM	Accuracy	0.6956
			Precision	0.7412
			Recall	0.6956
			F1	0.7
		NBM	Accuracy	0.6966
			Precision	0.7157
			Recall	0.6966
			F1	0.7018
	AdaCost	SVM	Accuracy	0.7651
			Precision	0.8155
			Recall	0.7651
			F1	0.7673

From Table 15, for the Mozilla dataset, the combination of the ROS and OVA method based on the SVM classifier had the greatest improvement, increasing by 0.9460, 0.9483, 0.9460, and 0.9450, respectively. A combination of the ROS and OVA method based on the NBM classifier also greatly improved compared with other combinations, increasing by 0.7424, 0.7683, 0.7424, and 0.7420, respectively. The combination of the RUS and OVA method based on the KNN classifier had the lowest improvement, 0.01346, 0.3396, 0.1346, and 0.1569, respectively. The combination of the SMOTE and OVA method based on the SVM classifier had higher improvement compared with the combination of the SMOTE and OVA method based on the NBM classifier, and the combination of the AdaCost and OVA method based on the SVM classifier increased by 0.4834, 0.5162, 0.4834, and 0.4901, respectively. From Table 16, for the GCC dataset, the combination of the ROS and OVA method based on the SVM classifier had the largest improvement, increasing by 0.9190, 0.9198, 0.9190 and 0.9185, respectively. A combination of the ROS and OVA method based on the NBM classifier also greatly improved compared to other combinations, increasing by 0.8260, 0.8320, 0.8260, and 0.8235, respectively. The combination of the RUS and OVA method based on the KNN classifier has the lowest improvement, 0.2222, 0.4332, 0.2222, and 0.1807, respectively. The combination of the AdaCost and OVA method based on the SVM classifier had higher improvement compared with the combination of the SMOTE and OVA methods based on the SVM and NBM classifiers, which increased by 0.7116, 0.6950, 0.7116, and 0.6899, respectively. The combination of the SMOTE and OVA methods based on the NBM classifier was more efficient than the combination of the SMOTE and OVA method based on the SVM classifier, with elevations of 0.6069, 0.6186, 0.6096, and 0.6114, respectively. From Table 17, for the Eclipse dataset, the combination of the ROS and OVA method based on the SVM classifier had the greatest improvement, increasing by 0.9460, 0.9483, 0.9460, and 0.9450, respectively. A combination of the ROS and OVA method based on the NBM classifier also greatly improved compared with other combinations, increasing by 0.8705, 0.8745, 0.8705, and 0.8694, respectively. The combination of the RUS and OVA method based on the KNN classifier had the lowest accuracy value, recall value, and F1 value of 0.2762, 0.2762, and 0.2967, respectively. The combination of the AdaCost and OVA method based on the SVM classifier had higher improvement compared with the

Table 18: The results of our classification using the ILS (RUS, ROS, SMOTE, and AdaCost) and OVA method based on SVM, KNN, and NBM classifiers for the OpenOffice dataset.

Project	ILS	Classifier	Evaluation metrics	OVA method
OpenOffice	RUS	SVM	Accuracy	0.4291
			Precision	0.4885
			Recall	0.4291
			F1	0.4325
		KNN	Accuracy	0.1497
			Precision	0.3197
			Recall	0.1497
			F1	0.1534
		NBM	Accuracy	0.4291
			Precision	0.438
			Recall	0.4291
			F1	0.4256
	ROS	SVM	Accuracy	0.93
			Precision	0.9335
			Recall	0.93
			F1	0.9265
		NBM	Accuracy	0.7723
			Precision	0.7838
			Recall	0.7723
			F1	0.7647
	SMOTE	SVM	Accuracy	0.4414
			Precision	0.4764
			Recall	0.4414
			F1	0.4305
		NBM	Accuracy	0.4204
			Precision	0.4438
			Recall	0.4204
			F1	0.4222
	AdaCost	SVM	Accuracy	0.4378
			Precision	0.4714
			Recall	0.4378
			F1	0.3749

Table 19: The results of our classification using the ILS (RUS, ROS, SMOTE, and AdaCost) and OVA method based on SVM, KNN, and NBM classifiers for the NetBeans dataset.

