IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 7, 2019, accepted July 30, 2019, date of publication August 9, 2019, date of current version August 22, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2934128

Ensemble MultiBoost Based on RIPPER Classifier for Prediction of Imbalanced Software Defect Data

HAITAO HE[1], XU ZHANG[1], QIAN WANG[1], JIADONG REN[1], JIAXIN LIU[1], XIAOLIN ZHAO[2], AND YONGQIANG CHENG[3]

[1]Computer Virtual Technology and System Integration Laboratory of Hebei Province, College of Information Science and Engineering, Yanshan University, Qinhuangdao 066000, China
[2]Beijing Key Laboratory of Software Security Engineering Technology, School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China
[3]Department of Computer Science, University of Hull, Hull HU6 7RX, U.K.

Corresponding author: Qian Wang (wangqianysu@163.com)

This work was supported in part by the National Key Research and Development Program of China under Grant 2016YFB0800700, and in part by the National Natural Science Foundation of China under Grant 61472341, Grant 61772449, Grant 61572420, Grant 61807028, and Grant 61802332.

ABSTRACT Identifying defective software entities is essential to ensure software quality during software development. However, the high dimensionality and class distribution imbalance of software defect data seriously affect software defect prediction performance. In order to solve this problem, this paper proposes an **E**nsemble **M**ultiBoost based on **R**IPPER classifier for prediction of imbalanced **S**oftware **D**efect data, called **EMR_SD**. Firstly, the algorithm uses principal component analysis (PCA) method to find out the most effective features from the original features of the data set, so as to achieve the purpose of dimensionality reduction and redundancy removal. Furthermore, the combined sampling method of adaptive synthetic sampling (ADASYN) and random sampling without replacement is performed to solve the problem of data class imbalance. This classifier establishes association rules based on attributes and classes, using MultiBoost to reduce deviation and variance, so as to achieve the purpose of reducing classification error. The proposed prediction model is evaluated experimentally on the NASA MDP public datasets and compared with existing similar algorithms. The results show that EMR_SD algorithm is superior to DNC, CEL and other defect prediction techniques in most evaluation indicators, which proves the effectiveness of the algorithm.

INDEX TERMS Software defect prediction, class imbalance, combined sampling, rule learning, MultiBoost.

I. INTRODUCTION

Software quality [1] is considered to be extremely important in the field of software engineering. With the expansion of software scale, software defect repair is often time-consuming and laborious, and it accounts for a large proportion of maintenance costs. Software test is to detect as many potential defects as possible before software release. Therefore, accurate and efficient prediction of software defects is of great significance for improving software quality and reliability. Software defect prediction can be seen as a 2-classification problem, that is, software modules can be divided into defective modules and non-defective modules based on historical data. Historical data can reflect attributes such as software module complexity, operands, and operators.

In the actual software defect prediction, there are 2 main challenges, namely the high dimensionality and the class distribution imbalance of the software defect data. However, some of these features may be more relevant to classes (defects or no-defects), and some may be redundant or irrelevant, which sometimes obscure the real important features for classification, and over-dimensionality leads to dimensionality disaster, so high correlation features need to be selected. Therefore, the introduction of feature selection and feature extraction in software defect prediction can not only improve the efficiency, but also improve the accuracy [2]. At the same time, the proportion of normal module is far more than the defective module, and the hidden characteristics of defect module make it more difficult to be tested out, which leads to the class distribution imbalance The non-defective module is considered as the majority class and the defective module is the minority class. Highly imbalanced

The associate editor coordinating the review of this article and approving it for publication was Xiaobing Sun.

data sets reduce the ability of machine learning algorithms to predict minority class [3]. At present, there are 2 common methods to solve the problem of class distribution imbalance: at the data view, the application of sampling technology to the data set is mainly to balance the class distribution by reducing most samples (i.e. under-sampling) or increasing a few samples (i.e. oversampling), this method has been widely studied [4]–[6]. At the algorithm view, the problem of class imbalance is solved by directly modifying its training mechanism, with the aim of improving the accuracy of minority groups. The traditional cost learning algorithm considers the overall misclassification cost, while the cost sensitive learning algorithm pays more attention to the misclassification cost of minority class samples, and believes that minority class samples have larger misclassification cost. In addition, the voting mechanism in ensemble learning reduces the residual effect caused by feature independence and redundancy.

In this paper, considering both data and algorithm views, First of all, data are processed by the PCA feature processing method and the combined sampling method of ADASYN and the random sampling without replacement, so as to solve software defect data redundancy and class distribution imbalance, and then, the rule-based RIPPER algorithm is used as the base classifier of MultiBoost ensemble learning, and the software defect prediction model is constructed to improve the prediction performance and efficiency.

The rest of this paper is organized as follows: Section 2 summarizes the relevant work. Section 3 describes the proposed learning model of software defect prediction. Section 4 presents the performance of EMR_SD algorithm. Section 5 concludes the paper and future work with concluding remarks.

II. RELATED WORK

Software defect prediction technology is to design software metrics related to software defects by analyzing software code, software development process, etc., and then establish the relationship between software metrics and software defects by using historical defect data. Many technologies based on machine learning have been used to predict software defects, including artificial neural network [7], bayesian network [8], SVM [9], dictionary learning [10], association rule [11], naive bayes [12], tree-based methods [13], evolutionary algorithm [14], etc. However, these algorithms ignore the high dimension and class distribution imbalance of the defect data set, which have a great impact on classification performance [15]. For example, for a given data set with less defective modules, you can achieve very high classification accuracy by simply classifying all modules into non-defective class. Obviously, the false alarm of defective modules is ignored, which proves that it is not enough to simply take the accuracy rate as the evaluation standard. And other indicators such as AUC value [16] based on the ROC curve can make further evaluation.

For the high dimensionality, previous studies on software defect prediction have shown [17] that feature selection and feature extraction methods can make great help. For example, Liu Shuyi *et al.* [18] propose a feature selection method based on multi-strategy. Firstly, a variety of feature evaluation algorithms are used to rank the features of defect data. Under different methods, the same features are divided into different grades according to its occurrence frequency, and the highest grade of each feature is chosen as the final rank. This method effectively avoids the loss of potentially effective information by just using a single algorithm for feature evaluation. In order to fully measure the correlation between different features and classes, Qiao Yu *et al.* [19] propose a software defect prediction feature selection algorithm based on similarity measure (SM). This algorithm first updates the feature weights based on the similarity of the different classes of samples. Secondly, the features are sorted in descending order according to the weights, so as to generate a feature sorting list, and all the feature subsets are sequentially selected from the sorted feature list. In order to further improve the performance of feature selection for software defect prediction, Gao *et al.* [20] combine the advantages of different feature selection, proposed a hybrid feature selection algorithm, combining feature sorting and feature subset selection. The results show that this algorithm is superior to other feature subset selection methods in software defect prediction. At the same time, Laradji *et al.* [21] combine feature selection and ensemble learning to improve the accuracy of software defect prediction considering the imbalanced class distribution of the software defect data set.

