Using Coding Based Ensemble Learning to Improve Software Defect Prediction

Zhongbin Sun, Qinbao Song*, Xiaoyan Zhu

***Abstract*—Using classification methods to predict software defect-proneness with static code attributes has attracted a great deal of attention. The class-imbalance characteristic of software defect data makes the prediction much difficult, thus a number of methods have been employed to address this problem. However, these conventional methods, such as sampling, cost sensitive learning, bagging and boosting, could suffer from the loss of important information, unexpected mistakes and overfitting because they alter the original data distribution. This paper presents a novel method that firstly converts the imbalanced binary-class data into balanced multi-class data and then builds a defect predictor on the multi-class data with a specific coding scheme. A thorough experiment with four different types of classification algorithms, three data coding schemes, and six conventional imbalance data handling methods was conducted over the 14 NASA data sets. The experimental results show that the proposed method with 1-against-1 coding scheme is averagely superior to the conventional methods.**

***Index Terms*—Software defect prediction, class-imbalance data, multi-classifier, meta learning**

I. Introduction

With the continuous increase of software size and complexity, software quality assurance is becoming increasingly important. One way to improve software quality is software defect prediction, which is also an efficient means to relieve the effort in software code inspection or testing. Under this circumstance, we only need to inspect or test a part of software artifacts and ignore the remainders. By doing so, an organization's limited resources could be reasonably allocated with the goal of detecting and correcting the greatest number of defects in software. Therefore, software defect prediction has been widely studied and a number of methods have been proposed for addressing this problem [1], [2], [3], [4], [5], [6], [7], [8].

Of these methods, classification is a popular approach for software defect prediction, including analogy-based approaches [9], [10], tree-based methods [11], [12] and statistical procedures [13], [14], etc. These classification methods are relatively straightforward transplants from the fields of data mining and machine learning, and they follow the same path of using software metrics, which are extracted from software source code, as candidate factors to categorize the software components as defective or non-defective. This is achieved by building a binary classifier with historical defect data to predict the defect-proneness of new software components.

However, class-imbalance is always a key characteristic of software defect data, which consists of only a few defective components and a large number of non-defective ones [15], [16]. Unfortunately, most traditional classification methods, which have been used to predict software defect, almost take no account of the class-imbalance problem. Many popular classification methods have been reported to be hindered when encountering the class-imbalance problem, such as decision trees [17], [18], nearest neighbor [19], neural networks [20], support vector machines [21], [22] and Bayesian network [23]. As these classification methods tend to generate models that maximize the overall classification accuracy, the more interesting minority class is usually ignored by such models [24], [25]. For example, given a data set where only 1% of the examples belong to the defective components, a simple model classifying all examples as the non-defective components would achieve an overall accuracy of 99%. However, the minority defective components, which we want to accurately predict, are all misclassified with this simple model though it achieves a very high accuracy. Therefore the class-imbalance problem usually undermines the binary classifiers, further makes these classifiers cannot effectively predict the minority defective software components.

Actually, class-imbalance problem also exist in many other real-world domains, such as detecting oil spills from satellite images [26] and identifying fraudulent credit card transactions [27], etc. Therefore, the community of data mining and machine learning have paid much attention to this problem. Two workshops on learning from imbalanced data sets have been held in 2000 and 2003 at the AAAI and ICML conferences, respectively. In addition, the sixth issue of SIGKDD Exploration 2004 was dedicated entirely to the imbalance problem. Researchers indicated that the skewness in a data set usually results in inefficient binary classifiers for classifying the more interesting minority class and some techniques, such as sampling, cost sensitive learning, bagging and boosting, might be effective in addressing the imbalance problem [28], [29]. Hence a great many of researchers have focused on these techniques for addressing the class-imbalance problem in software defect prediction.

However, we argue that the above mentioned strategies might encounter problems when applied to the class-imbalanced software defect data. For example, sampling methods alter the distribution of the original data, where the under-sampling methods may discard some potentially useful data that could be important for the induction process, and the over-sampling methods may increase the likelihood of occurring overfitting. Moreover, the best sampling rate is difficult to be determined. For the cost sensitive learning methods, it is very difficult to know the accurate misclassifying cost, even

Z. Sun, Q. Song and X. Zhu are with Dept. of Computer Science & Technology, Xi'an Jiaotong University, China.
$*$ Corresponding author, email: qbsong@mail.xjtu.edu.cn

for the experienced domain experts. Furthermore, bagging and boosting still deal with the imbalance problem in context of traditional binary classification and may suffer from overfitting.

This paper presents a novel method to predict defective software components. The proposed method addresses the binary class-imbalance problem in software defect prediction by converting it into a multi-classification problem that does not suffer from the skewed data. Moreover, three different coding schemes including 1-against-1 [30], Random Correction Code [31], [32], and 1-against-all [33] are employed to generate an ensemble of multiple binary classifiers for addressing the multi-classification problem.

The experimental results on the 14 NASA data sets show that the 1-against-1 based method significantly outperforms the other two methods as well as the simple multi-class learning on the multi-class data. Furthermore, multi-classification learning with the 1-against-1 coding scheme outperforms most binary classifiers with original data, and can effectively improve the performance of the three different types of classifiers C4.5, Ripper, and Random Forest. In addition, compared with the conventional imbalance data dealing with methods (i.e., sampling, cost sensitive learning, bagging and boosting), the performance of our proposed method with 1-against-1 code outperforms most of them, especially for highly imbalanced data sets.

The remainder part of this paper is organized as follows. Section 2 introduces the related work. Section 3 presents our proposed method including the three coding based multi-class modeling methods in details. Section 4 is devoted to the experiments, discusses the setup and analyzes the results. Section 5 identifies the threats to validity of this study, and finally conclusions and future work are given in Section 6.

II. Related work

Many traditional classification methods have been directly used to predict software defect, including tree-based methods [7], [8], [11], analogy-based approaches [9], [10], neural networks [34], [35], and Bayes methods [36], [37], etc.

Menzies et al. [7] empirically found that Naive Bayes with a log-filtering preprocessor on the numeric data outperformed those of rule or tree based learning methods and furthermore claimed that "the choice of learning method is far more important than which subset of the available data is used for learning". However, Lessmann et al. [38] drew an inconsistent conclusion with a large scale empirical comparison of 22 binary classifiers over 10 public NASA data sets, and indicated that the importance of a particular learning method might be less than previously assumed by Menzies et al. [7] as there was no significant performance difference among the top 17 binary classifiers. In addition, Song et al. [8] argued that there might be potential bias and misleading results with the framework proposed by Menzies et al. [7], and proposed a more general and reliable framework for software defect prediction.

However, the aforementioned software defect prediction works clearly take no consideration of the class-imbalance problem. When predicting the occurrences of defect in software where the majority of components is non-defective, the binary classifiers usually fail to detect the minority defective components. Menzies et al. [39] realized that software defect data are highly prone to suffering from the class-imbalance problem, and pointed out that methodologies such as data sampling and boosting may help to improve the performance of binary classifiers. Besides this, a variety of studies have been conducted on the enhancement of classification performance with techniques including sampling [40], [41], [42], cost sensitive learning [43], [44], bagging [45], [46] and boosting [47], [48].

Sampling methods, which are employed to balance class distribution for the class-imbalanced data sets, can be divided into two groups: under-sampling and over-sampling. The under-sampling method eliminates the majority class examples while the over-sampling method increases the minority class examples for the purpose of obtaining a desired class distribution. Both sampling methods have been applied to alleviating the class-imbalance problem in software defect prediction. Pelayo and Dick [40] proposed a stratification-based resampling strategy in predicting software defect-proneness. They randomly under-sampled the majority class examples without replacement and over-sampled the minority class examples with the SMOTE technique [49]. Results of their experiment showed an improvement of 23% in the average geometric mean classification accuracy on four benchmark data sets. Seiffert et al. [41] conducted a comprehensive study on the effects of class imbalance and noise on different classification algorithms and data sampling techniques when used to predict software quality. They showed that not all classification algorithms were greatly affected by the application of sampling techniques, and the random over-sampling methods outperformed other sampling methods at different levels of noise and imbalance. However, the sampling methods alter the original class distribution of software defect data. For example, under-sampling may drop some important information and oversampling may lead to overfitting.

Generally, data mining and machine learning algorithms assume that all errors in classification are equally important. However, the cost of misclassification errors is often higher when the errors are associated with the minority class in contrast with the majority class. For example, the misclassification cost of the defective software components is usually much higher than that of the non-defective software components. Cost-sensitive classifiers explicitly account for these varying costs and have been applied to the domain of software defect prediction. Khoshgoftaar et al. [43] compared the Boosting with Cost-Boosting algorithms for software quality modeling. The Cost-Boosting algorithm is an improvement of Boosting, which takes into account the misclassification cost in weight updating of Boosting algorithm. Zheng [44] employed three cost-sensitive boosting neural networks algorithms for software defect prediction and suggested that threshold-moving algorithm was the best choice among the three algorithms. However, it is hard to determine the misclassification cost when applying the cost-sensitive learning method to software defect prediction.

