applied sciences

MDPI

Article

LIMCR: Less-Informative Majorities Cleaning Rule Based on Naïve Bayes for Imbalance Learning in Software Defect Prediction

Yumei Wu [1,*,†], Jingxiu Yao [1,†], Shuo Chang [2] and Bin Liu [1]

[1] School of Reliability and Systems Engineering, Beihang University, Beijing 100191, China; yjx1080@126.com (J.Y.); liubin@buaa.edu.cn (B.L.)
[2] Beijing Institute of Remote Sensing Equipment, Beijing 100039, China; changshuo@buaa.edu.cn
* Correspondence: wuyumei@buaa.edu.cn; Tel.: +86-010-8231-6640
† These authors contributed equally to this work.

Received: 16 October 2020; Accepted: 19 November 2020; Published: 24 November 2020

Abstract: Software defect prediction (SDP) is an effective technique to lower software module testing costs. However, the imbalanced distribution almost exists in all SDP datasets and restricts the accuracy of defect prediction. In order to balance the data distribution reasonably, we propose a novel resampling method LIMCR on the basis of Naïve Bayes to optimize and improve the SDP performance. The main idea of LIMCR is to remove less-informative majorities for rebalancing the data distribution after evaluating the degree of being informative for every sample from the majority class. We employ 29 SDP datasets from the PROMISE and NASA dataset and divide them into two parts, the small sample size (the amount of data is smaller than 1100) and the large sample size (larger than 1100). Then we conduct experiments by comparing the matching of classifiers and imbalance learning methods on small datasets and large datasets, respectively. The results show the effectiveness of LIMCR, and LIMCR+GNB performs better than other methods on small datasets while not brilliant on large datasets.

Keywords: software defect prediction; naïve bayes; class imbalance learning; resampling methods

1. Introduction

Software defect prediction (SDP) is an effective technique to lower software module testing costs. It can efficiently identify defect-prone software modules by learning information from defect datasets of the previous release. Existing SDP studies can be divided into four categories: (1) Classification, (2) Regression, (3) Mining association rules, (4) Ranking [1]. Studies in the first category use classified algorithms (also called classifiers) as the prediction algorithms to classify software modules into defect-prone classes (positive or minority class) and non-defective classes (negative or majority class) or various levels of defect severity. The imbalance learning we focus on is based on binary classification study in this paper.

Commonly in software defect datasets, the number of samples (samples usually refer to software modules in SDP) in defect-prone classes is naturally smaller than the number of samples in non-defective classes. However, most prediction algorithms assume that the number of samples in any class are equally balanced. This contradiction makes the prediction algorithms trained in imbalanced software defect datasets are generally biased towards the samples in non-defect-prone classes and ignore the samples in defect-prone classes, i.e., many defect-prone samples might be classified into non-defect-prone class based on prediction algorithms trained by imbalanced datasets. This problem widely occurs in SDP and it has proved that reducing the influence of the imbalance problem can improve prediction performance efficiently.

Numerous methods [2,3] are proposed for tackling imbalance problems in SDP. In imbalance learning research [4,5], methods are divided into two categories: data level and algorithm level. Methods within the data level mainly consist of various data resampling techniques. The resampling technique is a kind of method to rebalance datasets by adding minorities (over-sampling methods) or removing majorities (under-sampling methods). For instance, SMOTE [6] is an over-sampling method to generate synthetic samples in minority class, NCL [7] is an under-sampling method to remove samples in majority class. Methods within the algorithm level improve classification algorithms based on existing algorithms to make them no more biased towards the samples in majority classes and ignore the samples in minority classes. Cost-sensitive methods combine both algorithm and data level methods. They consider the different misclassified cost of samples in different classes. For instance, RAMOBoost [8] is an improvement of Boosting algorithm [9], NBBag [5] is an improved algorithm based on Bagging algorithm [10], AdaCost [11] modifies the weight updated by adding a cost adjustment function based on AdaBoost algorithm [12]. Boosting, bagging and other ensemble classifiers are frequently selected as the basic classification algorithms for improving because of the high performance in classification they have [4,5]. A proper basic prediction algorithm could perform better with the imbalanced dataset after improving by imbalance learning. Obviously, basic classification algorithm selection is one of the most important steps for imbalance learning methods within algorithm level.

Different from algorithm level methods, methods in data level can choose and change classifiers flexibly. With the increasing number of imbalance learning studies, researchers notice the influence of classifier selection. Numerous empirical studies compare the performance of different techniques to find the rules of classifier selection, and the influence factors within consideration include the researcher group [13], levels of class imbalance [14], diversity [15,16], and others [17,18].

Most empirical studies focus on the comparison between resampling methods and the influence factors while less notice on the applicability of resampling methods and the connection between classifiers and them. In addition, there is almost no resampling method by quantifying the sample information. Motivated by this, we aim to investigate how resampling methods work in datasets with different sample size, and how they cooperate with various classifiers. Moreover, we aim to propose a novel and effective resampling method to remove less-informative samples of majority class for rebalancing the data distribution. The main contributions of this paper are divided into the following three aspects:

1. We perform an empirical study to investigate the influence of datasets sample size on popular comment classifiers.
2. We present a novel resampling method LIMCR based on Naïve Bayes to solve the class imbalance problem in SDP datasets. The new method outperforms other resampling methods on datasets with small sample size.
3. We evaluate and compare the proposed method with existing well performance imbalance learning methods including both methods from data level and algorithm level. The experiment presents the effective performance of specified methods on different datasets, respectively.

The remainder of this paper is organized as follows. Section 2 summarizes related work in the area of imbalance learning. In Section 3, we describe the methodology and procedure of our LIMCR. In Section 4, the experimental setup and results are explained, respectively. Finally, the discussion and conclusion are presented in Sections 5 and 6.

2. Related Works

2.1. Imbalanced Learning Methods

A large number of methods have been developed to address imbalance problems currently, and all of these methods are classified into two basic categories: data level and algorithm level. The methods

in data level mainly study the effect of changing class distribution to deal with imbalanced datasets. It has been empirically proved that the application of a preprocessing process for rebalancing class distribution is usually a positive solution [19]. The main advantage of methods in data level is that they are independent of the classifier [4]. Moreover, data level methods can be easily embedded in ensemble learning algorithms as algorithm level methods. Hereafter, representative imbalanced learning methods will be introduced in this section.

Among all the data resampling methods, random over-sampling and random under-sampling are the simplest resampling methods for rebalancing datasets [20]. Although random sampling methods have some defects, they indeed improve the performance of classifiers. To avoid the drawbacks caused by random methods, researchers attempt to generate new synthetic samples based on original dataset and attain to great success. SMOTE [6] is one of the most classical methods among synthetic over-sampling methods. Numerous methods are proposed based on SMOTE, like ADASYN [21], Borderline-SMOTE [22], MWMOTE [23], Safe-level-SMOTE [24]. The generated samples add essential information to the original dataset so that the additional bias to classifier can be alleviated and the overfitting problem to which random over-sampling might lead can be avoided [25].

On the other side, resampling methods we introduced above have been proved to show remarkable performance after being embedded in an ensemble algorithm [26,27]. So researchers integrate an oversampling method with an appropriate ensemble method to achieve a stronger approach for solving class imbalance problems. The most widely used ensemble learning algorithms are AdaBoost and Bagging which are usually combined with the resmapling methods to form new typical algorithms, such as SMOTEBoost [28], SMOTEBagging [15], RAMOBoost [8], etc., which perform well in the imbalanced dataset.

Undersampling methods are also widely used in imbalance learning, especially in SDP, research [29] has proved static code features have limited information content and undersampling performs better than others. In earlier studies, researchers prefer identifying redundant samples by cluster or K-nearest neighborhoods algorithms, for instance, Condensed Nearest Neighbor Rule (CNN) [30], Tomek links [31], Edited Nearest Neighbor Rule (ENN) [32], One-Sided Selection (OSS) [33], Neighborhood Cleaning Rule (NCL) [7]. With the increasing number of distribution problems are found in datasets, more new stronger undersampling methods are proposed currently. Research [34] proposes a set of sample hardness measurements to understand why some samples are harder to classify correctly and remove samples that are suspected hard to learn. A similar study [35] is also proved effective for imbalance learning. Undersampling can be also embedded in ensemble algorithms. Two algorithms embedded by undersampling methods: EasyEnsemble and BalanceCascade [36] are proposed for preserving information to a maximum degree and reducing the data complexity for efficient computation.

2.2. Software Defect Prediction

The classification problem in SDP is a typical learning problem. Bohem and Basili pointed out that in most cases, 20% of the modules can result in 80% of the software defects [37], this means software defect data has a natural imbalanced distribution.

SDP research starts with software defective metrics selection. The original defect data is obtained by using specified static software metrics[38]. For instance, McCabe [38] and Halstead [39] metrics are widely used, Chidamber and Kemerer's (CK) metrics are proposed for fitting the demand of object-orientation (OO) software. Lots of empirical studies are conducted for the imbalance problem in SDP. A comprehensive experiment to study the effect of imbalance learning in SDP emphasizes the importance of method selection [40]. The result of the study [41] advocates resampling method for effective imbalance learning. Meanwhile, many new imbalance leaning methods are proposed for SDP. L. Chen et al. [2] consider the class imbalance problem together with class overlap and integrate neighbor cleaning learning (NCL) and ensemble random under-sampling (ERUS) methods as a novel approach for SDP. H. N. Tong et al. [1] propose a novel ensemble learning approach for imbalance

and overfitting problems, ensemble it with the deep learning algorithm, and solve the imbalance problem and high dimensionality simultaneously. S. Kim et al. [42] propose an approach to detect and eliminate noises in defect data. N. Limsettho et al. [3] propose a novel approach named Class Distribution Estimation with Synthetic Minority Oversampling Technique (CDE-SMOTE) to modify the distribution of the training data for a balanced distribution.

2.3. Classification Algorithms for Class Imbalance

Classification is a form of data analysis that can be used to build a model that minimizes the number of classification errors on a training dataset [43]. Some classifiers are commonly used because of their outstanding performance, e.g., Naïve Bayes [44], multilayer perceptron [45], K-nearest neighbors [46], and logistic regression [47], decision trees [48], support vector machines [49], backpropagation neural networks [50]. However, it is confirmed that ensemble algorithms by a few weak classifiers outperform a common classifier [4,16] when the training dataset has a class imbalance problem. Random forest is a frequently used ensemble method in machine learning, which ensemble a certain amount of decision trees together for classification. However, it is still negatively influenced by imbalanced class distribution [51]. Facing the imbalance problem, F. Herrera et al. [52] evaluate the performance of the diverse approaches for imbalanced classification and use the MapReduce framework to solve the imbalance problem in big data. J. Xiao et al. [51] propose a dynamic classifier ensemble method for imbalanced data (DCEID). This method combines ensemble learning with cost-sensitive learning which improves classification accuracy effectively.