Project	ILS	Classifier	Evaluation metrics	OVA method
NetBeans	RUS	SVM	Accuracy	0.4683
			Precision	0.4701
			Recall	0.4683
			F1	0.4586
		KNN	Accuracy	0.1476
			Precision	0.181
			Recall	0.1476
			F1	0.117
		NBM	Accuracy	0.3713
			Precision	0.3804
			Recall	0.3713
			F1	0.3699
	ROS	SVM	Accuracy	0.9197
			Precision	0.9178
			Recall	0.9197
			F1	0.9172
		NBM	Accuracy	0.7381
			Precision	0.7463
			Recall	0.7381
			F1	0.735
	SMOTE	SVM	Accuracy	0.5329
			Precision	0.568
			Recall	0.5329
			F1	0.5239
		NBM	Accuracy	0.4775
			Precision	0.4809
			Recall	0.4775
			F1	0.4723
	AdaCost	SVM	Accuracy	0.5346
			Precision	0.5869
			Recall	0.5346
			F1	0.5217

combination of the SMOTE and OVA method based on the SVM and NBM classifiers, which increased by 0.7651, 0.8155, 0.7651, and 0.7673, respectively. The combination of the SMOTE and OVA method based on the NBM classifier had higher accuracy value, recall rate, and F1 value compared with the combination of the SMOTE and OVA method based on the SVM classifier, with elevations of 0.6966, 0.6966, and 0.7018, respectively. From Table 18, for the OpenOffice dataset, the combination of the ROS and OVA method based on the SVM classifier had the greatest improvement, increasing by 0.9300, 0.9335, 0.9300, and 0.9265, respectively. A combination of the ROS and OVA method based on the NBM classifier also greatly improved compared with other combinations, increasing by 0.7723, 0.7838, 0.7723, and 0.7647, respectively. The combination of the RUS and OVA method based on the KNN classifier had the lowest improvement, 0.1497, 0.3197, 0.1497, and 0.1534, respectively. The combination of the SMOTE and OVA method based on the SVM classifier had higher improvement compared with the combination of the SMOTE and OVA method based on the NBM classifier, and the combination of the AdaCost and OVA methods based on the SVM classifier increased by 0.4414, 0.4764, 0.4414, and 0.4305, respectively. From Table 19, for the NetBeans dataset, the combination of the ROS and OVA method based on the SVM classifier had the greatest improvement, increasing by 0.9197, 0.9178, 0.9197, and 0.9172, respectively. A combination of the ROS and OVA method based on the NBM classifier also greatly improved compared with other combinations, increasing by 0.7381, 0.7463, 0.7381, and 0.7350, respectively. The combination of the RUS and OVA method based on the KNN classifier had the lowest improvement, 0.1476, 0.1810, 0.1476, and 0.1170, respectively. The combination of the AdaCost and OVA method based on the SVM classifier had the highest accuracy value, the precision rate value, and the recall rate value compared with the combination of the SMOTE and OVA method based on the SVM and NBM classifiers which increased by 0.5346, 0.5869, and 0.5346, respectively.

Therefore, the combination of the imbalanced leaning strategy RUS and the OVA method based on SVM and NBM classifiers had higher efficiency in solving the bug reports component classification problem compared with other combinations for the Mozilla, GCC, NetBeans, OpenOffice, and Eclipse datasets.

6. Conclusion

In this article, we propose a new method by combining imbalanced learning technologies with multiclass classification methods to implement bug reports component classification problems. We use four imbalanced processing strategies, RUS, ROS, SMOTE, and AdaCost, to process the data and obtain a balanced dataset. Then, we use the multiclass classification methods, OVO and OVA, based on NBM, KNN, and SVM classifiers for the balanced dataset to classify the bug reports component and solve the bug reports component classification problem. We explored the optimal performance of bug reports component classifications by different combinations of imbalanced learning strategies and classification algorithms. We can better solve the problem of bug reports classification by using the bug component classification to determine the appropriate developer for the bug report. In our work, we could not only reduce the word dimension of the original training set that improves the quality of the training set but also improve the classification performance for bug reports.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (nos. 61902050, 61672122, 61602077, 61771087, 51879027, 51579024, and 71831002), Program for Innovative Research Team in University of Ministry of Education of China (no. IRT 17R13), Fundamental Research Funds for the Central Universities (nos. 3132019501 and 3132019502), and Next-Generation Internet Innovation Project of CERNET (no. NGII20190627).