Bagging [50] and Boosting [51], which are not specially designed to address class-imbalance problem, have been shown to be very effective in solving the class-imbalance problems [52]. These two ensemble methods have also been applied into software defect prediction. Seliya et al. [45] investigated the roughly balanced bagging (RBBag) algorithm for building software defect prediction models. The RBBag algorithm combines bagging and sampling techniques to improve the performance of bagged classifiers when learning from imbalanced data. The experimental results demonstrated that the RBBag algorithm could effectively address the class imbalance problem when building defect prediction models, and it significantly performed better than the benchmark binary classifiers without any bagging or data sampling techniques. Seiffert et al. [47] conducted a comprehensive study comparing sampling methods with Boosting for improving the performance of decision trees model built for identifying the defective modules. Their results showed that sampling methods were effective in improving the performance of such models while Boosting outperformed even the best data sampling methods. However, for each iteration of Bagging and Boosting methods, it may still suffer from the class imbalance problem as the sampled subset in a given iteration has the similar class distribution with the original data set.

We address the potential shortcomings of the methods mentioned above by converting the binary class-imbalance problem into a multi-class problem that no longer suffers from the class-imbalance curse, thus it is quite different from the existing methods.

III. Proposed defect prediction method

A. Overview of the method

Our proposed method addresses the class-imbalance problem in software defect prediction by turning a binary classification problem into a multi-classification problem.

Specifically, the majority class data (non-defective components) is firstly split into several bins. Each bin then has the similar number of examples to that of the minority class (defective components), and is assigned a new class label. After that, the relabeled data are employed to build a multi-classifier, which no more suffers from the skewed data because the new class distribution is balanced. Finally the multi-classification results are converted into the original binary, namely either defective or non-defective.

Our proposed method consists of three components: *Data balancing*, *Multi-classifier modeling*, and *Defect prediction*. Fig. 1 shows the details.

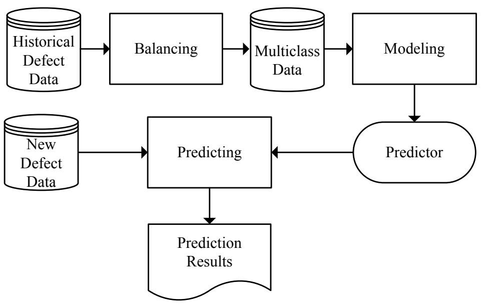

Fig. 1. Proposed software defect prediction method

(1) Data balancing

A software component can be labeled as non-defective or defective according to whether it contains defects or not. It has been investigated that the majority of software system's defects are contained in a small number of components. This means that software defect data are usually class imbalanced, and the number of non-defective components may severely outnumber that of the defective components in a given data set.

When balancing the skewed data, taking into account the fact that non-defective software components have no distinct difference and clustering might not result in groups with similar number of instances, we first randomly split the non-defective components into several bins where each bin has the similar number of components as that of the defective components. For example, suppose there are 100 defective components and 1000 non-defective components in the historical defect data. With our data balancing method, the 1000 non-defective components are randomly split into 1000/100=10 bins and each bin has 100 non-defective components. Then each bin is assigned a new label. By this way, multi-class data are obtained and the size of each class has almost the same number of instances.

(2) Multi-classifier modeling

Besides multi-class learning to learn multiple concepts with the direct application of a multi-class learning algorithm, there are many other methods to reduce a multi-class problem to multiple binary classification problems that could be solved separately, such as 1-against-all based method [53], [54], error-correcting output code based method [55], and 1-against-1 based method [53], [56].

When modeling the multi-classifier, instead of directly applying a multi-classification algorithm over multi-class data, we use one of the above mentioned three kinds of coding based methods to create diverse binary data from the multi-class data, and then employ a classification algorithm to construct binary predictors with the diverse binary data. Finally these binary predictors are combined into a multi-class predictor via a certain combination scheme that is corresponding to the specific coding schemes.

Thus, our multi-classifier modeling method can be viewed as a combination of the three steps: creating diverse binary data sets, learning diverse binary predictors from these binary data sets, and constructing a multi-class predictor. Refer to Section III-B for the further details.

(3) Defect prediction

When conducting the defect prediction for a given new software component, firstly the multi-classifier is applied and the classification result is obtained. Since the prediction result of the multi-classifier is not the desired binary result, it needs to be converted into the binary form, namely defective and non-defective. This is achieved by identifying which bins the result falls in. If it falls into the bins corresponding to the non-defective components, we regard it as a non-defective component, otherwise a defective component.

For example, suppose that after data balancing there are five classes in the multi-class data, in which the class label C_1 is corresponding to the original defective components and the class labels C_2, C_3, C_4, and C_5 are corresponding to the original non-defective components. For a new unknown-class software component, if it is predicted as C_1 by the multi-classifier, it will be labeled as defective, otherwise non-defective.

B. Coding based multi-classifier modeling method

When modeling a multi-classifier, we first create diverse binary data sets with a certain coding scheme. The coding schemes include the 1-against-all, the Random Correction Code (one popular error-correcting output code), and the 1-against-1. Each of them will be detailedly introduced later. Then a binary classification algorithm is employed to construct diverse binary predictors. After that, these binary predictors are combined to form a multi-class predictor. Fig. 2 shows the detailed procedure of our multi-classifier modeling method based on a specific coding scheme.

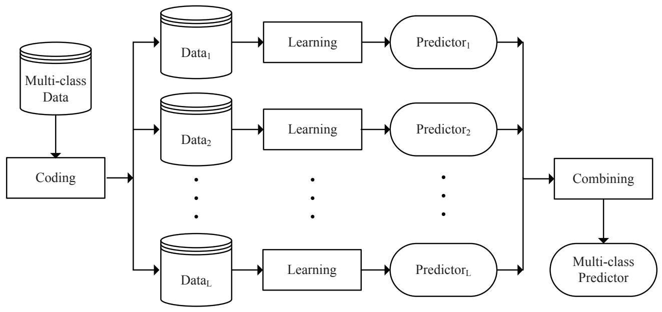

Fig. 2. Coding based multi-classifier modeling method

In order to facilitate the illustration of these three methods, we suppose that there are *K* classes in the multi-class data set and the corresponding labels are $C_1, C_2, ..., C_K$. Furthermore, for each of the binary data sets transformed from the multi-class data, the corresponding instances fall into two categories: *pos* and *neg*.

(1) 1-against-all based method

This is a 1-against-all coding scheme based multi-classifier modeling method proposed by Nilsson et al. [33].

This method learns *K* individual binary predictors P_1, P_2, ..., P_K for each class from the *K* diverse binary-class data sets, which can be obtained from the given *K*-class data set as follows.

The 1-against-all scheme first encodes each class with *K* elements whose values are either 1 or 0, thus a *K*×*K* binary matrix is obtained. For the *i*-th row, the code element on the *i*-th column is 1 while other code elements are 0. Table I shows the binary matrix for a 4-class data set with the 1-against-all code.

TABLE I
1-AGAINST-ALL CODE FOR A 4-CLASS DATA SET

Class	Code Word			
C_1	1	0	0	0
C_2	0	1	0	0
C_3	0	0	1	0
C_4	0	0	0	1

Each row of the matrix is referred to as a code word for a specific class. For example, in Table I, the code word for the class C_2 is (0, 1, 0, 0). Each column of the matrix is corresponding to a binary-class data set that consists of the original features and the new labels. In each column, the instances belonging to the class whose code element is 1 are relabeled as *pos* while other instances as *neg*. Thus the *K* binary-class data sets are obtained, and are utilized to learn *K* binary predictors P_1, P_2, ..., P_K[1].

For a new instance, each P_i $(i = 1, 2, ..., K)$ predicts it as *pos* with probability Pr_i and as *neg* with probability $1 - Pr_i$. That is, the new instance is predicted as class C_i with probability Pr_i while as other classes with probability $1-Pr_i$. Then for each class, the corresponding probabilities are added up, and the class label with the maximum probability is finally assigned to the new instance.

(2) Random Correction Code based method

Random correction code is a kind of randomly generated error-correcting output code [55]. This code performs as well as other excellent error-correcting output coding schemes [31], [32], and is an alternative to the widely used BCH code (Bose & Ray-Chaudhuri [57], Hocquenghem [58]) and furthermore is much easier to generate than the latter [59].

This method learns L ($L = R \cdot K$ where $R \in N^+ \wedge R \geq 2$) individual binary predictors P_1, P_2, ..., P_L for each class from the *L* diverse binary-class data sets, which can be obtained from the given *K*-class data set as follows.