All of these methods have been proved to improve the classifier performance efficiently, but the sample size of SDP defect data and its relationship with the classifiers are unexplored. Moreover, the cooperation between resampling methods and classifiers is less to be noticed. Therefore, in this paper, we firstly empirically study the influence of sampling size on classifiers and resampling methods; then, we investigate the cooperation between resampling methods and classifiers. Finally, based on the results of the empirical study, we propose a novel resampling method for imbalanced learning in software defect prediction, which can improve prediction results for SDP datasets.

3. Research Methodology

3.1. Overall Structure

In order to solve the class imbalance problem rationally and effectively, we choose to remove less-informative samples of majority class instead of randomly deleting for rebalancing the data distribution. Furthermore, we define the informative degree of a specified sample by measure the difference of samples with same feature values between defective and non-defective classes, which is the main idea of LIMCR. The proposed LIMCR involves three key phases. In the first phase, LIMCR defines the sample information calculating rule on one feature based on Naïve Bayes. In the second phase, LIMCR summarizes the variable of sample informative degree on one feature and proposes a new variable for describing sample informative degree. In the third phase, LIMCR analyzes the relationship between variable and sample distribution and proposes the definition of less informative majorities. The structure of the proposed method LIMCR is shown in Figure 1.

3.2. Assumption of Porposed Methods

In order to make the calculation of LIMCR more efficient and applicable to more datasets, the method we proposed is based on the following assumptions:

1. All features are independent for the given class label;
2. All features are continuous variables and the likelihood of the features is assumed to be Gaussian;
3. There is only one majority class in datasets.

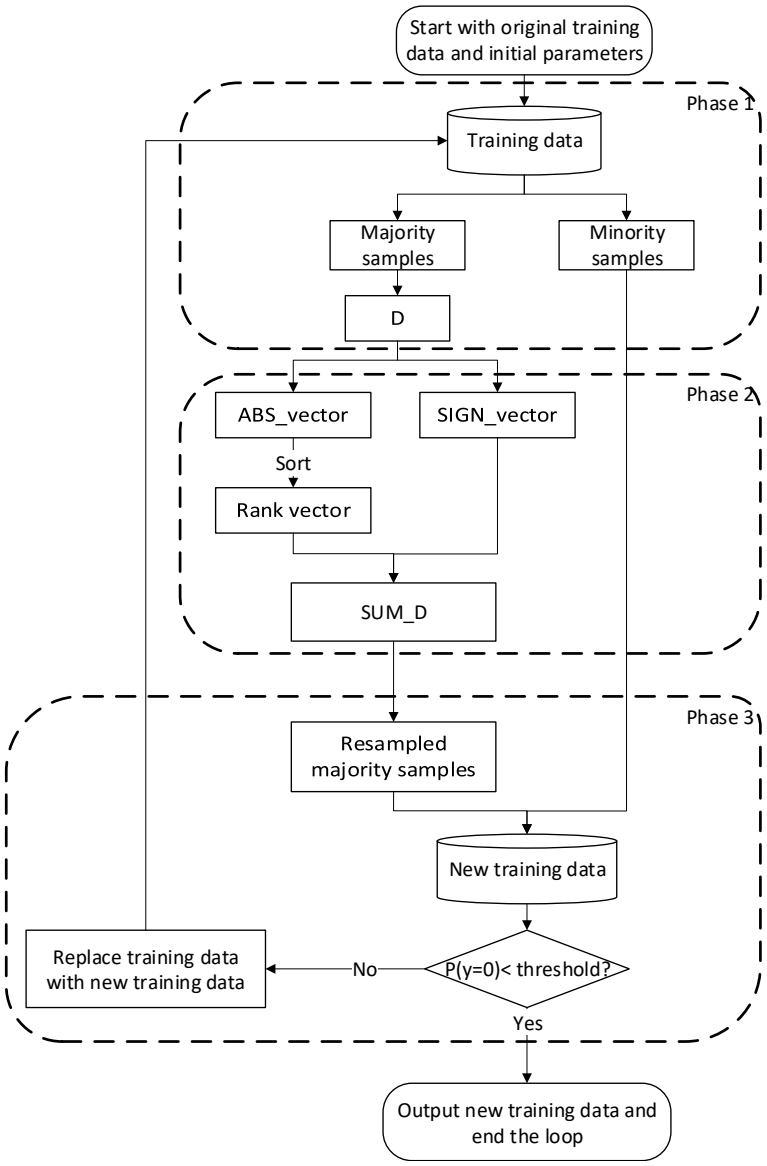

Figure 1. Overall structure of the proposed method LIMCR.

3.3. Variable of Sample Information for One Feature

A sample E is represented by a set of feature values $X(x_1, x_2, \cdots, x_m)$ and a class label Y, the value of Y can only be 1 or 0. According to Bayes probability, posterior probability of Y can be calculated as

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)} \tag{1}$$

Then posterior probability of a sample E_i with m features being class y can be calculated as

$$p(Y = y|X_i) = \frac{p(X_i = (X_{i1} = x_{i1}, X_{i2} = x_{i2}, \cdots, X_{im} = x_{im})|Y = y)p(Y = y)}{p(X_i = (X_{i1} = x_{i1}, X_{i2} = x_{i2}, \cdots, X_{im} = x_{im}))} \tag{2}$$

Because of the assumption that all features are independent for the given class label, the conditional probability $p(X_i \mid Y = y)$ can be calculated as

$$p(X_i = (X_{i1} = x_{i1}, X_{i2} = x_{i2}, \cdots, X_{im} = x_{im})|Y = y) = \prod_{j=1}^{m} p(x_{ij} \mid y) \tag{3}$$

The Naïve Bayes classifier is expressed as

$$f_b(X_i) = \frac{p(Y=1)}{p(Y=0)} \prod_{i=1}^{n} \frac{p(x_{ij} \mid Y=1)}{p(x_{ij} \mid Y=0)} = IR \prod_{i=1}^{n} \frac{p(x_{ij} \mid Y=1)}{p(x_{ij} \mid Y=0)} \tag{4}$$

where n is the number of samples and IR represents the imbalance radio. Then the class label Y of a sample with feature X_i is predicted according to $f_b(X_i)$

$$\begin{cases} Y=1, f_b(X_i) \geq 1 \\ Y=0, f_b(X_i) < 1 \end{cases} \tag{5}$$

For the single feature, the bigger the gap between $p(x_{ij} \mid Y=1)$ and $p(x_{ij} \mid Y=0)$, the larger the value of $\frac{p(x_{ij}|Y=1)}{p(x_{ij}|Y=0)}$ is, simultaneously the easier the sample X_i can be correctly classified by Naïve Bayes classifier. Correspondingly, the easier a sample is misclassified the more informative it is. Generally, the value of conditional probabilities $p(x_{ij} \mid Y=0)$ and $p(x_{ij} \mid Y=1)$ are calculated based on samples in dataset, and the likelihood of the features is assumed to be Gaussian. Then when there is only one feature, the conditional probabilities are calculated as

$$p(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right) \tag{6}$$

If a sample with one feature can be precisely classified into corresponding class (for instance $y=0$), the conditional probability $p(x_i \mid y=0)$ should be close to 1 and $p(x_i \mid y=1)$ should be close to 0, then the difference between them should be close to 1. This kind of samples cannot provide much effective information for classifier except making sample variance larger.

Figure 2 presents two curves in each figure which mean the probability density functions of two probabilities in one dimension, respectively. It is known that samples in overlapping area are hard to be classified correctly and may disturb model training. The data distribution of original dataset in one dimension is like curves in Figure 2a, the distribution of majorities is dispersive and creates a large overlapping area with minorities. After removing less informative samples in majorities, the variance of majorities turns small and the overlapping area turns smaller as well in Figure 2b. The two figures illustrate that the increase of sample variance of one class would lead sample overlapping area larger and make the learning phase harder to classify. Considering the amount of majority samples in datasets with imbalanced distribution is larger than minority samples, we define an informative variable D for evaluating how informative a majority sample is for one feature.

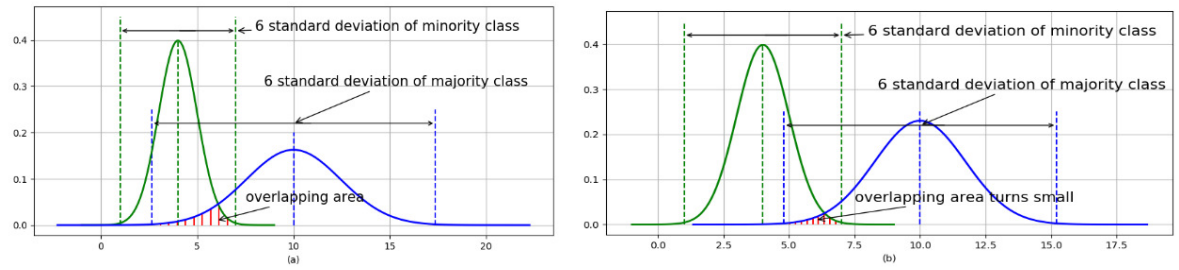

Figure 2. Distribution of overlapping area. (**a**) Distribution of original dataset in one dimension; (**b**) Removing less informative samples in majorities

Definition 1. *Informative variable D. In imbalanced datasets, the variable for evaluating information of a majority sample for one feature is defined as its difference between conditional probabilities* $D_{ik} = p(x_{ik} \mid y_i = 0) - p(x_{ik} \mid y_i = 1)$, *where i represents the ith sample and k represents the kth feature.*

3.4. Variable of Sample Informative Degree

In the feature space, variable D can only indicate the distribution of one feature which cannot refer to sample distribution characteristics. Since the Naïve Bayes algorithm assumes that all features are independent for the given class label, the relationship between features are not involved in our method. Under this assumption, we propose a rule to sum up the informative variable D of each feature distribution to get the informative degree SUM_D of each majority sample.

The construction of the informative degree SUM_D mainly consider two aspects. One is the difference between two conditional probabilities $p(X \mid Y = 0)$ and $p(X \mid Y = 1)$ might be too small to split samples with different labels. The other is D from different features might be offset after summation. To avoid the above two possible problems, we sum up the rank values of D instead of the variable itself. The steps proposed to calculate informative degree SUM_D of each majority sample are as follows.

1. Order m variables D_i to $D_{order} = \{D_{i1}, D_{i2}, \cdots, D_{im}\}$ by absolute values from the smallest absolute difference to the largest, where $\mid D_{i1} \mid \leq \mid D_{i2} \mid \leq \cdots \leq \mid D_{im} \mid$. Let ABS_vector denote the absolute values and $SIGN\_vector$ denote the signs of m variables.
2. Rank the m variables of D_{order} with the smallest as 1, if there are elements which have the same value, calculate average rank of these elements. Let $Rank\_vector$ denotes the rank.
3. Sum the product of $SIGN\_vector$ and $Rank\_vector$ as SUM_D. Let $SUM\_D(X_i)$ denote the informative degree of majority sample X_i.