In order to solve the imbalance problem of the defect data set, at the data view, Chawla *et al.* [22] propose the SMOTE algorithm, which increases the number of samples by interpolating between minority class samples that are close together. However, due to the large number of large class of samples in the original data set, in order to achieve a relative balance of the overall data sample distribution, a large number of minority class of samples need to be synthesized, resulting in a large increase in the number of data sets compared with previous one, which will reduce the prediction efficiency to a certain extent. Following the SMOTE algorithm, Wang Shuo *et al.* [23] use re-sampling, threshold shifting and other methods to further deal with the data distribution and solve the class imbalance of software defect data. In addition, Menzies T and Turhan B [24] achieve a balance by taking a SMOTE oversampling of the minority class and performing a simple random under-sampling (RUS) on the majority class. Existing studies have shown that [25], RUS and RUS-bal (under-sampling and over-sampling) are superior to SMOTE and random oversampling, and have achieved good classification results. At the algorithm view, one class learning [26], cost sensitive learning [27], ensemble learning and improved SVM are adopted to improve the classification performance of the minority class. Ensemble learning can combine the advantages of individual single classifiers to improve overall classification performance. For example, Seliya *et al.* [28] propose an early extension of bagging ensemble learning algorithm, known as rough balanced

bagging (RBBAG) algorithm. They also make empirical evaluation of two basic classifiers, naive bayes and C4.5 decision tree, the experimental results show that the performance based on naive bayes is better than those based on C4.5 decision tree. In order to further improve the accuracy of the model, Chawla N V *et al.* [29] not only study the imbalance in the data view, but also process it from the algorithm view, and integrate the SMOTE oversampling method and boosting technology. Sampling techniques can balance the distribution of data, but there still exist the incorrect classification of defective modules. Therefore, Khoshgoftaar *et al.* [30] propose a cost sensitive enhancement technique combining enhanced ensemble learning algorithm and cost sensitive features, with a focus on the cost of misclassification. At present, there are some rule-based classification techniques, such as CBA2 used by Ma *et al.* [16] and DPRAR algorithm proposed by Gabriela Czibula [11], which are used to predict the defects of software modules and have achieved good results in most evaluation indicators. In summary, feature selection or feature extraction, combined sampling and ensemble learning have their own advantages in the prediction of software defects. And combining the three is expected to achieve better classification performance.

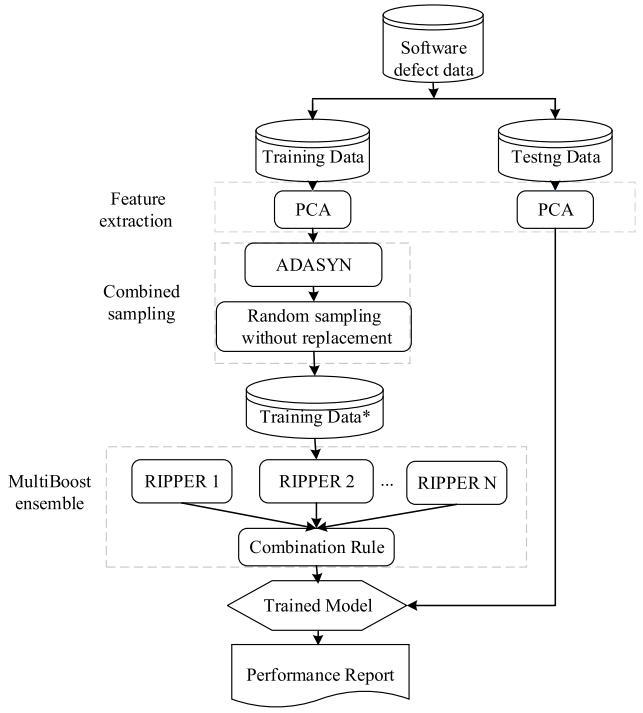

FIGURE 1. Software defect prediction model of EMR_SD.

III. MODEL OF OUR APPROACH EMR_SD

EMR_SD is a software defect imbalance data prediction model based on RIPPER classifier for MultiBoost ensemble. The main framework is shown in Figure 1. When making software defect prediction, first consider data pre-processing phase for the input training sample, this is by PCA feature extraction and combined sampling. And then, selecting a base classifier for the preprocessed data, and MultiBoost is used to integrate multiple base classifiers to obtain a higher accuracy of diagnosis result.

A. DATA PRE-PROCESSING

In data pre-processing, there are always missing values in the data sets, the average calculation mode is used for filling the missing ones to reduce the influence of the prediction effect. Then, statistical analysis is carried out on the defective modules so as to select the appropriate features to process the data, which retains the features with strong dependence the to the target output class, and achieves the purpose of dimensionality reduction and denoising. In this paper, principal component analysis (PCA) is used for feature extraction. Each module in the software defect data set is marked as defective or non-defective, and the number of defects is far less than that of no-defects, which shows a high degree of imbalance. Therefore, a new combined sampling method is proposed to balance the training data after dimension reduction, that is, the combination of ADASYN and random sampling without replacement. This combined sampling method not only provides a balanced data distribution, but also enables the learning algorithm to focus on samples that are difficult to learn in minority class.

1) FEATURE EXTRACTION

In software defect data, excessive dimensionality may degrade the performance of the algorithm and reduce the effectiveness of the model. Removing the irrelevant and redundant features of the software defect data to achieve the purpose of data dimensionality reduction is an effective way to improve the performance of the learning algorithm, and is also key to the training model.

EMR_SD uses the principal component analysis (PCA) feature extraction method [31], which achieves dimensionality reduction. Suppose a data set consists of tuples or vectors with n features, the PCA searches for the d-dimensional ($d < n$) orthogonal vectors which represent the data best. PCA makes the original data projected from an $n-$dimensional space into a small d-dimensional and it can always be able to reveal previously undetected connections. The PCA algorithm is described as follows:

Algorithm 1 PCA Feature Extraction Algorithm

Input: Data Sets $D = \{x_1, x_2, \ldots, x_n\}$.
Output: Projection Matrix $w^* = (\omega_1, \omega_2, \ldots . \omega_d)$.

1. Centralize all samples: $x_i \leftarrow x_i - \frac{1}{n}\sum_{i=1}^{n} x_i$;
2. Calculate the covariance matrix of the sample: XX^T;
3. Decompose the covariance matrix XX^T into eigenvalues, and sort the eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$;
4. Set value d to make its contribution rate more than μ: $\frac{\sum_{i=1}^{d} \lambda_i}{\sum_{i=1}^{n} \lambda_i} \geq \mu$;
5. Take the eigenvector corresponding to the largest d eigenvalues $\omega_1, \omega_2, \ldots, \omega_d$.

The PCA algorithm can perform simple vector subtraction and linear mapping by preserving the vector of W* and the mean vector of samples, which projects the sample from the original high-dimensional space to the low-dimensional space. The eigenvector corresponding to the smallest eigenvalue is often correlated with noise. When the data are affected by noise, the PCA algorithm discards the eigenvector corresponding to the smallest $(n-d)$ eigenvalues. Therefore, achieve denoising effect and guarantee the quality of the sampled data.

2) COMBINED SAMPLING

Class imbalance distribution, that is, the number of samples of one class in the data set is significantly less than the number of samples of other classes, for a 2-classification problem, the classes are named as the majority class and the minority class. Class imbalance will decline classification performance, which makes it difficult to meet the accuracy requirements in the classification process. Therefore, considering combined sampling method of adaptive synthetic sampling (ADASYN) and random sampling without replacement to solve the problem of data class imbalance.