The random correction coding scheme first encodes each class with *L* elements whose values are randomly set to 1 or 0, thus a *K*×*L* binary matrix is obtained. Table II shows the binary matrix for a 4-class data set with the random correction code (*R*=2).

Just like the 1-against-all scheme, each row of the matrix is referred to as a code word for a specific class, and each column of the matrix is corresponding to a binary-class data set that consists of the original features and the new labels. Then the *L* binary data sets are utilized to construct *L* binary predictors P_1, P_2, ..., P_L.

TABLE II
RANDOM CORRECTION CODE (R=2) FOR A 4-CLASS DATA SET

Class	Code Word							
C_1	1	1	0	1	0	0	1	0
C_2	0	1	1	0	0	0	1	1
C_3	1	0	1	1	1	1	1	1
C_4	1	0	1	0	1	0	0	0

For a new instance, each P_i $(i = 1, 2, ..., L)$ predicts it as *pos* with probability Pr_i and as *neg* with probability $1 - Pr_i$. If $Pr_i > 0.5$, 1 is outputted, otherwise 0. Thus with the *L* binary predictors we obtain a *L*-bit string comprised of 0 and 1. This string is then compared to the code word of each class by calculating the corresponding Hamming distance and the class label with the minimum Hamming distance is assigned to the new instance.

(3) 1-against-1 based method

[1]Note that for the binary-class data set used to construct the predictor P_i, the *pos* class in it is corresponding to the class C_i.

The 1-against-1 [30] has been widely used to address multi-class problems [60], [61]. This method converts a multi-class problem into a series of binary problems, one for each pair of classes. It works as follows.

For each pair of classes $\{C_i, C_j\}$ (1≤i<j≤K) of the given K-class data set, the instances belonging to the classes C_i and C_j are chosen to create a binary data set. The instances belonging to C_i are relabeled as *pos*, while those belonging to C_j are relabeled as *neg*. Finally, K(K-1)/2 binary-class data sets are generated.

This method encodes each class with K(K-1)/2 elements whose values can be 1, 0, or null, thus a K×K(K-1)/2 binary matrix, where each column corresponds to a class pair, is obtained. For each pair of classes $\{C_i, C_j\}$ (i.e., the column corresponding to the class pair), the code element corresponding to C_i is 1 and corresponding to C_j is 0, while other code elements are null. Table III shows the binary matrix for a 4-class data set with the 1-against-1 code.

TABLE III
1-AGAINST-1 CODE FOR A 4-CLASS DATA SET

Class	Code Word					
C_1	1	1	1	\	\	\
C_2	0	\	\	1	1	\
C_3	\	0	\	0	\	1
C_4	\	\	0	\	0	0

From the K(K-1)/2 binary data sets, K(K-1)/2 binary predictors P_{ij} (1≤i<j≤K) can be learned.

For a new instance, predictor P_{ij} (1≤i<j≤K) asserts it as C_i with the probability of Pr_i, and as C_j with $Pr_j = 1 - Pr_i$. The estimates of all the predictors on the new instance are combined and vector $\vec{P} = (Pr'_1, Pr'_2, ..., Pr'_K)$ is obtained. Finally, the class label C_m with the maximum probability $Pr'_m \in \vec{P}$ is assigned to the new instance.

In order to couple the estimates of the K(K-1)/2 binary predictors, Friedman proposed the popular 'max wins' algorithm [62]. In the 'max wins' algorithm each classifier assigns one vote for its preferred class, and the final result is the class with most votes. However, Hastie and Tibshirani [63] proposed a novel pairwise coupling method by combining the pairwise class probability estimates into a joint probability estimate for all K classes, which could improve on the Friedman's 'max wins' algorithm. Thus in our present study, Hastie and Tibshirani's method is employed and it will be introduced detailedly next.

Let n_{ij} be the number of instances for the binary-class data set that corresponding to the pair of classes $\{C_i, C_j\}$ (1≤i<j≤K) and $u_{ij} = Pr'_i/(Pr'_i + Pr'_j)$, vector $\vec{P}$ can be obtained by minimizing the Kullback-Leibler distance of $l(\vec{P})$ between Pr_i and u_{ij} [63]. $l(\vec{P})$ is defined as:

$$l(\vec{P}) = \sum_{1 \leq i < j \leq K} n_{ij}(Pr_i \log \frac{Pr_i}{u_{ij}} + (1 - Pr_i) \log \frac{1 - Pr_i}{1 - u_{ij}}) \quad (1)$$

To minimize $l(\vec{P})$, vector $\vec{P}$ has to satisfy the following conditions:

$$\sum_{i \neq j} n_{ij} Pr_i = \sum_{i \neq j} n_{ij} u_{ij} \quad (1 \leq i \leq K), \quad \sum_{i=1}^{K} Pr'_i = 1, and\ Pr'_i > 0. \quad (2)$$

Algorithm 1 shows the details of how to calculate the vector $\vec{P}$.

Algorithm 1: Iterative algorithm for calculating $\vec{P}$

```
   inputs: $\vec{P}$ - assigning randomly positive values
   output: $\vec{P}$ - representing the final prediction probability
1  Compute the initial $u_{ij} = Pr'_i/(Pr'_i + Pr'_j)$ ;
2  for $i = 1$ to $K$ do
3      $Pr'_i \leftarrow Pr'_i \times (\sum_{i \neq j} (n_{ij} \times Pr_i) / \sum_{i \neq j} (n_{ij} \times u_{ij}))$ ;
4  end
5  Recompute the corresponding $u_{ij} = Pr'_i/(Pr'_i + Pr'_j)$;
6  if the convergence condition Equation 2 is not satisfied then
7      $\vec{P} \leftarrow \vec{P} / (\sum_{i=1}^{K} Pr'_i)$ ;
8      goto step 2 ;
9  end
10 $\vec{P} \leftarrow \vec{P} / (\sum_{i=1}^{K} Pr'_i)$ ;
```

IV. EXPERIMENTAL RESULTS AND ANALYSIS

A. Data sets

All the 14 NASA data sets were used in our experiment. These 14 data sets are public and have been widely used for software defect prediction [7], [8], [11], [40], [44], [45], [47].

Each data set is comprised of the number of defects and some static code metrics including LOC (lines of code) counts, McCabe complexity measures and Halstead attributes etc. Software components that contain one or more defects are labeled as defective, while the others are labeled as non-defective.

Among the 14 data sets, the maximum number of total software components is 17186 while the minimum number of total software components is 125. Moreover, the minimum percentage of defective components is 0.41% and the maximum percentage of defective components is 48.80%. Table IV shows the statistic summary of the 14 defect data sets.

TABLE IV
STATISTIC SUMMARY OF THE NASA DATA SETS

Dataset	Language	LOC	number of attributes	number of components	number of defective components	percentage of defective components
CM1	C	17K	40	505	48	9.50%
JM1	C	457K	21	10878	2102	19.32%
KC1	C++	43K	21	2107	325	15.42%
KC2	C++	19K	22	522	107	20.50%
KC3	Java	8K	40	458	43	9.39%
KC4	Perl	25K	40	125	61	48.80%
MC1	C++	66K	39	9466	68	0.72%
MC2	C++	6K	40	161	52	32.30%
MW1	C	8K	40	403	61	7.69%
PC1	C	26K	40	1107	76	6.87%
PC2	C	25K	40	5589	23	0.41%
PC3	C	36K	40	1563	160	10.24%
PC4	C	30K	40	1458	178	12.21%
PC5	C++	162K	39	17186	516	3.00%

B. Experimental setup

For the purpose of obtaining reliable and stable results, in the experiment, 10-fold cross-evaluation strategy was used. However, Fisher et al. [64] reported that many learning algorithms exhibit order effects where certain ordering possibly improves or degrades performance. Thus the 10-fold cross-evaluation was repeated 10 times, and each time the ordering of instances was shuffled. Therefore each result is the average value of 100 experiments.

Four different types of classification algorithms including Random Forest, C4.5, Ripper and Naive Bayes were chosen to use in the study. These four basic learning algorithms could be used to learn not only from binary-class data but also from multi-class data.

Three coding schemes, which include 1-against-all, 1-against-1, and Random Correction Code, were used to create binary-class data.

Six traditional imbalance data dealing with methods including random under sampling, random over sampling, synthetic minority over-sampling technique, cost sensitive learning, bagging, and boosting were selected to compare with our proposed method.

The area under the ROC (Receiver Operating Characteristic) curve (AUC) was used as the measure of classification performance.

C. Experimental design

The experiment consists of the four investigations. The first investigation finds out which coding scheme performs best, while the second investigation explores whether or not the proposed defect prediction method with the best coding scheme outperforms the conventional imbalance data dealing with methods in the context of software defect prediction. The third investigation explores the impact of the different imbalance rates on our proposed defect prediction method and the fourth investigation identifies which classification algorithms benefit most from our proposed method.