$SUM\_D(X_i)$ quantifies how informative a sample is, especially when the classifier is Naïve Bayes, this variable denotes how difficult a Naïve Bayes classifier learns information for classification from a sample. The rank value recorded in $Rank\_vector$ can differ variable D from different features clearly and the product of $Rank\_vector$ and $SIGN\_vector$ can avoid offset of value D from different features efficiently.

3.5. Finding the Less Informative Majorities

Generally, the bigger the SUM_D is, the less informative the majority sample is, so we try to find out and remove the majority samples with big SUM_D values. However, there is another situation to be noticed, when SUM_D value is negative, it means this majority sample is in overlapping area or even in the minority class area. Samples like these are overlapping samples or noises, both possible results might have bad influence on performance of classification. Summarizing the rules above, we give the definition of less informative majorities.

Definition 2. *Majority samples in datasets which have a too large or too small SUM_D value are defined as the less informative samples.*

Order majorities with SUM_D, remove specified number of the first few and last few samples of the sequence from majorities. After removing, recalculate data distribution variables and repeat procedures introduced above until the imbalance problem is solved. The main components of LIMCR are described in Algorithm 1.

Algorithm 1: LIMCR: Less-informative majorities cleaning rule.

Input: Threshold to stop iteration *threshold*. Step of iteration *step_a* and *step_b*. Original training set S_o

Output: Resampled dataset S_{new}

Split S_o into majority class set S_{Maj} and minority class set S_{Min}.

Calculate prior probabilities of majority class and minority class: $Prior\_P0$ and $Prior\_P1$: $Prior\_P0 = \frac{|S_{Maj}|}{|S_o|}; Prior\_P1 = \frac{|S_{Min}|}{|S_o|}$.

while $Prior\_P0$>*threshold* **do**

  Calculate mean value $\overline{X}_{Majk}$ and variance S^2_{Majk} of feature k for samples in S_{Maj} ($k = 1, 2, \cdots, m$; m =number of features).

  Define $P(X_{ik} = x_{ik} \mid Y = 0) = \frac{1}{\sqrt{2\pi S^2_{Majk}}} \exp(-\frac{(x_{ik} - \overline{X}_{Majk})^2}{2S^2_{Majk}})$.

  Calculate mean value $\overline{X}_{Mink}$ and variance S^2_{Mink} of feature k for samples in S_{Min} ($k = 1, 2, \cdots, m$; m =number of features).

  Define $P(X_{ik} = x_{ik} \mid Y = 1) = \frac{1}{\sqrt{2\pi S^2_{Mink}}} \exp(-\frac{(x_{ik} - \overline{X}_{Mink})^2}{2S^2_{Mink}})$.

  $S_{tempnew} = S_0$.

  $S_{new} = S_{tempnew}$.

  for *sample* $X_i \in S_{Maj}$ **do**

    for $x_{ik} \in X_i$(x_{ik} *is the value of the kth feature of sample* X_i) **do**

      $D_{ik} = P(X_{ik} = x_{ik} \mid Y = 0) - P(X_{ik} = x_{ik} \mid Y = 1)$ (hold 3 digits after the decimal point).

      $D_i = \{D_{ik} \mid i = 1, 2, \cdots, m\}$.

      $SIGN\_vector(i) = sign(D_i)$.

      $ABS\_vector(i) = abs(D_i)$.

      Sort element in ABS_vector from smallest to largest and record the rank value into $Rank\_vector$. If there are elements which have the same value, calculate the average rank of these elements.

      $SUM\_D_i = SIGN\_vector(i)^T \times Rank\_vector(i)$.

      Number of samples need to be remove in this iteration: $N_a = \mid S_{Maj} \mid \times step\_a; N_b = \mid S_{Maj} \mid \times step\_b$.

      Sort $X_i \in S_{Maj}$ with SUM_D_i from largest to smallest.

      Remove the first N_b and the last N_a samples in S_{Maj}.

      Update S_{Maj}.

      Renew training datasets $S_{tempnew} = S_{Maj} + S_{Min}$. Update $Prior\_P0 = \frac{|S_{Maj}|}{|S_{new}|}; Prior\_P1 = \frac{|S_{Min}|}{|S_{new}|}$.

    end

  end

end

return S_{new}.

4. Experiments

4.1. Benchmark Datasets

Datasets we choose in this research are software defect datasets from Marian Jureczko Datasets [53], NASA MDP datasets from Tim Menzies [54] and Eclipse bug datasets from Thomas Zimmermann [55] as benchmark data. Datasets in the first two research can be obtained from the website (https://zenodo.org/search?page=1&size=20&q=software%20defect%20predictio) and the

Eclipse bug datasets are downloaded from Eclipse bug repository (https://www.st.cs.uni-saarland.de/softevo/bug-data/eclipse/). We investigate the sample size and IR value of each set (totally 108), and statistical results are shown in pie graph in Figure 3. Two basic distribution characteristics are listed as follows:

1. Most IR (imbalanced ratio) of SDP datasets range from 2 to 100.
2. The sample size of SDP datasets in different projects has a huge disparity. Some small datasets are less than 100, while some large datasets are more than 10,000.

In experiments, we choose 29 datasets from SDP datasets we investigated above for the following experiments as benchmark data. The information of selected datasets are presented in Table 1. To solve binary classification problem, we regard the samples of which the label is the number of bugs and greater than 1 as the same class and redefine the label as "1", meanwhile, the samples with the "0" label are not changed.

The definition of imbalanced ratio is the ratio of number of negative samples to number of positive samples, and the sample size is the number of samples of a dataset.

$$IR = \frac{number\ of\ negative\ samples}{number\ of\ positive\ samples} \tag{7}$$

Table 1. Description of software defect datasets.

Dataset	Abbreviation	Feature	Sample Size	Label = 1	Label = 0	IR
synapse-1.2	synapse-1.2	20	256	86	170	1.977
jedit-3.2	jedit-3.2	20	272	90	182	2.022
synapse-1.1	synapse-1.1	20	222	60	162	2.700
log4j-1.0	log4j-1.0	20	135	34	101	2.971
jedit-4.0	jedit-4.0	20	306	75	231	3.080
ant-1.7	ant-1.7	20	745	166	579	3.488
camel-1.6	camel-1.6	20	965	188	777	4.133
camel-1.4	camel-1.4	20	872	145	727	5.014
Xerces-1.3	Xerces-1.3	20	453	69	384	5.565
Xalan-2.4	Xalan-2.4	20	723	110	613	5.573
jedit-4.2	jedit-4.2	20	367	48	319	6.646
arc	arc	20	234	27	207	7.667
synapse-1.0	synapse-1.0	20	157	16	141	8.813
tomcat	tomcat	20	858	77	781	10.143
camel-1.0	camel-1.0	20	339	13	326	25.077
PC5	PC5	38	1694	458	1236	2.699
KC1	KC1	21	1162	294	868	2.952
JM1	JM1	21	7720	1612	6108	3.789
prop-5	prop-5	20	8516	1299	7217	5.556
prop-1	prop-1	20	18,471	2738	15,733	5.746
eclipse-metrics-files-3.0	ec -3.0	198	10,593	1568	9025	5.7557
eclipse-metrics-files-2.0	ec2.0	198	6729	975	5754	5.902
PC4	PC4	37	1270	176	1094	6.216
PC3	PC3	37	1053	130	923	7.100
prop-3	prop-3	20	10,274	1180	9094	7.707
eclipse-metrics-files-2.1	ec -2.1	198	7888	854	7034	8.237
prop-2	prop-2	20	23,014	2431	20,583	8.467
prop-4	prop-4	20	8718	840	7878	9.379
MC1	MC1	38	1952	36	1916	53.222

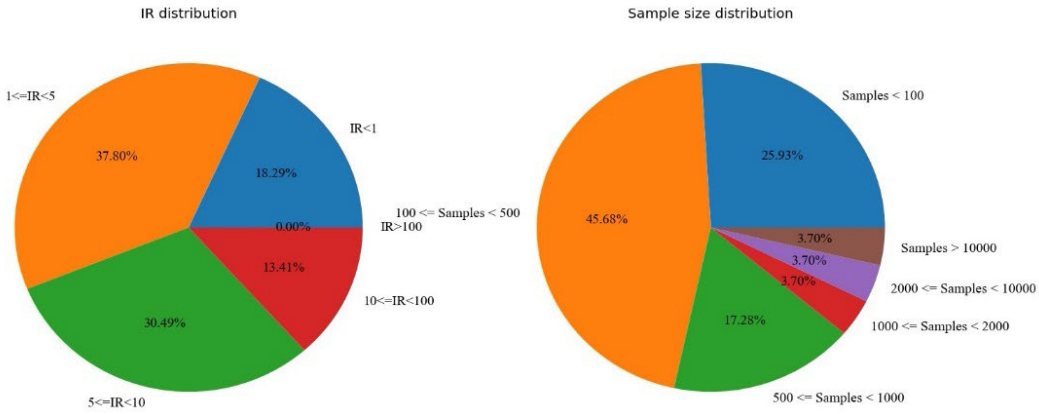

Figure 3. Data distribution of SDP datasets.

4.2. Performance Metrics

In experiments, we exploit four common performance metrics: *recall*, *G-mean*, *AUC* and Balanced Accuracy Score (*balancedscore*) [56]. The larger value of the four metrics is, the better the performance of the classifier is. All these metrics are based on confusion matrix (Table 2).

Table 2. Confusion matrix.

	Actual Buggy	**Actual Clean**
Predict Buggy (Positive)	*TP*	*FP*
Predict Clean (Negative)	*FN*	*TN*

Where the defective modules are regarded as buggy (or positive) samples and non-defective modules as clean (or negative) samples. According to confusion matrix, the definition of *PD* (the probability of detection, also called *recall*, *TPR*), *PF* (the probability of false alarm, also called *FPR*) and *precision* are as follows.

$$PD = recall = \frac{TP}{TP + FN} \tag{8}$$

$$PF = \frac{FP}{FP + TN} \tag{9}$$

$$precision = \frac{TP}{TP + FP} \tag{10}$$

recall and *G-mean* are proved more suitable for imbalanced learning [20].

$$G - mean = \sqrt{recall \times precision} \tag{11}$$

AUC measures the area under the *ROC* curve which describes the trade-off between *PD* and *PF*, it can be calculated as follows: [1]

$$AUC = \frac{\sum_{buggy_i} rank(buggy_i) - \frac{M(M+1)}{2}}{M * N} \tag{12}$$

where $\sum_{buggy_i} rank(buggy_i)$ represents the sum of ranks of all buggy(or positive) samples, M and N are the number of buggy samples and clean samples, respectively.