ADASYN is an adaptive synthetic sampling method proposed in Literature [32]. This method adaptively synthesizes the minority samples according to its distribution in the training data set, which focuses on more of the samples difficult to learn and less of the samples easy to learn. The key of the ADASYN method is to find a probability distribution, it will be used as the criterion for the number of samples to be synthesized for each minority samples, and finally obtain the new sample set. The specific algorithm is as shown in Algorithm 2.

By using ADASYN sampling, the synthesized samples of the minority class are added to the original sample set to achieve a balance of the data set. Then random sampling without replacement method can be further used to process the data to extract a smaller data set with a balanced guarantee. Therefore, the data quality is improved, and a data set with reasonable scale and distribution is ready for the classification and the prediction performance of EMR_SD is improved.

Algorithm 2 ADASYN Algorithm

Input: Data set D_1, which includes m samples $\{x_i, y_i\}$, $i = 1, 2, \ldots, m$, x_i is a sample of n-dimensional space, $y_i \in \{0, 1\}$ is label, $y_i = 0$ represents minority class and $y_i = 1$ is majority class. Where m_0 and m_1 represent the number of samples of few and many classes respectively.

Output: New composite samples.

1. Calculate the class imbalance $l = m_0/m_1$, where $l \in (0, 1]$;
2. Calculate the total amount of samples to be synthesized: $G = (m_1 - m_0) \times \beta$, where $\beta \in [0, 1]$ is a coefficient. G is the number of samples of the minority class to be synthesized;
3. For each of the minority samples x_i, find the K-nearest neighbor points and calculate: $\Gamma_i = \Delta_i/K$, $i = 1, 2, \ldots, m$,where Δ_i is the number of multi-class samples in the K-nearest neighbor, therefore, $\Gamma_i \in (0, 1]$;
4. According to $\hat{\Gamma}_i = \Gamma_i / \sum_{i=1}^{m_0} \Gamma_i$, regularization Γ_i, $\hat{\Gamma}_i$ is the probability distribution, and ($\sum \hat{\Gamma}_i = 1$);
5. Calculate the number of samples that need to be synthesized for each of the minority class of samples x_i: $g_i = \hat{\Gamma}_i \times G$ where G is the total number of samples that need to be synthesized;
6. For each minority class sample x_i, the following samples g_i are synthesized;
7. for $j = 1 : g_i$
8. Select a minority class of samples x_j randomly from the K nearest neighbors of x_i;
9. According to neighbors of $S_z = x_i + (x_j - x_i) \times \lambda$; synthesis sample neighbors of S_z, $\lambda \in [0, 1]$ is a random number;
10. end

B. BASE CLASSIFIER OF RIPPER

EMR_SD applies rule learning RIPPER as the base classifier to classify software defect data sets. There are two phases, the first is to summarize the correlation between features and classes by the induction and learning of training samples, so as to form if-then rules. The second phase is to use the formed rules to perform matching detection on the unknown sample to achieve the purpose of classification. The general rule generation is as Format (1).

$$\bigwedge_{i=1}^{n} (x_i = a_i) \rightarrow y_i \tag{1}$$

The left of the arrow is called the rule antecedent, x_i represents feature, the right of the arrow is called the rule postscript, y_i is a conclusion, which represents the class.

For the binary classification problem in software defect prediction, RIPPER algorithm firstly carries out ascending ranking according to the frequency of class occurrence. Because there are relatively few non-defective modules in the software defect data set, defect class is ahead of the non-defective modules. In this way, the RIPPER algorithm can first deal with the least frequent class, and finally deal with the most frequent class. It is the particularity of the rule of RIPPER that makes a good performance in handing unbalanced data. The flow chart of RIPPER algorithm is shown in Figure 2.

RIPPER algorithm mainly includes rule generation and rule pruning. When generating rules, firstly empty the rule set and then add antecedents to the rule set until the rule set can be extended to cover the entire data set. During rule pruning, rules and antecedents are continually deleted. Finally, use Format (2) to determine whether the most streamlined rule

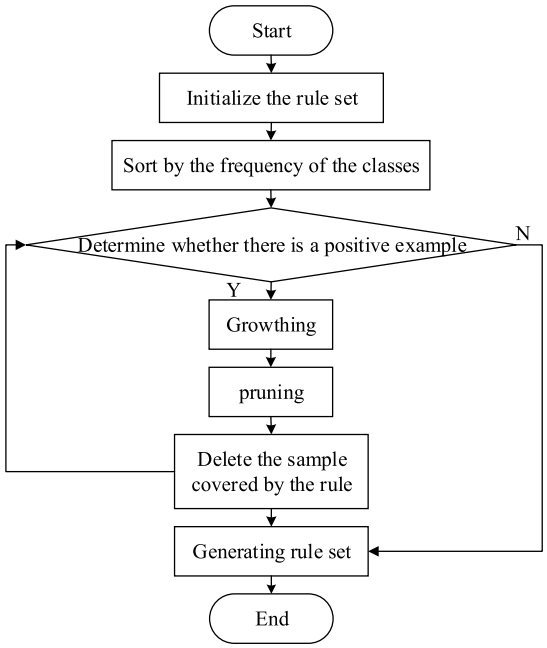

FIGURE 2. RIPPER algorithm flow chart.

is reached.

$$C = \frac{h_k - h_p}{h_k + h_p} \tag{2}$$

h_k is the number of samples covered by the rule, h_p is the number of samples not covered by the rule. Stop pruning when function C no longer grows.

C. MULTIBOOST ENSEMBLE CLASSIFICATION

When constructing the prediction model, the smaller the variance, deviation and mean square error are, the more stable the model is. Therefore, EMR_SD considers the MultiBoost ensemble learning method, which combines wagging (a technique for variance reduction) and AdaBoost (a technique for both bias and variance reduction) [33]. EMR_SD achieves a good perform by wagging a set of base classifier groups, and the base classifiers use the AdaBoost strategy inside each group.

Let T be the number of base classifiers combined using the MultiBoost method, k is the number of base classifier groups and I_i is an iterative termination flag variable. I_i can be obtained as Formula (3).

$$\begin{cases} k = \lfloor \sqrt{T} \rfloor \\ I_i = \lceil i \times T/k \rceil, & i = 1, \ldots, k-1 \\ I_i = T, & i = k, \ldots, \infty \end{cases} \tag{3}$$

As scholars have proved that the base classifier trained by each iteration improves the performance of the whole combined model less than its previous iteration [34]. The MultiBoost method improves AdaBoost, AdaBoost always makes T iterations according to the number of base classifiers, and MultiBoost uses the iterative termination flag variable for iteration rounds. The MultiBoost algorithm is as follows

Algorithm 3 MultiBoost Ensemble Algorithm

Input: Data set D: data set of m training samples;
Cycle number T: number of base classifiers;
Integer I_i: base classifier group iteration terminates marker variable.