(1) Investigation 1: which data coding scheme performs best?

This investigation intended to find out which coding scheme performs best by evaluating the proposed software defect prediction method with the three different coding schemes, and compare them with the multi-class learning method that does not need any coding work.

For this purpose, four different types of classification algorithms were used as defect predictors. Specifically, the four classification algorithms were separately employed as the base binary classifiers when evaluating the proposed method with the three coding schemes, while we directly applied them to the balanced multi-class data when conducting the experiment with the multi-class learning method.

(2) Investigation 2: is our proposed method more effective than the conventional ones?

This investigation was used to explore whether our proposed method is really effective or not. This was achieved by comparing our proposed method with six traditional imbalance data dealing with methods.

Random under sampling (RUS) decreases the number of majority cases in the data set by deleting majority cases randomly. Random over sampling (ROS) increases the number of minority cases in the data set by duplicating minority cases randomly. Synthetic minority over-sampling technique (SMOTE) [49] generates new minority cases based on the corresponding k-nearest neighbors. All these three sampling techniques can control the percentage of defect-prone components to all components, which was set to 50% as default in our experiment.

Cost sensitive learning assigns different misclassification costs for FP (false positive) and FN (false negative) in binary classifiers. In the experiment, we employed the MetaCost algorithm [65], and the ratio of costs for FP to FN was set as the number of positive examples to that of negative examples.

Ensemble learning methods including bagging and boosting were also used in the experiment. AUC instead of error rate was used to reweight the misclassified instances in the each iteration of boosting, and the default number of iterations was set to 10.

In addition, the binary classifiers were also evaluated which only employs the four basic classification algorithms directly on the original binary data.

Therefore, for this investigation, a total of 28 learning schemes (six traditional imbalance data dealing with methods $\times$ four basic learning algorithms + four binary classifiers with original data) were evaluated. All the above methods were implemented in the context of the machine learning tool Weka.

(3) Investigation 3: impact of the different imbalance rates on our proposed defect prediction method

This investigation was to explore whether or not our proposed method can effectively deals with class-imbalance problem in the context of software defect prediction.

From Table IV we know that the imbalance rates of the NASA data sets used in the experiments vary from 0.41% to 48.80%. This provides us a chance to explore how the imbalance rate affects our proposed defect prediction method by analyzing the prediction results of the investigations 1 and 2 under the different imbalance rates, and further to investigate the effectiveness of our proposed method on dealing with the class-imbalance problem.

(4) Investigation 4: which learning algorithms benefit most from our proposed method?

This investigation was to study which leaning algorithms improve their performance most in the context of our proposed defect prediction method compared with the corresponding binary classifiers.

This was achieved by the further analysis of the results obtained by the investigations 1 and 2 from a different perspective.

D. Experimental results and analysis

(1) Defect prediction results with different coding schemes

Table V shows the defect prediction results of the proposed method with different coding schemes over the 14 NASA data sets in terms of AUC. Note that the AUC values are the average values of the 10×10-fold cross-evaluation.

From Table V we observe that for a specific classifier, different coding schemes result in different AUC values; and the 1-against-1 coding scheme performs best in terms of AUC. Compared with that without any code, the prediction performance has been improved by the proposed method with the 1-against-1 code by 7.69% for Random Forest, by 25.76%

TABLE V

THE AUC VALUES OF THE PROPOSED PREDICTION METHOD WITH DIFFERENT BASE CLASSIFIERS AND DIFFERENT CODING SCHEMES

NASA	Random Forest				C4.5				Ripper				Naive Bayes			
	None	1-all	RCC	1-1	None	1-all	RCC	1-1	None	1-all	RCC	1-1	None	1-all	RCC	1-1
CM1	0.72	0.72	0.73	0.79	0.53	0.58	0.68	0.77	0.61	0.53	0.68	0.76	0.70	0.64	0.67	0.74
JM1	0.72	0.72	0.71	0.75	0.61	0.67	0.67	0.73	0.65	0.57	0.63	0.71	0.68	0.66	0.65	0.68
KC1	0.79	0.76	0.77	0.83	0.64	0.71	0.71	0.79	0.72	0.59	0.78	0.78	0.78	0.76	0.74	0.78
KC2	0.80	0.79	0.78	0.82	0.67	0.72	0.72	0.80	0.76	0.70	0.77	0.80	0.84	0.82	0.81	0.83
KC3	0.76	0.74	0.76	0.83	0.60	0.64	0.67	0.80	0.69	0.58	0.74	0.78	0.80	0.72	0.75	0.80
KC4	0.79	0.79	0.79	0.79	0.78	0.78	0.78	0.78	0.76	0.76	0.76	0.76	0.75	0.75	0.75	0.75
MC1	0.86	0.72	0.79	0.97	0.77	0.78	0.75	0.97	0.80	0.65	0.91	0.96	0.90	0.90	0.86	0.91
MC2	0.70	0.70	0.69	0.71	0.63	0.68	0.65	0.67	0.65	0.61	0.61	0.63	0.70	0.68	0.68	0.71
MW1	0.69	0.68	0.68	0.77	0.62	0.51	0.68	0.77	0.64	0.60	0.72	0.75	0.75	0.73	0.76	0.75
PC1	0.81	0.79	0.81	0.88	0.68	0.70	0.76	0.87	0.73	0.62	0.74	0.83	0.73	0.63	0.64	0.73
PC2	0.59	0.44	0.54	0.90	0.52	0.52	0.40	0.90	0.54	0.53	0.85	0.88	0.79	0.80	0.44	0.86
PC3	0.81	0.79	0.81	0.85	0.65	0.63	0.74	0.82	0.70	0.55	0.73	0.81	0.76	0.78	0.63	0.77
PC4	0.91	0.90	0.91	0.94	0.77	0.80	0.86	0.92	0.85	0.74	0.86	0.91	0.83	0.72	0.73	0.83
PC5	0.94	0.83	0.89	0.98	0.75	0.78	0.90	0.97	0.88	0.73	0.94	0.97	0.95	0.91	0.90	0.94
AVG	0.78	0.74	0.76	**0.84**	0.66	0.68	0.71	**0.83**	0.71	0.63	0.77	**0.81**	0.78	0.75	0.72	**0.79**

* None: without any coding scheme, 1-all: 1-against-all code, RCC: random correction code, 1-1: 1-against-1 code, AVG: the average AUC values over the 14 NASA data sets.

for C4.5, by 14.08% for Ripper, and by 1.28% for Naive Bayes, respectively.

We also observe that given the 1-against-1 coding scheme, Random Forest outperforms other three classifiers. The performance of Naive Bayes, C4.5, and Ripper has been improved by Random Forest by 6.33%, 1.2%, and 3.7%, respectively. Obviously, the tree based methods outperform Naive Bayes. This is quite different from that of Menzies et al. [7].

For the purpose of formally confirming that the 1-against-1 coding scheme outperforms others in terms of AUC, we conducted a Wilcoxon signed rank test [66]. The alternative hypotheses are that for a given base classifier, the 1-against-1 coding scheme is superior to each of the other three schemes at significance level $\alpha = 0.05$.

The p-values of all the hypotheses are far less than 0.05 except the hypothesis 1-1 superior to None for Naive Bayes. This means that for the four base classifiers, the 1-against-1 coding scheme is significantly better than the other three coding schemes except None for Naive Bayes. This can be explained as follows. From the coding methods we can know that for the 1-against-all based method, the K diverse binary-class data sets obtained via encoding are still imbalanced; and for the random correction code based method, the L binary-class data sets can be either imbalanced or balanced as the code word of each class is generated randomly. Imbalance data usually undermines binary classifiers. However, the 1-against-1 coding scheme guarantees the resulted binary-class data sets are balanced, thus is free from class-imbalance problem.

(2) Comparison of our proposed method and the conventional imbalance data handling methods

Now that we have concluded that 1-against-1 is the best coding scheme in the context of defect prediction with our proposed method, here we only compare the prediction results of our prediction method with the 1-against-1 coding scheme with those of the conventional imbalance data handling methods.

Tables VI, VII, VIII, and IX show the AUC values of the 1-against-1 coding scheme and the conventional imbalance data handling methods with classifiers Random Forest, C4.5, Ripper, and Naive Bayes, respectively. Note that the AUC values are the average values of the 10×10-fold cross-evaluation.

From Table VI we observe that 1-against-1 and Bagging rank first, and Boosting obtains the last position. The prediction performance has been improved by our proposed method with 1-against-1 code by 6.33% for Orig, by 3.70% for RUS, by 6.33% for ROS, by 2.44% for SMOTE, by 3.70% for Cost, and by 7.69% for Boost, respectively. This means the proposed method outperforms the learning method with the original binary data sets and those with conventional imbalance data dealing with methods except Bagging.