Balanced Accuracy Score (called *balancedscore*) as another accuracy metric is defined as the average recall obtained from each class, the metric can avoid inflated accuracy resulted from imbalanced class. Assume that y_i is the true value of the ith sample, and ω_i is the corresponding

sample weight, then we adjust the sample weight to $\widehat{\omega_i} = \frac{\omega_i}{\sum l(y=y_i)\omega_i}$, where $l(x)$ is the indicator function. Given predicted $\widehat{y}_i$, *balancedscore* is defined as

$$balancedscore(y, \widehat{y}, \widehat{\omega}) = \frac{1}{\sum \widehat{\omega_i}} \sum l(\widehat{y}_i = y_i)\widehat{\omega_i} \tag{13}$$

4.3. Research Question and Results

RQ1: Which baseline classifiers do we choose to match the imbalance learning methods with different sample sizes?

Motivation: Classification effect can be affected by classifiers, imbalance learning methods, sample size and number of features. To improve the efficiency of the experiments, we perform an empirical study to give priority to classifiers that perform well on SDP datasets. On the other hand, we need explore the impact of different classifers on different sample size.

Approach: We first do a preliminary experiment to show which baseline classifier performs better without any resampling method on 29 benchmark datasets. We choose nine baseline classifiers with parameters which are listed in Table 3. All baseline classifiers [57] are implemented in scikit-learn v0.20.3 [58]. The parameters of each classifier are decided by pre-experiments, meanwhile, parameters which make no influence on classification performance, are used as default value.

Table 3. Selection of classifiers and parameters.

Classifier	Abbreviation	Parameters
AdaBoostClassifier	ABC	n_estimators = 5
BaggingClassifier	Bgg	base_estimator = DecisionTree, n_estimators = 18
GaussianNB	GNB	var_smoothing = 1×10^{-5}
KNeighborsClassifier	Knn	n_neighbors = 10
RandomForest	RF	n_estimators = 50, max_depth = 3, min_samples_leaf = 4
SVC	SVC	default
DecisionTreeClassifier	DTC	max_depth = 3, min_samples_leaf = 4
MLPClassifier	MLP	solver = 'lbfgs', hidden_layer_sizes = (10,500)
LogisticRegression	LR	default

Results: Table 4 summarize the results of nine classifiers on 29 datasets. The best result is highlighted with bold-face type. The differences between results of each classifier are analyzed by using Friedman and Wilcoxon statistical test [59].

The performance of classifiers is measured by recall, G-mean and AUC, the result of these three metrics are quite similar so we present recall value of classifiers in Table 4. The average values of each algorithm are listed after the result of each datasets, and the average ranks calculated as in the Friedman test are followed with it, the lower the average rank is, the better the classifier is. From recall value of each dataset, we can see clearly that GNB performs better than other basic classifiers in most of the datasets when the size of datasets is around 100 to 1100, and when sample size is larger than 1100, ABC and DTC perform better in most of datasets. Moreover, from the average result in the last two rows in Table 4, GNB attains to the highest average recall value from all datasets but ABC gets the best Friedman rank value among all nine classifiers comparison. We present result of G-mean and AUC together with recall in Figures 4 and 5, the figure shows the similar trend of the three metrics.

For more details, we divide the datasets into two parts according to sample size, and analyze the differences among these classifiers on two parts of datasets, respectively. The datasets with sample size smaller than 1100 are called small datasets, otherwise, we call them large dataset. The average value and Friedman rank are recalculated for the two parts of datasets in Table 5.

From Table 5 we discover law of these classifiers clearly that GNB performs best among the nine selected classifiers when the sample size of a dataset is small, and the exact boundary of small and

large sample size is 1100. ABC and DTC perform best in datasets of large sample size, the differences among classifier average results are analyzed in Table 6.

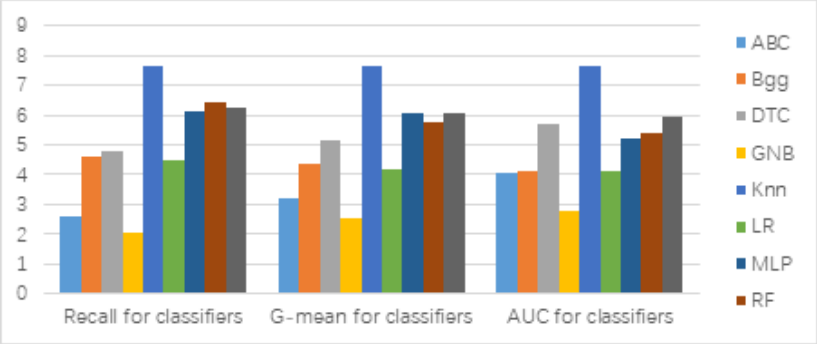

Figure 4. Average Friedman rank value of 3 metrics for 9 classifiers.

Table 4. Recall of comparison among basic classifiers.

Data	Sample Size	ABC	Bgg	DTC	GNB	Knn	LR	MLP	RF	SVC
log4j-1.0	135	0.563	**0.611**	0.441	0.608	0.26	0.472	0.472	0.432	0.392
synapse-1.0	157	0.26	0.086	0.181	**0.646**	0.205	0.256	0.164	0.037	0
synapse-1.1	222	0.446	0.374	0.379	**0.724**	0.164	0.462	0.45	0.415	0.486
arc	234	0.254	0.192	0.155	**0.762**	0.05	0.232	0.263	0.125	0.167
synapse-1.2	256	**0.559**	0.537	0.554	0.473	0.506	0.464	0.46	0.503	0.01
jedit-3.2	272	0.624	**0.662**	0.584	0.586	0.444	0.613	0.587	0.61	0.662
jedit-4.0	306	**0.512**	0.454	0.453	0.408	0.323	0.375	0.272	0.44	0.353
camel-1.0	339	0.025	0.033	0	**0.341**	0	0.06	0.25	0	0
jedit-4.2	367	0.341	0.257	0.275	**0.418**	0.118	0.335	0.13	0.192	0.274
Xerces-1.3	453	0.393	0.295	0.39	**0.457**	0.033	0.319	0.047	0.298	0.409
Xalan-2.4	723	0.253	0.111	0.209	**0.382**	0.126	0.202	0.09	0.104	0.089
ant-1.7	745	0.459	0.475	0.445	**0.555**	0.413	0.358	0.405	0.422	0.388
tomcat	858	0.229	0.125	0.177	**0.38**	0.006	0.197	0.082	0.088	0.228
camel-1.4	872	0.268	0.069	0.132	**0.288**	0.034	0.119	0.09	0.036	0.004
camel-1.6	965	0.194	0.119	0.144	**0.248**	0.025	0.138	0.104	0.044	0.036
PC3	1053	0.325	0.131	0.317	**0.876**	0.137	0.142	0.035	0.002	0
KC1	1162	**0.374**	0.301	0.36	0.234	0.258	0.238	0.211	0.2	0.01
PC4	1270	0.516	0.467	**0.54**	0.064	0.097	0.428	0.074	0.035	0.01
PC5	1694	**0.482**	0.387	0.47	0.112	0.335	0.228	0.217	0.228	0.029
MC1	1952	**0.297**	0.126	0.262	0.272	0.02	0.026	0.14	0	0
ec -2.1	6729	0.222	0.173	**0.289**	0.253	0.212	0.134	0.043	0.066	0.009
JM1	7720	**0.344**	0.191	0.332	0.062	0.198	0.102	0.154	0.073	0.011
ec2.0	7888	0.389	0.352	**0.422**	0.304	0.339	0.248	0.114	0.186	0.047
prop-5	8516	**0.232**	0.183	0.228	0.157	0.172	0.047	0.023	0.016	0.037
prop-4	8718	0.172	0.161	0.199	**0.233**	0.135	0.113	0.06	0.055	0.016
prop-3	10,274	0.148	0.13	0.137	**0.156**	0.101	0.029	0.018	0.002	0.037
ec3.0	10,593	0.297	0.234	**0.358**	0.253	0.266	0.157	0.141	0.104	0.025
prop-1	18,471	**0.353**	0.315	0.349	0.258	0.289	0.107	0.051	0.073	0.136
prop-2	23,014	0.33	0.317	**0.362**	0.208	0.203	0.037	0.024	0.014	0.063
Average recall		0.340	0.271	0.315	**0.370**	0.189	0.229	0.178	0.166	0.135
Average Friedman rank		**2.172**	4.259	3.362	3.103	6.086	5.138	6.569	7.121	7.190

Table 5. Average recall value and Friedman rank of 9 classifiers on different datasets.

Datasets	Average	ABC	Bgg	DTC	GNB	Knn	LR	MLP	RF	SVC
Small datasets	Recall value	0.357	0.283	0.302	**0.510**	0.178	0.297	0.244	0.234	0.219
	Rank value	2.563	4.719	4.656	**2.000**	7.469	4.469	6.156	6.531	6.438
Large datasets	Recall value	0.320	0.257	**0.331**	0.197	0.202	0.146	0.098	0.081	0.033
	Rank value	**1.692**	3.692	1.769	4.462	4.385	5.962	7.077	7.846	8.115

Table 6. *p*-Value and CD value of differences among classifier result on different datasets.

	Metric	Friedman Test	Nemenyi Analysis (CD)	α
Small datasets	Recall	<<0.00001	3.003	$\alpha = 0.05$
	G-mean	<<0.00001	3.003	$\alpha = 0.05$
	AUC	<<0.00001	3.003	$\alpha = 0.05$
Large datasets	Recall	<<0.00001	3.332	$\alpha = 0.05$
	G-mean	<<0.00001	3.332	$\alpha = 0.05$
	AUC	<<0.00001	3.332	$\alpha = 0.05$

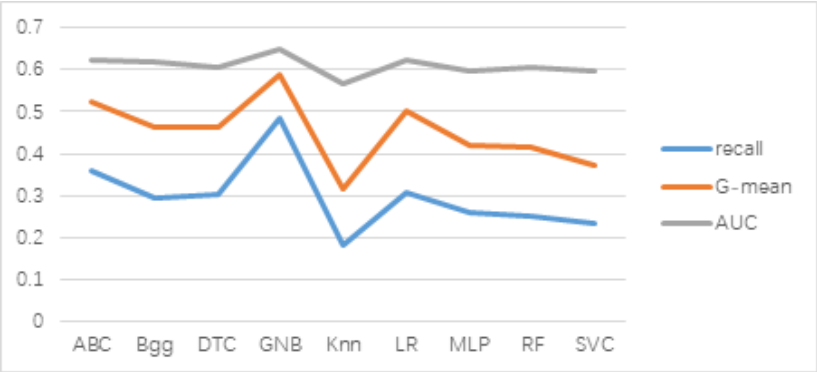

Figure 5. Average result value of 3 metrics for 9 classifiers.