Output: Composite model

$$M^*(x) = \arg\max_{y \in Y} \sum_{t: M_t(x) = y} \log \frac{1}{\beta_t}$$

1. First, the weights $\omega(x_j), j = 1, 2, \ldots m$ of each sample assigned to be 1/m in the data set $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$, where $y_j \in Y$;
2. Set $i = 1$; //The initial value of i is 1
3. For $t = 1$ to T {//Set the maximum number of cycles to T
4. calculate the value of I_i according to (3).
5. If $I_k = t$ then rest D to random weights drawn from the continuous Poisson distribution, and each weight value needs to be normalized;
6. $i++$; //After each base classifier group terminates, wagging iteration is conducted once.
7. $M_t = C_j(D)$; //Use the classification model to get each base classifier.
8. The error of each base classifier is calculated $\varepsilon_t = \frac{\sum_{x_j \in D: M_t(x_j) \neq y_j} w(x_j)}{m}$;
9. If $\varepsilon_t > 0.5$ then rest D to random weights drawn from the continuous Poisson distribution;
10. $i++$;
11. go to Step 7;
12. else If $\varepsilon_t = 0$ then
13. set $\beta_t = 10^{-10}$;
14. reset D to random weights drawn from the continuous Poisson distribution;
15. $i++$;
16. else
17. $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$;
18. For each $x_j \in D$
19. divide $\omega(x_j)$ by $2\,\varepsilon_t$ if $M_t(x_j) \neq y_j$ and $2\,(1-\varepsilon_t)$ otherwise;
20. If $\omega(x_j) < 10^{-8}$, set $\omega(x_j)$ to 10^{-8}.
21.}

IV. EXPERIMENTS AND RESULTS

A. DATA SETS

The data set used in this experiment is the MDP dataset from NASA, which is widely used in software defect prediction research [35]. This data set consists of 13 data sets, each of which is collected from an actual software system project and consists by the class label and static code features based on Loc, McCabe, Halstead, and so on. Loc line measurement is a measure of program complexity based on the number of lines

TABLE 1. NASA MDP dataset.

Data set	Defect	No-defect	Features	Proportion of defective modules
CM1	48	457	40	9.50%
MC2	52	109	40	32.30%
PC5	516	16670	39	3.00%
PC1	76	1031	40	6.87%
MW1	31	372	40	7.69%
PC2	23	5566	40	0.41%
JM1	2102	8776	21	19.32%
KC1	325	1782	21	15.42%
KC3	43	415	40	9.40%
KC4	61	64	40	48.80%
PC3	160	1403	40	10.24%
PC4	178	1280	40	12.21%
MC1	68	9398	39	0.72%

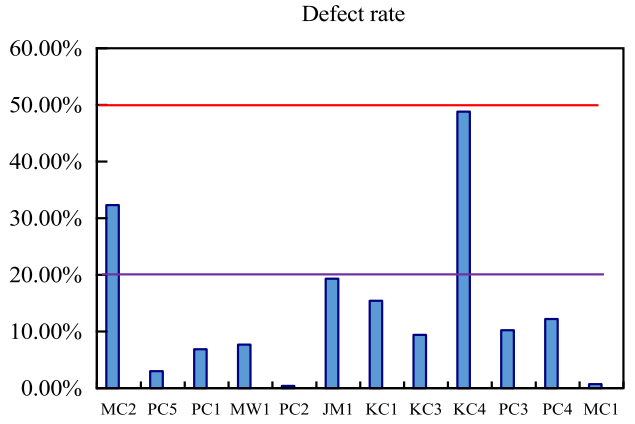

FIGURE 3. Defect rate.

of source code. McCabe loop complexity measurement is a method based on the complexity of program control flow. It is believed that the more loops, the more complex the program is. The Halstead science metric calculates the complexity of a program based on the number of operators and operands in the executable lines of code in the program. Table 1 shows the basic information of all NASA MDP data sets. From the Table 1, it can be seen that the proportion of defective modules is extremely low, which shows the imbalance of class distribution of data. The distribution of the imbalance rate of the data set is shown in Figure 3. As shown in Figure 3, the proportion of defective modules in all data sets is less than 50%, and most of them are concentrated below 20%, which again presents a serious imbalance in the NASA MDP data set. This paper selects some representative data sets for experiments. In order to prove the validity of the constructed prediction model, the proposed algorithm is compared with other similar algorithms.

B. EVALUATION MEASUREMENTS

The software defect prediction data has a class imbalance distribution, and when the classifier performance is evaluated only from the overall perspective, the classification effect of majority class will overshadow the minority class. For the evaluation of the classifier performance, it is necessary to take into account the defect detection rate while paying attention to the overall performance. Therefore, a variety of performance indicators are usually adopted to evaluate the prediction model, such as Accuracy, Recall, F-measure and AUC. These indicators are based on the definition of confusion matrix, as shown in Table 2.

TABLE 2. Confusion matrix.

Actual	Predicted	
	Positive	Negative
Positive	TP (true positive)	FN (false negative)
Negative	FP (false positive)	TN (true negative)

Accuracy is represented by the ratio of the number of correctly predicted modules to the total number of modules. Recall is the ratio of the number of defective modules correctly predicted to the actual number of defective modules. F-measure is the combination of Recall and Precision for evaluation. Precision is the ratio of the number of modules correctly predicted to be defective modules to the total number of defective modules. AUC (Area Under the Curve) is defined as the ROC Curve (Receiver Operating Characteristic) of the Area. AUC means that given a defective sample and a non-defective sample randomly, the probability value that the classifier outputs the defective sample as defect is greater than the probability value that the classifier outputs the non-defective sample as defect. AUC value can effectively evaluate the classification effect of 2- classification problem. The larger the value, the better the performance of the software defect prediction model. The calculation formula is as follows

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

$$Recall = \frac{TP}{(TP + FN)} \tag{5}$$

$$Precision = \frac{TP}{(TP + FP)} \tag{6}$$

$$F - measure = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{7}$$

The ROC curve is shown in Figure 4.

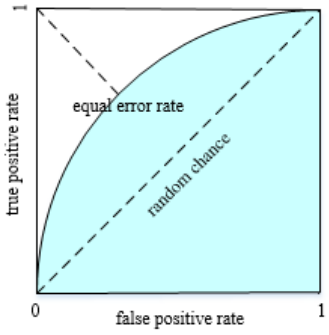

FIGURE 4. ROC curve.

For a specific prediction model and training data set, the prediction result corresponds to a point on the ROC curve, AUC, varying in [0, 1], measures the area under the ROC

TABLE 3. Experimental results.

Data	Acc	Re	PF	Pre	F-mea	AUC	Bal
CM1	90.99%	0.95	0.13	0.88	0.91	0.97	0.90
MC2	82.05%	0.86	0.22	0.79	0.82	0.89	0.82
PC1	95.74%	0.97	0.06	0.94	0.95	0.98	0.95
MW1	93.72%	0.97	0.10	0.91	0.93	0.97	0.93
KC1	84.35%	0.88	0.19	0.82	0.85	0.91	0.84
KC3	92.10%	0.95	0.11	0.89	0.92	0.97	0.91
PC3	91.36%	0.96	0.13	0.87	0.91	0.97	0.90
PC4	94.48%	0.90	0.02	0.98	0.94	0.98	0.93

curve which describes the trade-off between Recall (true positive rate) and PF (false positive rate). The larger AUC, the better the performance of the classifier is [36]. A new evaluation indicator Balance can be introduced by calculating the Euclidean distance from the (PF, Recall) point to the (0,1) point. The definition of Balance is as follows

$$Balance = 1 - \frac{\sqrt{(0 - PF)^2 + (1 - Recall)^2}}{\sqrt{2}} \tag{8}$$

where PF represents the proportion of defect modules misclassified in non-defective class.