TABLE VI

THE AUC VALUES OF THE DIFFERENT IMBALANCE DATA HANDLING METHODS WITH RANDOM FOREST

NASA	1-1	Orig	RUS	ROS	SMOTE	Cost	Bag	Boost
CM1	0.79	0.75	0.74	0.74	0.76	0.76	0.77	0.74
JM1	0.75	0.71	0.71	0.71	0.73	0.72	0.75	0.68
KC1	0.83	0.79	0.78	0.78	0.81	0.74	0.82	0.75
KC2	0.82	0.80	0.80	0.79	0.81	0.80	0.81	0.74
KC3	0.83	0.79	0.78	0.80	0.82	0.79	0.83	0.68
KC4	0.79	0.79	0.79	0.79	0.80	0.80	0.81	0.77
MC1	0.97	0.87	0.94	0.88	0.90	0.95	0.94	0.89
MC2	0.71	0.68	0.67	0.69	0.73	0.64	0.73	0.76
MW1	0.77	0.74	0.74	0.70	0.73	0.77	0.78	0.71
PC1	0.88	0.81	0.84	0.82	0.85	0.86	0.89	0.80
PC2	0.90	0.68	0.85	0.64	0.76	0.87	0.80	0.72
PC3	0.85	0.78	0.81	0.81	0.83	0.84	0.85	0.80
PC4	0.94	0.92	0.91	0.92	0.92	0.91	0.95	0.91
PC5	0.98	0.94	0.97	0.94	0.96	0.95	0.98	0.93
AVG	**0.84**	0.79	0.81	0.79	0.82	0.81	**0.84**	0.78

* 1-1: 1-against-1 code, Orig: learning with the original binary-class data sets, RUS: random under-sampling, ROS: random over-sampling, SMOTE: synthetic minority over-sampling technique, Cost: cost sensitive learning, Bag: Bagging, Boost: Boosting.

The Wilcoxon signed rank test at the significance level 0.05 was performed, in which the alternative hypotheses are that 1-1 is superior to Orig, RUS, ROS, SMOTE, Cost and Boost but unequal to Bag. All the corresponding p-values are less than 0.05 except the hypothesis 1-1 unequal to Bag, which means that when Random Forest is used as the base classifier, 1-against-1 is equivalent to Bagging but significantly better than other six imbalance data handling methods.

From Table VII we observe that 1-against-1 ranks first again, and Orig and ROS rank last. This means the our proposed method with 1-against-1 code outperforms the learning method with the original binary data sets and those with the conventional imbalance data handling methods employed in our experiment. The prediction performance has been improved by our proposed method by 27.69% for Orig, by 12.16% for RUS, by 23.88% for ROS, by 18.57% for SMOTE, by 5.06% for Cost, by 2.47% for Bag, and by 6.41% for Boost, respectively.

For the purpose of validating our proposed method with the

TABLE VII
THE AUC VALUES OF THE DIFFERENT IMBALANCE DATA HANDLING METHODS WITH C4.5

NASA	1-1	Orig	RUS	ROS	SMOTE	Cost	Bag	Boost
CM1	0.77	0.52	0.67	0.59	0.65	0.76	0.79	0.68
JM1	0.73	0.67	0.65	0.59	0.66	0.68	0.74	0.68
KC1	0.79	0.73	0.72	0.58	0.69	0.73	0.80	0.75
KC2	0.80	0.72	0.61	0.72	0.70	0.75	0.80	0.73
KC3	0.80	0.60	0.70	0.66	0.63	0.77	0.78	0.71
KC4	0.78	0.77	0.76	0.77	0.78	0.84	0.81	0.78
MC1	0.97	0.80	0.88	0.81	0.85	0.84	0.87	0.96
MC2	0.67	0.68	0.62	0.63	0.68	0.74	0.74	0.74
MW1	0.77	0.45	0.69	0.64	0.63	0.77	0.74	0.78
PC1	0.87	0.71	0.76	0.69	0.73	0.83	0.84	0.83
PC2	0.90	0.45	0.80	0.58	0.63	0.80	0.74	0.62
PC3	0.82	0.62	0.71	0.64	0.65	0.79	0.82	0.77
PC4	0.92	0.79	0.83	0.75	0.77	0.89	0.92	0.91
PC5	0.97	0.76	0.93	0.66	0.81	0.93	0.96	0.93
AVG	**0.83**	0.65	0.74	0.67	0.70	0.79	0.81	0.78

1-against-1 code significantly outperforms other methods, we conducted the Wilcoxon signed rank test at the significance level 0.05. The obtained *p*-values of all the hypotheses except the 1-1 better than Bag are less than 0.05. This means that the 1-against-1 method is significantly better than five out of the six conventional imbalance data handling methods, including RUS, ROS, SMOTE, Cost and Boost, and the learning method with the original binary-class data.

TABLE VIII
THE AUC VALUES OF THE DIFFERENT IMBALANCE DATA HANDLING METHODS WITH RIPPER

NASA	1-1	Orig	RUS	ROS	SMOTE	Cost	Bag	Boost
CM1	0.76	0.55	0.69	0.61	0.67	0.70	0.78	0.75
JM1	0.71	0.58	0.68	0.67	0.64	0.50	0.74	0.70
KC1	0.78	0.60	0.73	0.67	0.70	0.69	0.80	0.76
KC2	0.80	0.72	0.71	0.72	0.76	0.75	0.82	0.75
KC3	0.78	0.59	0.72	0.63	0.67	0.72	0.73	0.74
KC4	0.76	0.76	0.76	0.79	0.78	0.79	0.79	0.79
MC1	0.96	0.65	0.88	0.80	0.83	0.91	0.90	0.93
MC2	0.63	0.51	0.61	0.63	0.65	0.58	0.67	0.77
MW1	0.75	0.55	0.68	0.61	0.64	0.67	0.70	0.70
PC1	0.83	0.60	0.74	0.69	0.72	0.78	0.85	0.82
PC2	0.88	0.53	0.80	0.61	0.62	0.84	0.74	0.65
PC3	0.81	0.55	0.74	0.64	0.72	0.76	0.83	0.79
PC4	0.91	0.76	0.86	0.78	0.86	0.86	0.94	0.94
PC5	0.97	0.75	0.94	0.89	0.83	0.94	0.97	0.95
AVG	**0.81**	0.62	0.75	0.70	0.72	0.75	0.80	0.79

From Table VIII we observe that our proposed method with 1-against-1 code ranks first again, and the learning method with the original binary-class data holds the last position. Specially, the prediction performance has been improved by our proposed method by 30.65% for Orig, by 8.00% for RUS, by 15.71% for ROS, by 12.50% for SMOTE, by 8.00% for Cost, by 1.25% for Bag, and by 2.53% for Boost, respectively. This means that our proposed method with 1-against-1 code outperforms all the conventional imbalance data handling methods.

The Wilcoxon signed rank test with significance level 0.05 was conducted to confirm our proposed method performs significantly better than other methods. The corresponding *p*-values for hypotheses 1-1 superior to Orig, RUS, ROS, SMOTE, and Cost are less than 0.05, which means that these alternative hypotheses are accepted and our proposed method with the 1-against-1 code significantly outperforms five out of the seven different imbalance data dealing with methods.

From Table IX we observe that the average AUC values of of Orig and Bag are 80%, both of which slightly improve the prediction performance of our proposed method by 1.27%.

TABLE IX
THE AUC VALUES OF THE DIFFERENT IMBALANCE DATA HANDLING METHODS WITH NAIVE BAYES

NASA	1-1	Orig	RUS	ROS	SMOTE	Cost	Bag	Boost
CM1	0.74	0.75	0.73	0.74	0.74	0.71	0.75	0.70
JM1	0.68	0.68	0.67	0.68	0.69	0.65	0.68	0.67
KC1	0.78	0.79	0.77	0.79	0.79	0.73	0.79	0.77
KC2	0.83	0.83	0.83	0.83	0.83	0.83	0.83	0.76
KC3	0.80	0.81	0.78	0.81	0.80	0.75	0.81	0.75
KC4	0.75	0.75	0.73	0.75	0.75	0.80	0.80	0.76
MC1	0.91	0.91	0.91	0.91	0.91	0.86	0.90	0.91
MC2	0.71	0.69	0.69	0.70	0.70	0.65	0.70	0.62
MW1	0.75	0.77	0.73	0.74	0.74	0.72	0.76	0.65
PC1	0.73	0.72	0.72	0.73	0.74	0.67	0.73	0.76
PC2	0.86	0.86	0.85	0.86	0.86	0.78	0.85	0.75
PC3	0.77	0.78	0.71	0.77	0.75	0.73	0.78	0.77
PC4	0.83	0.86	0.81	0.83	0.85	0.82	0.86	0.87
PC5	0.94	0.93	0.93	0.94	0.94	0.91	0.93	0.94
AVG	0.79	**0.80**	0.78	0.79	0.79	0.76	**0.80**	0.76

However, we observe that the AUC of our method is greater than those of Cost and Boost, whose performance has been improved by 3.95%, respectively. In addition, the performance of RUS has also been improved by our method by 1.28%.