From the results on small datasets, in the Friedman test, we reject the null hypothesis that the nine classifiers have no significant difference (p-values of the three metrics are all smaller than 0.00001). Carrying out the Nemenyi post hoc analysis (Critical Difference of three metrics CD = 3. 003, $\alpha = 0.05$) shows that ABC, Bgg, GNB, LR and DTC are significantly better than others. According to average rank and results of datasets, ABC and GNB seem to be slightly better than others and GNB is slightly better than ABC intuitively. In order to verify whether GNB is significantly better than ABC, we perform a further paired Wilcoxon test which null hypothesis is no significant difference between ABC and GNB. The p-value of three metrics are 0.017, 0.053, 0.088, respectively. According to the Wilcoxon test, for defective sample detecting metric (recall) GNB performs better than ABC significantly, and for overall accuracy metrics (G-mean and AUC) GNB performs a little better than ABC. Considering the cost of false negative is much more than false positive, we attach more importance to recall. Therefore, we regard GNB as the best classifier among nine basic classifiers and better than other eight classifiers on datasets with small sample size. However, we also see the bad performance of all the nine classifiers, even GNB have a relatively low AUC on small datasets. This indicates that performance of classifiers are restricted by the class imbalance problem, and there is a great space of improvement in performance of basic classifiers after overcoming the class imbalance problem reasonably. From the results on large datasets, we can extract observations that GNB performs worse than DTC and ABC. In the Friedman test, all the p-values of three metrics are smaller than 0.00001, which shows there is significant difference between nine classifiers, then the Nemenyi post hoc analysis (critical difference of three metric CD = 3.332, $\alpha = 0.05$) shows that LR, MLP, RF and SVC perform significantly worse than other classifiers, these classifiers are unsuitable for SDP datasets with a large sample size. Then the paired Wilcoxon test turns out that differences between ABC and DTC are not significant because the null hypothesis on no significant difference between ABC and DTC cannot be rejected (*p*-value of recall, G-mean and AUC are 0.433, 0.396, 0.753, respectively). Furthermore, combined with the average rank we have found that ABC performs well in both small and large datasets (rank second in small-sample-size datasets and rank first in large-sample-size datasets), the result of Wilcoxon rank sum test support the null hypothesis that no significant difference between large and small datasets for ABC (*p*-value of recall, G-mean and AUC are 0.558, 0.661, 0.539, respectively). The result reflects that the performance of ABC is not affected by sample size of datasets.

RQ2: How does LIMCR perform compared with other imbalance learning methods in small sample size of datasets?

Motivation: We have selected the baseline classifier GNB in small sample size in RQ1. Now, we need to validate the effectiveness of our proposed LIMCR.

Approach: To solve the class imbalance problem, researchers usually use two kinds of methods: resampling methods based on data level and classification methods based on algorithm level. Resampling methods are always sorted into three categories: over-sampling methods, under-sampling methods and combination of over- and under-sampling methods. The question that which method is more suitable for the class imbalance problem has been discussed in many research [4,60,61]. To evaluate the effectiveness of LIMCR, we employ six baseline imbalance learning methods with parameters which are listed in Table 7. All these baseline methods are implemented in the Imbalanced-learn model in Python [62].

Table 7. Baseline imbalance learning methods.

Level	Category	Imbalance Learning Method	Abbreviation	Parameters
Data	Over-sampling	Borderline-SMOTE [22]	B-SMO	Default
	Under-sampling	Neighbourhood Cleaningrule [7]	NCL	Default
		Instance hardness threshold [34]	IHT	Default
	Combination of over- and under-sampling methods	SMOTE+ENN [25]	SMOE	Default
Algorithm	Algorithm	RUSBoost	RUSB	Default
		EasyEnsemble	EasyE	Default

In this experiment, we compare performance metrics (balancedscore and G-mean) of our LIMCR with baseline imbalance learning methods in small sample size of benchmark datasets. All imbalance learning methods including LIMCR are combined with baseline classifier GNB.

Results: Tables 8 and 9 present the results of our LIMCR and the baseline imbalance learning methods on small datasets in terms of balancedscore and G-mean, respectively. We notice that the average balancedscore and G-mean of LIMCR are 0.701 and 0.69 which perfoms better than other baseline imbalance learning methods.

Table 8. Balancedscore of GNB with different imbalance learning methods on small datasets.

	LIMCR	SMOE	IHT	B-SMO	NCL	RUSB	EasyE
synapse-1.2	0.729	0.699	0.715	0.627	0.711	0.537	0.682
jedit-3.2	0.703	0.669	0.72	0.73	0.689	0.53	0.645
synapse-1.1	0.692	0.684	0.643	0.617	0.594	0.568	0.522
log4j-1.0	0.772	0.645	0.602	0.684	0.782	0.511	0.623
jedit-4.0	0.758	0.708	0.64	0.692	0.708	0.556	0.66
ant-1.7	0.78	0.754	0.717	0.709	0.763	0.588	0.668
camel-1.6	0.608	0.616	0.59	0.594	0.605	0.543	0.58
camel-1.4	0.656	0.665	0.658	0.578	0.605	0.56	0.603
Xerces-1.3	0.693	0.787	0.761	0.719	0.65	0.502	0.493
Xalan-2.4	0.68	0.65	0.68	0.68	0.684	0.55	0.668
jedit-4.2	0.758	0.759	0.727	0.73	0.722	0.705	0.76
arc	0.561	0.659	0.583	0.443	0.616	0.622	0.554
synapse-1.0	0.741	0.754	0.723	0.658	0.554	0.402	0.679
tomcat	0.735	0.857	0.605	0.687	0.677	0.488	0.691
camel-1.0	0.654	0.54	0.608	0.811	0.679	0.624	0.698
Average	**0.701**	0.696	0.665	0.664	0.669	0.552	0.635
Friedman rank	**2.467**	2.767	3.933	4	3.567	6.4	4.867

Table 9. G-mean of GNB with different imbalance learning methods on small datasets.

	LIMCR	SMOE	IHT	B-SMO	NCL	RUSB	EasyE
synapse-1.2	0.721	0.698	0.714	0.603	0.707	0.341	0.681
jedit-3.2	0.692	0.669	0.703	0.709	0.671	0.377	0.639
synapse-1.1	0.672	0.677	0.643	0.616	0.59	0.402	0.517
log4j-1.0	0.76	0.619	0.595	0.666	0.779	0.26	0.6
jedit-4.0	0.755	0.702	0.638	0.681	0.698	0.361	0.628
ant-1.7	0.78	0.749	0.71	0.697	0.755	0.481	0.668
camel-1.6	0.555	0.578	0.589	0.501	0.531	0.431	0.569
camel-1.4	0.65	0.663	0.653	0.504	0.549	0.41	0.594
Xerces-1.3	0.675	0.786	0.732	0.694	0.619	0.281	0.47
Xalan-2.4	0.665	0.635	0.63	0.649	0.658	0.444	0.667
jedit-4.2	0.757	0.759	0.673	0.73	0.699	0.699	0.75
arc	0.558	0.649	0.581	0.383	0.612	0.563	0.553
synapse-1.0	0.741	0.749	0.711	0.658	0.539	0	0.678
tomcat	0.735	0.857	0.604	0.661	0.659	0.162	0.67
camel-1.0	0.635	0.521	0.598	0.808	0.655	0.582	0.698
Average	**0.69**	0.687	0.652	0.637	0.648	0.386	0.625
Friedman rank	**2.533**	2.6	3.8	4.2	3.833	6.633	4.4

In further study, p-value in Friedman test in these two performance metrics are all smaller than 0.00001, which shows that significant difference is existing among the seven methods. Then, the Nemenyi post hoc test shows CD = 2.326, $\alpha < 0.05$. According to Nemenyi post hoc test, we underline the average ranks which significantly worse than our LIMCR. The results reflect that ensemble algorithms performs significantly worse than resampling method combined with GNB. In order to find out if there is any significant difference between resampling methods, we perform Wilcoxon signed-rank test between our LIMCR and other resampling methods, the p-values of each test are listed in Table 10.

Table 10. p-Value of Wilcoxon signed-rank tests of resampling methods.

	LIMCR vs. SMOE	LIMCR vs. IHT	LIMCR vs. B-SMO	LIMCR vs. NCL
Balancedscore	0.532	0.043	0.031	0.041
G-mean	0.691	0.047	0.198	0.011

From Table 10 we can learn that SMOE performs no significant difference from LIMCR, but for the Friedman average ranks, LIMCR performs slightly better than SMOE. The overall metrics balancedscore and G-mean of IHT shows significantly worse than LIMCR. Another two methods B-SMO and NCL perform significantly worse than LIMCR in most of datasets for balancedscore and G-mean. Therefore we conclude that LIMCR performs better than most of imbalance learning methods.

RQ3: How does LIMCR work with other classifiers?

Motivation: In principle analyses, our LIMCR is based on Bayesian Probability, and GNB is the most suitable classifier for it. However, we expect that LIMCR still has good performance when it is combined with other classifiers.

Approach: In this experiment, we choose another two well performed classifiers ABC and DTC, with the combination of three resampling methods LIMCR, SOME and IHT on small datasets in terms of balancedscore and G-mean for further comparison.

Results: Tables 11 and 12 show the results of matching of three classifiers with three resampling methods on small datasets in terms of balacedscore and G-mean. We notice that the average balacedscore value of LIMCR+GNB is 0.701 which is equal to IHT+ABC and higher than other combinations. Simultaneously, the average G-mean of LIMCR+GNB is 0.69 while it is 0.696 for IHT+ABC.

Table 11. Balancedscore of matching of classifiers with resampling methods on small datasets.

Datasets	ILMCR			SMOE			IHT		
	GNB	DTC	ABC	GNB	DTC	ABC	GNB	DTC	ABC
synapse-1.2	0.729	0.701	0.689	0.699	0.713	0.719	0.715	0.663	0.663
jedit-3.2	0.703	0.743	0.73	0.669	0.698	0.716	0.72	0.731	0.733
synapse-1.1	0.692	0.64	0.691	0.684	0.537	0.657	0.643	0.596	0.742
log4j-1.0	0.772	0.732	0.726	0.645	0.75	0.762	0.602	0.75	0.685
jedit-4.0	0.758	0.666	0.734	0.708	0.63	0.652	0.64	0.659	0.71
ant-1.7	0.78	0.778	0.718	0.754	0.739	0.687	0.717	0.768	0.779
camel-1.6	0.608	0.599	0.595	0.616	0.635	0.657	0.59	0.599	0.643
camel-1.4	0.656	0.525	0.529	0.665	0.588	0.612	0.658	0.591	0.597
Xerces-1.3	0.693	0.659	0.666	0.787	0.698	0.745	0.761	0.684	0.682
Xalan-2.4	0.68	0.639	0.549	0.65	0.657	0.7	0.68	0.708	0.725
jedit-4.2	0.758	0.674	0.636	0.759	0.756	0.737	0.727	0.686	0.734
arc	0.561	0.604	0.569	0.659	0.502	0.649	0.583	0.526	0.507
synapse-1.0	0.741	0.679	0.652	0.754	0.688	0.696	0.723	0.732	0.759
tomcat	0.735	0.639	0.726	0.857	0.776	0.786	0.605	0.752	0.779
camel-1.0	0.654	0.751	0.694	0.54	0.782	0.624	0.608	0.74	0.778
Average	**0.701**	0.669	0.66	0.696	0.677	0.693	0.665	0.679	**0.701**
Rank	**3.567**	5.9	6.467	4.2	5.567	4.2	5.9	5.367	3.833

Table 12. G-mean of matching of classifiers with resampling methods on small datasets.