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

C. EXPERIMENTAL DESIGN AND ANALYSIS OF RESULT

In order to verify the prediction performance of the model, the experiment is based on the WEKA and Matlab platform, and the final results are verified by the 10-fold cross validation method. The sampling rate of the random sampling without replacement and the number of iterations of MultiBoost ensemble learning are set to empirically experimental values of 0.9 and 20. The sampling rate is originally set to [0.7, 1.0], the step size is 0.1, and the number of iterations is set to [10], [30], the step size is 10. The experimental results achieve the best when the sampling rate is 0.9 and the number of iterations is 20.

The experimental results are shown in Table 3.

Table 3 summarizes the experimental results of EMR_SD. The corresponding values of the evaluation indicators Accuracy, Recall, Precision, F-measure, and Balance are obtained based on the confusion matrix.

Because Accuracy is an indicator of overall evaluation, it represents the proportion of the number of modules correctly predicted in the total number of modules, and the number of correctly predicted modules includes the correct prediction of the defective module and the total number of correct predictions of the non-defective module. In the case of imbalanced software defect data, due to the relatively small number of defective modules, there may be case where all defective modules are misclassified and the accuracy is still high. Therefore, simply using Accuracy as an indicator for evaluating the performance of a model is not comprehensive. Further combining F-measure and AUC value Table 3 shows the prediction performance on F-measure and AUC. Except that the F-measure and the AUC on MC2 data set are slightly lower than other data sets, the remaining values are all above 0.9, indicating that the preprocessing method proposed in this paper can conduct more effective data processing for imbalanced data and that the integrated model can better adapt to the software defect data set. For each data set, When the point (PF, Recall) is closer to the (0, 1) point, the performance of the prediction model is better, according to which, the indicator Balance can be calculated, and it is frequently used by software engineers in practice [37].

Firstly, in order to explain how EMR_SD improves the prediction performance, we carry out the experiments stage by stage. The experimental results are shown in Table 4 Secondly, in comparison with baseline methods, in order to make the comparison fair, preprocessing should also be performed on baseline methods. In addition, three classical algorithms widely accepted by the public in the field of software defect prediction, namely naive bayes, J48 and SVM are selected for comparative experiments to prove the effectiveness of EMR_SD. NB is a simple probabilistic classifier, which

TABLE 4. Results of each stage.

Measure	Stage	CM1	MC2	PC1	MW1	KC1	KC3	PC3	PC4	Average
Accuracy (%)	Multboost	88.91	72.05	93.50	92.31	85.33	89.08	89.70	90.88	87.72
	PCA+Multboost	90.10	75.78	93.22	92.56	85.38	90.83	90.02	91.36	88.66
	PCA+Combined sampling+ Multboost	90.99	82.05	95.74	93.72	84.35	92.10	91.36	94.48	**90.99**
F-measure	Multboost	0.10	0.53	0.38	0.28	0.39	0.24	0.32	0.56	0.35
	PCA+Multboost	0.17	0.54	0.51	0.54	0.40	0.30	0.36	0.54	0.42
	PCA+Combined sampling+ Multboost	0.91	0.82	0.95	0.93	0.85	0.92	0.91	0.94	**0.90**
AUC	Multboost	0.76	0.73	0.84	0.79	0.79	0.76	0.84	0.93	0.81
	PCA+Multboost	0.77	0.78	0.85	0.76	0.79	0.81	0.82	0.90	0.81
	PCA+Combined sampling+ Multboost	0.97	0.89	0.98	0.97	0.91	0.97	0.97	0.98	**0.96**
Balance	Multboost	0.34	0.61	0.50	0.43	0.50	0.43	0.46	0.62	0.49
	PCA+Multboost	0.36	0.60	0.52	0.46	0.51	0.44	0.49	0.59	0.50
	PCA+Combined sampling+ Multboost	0.90	0.82	0.95	0.93	0.84	0.91	0.90	0.93	**0.90**

TABLE 5. Results of comparison with baseline algorithms.

Measure	Algorithm	CM1	MC2	PC1	MW1	KC1	KC3	PC3	PC4	Average
Accuracy (%)	data processing+naive bayes	73.24	69.74	6.0.54	73.24	62.77	70.15	70.65	76.30	70.87
	data processing+J48	86.74	71.94	89.87	89.39	81.57	89.02	87.01	91.06	85.83
	data processing+SVM	76.52	62.05	68.03	69.96	88.90	74.83	74.65	76.04	73.87
	EMR_SD	90.99	82.05	95.74	93.72	84.35	92.10	91.36	94.48	**90.99**
F-measure	data processing+naive bayes	0.72	0.65	0.44	0.77	0.67	0.62	0.73	0.79	0.67
	data processing+J48	0.87	0.72	0.90	0.99	0.83	0.89	0.88	0.91	0.87
	data processing+SVM	0.79	0.55	0.61	0.67	0.89	0.76	0.74	0.78	0.72
	EMR_SD	0.95	0.86	0.97	0.97	0.88	0.95	0.96	0.90	**0.95**
AUC	data processing+naive bayes	0.80	0.62	0.76	0.81	0.90	0.80	0.77	0.88	0.79
	data processing+J48	0.88	0.76	0.91	0.92	0.85	0.92	0.88	0.92	0.88
	data processing+SVM	0.77	0.70	0.68	0.70	0.89	0.75	0.75	0.76	0.85
	EMR_SD	0.97	0.89	0.98	0.97	0.91	0.97	0.97	0.98	**0.96**
Balance	data processing+naive bayes	0.73	0.64	0.50	0.72	0.56	0.67	0.68	0.75	0.66
	data processing+J48	0.86	0.72	0.89	0.89	0.80	0.88	0.85	0.91	0.85
	data processing+SVM	0.74	0.59	0.63	0.69	0.89	0.74	0.74	0.74	0.72
	EMR_SD	0.90	0.82	0.95	0.93	0.84	0.91	0.90	0.93	**0.90**

TABLE 6. Comparisons of different algorithms on Accuracy, F-measure, AUC and Balance.