In order to formally confirm whether or not the two methods with 80% of the average AUC are significantly better than our proposed method with the 1-against-1 code and our proposed method significantly outperforms RUS, Cost and Boost, we conducted the Wilcoxon signed rank test at the significance level 0.05. There are only three *p*-values corresponding to hypotheses that our proposed method with the 1-against-1 code significantly outperforms RUS, Cost and Boost are less than the significance level 0.05, which means that our proposed method performs significantly better than these three imbalance data handling methods. At the same time, Orig and Bagging are not significantly better than our proposed method as their corresponding *p*-values are 0.0747 and 0.1039 respectively. The little performance difference of 0.01 may be due to chance.

Based on the above results and in order to provide readers a big picture about the performance of our proposed method, we rank the eight imbalance data dealing with methods according to the prediction performance AUC of a given classifier on the NASA data. Then we summarize the ranks of the prediction methods under the four different classifiers, and give the final ranks. Table X shows the details.

TABLE X
RANK OF THE EIGHT METHODS WITH DIFFERENT TYPES OF CLASSIFIERS OVER THE NASA DATA

	1-1	Orig	RUS	ROS	SMOTE	COST	Bag	Boost
Random Forest	1	4	3	4	2	3	1	5
C4.5	1	8	5	7	6	3	2	4
Ripper	1	7	4	6	5	4	2	3
Naive Bayes	2	1	3	2	2	4	1	4
Sum	5	20	15	19	15	14	6	16
Rank	1	7	4	6	4	3	2	5

From Table X we observe that (i) all the imbalance data dealing with methods perform better than the learning method with the original binary-class data. This means that when predicting software defects, the methods that are able to deal with imbalance data should be employed. (ii) Among the imbalance data dealing with methods, our proposed method obtains the rank of 1, and Bagging is a good alternative. These two methods are much better than the other methods.

(3) Impact of imbalance rate on our proposed defect prediction method

From the investigations 1 and 2 we observe that, for a specific (base) classifiers, the performance improvements of 1-against-1 over None, Orig and the six conventional imbalance data handling methods vary with different NASA data sets. As the imbalance rates of these data sets vary from 0.41% to 48.80%, so this reveals that the performance improvements are different for different imbalance rates as well.

Fig. 3[2] shows the tendencies of performance improvements of our proposed method with the 1-against-1 coding scheme over the other eight methods Orig, None, RUS, ROS, SMOTE, Cost, Bagging and Boosting, respectively. In this figure, the data sets are sorted in the increasing order of percentage of defective components, i.e., the descending order of imbalance rate.

Fig. 3 (a) shows the performance improvements of our proposed method with 1-against-1 over the binary classifiers with original data. From it we observe that with the decrease of the imbalance rate (i.e., the ascending of the percentage of defective components), the performance improvements get smaller for the classifiers except Naive Bayes. This means that our proposed method (i) can conspicuously improve the performances of the classifiers Random Forest, C4.5 and Ripper when no any imbalance data handling method is adopted especially for the highly imbalanced data; (ii) cannot improve the performance of Naive Bayes with original data regardless the imbalance rate.

Fig. 3 (b) shows the performance improvements of our proposed method with 1-against-1 over the multi-classifiers with the balanced non-coding data. From it we observe that with the decrease of the imbalance rate, the performance improvements get smaller for all the four classifiers and even become negative for three of them. This means our proposed method (i) can conspicuously improve the performances of the four classifiers with balanced non-coding data for the highly skewed data even the data are balanced; (ii) generally cannot used for the classifiers with balanced data when the original imbalance rate is minor except C4.5, as these classifiers work well with the balanced or nearly balanced data. It also reveals that the data balancing method has two-sided effects: it can improve the predictions of Naive Bayes, Random Forest and Ripper when the imbalance rate is minor while cannot for highly skewed data. So it is recommended to use when the imbalance rate is small.

Figs. 3 (c)-(h) show the performance improvements of our proposed method with 1-against-1 over the binary classifiers with the data that preprocessed by the six conventional imbalance data handling methods RUS, ROS, SMOTE, Cost, Bagging and Boosting, respectively. From these figures we observe that:

- For the very highly skewed data with imbalance rate from 90% to 99.6% (i.e., the percentage of the defective components from 10% to 0.4%), our proposed method with 1-against-1 conspicuously improves the performances of the four classifiers with RUS, ROS, SMOTE, Cost, Bagging and Boosting, and the improvement increases with the increase of the imbalance rate except for Ripper with RUS and Cost and Random Forest with Cost. This reveals compared with the conventional methods, our proposed method is very effective for the highly skewed data in the context of software defect prediction.
- For the skewed data with imbalance rate less than 90%, our proposed method with 1-against-1 conspicuously improves the performances of the four classifiers with the five conventional methods RUS, ROS, SMOTE, Cost

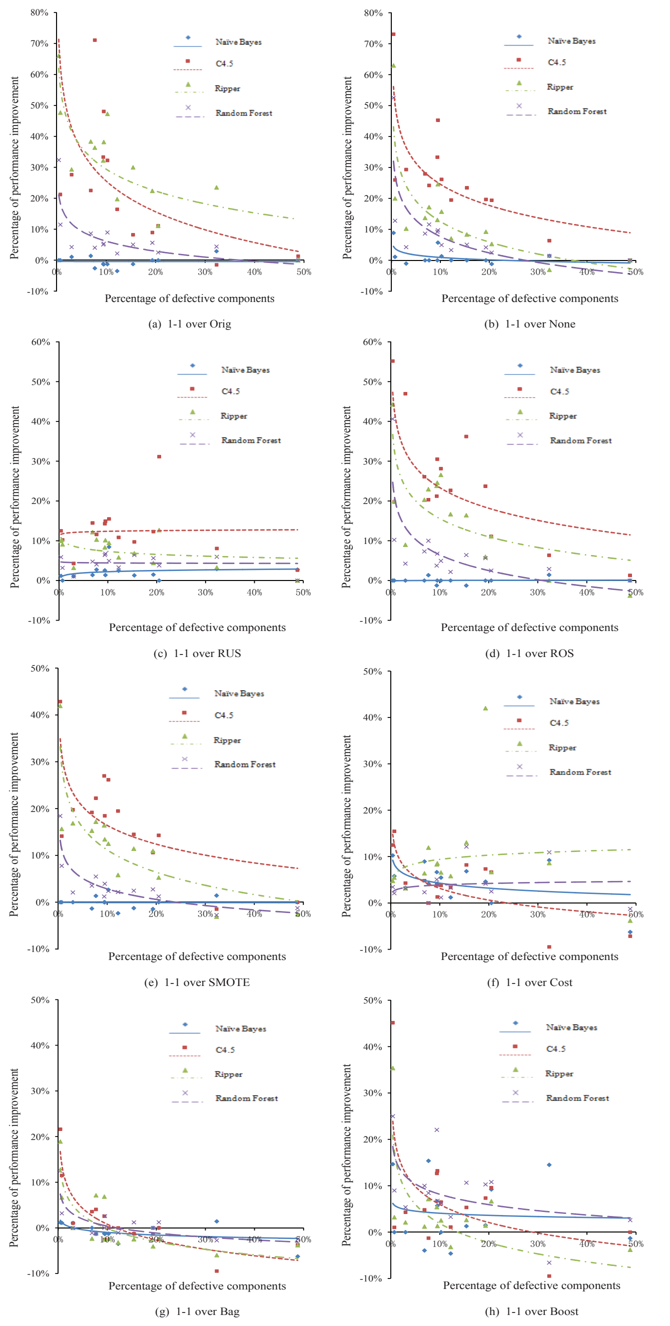

Fig. 3. Performance improvements of our proposed method with 1-against-1 over other methods for the four classifiers

[2]For more clearly depicts the main content, the scale has been truncated.

and Boosting except for Naive Bayes with ROS and SMOTE, Random Forest with ROS and SMOTE, C4.5 with Cost and Boosting, and Ripper with Boosting. This means our proposed method generally outperforms the other methods, and some times other methods such as ROS and Boosting can effectively handle skewed data when the imbalance rate is not very high.

- Our proposed method cannot improve the performances of the four classifiers with Bagging when the imbalance rate is smaller than 90%. However, for the very highly skewed data, our method outperforms Bagging. This means the two methods are complementary. That is, for very highly imbalanced data, our method should be the first choice; otherwise, Bagging is a good alternative.