Datasets	ILMCR			SMOE			IHT		
	GNB	DTC	ABC	GNB	DTC	ABC	GNB	DTC	ABC
synapse-1.2	0.721	0.688	0.689	0.698	0.713	0.717	0.714	0.66	0.66
jedit-3.2	0.692	0.74	0.73	0.669	0.64	0.685	0.703	0.725	0.73
synapse-1.1	0.672	0.639	0.69	0.677	0.511	0.638	0.643	0.552	0.734
log4j-1.0	0.76	0.732	0.724	0.619	0.745	0.759	0.595	0.745	0.677
jedit-4.0	0.755	0.632	0.734	0.702	0.623	0.622	0.638	0.656	0.709
ant-1.7	0.78	0.772	0.717	0.749	0.723	0.671	0.71	0.766	0.778
camel-1.6	0.555	0.592	0.594	0.578	0.606	0.644	0.589	0.599	0.638
camel-1.4	0.65	0.522	0.514	0.663	0.559	0.586	0.653	0.591	0.587
Xerces-1.3	0.675	0.657	0.666	0.786	0.687	0.719	0.732	0.682	0.678
Xalan-2.4	0.665	0.613	0.542	0.635	0.645	0.671	0.63	0.682	0.724
jedit-4.2	0.757	0.674	0.636	0.759	0.743	0.73	0.673	0.686	0.725
arc	0.558	0.604	0.538	0.649	0.501	0.647	0.581	0.506	0.492
synapse-1.0	0.741	0.631	0.647	0.749	0.666	0.678	0.711	0.721	0.753
tomcat	0.735	0.639	0.723	0.857	0.776	0.786	0.604	0.746	0.777
camel-1.0	0.635	0.75	0.686	0.521	0.781	0.582	0.598	0.737	0.778
Average	0.69	0.659	0.655	0.687	0.661	0.676	0.652	0.67	**0.696**
Rank	3.933	6	6.1	4.2	5.433	4.733	5.8	4.933	**3.867**

From the average score and Friedman average ranks we see the combination of LIMCR and GNB still performs better than others except for G-mean, IHT+ABC ranks first on G-mean and LIMCR+GNB is slightly worse than it.

Friedman test results are shown in Table 13, column named Total is the Friedman test among all nine methods, p-value of metric G-mean is 0.123 larger than 0.05 means there is no significant difference among all nine methods, i.e., LIMCR with other classifiers can perform as well as it with GNB and other methods for G-mean. The result of Nemenyi post hoc test (CD = 3.12, $\alpha < 0.05$) supports this result. Column LIMCR, SMOE and IHT represent the Friedman tests among classifiers with same resampling method, respectively, for instance, p-value in column LIMCR is the result of Friedman test among GNB, DTC, ABC with the same resampling method LIMCR. For LIMCR on balancedscore p-value is smaller than 0.05, which suggests the combination of LIMCR + GNB performs significantly better than other combinations of LIMCR. For other resampling methods, p-value of balancedscore and G-mean

are all larger than 0.05, which suggests that for SMOE and IHT, there are no significant difference among different combinations of them with different classifiers on the perspective of Balancedscore and G-mean. We can draw a conclusion from the experiment, LIMCR, SMOE and IHT all perform no significant difference on some metrics by being combined with different classifiers but inversely on other metrics. We take more consideration on the difference, performance of data resampling methods may change by using different classifiers in imbalance learning, therefore, when the dataset has a small sample size, it is necessary to choose GNB or Naïve Bayes as the basic classifier in imbalance learning.

Table 13. *p*-Value of Friedment test between different classifiers for resampling methods.

Metrics	Total	LIMCR	SMOE	IHT
Balancedscore	0.024	0.011	0.165	0.051
G-mean	0.123	0.085	0.154	0.07

RQ4: Does the number of features of datasets have an influence in LIMCR?

Motivation: Our proposed method LIMCR is strongly related to the features of datasets. In former experiments we use datasets with the same feature dimension and have no idea if the number of features have influence on the performance of LIMCR.

Approach: In this experiment, we retain k (k = 4, 8, 12, 16 and 20) highest scoring features to observe the variation of performance on LIMCR. We exploit a feature selection method named SelectKBest from model feature selection in scikit-learn v0.20.3 [58], and the dependence score between each feature and class label is measured by chi-square [57]. The main reason we choose this method is the convenience for selecting certain number of features in the experiment, and it removes features by univariate analyze suits for datasets.

Results: The results are shown in Tables 14 and 15. We notice that the average balancedscore values vary from 0.636 to 0.662 and the average G-mean from 0.534 to 0.631. When the number of feature is 16, the values of average balancedscore and G-mean are highest.

Table 14. Balancedscore of different number features in GNB with LIMCR on small datasets.

Dataset	4	8	12	16	20
synapse-1.2	0.642	0.649	0.638	0.659	0.68
jedit-3.2	0.659	0.738	0.762	0.741	0.772
synapse-1.1	0.637	0.661	0.661	0.623	0.622
log4j-1.0	0.643	0.649	0.673	0.542	0.577
jedit-4.0	0.571	0.571	0.579	0.621	0.657
ant-1.7	0.716	0.768	0.773	0.761	0.767
camel-1.6	0.531	0.539	0.562	0.569	0.568
camel-1.4	0.518	0.564	0.587	0.622	0.63
Xerces-1.3	0.607	0.609	0.692	0.674	0.675
Xalan-2.4	0.588	0.589	0.619	0.686	0.661
jedit-4.2	0.616	0.639	0.685	0.683	0.726
arc	0.699	0.693	0.646	0.631	0.59
synapse-1.0	0.922	0.822	0.853	0.707	0.707
tomcat	0.614	0.621	0.64	0.666	0.704
camel-1.0	0.576	0.537	0.523	0.745	0.559
Average	0.636	0.643	0.66	**0.662**	0.66
Rank	3.9	3.333	2.567	2.767	**2.433**

Table 15. G-mean of different number features in GNB with LIMCR on small datasets.

Dataset	4	8	12	16	20
synapse-1.2	0.59	0.606	0.623	0.646	0.673
jedit-3.2	0.593	0.719	0.758	0.736	0.771
synapse-1.1	0.552	0.607	0.654	0.584	0.607
log4j-1.0	0.563	0.606	0.622	0.456	0.476
jedit-4.0	0.449	0.449	0.514	0.582	0.638
ant-1.7	0.69	0.762	0.772	0.761	0.767
camel-1.6	0.314	0.354	0.467	0.45	0.449
camel-1.4	0.27	0.415	0.505	0.622	0.626
Xerces-1.3	0.515	0.537	0.664	0.64	0.675
Xalan-2.4	0.477	0.507	0.56	0.68	0.638
jedit-4.2	0.512	0.553	0.627	0.644	0.69
arc	0.651	0.647	0.614	0.626	0.589
synapse-1.0	0.919	0.822	0.841	0.643	0.643
tomcat	0.508	0.536	0.587	0.658	0.698
camel-1.0	0.405	0.389	0.523	0.741	0.487
Average	0.534	0.567	0.622	**0.631**	0.628
Rank	4.233	3.6	2.267	2.7	**2.2**

From the *p*-value of Friedman test in Table 16 we know for metric balancedscore there is no significant difference between the number of features in five levels. According to the result of Nemenyi post hoc test (CD = 1.575), the Friedman average ranks which larger than the best one more than 1.575. For metric G-mean, only datasets with four features perform significantly worse the best one and others has no significant difference. When the number of features declines, the precision score increases and the overall performance is declining. In summary, it can be believed that the number of features has no influence on the performance of LIMCR unless the number of features below a certain value. The value in this experiment is 4 or 5.

Table 16. *p*-Value of Friedman test for RQ4.

Metric	Balancedscore	G-Mean
p-value	0.054	<0.001

RQ5: How does LIMCR work with datasets with large sample sizes?

Motivation: From RQ1 we know sample size has a great influence on classifier selecting in imbalance learning and our proposed LIMCR is proved performing well with small datasets. However, how LIMCR performs on datasets with large sample size is also needed to know.

Approach: In this experiment we combine ILMCR with three classifiers, GNB, ABC and DTC. IHT combined with the same classifiers are used as comparison. The datasets are large datasets introduced in RQ1.

Results: The results of comparison between six combined methods are listed in Tables 17 and 18.

From Tables 17 and 18, for both metrics balancedscore and G-mean, IHT get higher results than LIMCR when using the same classifier. In other words, resampling method IHT performs better than our LIMCR according to average score and Friedman average ranks. Meanwhile, the Nemenyi post hoc test (CD = 2.015) result declares that all classifiers combined with LIMCR perform significantly worse than the best performance on balancedscore. From these we can conclude that the proposed LIMCR performs well when the sample size is small (generally smaller than 1100), but it turns worse than IHT when sample size increases (generally larger than 1100). So LIMCR can achieve better performance with typical classifier when the sample size is smaller.

Table 17. Balancedscore of matching of classifiers with LIMCR and IHT on large datasets.

Datasets	ILMCR+GNB	ILMCR+DTC	ILMCR+ABC	IHT+GNB	IHT+DTC	IHT+ABC
JM1	0.562	0.515	0.522	0.615	0.64	0.612
KC1	0.631	0.634	0.617	0.628	0.62	0.647
MC1	0.655	0.536	0.57	0.697	0.563	0.617
PC3	0.486	0.53	0.538	0.718	0.755	0.741
PC4	0.577	0.717	0.759	0.65	0.826	0.799
PC5	0.554	0.556	0.595	0.623	0.679	0.669
prop-1	0.612	0.652	0.626	0.63	0.69	0.692
prop-2	0.593	0.539	0.53	0.595	0.573	0.7
prop-3	0.539	0.589	0.542	0.58	0.64	0.665
prop-4	0.606	0.607	0.597	0.632	0.691	0.704
prop-5	0.544	0.56	0.551	0.605	0.635	0.681
ec-2.0	0.694	0.624	0.683	0.674	0.686	0.685
ec-2.1	0.643	0.628	0.651	0.683	0.677	0.659
ec-3.0	0.679	0.638	0.648	0.672	0.65	0.651
Average	0.598	0.595	0.602	0.643	0.666	**0.68**
Rank	4.286	4.643	4.714	3	2.429	**1.929**

Table 18. G-mean of matching of classifiers with LIMCR and IHT on large datasets.