Measure	Algorithm	CM1	MC2	PC1	MW1	KC1	KC3	PC3	PC4
Accuracy (%)	DNC	75.04	62.86	78.30	82.22	74.61	76.14	78.35	84.22
	CEL	79.58	59.89	83.48	82.22	73.83	77.05	81.26	87.67
	AKPCC	76.55	65.38	86.45	80.39	71.78	76.24	80.38	83.91
	MEKL	68.03	74.76	69.00	77.41	65.48	66.63	69.13	73.96
	DPRAR	87.16	**89.60**	95.60	**94.10**	82.30	83.00	**96.70**	**96.10**
	CBA2	80.36	69.81	91.78	91.04	83.71	91.91	86.48	83.96
	ACAR	73.68	76.36	68.17	79.56	72.77	76.72	69.97	76.21
	EMR_SD	**90.99**	82.05	**95.74**	93.72	**84.35**	**92.10**	91.36	94.48
F-measure	DNC	0.32	0.48	0.38	0.31	0.47	0.33	0.40	0.16
	CEL	0.27	0.49	0.32	0.27	0.36	0.33	0.36	0.48
	AKPCC	0.21	0.49	0.35	0.33	0.35	0.29	0.39	0.43
	MEKL	0.40	0.63	0.50	0.49	0.50	0.44	0.46	0.55
	DPRAR	0.65	**0.84**	0.78	0.76	0.71	0.66	0.86	0.85
	CBA2	0.16	0.41	0.43	0.46	0.47	0.41	0.29	0.52
	ACAR	0.39	0.63	0.27	0.43	0.48	0.45	0.36	0.52
	EMR_SD	**0.91**	0.82	**0.95**	**0.93**	**0.85**	**0.92**	**0.91**	**0.94**
AUC	DNC	0.79	0.65	0.87	0.71	0.82	0.80	0.82	0.92
	CEL	0.70	**0.91**	0.83	0.71	0.81	0.82	0.80	0.89
	AKPCC	0.71	0.86	0.78	0.69	0.77	0.78	0.79	0.91
	MEKL	0.71	0.76	0.71	0.72	0.70	0.66	0.64	0.77
	DPRAR	0.90	0.87	0.92	0.92	0.82	0.85	0.92	0.90
	CBA2	0.60	0.67	0.83	0.86	0.84	0.70	0.82	0.89
	ACAR	0.80	0.73	0.74	0.89	0.75	0.87	0.75	0.86
	EMR_SD	**0.97**	0.89	**0.98**	**0.97**	**0.91**	**0.97**	**0.97**	**0.98**
Balance	DNC	0.65	0.57	0.68	0.62	0.73	0.66	0.74	0.85
	CEL	0.54	0.60	0.61	0.46	0.55	0.49	0.57	0.63
	AKPCC	0.49	0.57	0.56	0.52	0.51	0.49	0.52	0.65
	MEKL	0.71	0.75	0.70	0.71	0.68	0.66	0.63	0.76
	DPRAR	**0.90**	**0.84**	0.92	0.91	0.82	0.85	0.89	0.87
	CBA2	0.43	0.52	0.60	0.64	0.60	0.53	0.47	0.73
	ACAR	0.79	0.70	0.73	0.84	0.75	0.82	0.74	0.81
	EMR_SD	**0.90**	0.82	**0.95**	**0.93**	**0.84**	**0.91**	**0.90**	**0.93**

assumes that the features are statistically independent of each other. It can provide good classification results, even though some of the features are inter-related. J48 is a decision tree based classifier. Decision trees use feature values for the classification of instances. Each feature is represented by a node in the decision tree, while the assumption values of each node is represented as branches. The classification of instances is performed by following a path through the tree from the root node to the leaf nodes by checking feature values against rules. The basic idea of SVM is solving sum functions and quadratic programming, and solving the nonlinear separable problem by mapping data to high-order feature space through sum function. The experimental results are shown in Table 5.

At last, this paper compare with other similar algorithms to further verify the effectiveness of EMR_SD, including Dynamic version of Adaboost.NC (DNC) [23], Coding based Ensemble Learning (CEL) [39], Asymmetric Kernel Principal Component Classification (AKPCC) [40], Multi-core kernel ensemble learning (MEKL) [41], Relational association rule mining (DPRAR) [11], Classification based on Association rules (CBA2) [16] Mining based on Atomic Association rules (ACAR) [42]. The performance evaluation of classifier with class imbalance data needs to be analyzed by using multiple evaluation indicators. The results are shown in Table 6.

It can be seen from Table 4, on the whole, the Accuracy, F-measure and Balance with PCA dimension reduction and denoising are improved comparing with those without any preprocessing. In particular, the values of F-measure on data set PC1 and MW1 are significantly improved. And the prediction performance is improved due to the dimension reduction. When the combined sampling method is further used to preprocess the data, the prediction performance improves more, all indicators are above 0.9. This indicates that each stage of EMR_SD has a certain effect on prediction performance.

As can be seen from Table 5, among the three classical algorithms, J48 has the best prediction performance, but still inferior to EMR_SD. Compared with J48 classifier, EMR_SD algorithm improves at least 5.9% in each indicator. This also proved the superiority of MultiBoost classifier.

It can be seen from Table 6 that EMR_SD has a mean value of 0.9 on the F-measure indicator, which is at least 18.42% higher than that of the similar algorithms. Among the similar algorithms, the performance of the DPRAR algorithm on the F-measure indicator is only to EMR_SD. The F-measure value of the CM1 data set is only 0.65, and the EMR_SD achieves the F-measure value of 0.91. The prediction performance of this data set is greatly improved, which makes up the shortcomings of the DPRAR algorithm for the prediction of the CM1 data set. From the comparison of the AUC values, it can be seen that compared with the similar algorithms, the AUC value of EMR_SD is at least 7.87% higher than the similar algorithms, up to 0.96, especially in the PC1 and PC4 data sets. The performance on AUC value is relatively high and reaches 0.98, because the EMR_SD algorithm considers both the correct classification of the non-defective modules and defective modules, and the goal of comprehensively improving the classification performance of the prediction model is achieved. It can be seen that the EMR_SD algorithm in the mean of Balance is higher than other similar algorithms and achieves 0.9. Although the Balance is slightly lower for the individual data set MC2, it still achieves 0.82. It can be clearly seen from Fig.5 that in the column diagram of F-measure, the column corresponding to EMR_SD algorithm is higher than other similar algorithms. From the comparison of the Balance value, it can be seen that the Balance curve of EMR_SD algorithm is located at the top. It can be seen from the radar diagram of AUC that the circle representing EMR_SD is almost located outside all

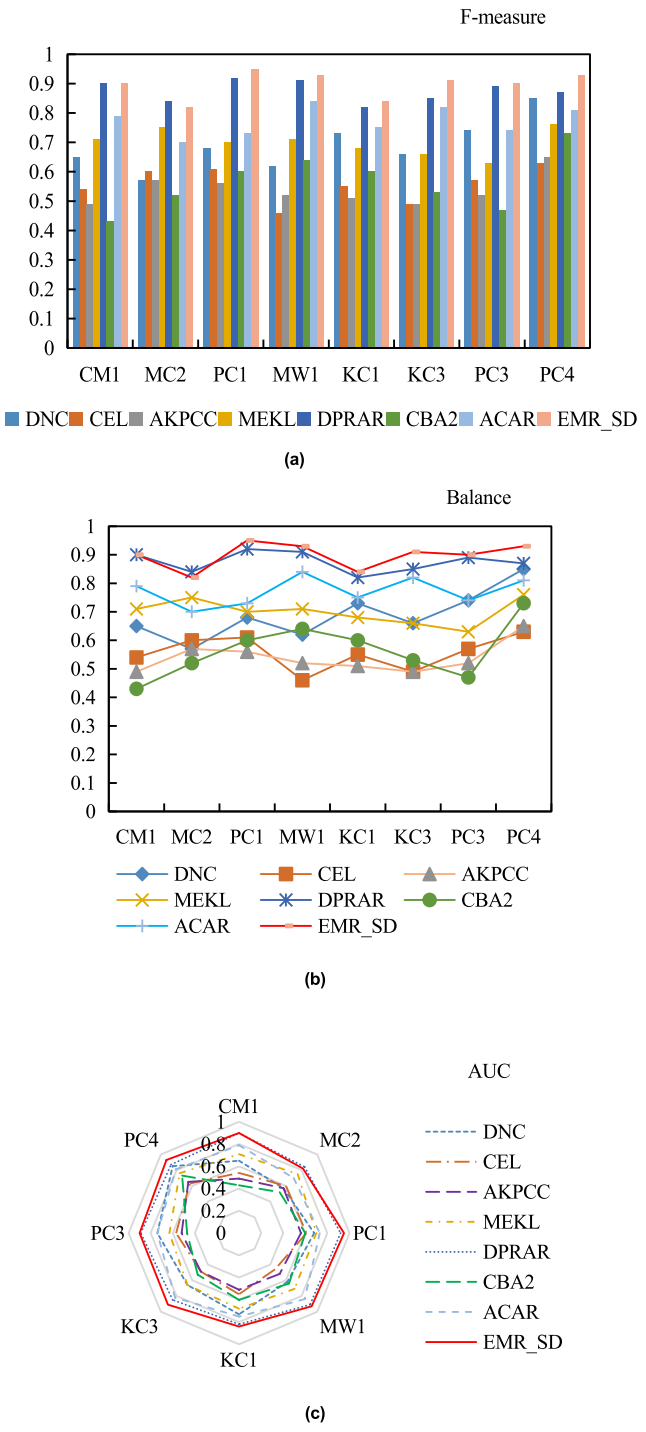

FIGURE 5. Compare other algorithms on F-measure, balance and AUC.