References

[1] W. Pan, B. Li, J. Liu, Y. Ma, and Bo Hu, "Analyzing the structure of Java software systems by weighted k-core decomposition," *Future Generation Computer Systems*, vol. 83, pp. 431–444, 2018.

[2] F. Rahman and P. Devanbu, "How, and why, process metrics are better," in *Proceedings of the 2013 35th International Conference on Software Engineering (ICSE)*, pp. 432–441, IEEE, San Francisco, CA, USA, May 2013.

[3] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *Proceedings of the 2013 35th International Conference on Software Engineering (ICSE)*, pp. 382–391, IEEE, San Francisco, CA, USA, May 2013.

[4] Y. Kamei, E. Shihab, B. Adams et al., "A large-scale empirical study of just-in-time quality assurance," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 757–773, 2012.

[5] S. Guo, R. Chen, M. Wei et al., "Ensemble data reduction techniques and Multi-RSMOTE via fuzzy integral for bug report classification," *IEEE Access*, vol. 6, pp. 45934–45950, 2018.

[6] J. Anvik, L. Hiew, and G. C. Murphy, "Coping with an open bug repository," in *Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology Exchange*, pp. 35–39, ACM, San Diego, CA, USA, 2005.

[7] X. Xie, W. Zhang, Y. Yang et al., "Dretom," in *Proceedings of the 8th International Conference on Predictive Models in Software Engineering*, pp. 19–28, ACM, Paris, France, 2012.

[8] X. Xia, D. Lo, X. Wang et al., "Accurate developer recommendation for bug resolution," in *Proceedings of the 2013 20th Working Conference on Reverse Engineering (WCRE)*, pp. 72–81, IEEE, Koblenz, Germany, October 2013.

[9] T. Menzies, Z. Milton, B. Turhan et al., "Defect prediction from static code features: current results, limitations, new approaches," *Automated Software Engineering*, vol. 17, no. 4, pp. 375–407, 2010.

[10] T. Jiang, L. Tan, and S. Kim, "Personalized defect prediction," in *Proceedings of the 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 279–289, IEEE, Silicon Valley, CA, USA, November 2013.

[11] 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.

[12] X. Yang, D. Lo, X. Xia et al., "Deep learning for just-in-time defect prediction," in *Proceedings of the 2015 IEEE International Conference on Software Quality, Reliability and Security*, pp. 17–26, IEEE, Vancouver, Canada, August 2015.

[13] X. Zhu and X. Wu, "Cost-constrained data acquisition for intelligent data preparation," *IEEE Transactions on Knowledge & Data Engineering*, vol. 17, no. 11, pp. 1542–1556, 2005.

[14] H. Jiang, X. Li, Z. Ren, J. Xuan, and Z. Jin, "Toward better summarizing bug reports with crowdsourcing elicited attributes," *IEEE Transactions on Reliability*, vol. 68, no. 1, pp. 2–22, 2019.

[15] W. Fan, S. J. Stolfo, J. Zhang et al., "AdaCost: misclassification cost-sensitive boosting," in *Proceedings of the ICML*, vol. 99, pp. 97–105, Bled, Slovenia, 1999.

[16] A. Tamrawi, T. T. Nguyen, J. M. Al-Kofahi et al., "Fuzzy set and cache-based approach for bug triaging," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, pp. 365–375, ACM, 2011.

[17] M. Alenezi, K. Magel, and S. Banitaan, "Efficient bug triaging using text mining," *JSW*, vol. 8, no. 9, pp. 2185–2190, 2013.

[18] C. Kang, Y. Huo, L. Xin et al., "Feature selection and tumor classification for microarray data using relaxed lasso and generalized multi-class support vector machine," *Journal of Theoretical Biology*, vol. 463, pp. 77–91, 2019.

[19] S. Guney and A. Atasoy, "Multiclass classification of n-butanol concentrations with *k*-nearest neighbor algorithm and support vector machine in an electronic nose," *Sensors and Actuators B: Chemical*, vol. 166, pp. 721–725, 2012.