Datasets	ILMCR+GNB	ILMCR+DTC	ILMCR+ABC	IHT+GNB	IHT+DTC	IHT+ABC
JM1	0.398	0.187	0.231	0.559	0.628	0.596
KC1	0.583	0.569	0.534	0.619	0.609	0.634
MC1	0.654	0.277	0.389	0.668	0.375	0.521
PC3	0.207	0.27	0.35	0.713	0.744	0.724
PC4	0.483	0.704	0.759	0.647	0.815	0.773
PC5	0.366	0.36	0.534	0.59	0.65	0.645
prop-1	0.573	0.617	0.588	0.613	0.687	0.69
prop-2	0.506	0.307	0.516	0.546	0.452	0.697
prop-3	0.466	0.576	0.511	0.514	0.639	0.663
prop-4	0.559	0.501	0.554	0.601	0.689	0.703
prop-5	0.396	0.558	0.546	0.575	0.633	0.66
ec-2.0	0.683	0.62	0.674	0.641	0.651	0.642
ec-2.1	0.62	0.628	0.648	0.672	0.635	0.6
ec-3.0	0.662	0.638	0.643	0.64	0.599	0.597
Average	0.511	0.487	0.534	0.614	0.629	**0.653**
Rank	4.286	4.857	3.929	3	2.571	**2.357**

5. Discussion

5.1. Why Hold 3 Digits for Informative Variable D?

As mentioned in Section 3.3, the rank value of informative variable D on features of a sample have a great effect on estimating how informative the sample is, moreover, the precision of variable D affect the rank value directly. Therefore, it is necessary to discuss a proper value of this parameter (precision for variable D). The aim of this discussion is to present the effects of different precision of D on the performance of proposed LIMCR. Considering the space limitation, here, we randomly select the result of three datasets with different sample size (sample sizes of synapse-1.0, PC4, prop2 are 157, 1270, 23014, respectively) and the average result of 29 datasets introduced in Section 3.1. We choose GNB as classifier and evaluate the performance with balancedscore, precision, recall, and G-mean. The value of the precision of D is varied from 0 to 5 with increment of 1. The experimental result are presented as shown in Figure 6.

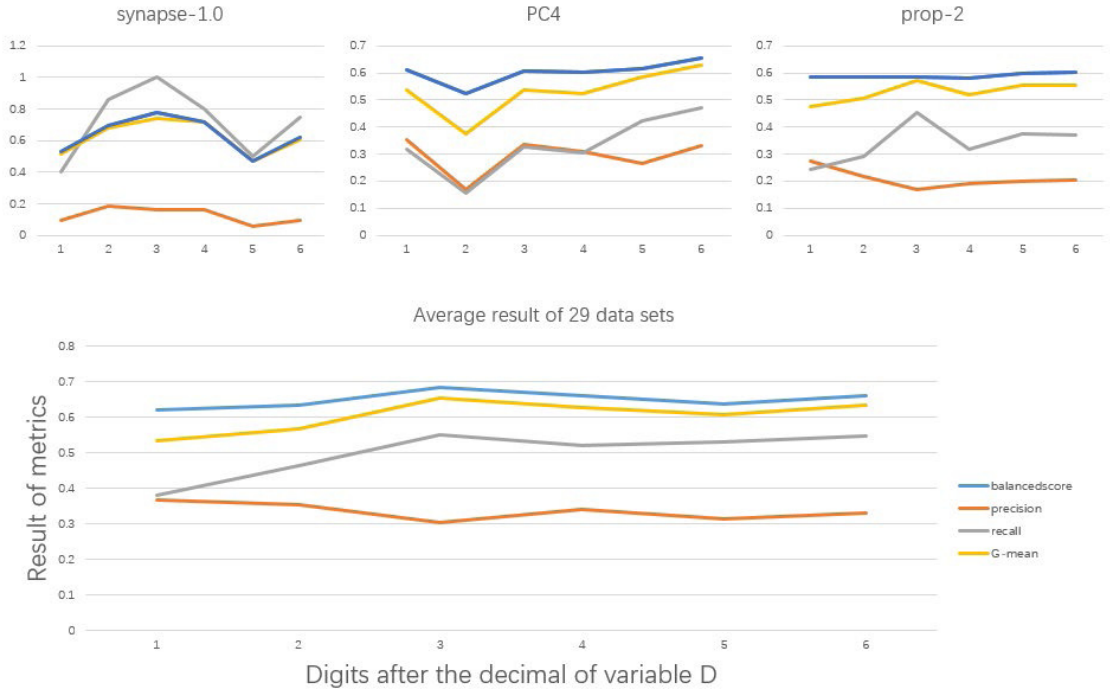

Figure 6. Result of LIMCR with different precision of variable *D*.

From the figures we notice that when this parameter equals to 3, the result of LIMCR performs stable and better than most of other values on average result. The metrics excepting precision have an increasing trend with the increase of this parameter value. Inversely, the metric precision decreases with the parameter value increasing. In order to obtaining the global optimum performance, we choose 3 as the generic value of this parameter.

5.2. Threats to Validity

There are still several potential limitations in our study which are shown as follows.

1. Quality and quantity of datasets for empirical study might be insufficient. Although we have collected more than 100 datasets for illustrating the distribution of sample size and imbalanced ratio in most of SDP datasets, and 29 datasets for investigations in empirical study. It is still hard to confirm if these datasets are typical to reflect characters of SDP data
2. The generalization of our method might be limited. The method we proposed focus on binary classification, it improves the performance of predicting if a sample (software model) has any defects but cannot predict the number of defects in it. More types of defect datasets should be considered in the future to reduce the threats.
3. The performance evaluating metrics we selected might be partial. There are many metrics such as PD and MCC have been used in binary classification for SDP research. At the same time, F1 is also widely used in SDP, but we do not employ it as it is proved to be biased and unreliable [63]. Although we have considered to select evaluating metrics from two aspects, overall performance and one-class accuracy, however, the limited number of metrics still pose some threats to the construct validity.
4. Practical significance of LIMCR in software engineering might be extended. Project members can obtain information on possible defect-prone modules of the software before failure occur by using defect prediction technique, LIMCR has not been applied to predict defect classes/severities [64]. In addition, it is worth studying the performance of LIMCR with different prediction models (within a single project, cross project predictions) [65]. Meanwhile, how to cooperate with instance deletion, missing values replacement, normalization issues mentioned in [66] and defect prediction cost effectiveness [67] also needs further research.

6. Conclusions

The performance of a defect prediction model is influenced by the sample size of dataset, selection of classifiers and data resampling methods. From our empirical study, we compared performance of nine popular classifiers in 29 software datasets with different sample size ranging from 100 to 20,000 to study the influence of sample size and classifiers. The major conclusion in this part is that GNB performs well with small datasets, but its performance deteriorates when sample size of datasets grow to 1100. Another classifier ABC performs stable with different sample size and obtains relatively better result with large datasets among classifiers. On this basis, in order to make an expected matching on small datasets, we proposed a new resampling method LIMCR motivated by the good performance of GNB. LIMCR is used for SDP datasets with small sample size and it is designed as the best resampling method cooperated with classifier GNB. The results of comparison experiments confirm that the performance of LIMCR is better than the other resampling methods, and the matching between GNB and LIMCR is the best solution for the imbalance problem in SDP datasets with small datasets. Besides, we also design experiments to research how it performs with other classifiers, feature selection and data with large sample size. The results can be summarized as follows.

1. LIMCR together with classifier GNB is a better solution for the imbalance problem on SDP datasets with small datasets which is slightly better than SMOE+GNB .
2. On aspect of metric G-mean, LIMCR has the same well performance when cooperates with other classifiers. On aspect of metric balancedscore, when cooperating with LIMCR, GNB performs significantly better than other classifiers.
3. Number of features in a datasets has no influence on LIMCR, but the performance turn significantly worse when the number of features less than 5.
4. When the sample size bigger than 1100, performance of LIMCR is worse than IHT, so when sample size bigger than 1100, IHT is recommended as the best imbalanced learning method for SDP.

Although our proposed LIMCR cannot outperform for all datasets, but the result of our research emphasizes the importance of the influence of datasets. There is no all-purpose imbalance learning methods, the way of choosing methods appropriately is also important. In the future, we plan to extend our research to cover other data distribution problems such as overlapping problem and high dimensionality. We will update our LIMCR to solve more combined problems and being suitable for more SDP datasets.

Author Contributions: Conceptualization, Y.W.; methodology, Y.W. and J.Y.; software, J.Y.; validation, Y.W. and J.Y.; formal analysis, J.Y.; investigation, J.Y. and S.C.; resources, S.C.; data curation, S.C.; writing–original draft preparation, J.Y. and S.C.; writing–review and editing, Y.W.; visualization, J.Y.; supervision, B.L.; project administration, Y.W.; funding acquisition, Y.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Aerospace Science Foundation of China (grant number 2017ZD51052).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Tong, H.; Liu, B.; Wang, S. Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning. *Inf. Softw. Technol.* **2018**, *96*, 94–111. [CrossRef]
2. Chen, L.; Fang, B.; Shang, Z.; Tang, Y. Tackling class overlap and imbalance problems in software defect prediction. *Softw. Qual. J.* **2018**, *26*, 97–125. [CrossRef]
3. Limsettho, N.; Bennin, K.E.; Keung, J.W.; Hata, H.; Matsumoto, K. Cross project defect prediction using class distribution estimation and oversampling. *Inf. Softw. Technol.* **2018**, *100*, 87–102. [CrossRef]