From the aforementioned observations and discussions we know that our proposed method with the 1-against-1 coding scheme can effectively deal with skewed data especially for the very highly imbalanced data.

(4) Statistical comparison results of the four learning algorithms with our proposed prediction method

From the AUC of our proposed method with the four different learning algorithms we know that Random Forest performs best. In this subsection we intend to statistically confirm the conclusion.

Firstly, the Friedman test [67], which is based on the ranked performance rather than the actual performance estimates and therefore is less susceptible to outliers, was conducted to compare several different classifiers over multiple data sets. The *p*-value is 1.90E-12, which indicates the performance differences among the eight models are not random and therefore confirms the differences of AUC values are significant. Then a post hoc test was performed to identify which particular models significantly perform best, as suggested by Demsar [68]. This was accomplished by applying the Nemenyi test [69] at the significance level $\alpha = 0.05$. Fig. 4 shows the results of the Nemenyi test.

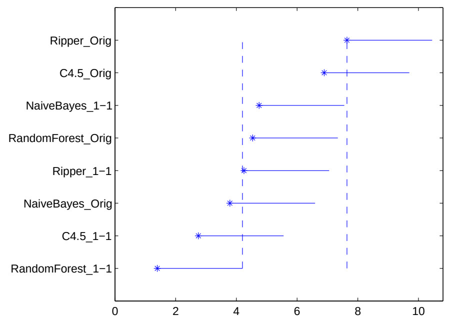

Fig. 4. Results of the pairwise comparisons of the eight models using Nemenyi post hoc test with $\alpha = 0.05$

Fig. 4 plots the prediction models against average performance ranks, where all models are sorted according to their ranks, the '$*$' denotes the respective average rank of each model and the line segment to the right of each model represents its critical difference, which means the models whose '$*$' on the right end of the line are outperformed significantly. The critical difference is highlighted with a vertical dotted line in two cases. The left vertical line is associated with RandomForest_1-1, and all models right to this line perform significantly worse than RandomForest_1-1. The right vertical line is associated with Ripper_Orig, and all models left to this line perform significantly better than Ripper_Orig.

From Fig. 4 we observe that (i) RandomForest_1-1 obtains the rank of 1, and performs significantly better than five out of the seven models. (ii) RandomForest_1-1 almost significantly outperforms NaiveBayes_Orig, but Nemenyi's test is unable to reject the null hypothesis that RandomForest_1-1 and NaiveBayes_Orign have the same average rank, which means that the performance differences between them are due to chance. However, it could also be caused by a Type II error as suggested by Lessmann et al. [38]: possibly the Nemenyi's test doesn't have enough power to detect the significant difference at $\alpha = 0.05$. In other words, the significant difference might be detected when α is slightly greater (i.e. $\alpha = 0.1$). (iii) RandomForest_1-1, Ripper_1-1, and C4.5_1-1 significantly outperform RandomForest_Orig, Ripper_Orig, and C4.5_Orig, respectively. This means that these binary classifiers could be improved effectively by our proposed method.

From the above results we can know that our proposed multi-class learning method to handle imbalance data via 1-against-1 coding is able to improve the performance of classifiers with original two-class data, and the Random Forest using the proposed method with 1-against-1 coding scheme performs best among the different models, though not significantly better than the C4.5 with the proposed method and Naive Bayes with the original binary data.

V. Threats to Validity

One threat to validity is the data used to generate the experimental results. There might be some differences among the NASA data between our present study and other studies, which are possibly attributed to the different preprocessing methods employed in different studies, such as whether discarding the components with missing data or not. Recently Gray et al. [70] have questioned the quality of the NASA software defect data and therefore other defect data could be employed to validate our method.

The employed classification algorithms are another possible source of threat to validity. In our present study, we only employed four different kinds of classification algorithms. As is known to all, data mining is a large and dynamic field with large numbers of new data mining algorithms continually being developed. Thus we could not validate other classification algorithms for space limitation and we would encourage other researchers interested in our study to repeat our study with more classification algorithms.

VI. Conclusion

In this paper, we have presented a novel method to handle the class-imbalance data in software defect prediction. Different from the conventional sampling, cost-sensitive learning, and ensemble learning methods, the proposed method does not change the original class distribution, and does not suffer from information loss or unexpected mistakes that may be caused by the conventional methods via increasing the minority instances or decreasing the majority ones.

The proposed method turns the class-imbalance binary classification into the balanced multi-class classification by randomly splitting the instances of the majority class into several smaller clusters. Three different coding schemes, which are 1-against-1, 1-against-all and Random Correction Code, were employed for this multi-class classification that performs essential ensemble learning and can improve the classification results effectively. The experimental results show that the 1-against-1 coding scheme significantly outperforms the other two coding schemes as well as no coding scheme.

We compared the proposed method with the conventional methods, such as sampling, cost sensitive learning, bagging and boosting, and four different types of base classifiers were employed. The experimental results on the 14 NASA data sets show that our proposed method significantly outperforms random under-sampling, random over-sampling, synthetic minority over-sampling, cost sensitive learning and boosting methods, and achieves comparable results to the bagging method. We find out that our proposed method can effectively handle skewed data especially the highly imbalanced data. We also conclude that the prediction performances of C4.5, Ripper and Random Forest are greatly improved while Naive Bayes is improved a little.

Future work will focus on generalizing our proposed approach to more domains suffering from class imbalance problems other than the domain of software defect prediction, which will help widen the usage of our proposed approach. In addition, we will also explore more basic classification algorithms with the use of our method for future study.

Acknowledgment

The authors would like to thank the editors and the anonymous reviewers for their insightful and helpful comments and suggestions. This work is supported by the National Natural Science Foundation of China under grant 61070006.

References

[1] P. Bishnu and V. Bhattacherjee, "Software fault prediction using quad tree based k-means clustering algorithm," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 6, pp. 1146–1150, 2012.

[2] T. Khoshgoftaar and E. Allen, "A practical classification-rule for software-quality models," *IEEE Transactions on Reliability*, vol. 49, no. 2, pp. 209–216, 2000.

[3] W. Yang and L. Li, "A rough set model for software defect prediction," in *International Conference on Intelligent Computation Technology and Automation*, 2008, pp. 747–751.

[4] N. Ohlsson, M. Zhao, and M. Helander, "Application of multivariate analysis for software fault prediction," *Software Quality Journal*, vol. 7, no. 1, pp. 51–66, 1998.

[5] M. Shepperd and G. Kadoda, "Comparing software prediction techniques using simulation," *IEEE Transactions on Software Engineering*, vol. 27, no. 11, pp. 1014–1022, 2001.

[6] T. Khoshgoftaar, E. Allen, and J. Deng, "Using regression trees to classify fault-prone software modules," *IEEE Transactions on Reliability*, vol. 51, no. 4, pp. 455–462, 2002.

[7] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 2–13, 2007.

[8] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A general software defect-proneness prediction framework," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 356–370, 2011.

[9] K. E. Emam, S. Benlarbi, N. Goel, and S. N. Rai, "Comparing case-based reasoning classifiers for predicting high risk software components," *Journal of Systems and Software*, vol. 55, no. 3, pp. 301–320, 2001.

[10] T. M. Khoshgoftaar and N. Seliya, "Analogy-based practical classification rules for software quality estimation," *Empirical Software Engineering*, vol. 8, no. 4, pp. 325–350, 2003.

[11] L. Guo, Y. Ma, B. Cukic, and H. Singh, "Robust prediction of fault-proneness by random forests," in *15th International Symposium on Software Reliability Engineering*, 2004, pp. 417–428.

[12] R. Selby and A. Porter, "Learning from examples: Generation and evaluation of decision trees for software resource analysis," *IEEE Transactions on Software Engineering*, vol. 14, no. 12, pp. 1743–1757, 1988.

[13] J. Munson and T. Khoshgoftaar, "The detection of fault-prone programs," *IEEE Transactions on Software Engineering*, vol. 18, no. 5, pp. 423–433, 1992.

[14] V. Basili, L. Briand, and W. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 751–761, 1996.

[15] C. Andersson, "A replicated empirical study of a selection method for software reliability growth models," *Empirical Software Engineering*, vol. 12, no. 2, pp. 161–182, 2007.

[16] N. Fenton and N. Ohlsson, "Quantitative analysis of faults and failures in a complex software system," *IEEE Transactions on Software Engineering*, vol. 26, no. 8, pp. 797–814, 2000.

[17] G. Batista, R. Prati, and M. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 20–29, 2004.

[18] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[19] J. Zhang and I. Mani, "knn approach to unbalanced data distributions: a case study involving information extraction," in *ICML Workshop on Learning from Imbalanced Datasets II*, 2003.

[20] N. Japkowicz and S. Stephen, "The class imbalance problem: A systematic study," *Intelligent Data Analysis*, vol. 6, no. 5, pp. 429–449, 2002.