[20] S. K. Ajagekar and V. Jadhav, "Automated approach for DDOS attacks detection based on naive Bayes multinomial classifier," in *Proceedings of the 2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)*, pp. 1–5, IEEE, Tirunelveli, India, May 2018.

[21] W. Deng, J. Xu, and H. Zhao, "An improved ant colony optimization algorithm based on hybrid strategies for

scheduling problem," *IEEE Access*, vol. 7, pp. 20281–20292, 2019.

[22] S. Guo, Y. Liu, R. Chen, X. Sun, and X. Wang, "Improved SMOTE algorithm to deal with imbalanced activity classes in smart homes," *Neural Processing Letters*, vol. 50, no. 2, pp. 1503–1526, 2019.

[23] G. Murphy and D. Cubranic, "Automatic bug triage using text categorization," in *Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*, Reading, MA, USA, 2004.

[24] W. Pan and C. Chai, "Measuring software stability based on complex networks in software," *Cluster Computing*, vol. 22, no. 2, pp. 2589–2598, 2019.

[25] Y. Liu, X. Wang, Z. Zhai, R. Chen, B. Zhang, and Y. Jiang, "Timely daily activity recognition from headmost sensor events," *ISA Transactions*, vol. 94, pp. 379–390, 2019.

[26] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?," in *Proceedings of the 28th International Conference on Software Engineering*, pp. 361–370, ACM, Shanghai, China, 2006.

[27] C. Sun, D. Lo, X. Wang et al., "A discriminative model approach for accurate duplicate bug report retrieval," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-ICSE '10*, pp. 45–54, 2010.

[28] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp. 111–120, ACM, 2009.

[29] W. Deng, H. Zhao, L. Zou, G. Li, X. Yang, and D. Wu, "A novel collaborative optimization algorithm in solving complex optimization problems," *Soft Computing*, vol. 21, no. 15, pp. 4387–4398, 2017.

[30] J. Xuan, H. Jiang, Z. Ren et al., "Automatic bug triage using semi-supervised text classification," 2017, https://arxiv.org/abs/1704.04769.

[31] T. Zhang, J. Chen, G. Yang, B. Lee, and X. Luo, "Towards more accurate severity prediction and fixer recommendation of software bugs," *Journal of Systems and Software*, vol. 117, pp. 166–184, 2016.

[32] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in *Proceedings of the 2008 IEEE International Conference on Software Maintenance*, pp. 346–355, IEEE, Beijing, China, September-October 2008.

[33] A. Lamkanfi, S. Demeyer, Q. D. Soetens et al., "Comparing mining algorithms for predicting the severity of a reported bug," in *Proceedings of the 2011 15th European Conference on Software Maintenance and Reengineering*, pp. 249–258, IEEE, Oldenburg, Germany, March 2011.

[34] Y. Tian, D. Lo, and C. Sun, "Information retrieval based nearest neighbor classification for fine-grained bug severity prediction," in *Proceedings of the 2012 19th Working Conference on Reverse Engineering*, pp. 215–224, IEEE, Kingston, Canada, October 2012.

[35] R. Chen, S. Guo, X. Wang et al., "Fusion of multi-RSMOTE with fuzzy integral to classify bug reports with an imbalanced distribution," *IEEE Transactions on Fuzzy Systems*, vol. 27, no. 12, pp. 2406–2420, 2019.

[36] H. Li, G. Gao, R. Chen, X. Ge, S. Guo, and L.-Y. Hao, "The influence ranking for testers in bug tracking systems," *International Journal of Software Engineering and Knowledge Engineering*, vol. 29, no. 1, pp. 93–113, 2019.

[37] S. Guo, R. Chen, H. Li, T. Zhang, and Y. Liu, "Identify severity bug report with distribution imbalance by CR-SMOTE and ELM," *International Journal of Software Engineering and Knowledge Engineering*, vol. 29, no. 02, pp. 139–175, 2019.

[38] W. Pan, B. Hu, J. Dong, K. Liu, and B. Jiang, "Structural properties of multilayer software networks: a case study in Tomcat," *Advances in Complex Systems*, vol. 21, no. 2, Article ID 1850004, 2018.

[39] H. Jiang, L. Nie, Z. Sun et al., "ROSF: leveraging information retrieval and supervised learning for recommending code snippets," *IEEE Transactions on Services Computing*, vol. 12, no. 1, pp. 34–46, 2019.