4. Galar, M.; Fernandez, A.; Barrenechea, E.; Bustince, H.; Herrera, F. A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches. *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.* **2011**, *42*, 463–484. [CrossRef]
5. Błaszczyński, J.; Stefanowski, J. Neighbourhood sampling in bagging for imbalanced data. *Neurocomputing* **2015**, *150*, 529–542. [CrossRef]
6. Chawla, N.V.; Bowyer, K.W.; Hall, L.O.; Kegelmeyer, W.P. SMOTE: Synthetic minority over-sampling technique. *J. Artif. Intell. Res.* **2002**, *16*, 321–357. [CrossRef]
7. Laurikkala, J. Improving identification of difficult small classes by balancing class distribution. In Proceedings of the Conference on Artificial Intelligence in Medicine in Europe, Cascais, Portugal, 1–4 July 2001; pp. 63–66.
8. Chen, S.; He, H.; Garcia, E.A. RAMOBoost: Ranked minority oversampling in boosting. *IEEE Trans. Neural Netw.* **2010**, *21*, 1624–1642. [CrossRef]
9. Freund, Y.; Schapire, R. Experiments with a new boosting algorithm. In Proceedings of the Thirteenth International Conference (ICML 1996), Bari, Italy, 3–6 July 1996; pp. 148–156.
10. Breiman, L. Bagging predictors. *Mach. Learn.* **1996**, *24*, 123–140. [CrossRef]
11. Fan, W.; Stolfo, S.J.; Zhang, J.; Chan, P.K. AdaCost: Misclassification Cost-Sensitive Boosting. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML '99)*; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 1999; Volume 99, pp. 97–105.
12. Freund, Y.; Schapire, R.E. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.* **1997**, *55*, 119–139. [CrossRef]
13. Lusa, L.; Blagus, R. Class prediction for high-dimensional class-imbalanced data. *BMC Bioinform.* **2010**, *11*, 523.
14. García, V.; Sánchez, J.S.; Mollineda, R.A. On the effectiveness of preprocessing methods when dealing with different levels of class imbalance. *Knowl. Based Syst.* **2012**, *25*, 13–21. [CrossRef]
15. Wang, S.; Yao, X. Diversity analysis on imbalanced data sets by using ensemble models. In Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Data Mining, Nashville, TN, USA, 30 March–2 April 2009; pp. 324–331.
16. Díez-Pastor, J.F.; Rodríguez, J.J.; García-Osorio, C.I.; Kuncheva, L.I. Diversity techniques improve the performance of the best imbalance learning ensembles. *Inf. Ences* **2015**, *325*, 98–117. [CrossRef]
17. Weiss, G.M.; Provost, F. *The Effect of Class Distribution on Classifier Learning*; Technical Report ML-TR-44; Department of Computer Science, Rutgers University: New Brunswick, NJ, USA, 2001.
18. Khoshgoftaar, T.M.; Van Hulse, J.; Napolitano, A. Comparing boosting and bagging techniques with noisy and imbalanced data. *IEEE Trans. Syst. Man Cybern. Part A Syst. Hum.* **2010**, *41*, 552–568. [CrossRef]
19. More, A. Survey of resampling techniques for improving classification performance in unbalanced datasets. *arXiv* **2016**, arXiv:1608.06048.
20. He, H.; Garcia, E.A. Learning from imbalanced data. *IEEE Trans. Knowl. Data Eng.* **2009**, *21*, 1263–1284.
21. He, H.; Bai, Y.; Garcia, E.A.; Li, S. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In Proceedings of the 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), Hong Kong, China, 1–8 June 2008.
22. Han, H.; Wang, W.; Mao, B. Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. In Proceedings of the International Conference on Intelligent Computing 2005, Hefei, China, 23–26 August 2005; pp. 878–887.
23. Barua, S.; Islam, M.M.; Yao, X.; Murase, K. MWMOTE–majority weighted minority oversampling technique for imbalanced data set learning. *IEEE Trans. Knowl. Data Eng.* **2012**, *26*, 405–425. [CrossRef]
24. Bunkhumpornpat, C.; Sinapiromsaran, K.; Lursinsap, C. Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem. In Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining, Bangkok, Thailand, 27–30 April 2009; pp. 475–482.
25. Batista, G.E.; Prati, R.C.; Monard, M.C. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD Explor. Newsl.* **2004**, *6*, 20–29. [CrossRef]
26. Sun, Z.; Song, Q.; Zhu, X.; Sun, H.; Xu, B.; Zhou, Y. A novel ensemble method for classifying imbalanced data. *Pattern Recognit.* **2015**, *48*, 1623–1637. [CrossRef]
27. Wang, S.; Yao, X. Using class imbalance learning for software defect prediction. *IEEE Trans. Reliab.* **2013**, *62*, 434–443. [CrossRef]

28. Chawla, N.V.; Lazarevic, A.; Hall, L.O.; Bowyer, K.W. SMOTEBoost: Improving prediction of the minority class in boosting. In Proceedings of the European Conference on Principles of Data Mining and Knowledge Discovery, Cavtat-Dubrovnik, Croatia, 22–26 September 2003; pp. 107–119.
29. Menzies, T.; Turhan, B.; Bener, A.; Gay, G.; Cukic, B.; Jiang, Y. Implications of ceiling effects in defect predictors. In Proceedings of the 4th International Workshop on Predictor Models in Software Engineering, Leipzig, Germany, 12–13 May 2008; pp. 47–54.
30. Hart, P. The condensed nearest neighbor rule (Corresp.). *IEEE Trans. Inf. Theory* **1968**, *14*, 515–516. [CrossRef]
31. Tomek, I. Two modifications of CNN. *IEEE Trans. Syst. Man Cybern.* **1976**. [CrossRef]
32. Wilson, D.L. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Trans. Syst. Man Cybern.* **1972**, 408–421. [CrossRef]
33. Kubat, M.; Matwin, S. Addressing the Curse of Imbalanced Training Sets: One-Sided Selection. In Proceedings of the 14th International Conference on Machine Learning (ICML 1997), Nashville, TN, USA, 8–12 July 1997; Volume 97, pp. 179–186.
34. Smith, M.R.; Martinez, T.; Giraud-Carrier, C. An instance level analysis of data complexity. *Mach. Learn.* **2014**, *95*, 225–256. [CrossRef]
35. Gupta, S.; Gupta, A. A set of measures designed to identify overlapped instances in software defect prediction. *Computing* **2017**, *99*, 889–914. [CrossRef]
36. Liu, X.Y.; Wu, J.; Zhou, Z.H. Exploratory undersampling for class-imbalance learning. *IEEE Trans. Syst. Man Cybern. Part B Cybern.* **2008**, *39*, 539–550.
37. Wang, T.; Zhang, Z.; Jing, X.; Zhang, L. Multiple kernel ensemble learning for software defect prediction. *Autom. Softw. Eng.* **2016**, *23*, 569–590. [CrossRef]
38. McCabe, T.J. A complexity measure. *IEEE Trans. Softw. Eng.* **1976**, 308–320. [CrossRef]
39. Maurice, H.H. *Elements of Software Science*; Elsevier: New York, NY, USA, 1977.
40. Song, Q.; Guo, Y.; Shepperd, M. A comprehensive investigation of the role of imbalanced learning for software defect prediction. *IEEE Trans. Softw. Eng.* **2018**, *45*, 1253–1269. [CrossRef]
41. Malhotra, R.; Khanna, M. An empirical study for software change prediction using imbalanced data. *Empir. Softw. Eng.* **2017**, *22*, 2806–2851. [CrossRef]
42. Kim, S.; Zhang, H.; Wu, R.; Gong, L. Dealing with noise in defect prediction. In Proceedings of the 2011 33rd International Conference on Software Engineering (ICSE), Honolulu, HI, USA, 21–28 May 2011; pp. 481–490.
43. Wang, H.; Khoshgoftaar, T.M.; Napolitano, A. Software measurement data reduction using ensemble techniques. *Neurocomputing* **2012**, *92*, 124–132. [CrossRef]
44. John, G.H.; Langley, P. Estimating continuous distributions in Bayesian classifiers. *arXiv* **2013**, arXiv:1302.4964.
45. Haykin, S. *Neural Networks: A Comprehensive Foundation*; Prentice-Hall, Inc.: Upper Saddle River, NJ, USA, 2007.
46. Aha, D.W.; Kibler, D.; Albert, M.K. Instance-based learning algorithms. *Mach. Learn.* **1991**, *6*, 37–66. [CrossRef]
47. Le Cessie, S.; Van Houwelingen, J.C. Ridge estimators in logistic regression. *J. R. Stat. Soc. Ser. C Appl. Stat.* **1992**, *41*, 191–201. [CrossRef]
48. Safavian, S.R.; Landgrebe, D. A survey of decision tree classifier methodology. *IEEE Trans. Syst. Man Cybern.* **1991**, *21*, 660–674. [CrossRef]
49. Saunders, C.; Stitson, M.O.; Weston, J.; Holloway, R.; Bottou, L.; Scholkopf, B.; Smola, A. Support Vector Machine. *Comput. Sci.* **2002**, *1*, 1–28.
50. Haykin, S.; Network, N. A comprehensive foundation. *Neural Netw.* **2004**, *2*, 41.
51. Xiao, J.; Xie, L.; He, C.; Jiang, X. Dynamic classifier ensemble model for customer classification with imbalanced class distribution. *Expert Syst. Appl.* **2012**, *39*, 3668–3675. [CrossRef]
52. Del Río, S.; López, V.; Benítez, J.M.; Herrera, F. On the use of MapReduce for imbalanced big data using Random Forest. *Inf. Sci.* **2014**, *285*, 112–137. [CrossRef]
53. Jureczko, M.; Madeyski, L. Towards identifying software project clusters with regard to defect prediction. In Proceedings of the 6th International Conference on Predictive Models in Software Engineering, Timisoara, Romania, 12–13 September 2010; pp. 1–10.
54. Menzies, T.; Di Stefano, J.S. How good is your blind spot sampling policy. In Proceedings of the Eighth IEEE International Symposium on High Assurance Systems Engineering, Tampa, FL, USA, 25–26 March 2004; pp. 129–138.

55. Zimmermann, T.; Premraj, R.; Zeller, A. Predicting defects for eclipse. In Proceedings of the Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007), Minneapolis, MN, USA, 20–26 May 2007; p. 9.
56. Brodersen, K.H.; Ong, C.S.; Stephan, K.E.; Buhmann, J.M. The balanced accuracy and its posterior distribution. In Proceedings of the 2010 20th International Conference on Pattern Recognition, Istanbul, Turkey, 23–26 August 2010; pp. 3121–3124.
57. Buitinck, L.; Louppe, G.; Blondel, M.; Pedregosa, F.; Mueller, A.; Grisel, O.; Niculae, V.; Prettenhofer, P.; Gramfort, A.; Grobler, J.; et al. API design for machine learning software: Experiences from the scikit-learn project. *arXiv* **2013**, arXiv:1309.0238.
58. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
59. Japkowicz, N.; Shah, M. *Evaluating Learning Algorithms: A Classification Perspective*; Cambridge University Press: Cambridge, UK, 2011.
60. Stefanowski, J.; Wilk, S. Selective pre-processing of imbalanced data for improving classification performance. In Proceedings of the International Conference on Data Warehousing and Knowledge Discovery, Turin, Italy, 1–5 September 2008; pp. 283–292.
61. Kamei, Y.; Monden, A.; Matsumoto, S.; Kakimoto, T.; Matsumoto, K.I. The effects of over and under sampling on fault-prone module detection. In Proceedings of the First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007), Madrid, Spain, 20–21 September 2007; pp. 196–204.
62. Lemaître, G.; Nogueira, F.; Aridas, C.K. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *J. Mach. Learn. Res.* **2017**, *18*, 559–563.
63. Yao, J.; Shepperd, M. Assessing software defection prediction performance: Why using the Matthews correlation coefficient matters. In Proceedings of the Evaluation and Assessment in Software Engineering, Trondheim, Norway, 15–17 April 2020; ACM: New York, NY, USA, 2020; pp. 120–129.
64. Janczarek, P.; Sosnowski, J. Investigating software testing and maintenance reports: Case study. *Inf. Softw. Technol.* **2015**, *58*, 272–288. [CrossRef]
65. Korpalski, M.; Sosnowski, J. Correlating software metrics with software defects. In Proceedings of the Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2018, Wilga, Poland, 1 October 2018; International Society for Optics and Photonics: Bellingham, WA, USA, 2018; Volume 10808, p. 108081P.
66. Pandey, S.K.; Mishra, R.B.; Tripathi, A.K. BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques. *Expert Syst. Appl.* **2020**, *144*, 113085. [CrossRef]
67. Hryszko, J.; Madeyski, L. Assessment of the software defect prediction cost effectiveness in an industrial project. In *Software Engineering: Challenges and Solutions*; Springer: New York, NY, USA, 2017; pp. 77–90.