circles, which indicates that EMR_SD algorithm has better prediction performance. However, since this paper adopts the same dimensionality reduction and combined sampling method for all data sets, it does not consider the characteristics of each data set itself, resulting in the prediction performance is not obvious for the MC2 dataset with less data and the KC1 dataset with fewer attributes. Combined with all evaluation indicators, EMR_SD algorithm can achieve better

prediction result in software defect prediction, and has strong theoretical and experimental basis for further research.

V. THREATS OF VALIDATION

In conducting empirical research, it is important to be aware of potential threats to the validity of the results and conclusions. Moreover, these potential threats exist in most studies. The potential threats for this paper include two aspects. First, sampling bias. Since EMR_SD algorithm has randomness to create different instances during the combined sampling, we mitigate this potential bias with 10 runs of the experiment for EMR_SD. Second, threats to construct validity are about whether the performance indicators are comprehensive for evaluation. To minimize the threads, we use multiple evaluation indicators to evaluate the prediction performance, such as Accuracy, F-measure, AUC and Balance, which are widely used in current software defect prediction research.

VI. CONCLUSION

This paper proposes an ensemble classification algorithm, EMR_SD, for software defect prediction. The main challenge affects the overall performance of the predictor is that there are high-dimension and imbalanced class distribution problems in the defect data sets. Therefore, EMR_SD uses PCA method to reduce the dimension of data set and combined sampling method to balance the class distribution. Moreover, EMR_SD uses the MultiBoost ensemble learning method as the classifier, which combines wagging (a technique for variance reduction) and AdaBoost (a technique for both bias and variance reduction), as the smaller the variance, deviation and mean square error are, the more stable the classifier is. To illustrate the prediction performance of the EMR_SD, the experiments are carried out stage by stage, and prove that each stage has contributions to the performance. Then, EMR_SD is contrasted with relative algorithms and performs the best on the indicators of Accuracy, F-measure, AUC, and Balance, which further shows great potential for software defect prediction. Because this paper adopts a unified data processing method for all data sets, and does not consider the unique characteristics of each data set. Therefore, in the future, targeted dimensionality reduction and combined sampling method will be used for different data set.

ACKNOWLEDGMENT

The authors are grateful to valuable comments and suggestions of the reviewers.

REFERENCES

[1] N. J. Pizzi, "A fuzzy classifier approach to estimating software quality," *Inf. Sci.*, vol. 241, pp. 1–11, Aug. 2013.
[2] T. M. Khoshgoftaar, K. Gao, A. Napolitano, and R. Wald, "A comparative study of iterative and non-iterative feature selection techniques for software defect prediction," *Inf. Syst. Frontiers*, vol. 16, no. 5, pp. 801–822, 2014.
[3] Z. Mahmood, D. Bowes, P. C. R. Lane, and T. Hall, "What is the impact of imbalance on software defect prediction performance?" in *Proc. 11th Int. Conf. Predictive Models Data Anal. Softw. Eng.*, Beijing, China, 2015, Art. no. 4.
[4] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A large-scale empirical study of just-in-time quality assurance," *IEEE Trans. Softw. Eng.*, vol. 39, no. 6, pp. 757–773, Jun. 2013.
[5] Y. Kamei, T. Fukushima, S. McIntosh, K. Yamashita, N. Ubayashi, and A. E. Hassan, "Studying just-in-time defect prediction using cross-project models," *Empirical Softw. Eng.*, vol. 21, no. 5, pp. 2072–2106, 2016.
[6] Y. Ma, W. Pan, S. Zhu, H. Yin, and J. Luo, "An improved semi-supervised learning method for software defect prediction," *J. Intell. Fuzzy Syst.*, vol. 27, no. 5, pp. 2473–2480, 2016.
[7] V. Vashisht, M. Lal, and G. S. Sureshchandar, "Framework for software defect prediction using neural networks," *J. Eng. Appl.*, vol. 2015, no. 8, pp. 384–394, 2015.
[8] A. Okutan and O. T. Yıldız, "Software defect prediction using Bayesian networks," *Empirical Softw. Eng.*, vol. 19, no. 1, pp. 154–181, 2014.
[9] M. Y. Ricky, B. Yulianto, and F. Purnomo, "Mobile application software defect prediction," in *Proc. IEEE Symp. Service-Oriented Syst. Eng. (SOSE)*, Oxford, U.K., Mar./Apr. 2016, pp. 307–313.
[10] X.-Y. Jing, S. Ying, Z.-W. Zhang, S.-S. Wu, and J. Liu, "Dictionary learning based software defect prediction," in *Proc. 36th Int. Conf. Softw. Eng. (ICSE)*, Hyderabad, India, 2014, pp. 414–423.
[11] G. Czibula, Z. Marian, and I. G. Czibula, "Software defect prediction using relational association rule mining," *Inf. Sci.*, vol. 264, pp. 260–278, Apr. 2014.
[12] A. Soleimani and F. Asdaghi, "An AIS based feature selection method for software fault prediction," in *Proc. Iranian Conf. Intell. Syst. (LCIS)*, Feb. 2014, pp. 1–5.
[13] B. Li, B. Shen, J. Wang, Y. Chen, T. Zhang, and J. Wang, "A scenario-based approach to predicting software defects using compressed C4.5 model," in *Proc. IEEE 38th Annu. Int. Comput., Softw. Appl. Conf.*, Jul. 2014, pp. 406–415.
[14] S. S. Rathore and S. Kumar, "Predicting number of faults in software system using genetic programming," in *Proc. Int. Conf. Soft Comput. Softw. Eng.*, vol. 62, 2015, pp. 303–311.
[15] Y. Sun, M. S. Kamel, A. K. C. Wong, and Y. Wang, "Cost-sensitive boosting for classification of imbalanced data," *Pattern Recognit.*, vol. 40, no. 12, pp. 3358–3378, 2007.
[16] B. Ma, K. Dejaeger, J. Vanthienen, and B. Baesens, "Software defect prediction based on association rule classification," Katholieke Univ. Leuven, Leuven, Belgium, Tech. Rep., 2011, vol. 14, pp. 396–402.
[17] J. Chen, S. Liu, W. Liu, X. Chen, Q. Gu, and D. Chen, "A two-stage data preprocessing approach for software fault prediction," in *Proc. 8th Int. Conf. Softw. Secur. Rel. (SERE)*, Jun./Jul. 2014, pp. 20–29.
[18] S. Y. Liu, Y. Zhai, and D. S. Liu, "Cross-project software defect prediction with multi-strategy feature fifiltering," *Comput. Eng. Appl.*, pp. 1002–8331, Nov. 2018. [Online]. Available: http://kns.cnki.net/kcms/detail/11.2127.tp.20181101.1808.051.html
[19] Q. Yu, S.-J. Jiang, R.-C. Wang, and H.-Y. Wang, "A feature selection approach based on a similarity measure for software defect prediction," *Front. Inf. Technol. Elect. Eng.*, vol. 18, no. 11, pp. 1744–1753, 2017.
[20] K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya, "Choosing software metrics for defect prediction: An investigation on feature selection techniques," *Softw.-Pract. Exper.*, vol. 41, no. 5, pp. 579–606, 2011.
[21] I. H. Laradji, M. Alshayeb, and L. Ghouti, "Software defect prediction using ensemble learning on selected features," *Inf. Softw. Technol.*, vol. 58, pp. 388–402, Feb. 2015.
[22] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, Jun. 2002.
[23] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Trans. Rel.*, vol. 62, no. 2, pp. 434–443, Jun. 2013.
[24] T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic, and Y. Jiang, "Implications of ceiling effects in defect predictors," in *Proc. 4th Int. Workshop Predictor Models Softw. Eng.*, 2008, pp. 47–54.
[25] L. Pelayo and S. Dick, "Evaluating stratification alternatives to improve software defect prediction," *IEEE Trans. Rel.*, vol. 61, no. 2, pp. 516–525, Jun. 2012.
[26] N. Japkowicz, C. Myers, and M. Gluck, "A novelty detection approach to classification," *IJCAI*, vol. 1, pp. 518–523, Aug. 1995.
[27] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1263–1284, Sep. 2009.
[28] N. Seliya, T. M. Khoshgoftaar, and J. Van Hulse, "Predicting faults in high assurance software," in *Proc. IEEE 12th Int. Symp. High-Assurance Syst. Eng. (HASE)*, Nov. 2010, pp. 26–34.