[40] J. Guo, Y. Mu, M. Xiong, Y. Liu, and J. Gu, "Activity feature solving based on TF-IDF for activity recognition in smart homes," *Complexity*, vol. 2019, Article ID 5245373, 10 pages, 2019.

[41] W. Pan, B. Song, K. Li, and K. Zhang, "Identifying key classes in object-oriented software using generalizedk-core decomposition," *Future Generation Computer Systems*, vol. 81, pp. 188–202, 2018.

[42] N. Bettenburg, S. Just, A. SchrÃter et al., "What makes a good bug report?," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 308–318, ACM, Atlanta, GA, USA, June 2008.

[43] W. Pan, H. Ming, C. K. Chang, Z. Yang, and D.-K. Kim, "ElementRank: ranking java software classes and packages using multilayer complex network-based approach," *IEEE Transactions on Software Engineering (IEEE TSE)*, 2019.

[44] W. Pan and C. Chai, "Structure-aware mashup service clustering for cloud-based Internet of things using genetic algorithm based clustering algorithm," *Future Generation Computer Systems*, vol. 87, pp. 267–277, 2018.

[45] W. Pan, J. Dong, K. Liu, and J. Wang, "Topology and topic-aware service clustering," *International Journal of Web Services Research*, vol. 15, no. 3, pp. 18–37, 2018.

[46] X. Chen, H. Jiang, Z. Chen, T. He, and L. Nie, "Automatic test report augmentation to assist crowdsourced testing," *Frontiers of Computer Science*, vol. 13, no. 5, pp. 943–959, 2019.

[47] J. Xuan, H. Jiang, Z. Ren et al., "Developer prioritization in bug repositories," in *Proceedings of the 2012 34th International Conference on Software Engineering (ICSE)*, pp. 25–35, IEEE, Zurich, Switzerland, June 2012.

[48] J. Ai, Z. Su, Y. Li, and C. Wu, "Link prediction based on a spatial distribution model with fuzzy link importance," *Physica A: Statistical Mechanics and its Applications*, vol. 527, Article ID 121155, 2019.

[49] W. Deng, H. Zhao, X. Yang, J. Xiong, M. Sun, and B. Li, "Study on an improved adaptive PSO algorithm for solving multi-objective gate assignment," *Applied Soft Computing*, vol. 59, pp. 288–302, 2017.

[50] G. Lang, Q. Li, and L. Guo, "Discernibility matrix simplification with new attribute dependency functions for incomplete information systems," *Knowledge and Information Systems*, vol. 37, no. 3, pp. 611–638, 2013.

[51] H. Zhao, R. Yao, L. Xu, Y. Yuan, G. Li, and W. Deng, "Study on a novel fault damage degree identification method using high-order differential mathematical morphology gradient spectrum entropy," *Entropy*, vol. 20, no. 9, p. 682, 2018.

[52] W. Pan, H. Jiang, H. Ming, C. Chai, Bi Chen, and H. Li, "Characterizing software stability via change propagation simulation," *Complexity*, vol. 2019, Article ID 9414162, 17 pages, 2019.

[53] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[54] S. Kim, K. Pan, and E. E. Whitehead Jr., "Memories of bug fixes," in *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 35–45, ACM, Portland, OR, USA, November 2006.
[55] E. Shihab, A. Mockus, Y. Kamei et al., "High-impact defects: a study of breakage and surprise defects," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, pp. 300–310, ACM, Szeged, Hungary, September 2011.
[56] H. Zhao, J. Zheng, J. Xu, and W. Deng, "Fault diagnosis method based on principal component analysis and broad learning system," *IEEE Access*, vol. 7, pp. 99263–99272, 2019.
[57] Y. Xiang, W. Pan, H. Jiang, Y. Zhu, and H. Li, "Measuring software modularity based on software networks," *Entropy*, vol. 21, no. 4, p. 344, 2019.
[58] http://Mozilla.apache.org/, March, 2019.
[59] http://GCC.apache.org/, March, 2019.
[60] http://NetBeans.apache.org/, March, 2019.
[61] http://OpenOffice.apache.org/, March, 2019.
[62] http://Eclipse.apache.org/, March, 2019.