[29] N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer, "SMOTEBoost: Improving prediction of the minority class in boosting," in *Proc. Eur. Conf. Knowl. Discovery Databases (PKDD)*, 2003, pp. 107–119.
[30] T. M. Khoshgoftaar, E. Geleyn, L. Nguyen, and L. Bullard, "Cost-sensitive boosting in software quality modeling," in *Proc. 7th IEEE Int. Symp. High Assurance Syst. Eng.*, Oct. 2002, pp. 51–60.
[31] C. Y. Zhu, X. Z. Chen, L. Yan, and X. M. Zhang, "Research on defect prediction model of artifificial immune recognition software based on principal component analysis," *Comput. Sci.*, vol. 44, no. 6A, pp. 483–518, 2017.
[32] H. He, Y. Bai, E. A. Garcia, and S. Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," in *Proc. Int. Joint Conf. Neural Netw.*, New York, NY, USA, Jun. 2008, pp. 1322–1328.
[33] E. Bauer and R. Kohavi, "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants," *Mach. Learn.*, vol. 36, pp. 105–139, Jul. 1999.
[34] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee, "Boosting the margin: A new explanation for the effectiveness of voting methods," *Ann. Statist.*, vol. 26, no. 5, pp. 1651–1686, 1998.
[35] T. Menzies, B. Caglayan, Z. He, E. Kocaguneli, J. Krall, F. Peters, and B. Turhans, "The promise repository of empirical software engineering data," Dept. Comput. Sci., West Virginia Univ., Beckley, WV, USA, Tech. Rep., 2012.
[36] H. Tong, B. Liu, and S. Wang, "Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning," *Inf. Softw. Technol.*, vol. 96, pp. 94–111, Apr. 2018.
[37] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 2–13, Jan. 2007.
[38] C. Ni, X. Chen, F. Wu, Y. Shen, and Q. Gu, "An empirical study on Pareto based multi-objective feature selection for software defect prediction," *J. Syst. Softw.*, vol. 152, pp. 215–238, Jun. 2019.
[39] Z. Sun, Q. Song, and X. Zhu, "Using coding-based ensemble learning to improve software defect prediction," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 42, no. 6, pp. 1806–1817, Nov. 2012.
[40] Y. Ma, G. Luo, and H. Chen, "Kernel based asymmetric learning for software defect prediction," *IEICE Trans. Inf. Syst.*, vol. E95.D, no. 1, pp. 267–270, 2012.
[41] T. Wang, Z. Zhang, X. Jing, and L. Zhang, "Multiple kernel ensemble learning for software defect prediction," *Automated Softw. Eng.*, vol. 23, no. 4, pp. 569–590, 2016.
[42] Y. Shao, B. Liu, S. Wang, and G. Li, "A novel software defect prediction based on atomic class-association rule mining," *Expert Syst. Appl.*, vol. 114, pp. 237–254, Dec. 2018.

HAITAO HE received the Ph.D. degree in mechanical design and manufacturing from Yanshan University, China, where she is currently a Professor with the School of Information Science and Engineering. Her research interests include data mining network information security and artificial intelligence.

XU ZHANG received the B.S. degree in mathematics and applied mathematics from the Hebei Normal University of Science and Technology, China, in 2017. She is currently pursuing the master's degree with the School of Information Science and Engineering, Yanshan University, China. Her major is computer science and technology. Her research interests include data mining and machine learning.

QIAN WANG received the B.S., M.S., and Ph.D. degrees from the School of Information Science and Engineering, Yanshan University, China, in 2009, 2012, and 2016, respectively, where she has been a Lecturer, since 2016.

She was a Visiting Scholar with the University of Hull, from 2015 to 2016. Her research interests include data mining, machine learning, software security, and digital health.

JIADONG REN received the B.S. and M.S. degrees from the Northeast Heavy Machinery Institute, in 1989 and 1994, respectively, and the Ph.D. degree from the Harbin Institute of Technology, in 1999.

He is currently a Professor with the School of Information Science and Engineering, Yanshan University, China. His research interests include data mining, complex networks, and software security. He is a Senior Member of the Chinese Computer Society and a member of IEEE SMC Society and ACM.

JIAXIN LIU received the B.S., M.S., and Ph.D. degrees from the School of Information Science and Engineering, Yanshan University, China, in 1997, 2001, and 2008, respectively, where she has been a Senior Experimentalist, since 2013. Her research interest includes data mining.

XIAOLIN ZHAO received the B.S., M.S., and Ph.D. degree from the School of Computer Science, Beijing Institute of Technology, Beijing, China, in 1993 and 2000, where he is currently an Associate Professor with the School of Software. His research interests include software security, complex networks, and software engineering theory.

YONGQIANG CHENG received the B.S. and M.S. degrees in control theory and control engineering from Tongji University, Shanghai, China, in 2001 and 2004, respectively, and the Ph.D. degree from the School of Engineering, Design and Technology, University of Brad-ford, U.K., in 2010.

He is currently a Senior Lecturer with the Department of Computer Science, University of Hull, U.K. His current research interests include smart systems and digital health, including ambient living robotics and non-invasive healthcare devices with predictive analysis on the collected data.