[21] R. Akbani, S. Kwek, and N. Japkowicz, "Applying support vector machines to imbalanced datasets," in *15th European Conference on Machine Learning*, 2004, pp. 39–50.

[22] G. Wu and E. Chang, "Class-boundary alignment for imbalanced dataset learning," in *ICML Workshop on Learning from Imbalanced Datasets II*, 2003, pp. 49–56.

[23] K. Ezawa, M. Singh, and S. Norton, "Learning goal oriented bayesian networks for telecommunications risk management," in *International Conference on Machine Learning*, 1996, pp. 139–147.

[24] Y. Sun, M. Kamel, A. Wong, and Y. Wang, "Cost-sensitive boosting for classification of imbalanced data," *Pattern Recognition*, vol. 40, no. 12, pp. 3358–3378, 2007.

[25] X. Liu, J. Wu, and Z. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 39, no. 2, pp. 539–550, 2009.

[26] M. Kubat, R. C. Holte, and S. Matwin, "Machine learning for the detection of oil spills in satellite radar images," *Machine Learning*, vol. 30, no. 2, pp. 195–215, 1998.

[27] T. Fawcett and F. Provost, "Adaptive fraud detection," *Data Mining and Knowledge Discovery*, vol. 1, no. 3, pp. 291–316, 1997.

[28] G. M. Weiss, "Mining with rarity: a unifying framework," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 7–19, 2004.

[29] S. Kotsiantis, D. Kanellopoulos, and P. Pintelas, "Handling imbalanced datasets: A review," *GESTS International Transactions on Computer Science and Engineering*, vol. 30, no. 1, pp. 25–36, 2006.

[30] S. Knerr, L. Personnaz, and G. Dreyfus, "Single-layer learning revisited: a stepwise procedure for building and training a neural network," *Neurocomputing: Algorithms, Architectures, and Applications*, vol. 68, pp. 41–50, 1990.

[31] G. James and T. Hastie, "The error coding method and picts," *Journal of Computational and Graphical Statistics*, vol. 7, no. 3, pp. 377–387, 1998.

[32] A. Berger, "Error-correcting output coding for text classification," in *IJCAI Workshop on Machine Learning for Information Filtering*, 1999.

[33] N. J. Nilsson, T. J. Sejnowski, T. J. Sejnowski, H. White, and H. White, "Learning machines," 1965.

[34] T. Khoshgoftaar, E. Allen, J. Hudepohl, and S. Aud, "Application of neural networks to software quality modeling of a very large telecommunications system," *IEEE Transactions on Neural Networks*, vol. 8, no. 4, pp. 902–909, 1997.
[35] M. Thwin and T. Quah, "Application of neural networks for software quality prediction using object-oriented metrics," *Journal of Systems and Software*, vol. 76, no. 2, pp. 147–156, 2005.
[36] N. Bouguila, J. H. Wang, and A. Ben Hamza, "A bayesian approach for software quality prediction," in *4th International IEEE Conference on Intelligent Systems*, 2008, pp. 11–49.
[37] B. Turhan and A. Bener, "Analysis of naive bayes' assumptions on software fault data: An empirical study," *Data & Knowledge Engineering*, vol. 68, no. 2, pp. 278–290, 2009.
[38] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, 2008.
[39] T. Menzies, A. Dekhtyar, J. Distefano, and J. Greenwald, "Problems with precision: A response to "comments on 'data mining static code attributes to learn defect predictors'"," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, pp. 637–640, 2007.
[40] L. Pelayo and S. Dick, "Applying novel resampling strategies to software defect prediction," in *Annual Meeting of the North American on Fuzzy Information Processing Society*, 2007, pp. 69–72.
[41] C. Seiffert, T. Khoshgoftaar, J. Van Hulse, and A. Folleco, "An empirical study of the classification performance of learners on imbalanced and noisy software quality data," *Information Sciences*, 2011.
[42] D. Drown, T. Khoshgoftaar, and N. Seliya, "Evolutionary sampling and software quality modeling of high-assurance systems," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 39, no. 5, pp. 1097–1107, 2009.
[43] T. Khoshgoftaar, E. Geleyn, L. Nguyen, and L. Bullard, "Cost-sensitive boosting in software quality modeling," in *7th IEEE International Symposium on High Assurance Systems Engineering*, 2002, pp. 51–60.
[44] J. Zheng, "Cost-sensitive boosting neural networks for software defect prediction," *Expert Systems with Applications*, vol. 37, no. 6, pp. 4537–4543, 2010.
[45] N. Seliya, T. M. Khoshgoftaar, and J. V. Hulse, "Predicting faults in high assurance software," in *12th IEEE International Symposium on High Assurance Systems Engineering*, 2010, pp. 26–34.
[46] T. Khoshgoftaar, E. Geleyn, and L. Nguyen, "Empirical case studies of combining software quality classification models," in *3rd International Conference on Quality Software*, 2003, pp. 40–49.
[47] C. Seiffert, T. Khoshgoftaar, and J. Van Hulse, "Improving software-quality predictions with data sampling and boosting," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 39, no. 6, pp. 1283–1294, 2009.
[48] C. Seiffert, T. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "Rusboost: A hybrid approach to alleviating class imbalance," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 40, no. 1, pp. 185–197, 2010.
[49] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.
[50] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.
[51] Y. Freund and R. Schapire, "Experiments with a new boosting algorithm," in *International Conference on Machine Learning*, 1996, pp. 148–156.
[52] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera, "A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 42, no. 4, pp. 463–484, 2012.
[53] G. Anthony, H. Gregg, and M. Tshilidzi, "Image classification using svms: One-against-one vs one-against-all," in *28th Asian Conference on Remote Sensing*, 2007.
[54] A. Beygelzimer, J. Langford, and B. Zadrozny, "Weighted one-against-all," in *National Conference on Artificial Intelligence*, 2005, pp. 720–725.
[55] T. G. Dietterich and G. Bakiri, "Solving multiclass learning problems via error-correcting output codes," *Journal of Artificial Intelligence Research*, vol. 2, pp. 263–286, 1995.
[56] J. Milgram, M. Cheriet, and R. Sabourin, ""one against one" or "one against all": Which one is better for handwriting recognition with svms?" in *10th International Workshop on Frontiers in Handwriting Recognition*, 2006.
[57] R. Bose and D. Ray-Chaudhuri, "On a class of error correcting binary group codes*," *Information and Control*, vol. 3, no. 1, pp. 68–79, 1960.
[58] A. Hocquenghem, "Codes correcteurs derreurs," *Chiffres*, vol. 2, no. 2, pp. 147–56, 1959.
[59] G. Bakiri, "Converting english text to speech: A machine learning approach," Oregon State University, Tech. Rep., 1991.
[60] R. Debnath, N. Takahide, and H. Takahashi, "A decision based one-against-one method for multi-class support vector machine," *Pattern Analysis & Applications*, vol. 7, no. 2, pp. 164–175, 2004.
[61] C. Hsu and C. Lin, "A comparison of methods for multiclass support vector machines," *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 415–425, 2002.
[62] J. H. Friedman, "Another approach to polychotomous classification," Stanford University, Tech. Rep., 1996.
[63] T. Hastie and R. Tibshirani, "Classification by pairwise coupling," *The Annals of Statistics*, vol. 26, no. 2, pp. 451–471, 1998.
[64] D. H. Fisher, L. Xu, and N. Zard, "Ordering effects in clustering," in *9th International Workshop on Machine Learning*, 1992, pp. 162–168.
[65] P. Domingos, "Metacost: A general method for making classifiers cost-sensitive," in *5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1999, pp. 155–164.
[66] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.
[67] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *Journal of the American Statistical Association*, vol. 32, no. 200, pp. 675–701, 1937.
[68] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *The Journal of Machine Learning Research*, vol. 7, no. 1, pp. 1–30, 2006.
[69] P. Nemenyi, "Distribution-free multiple comparisons," Ph.D. dissertation, Princeton University, 1963.
[70] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "The misuse of the nasa metrics data program data sets for automated software defect prediction," in *15th Annual Conference on Evaluation & Assessment in Software Engineering*, 2011, pp. 96–103.

Zhongbin Sun received the BS degree in computer science from the Xi'an Jiaotong University, Xi'an, China, in 2010. He is currently a doctor student in the Department of Computer Science and Technology, Xi'an Jiaotong University. His research focuses on software engineering and data mining.

Qinbao Song received the Ph.D. degree in computer science from Xi'an Jiaotong University, Xi'an, China, in 2001. He is currently a Professor of software technology in the Department of Computer Science and Technology, Xi'an Jiaotong University, where he is also the Deputy Director of the Department of Computer Science and Technology. He is also with the State Key Laboratory of Software Engineering, Wuhan University, Wuhan, China. He has authored or coauthored more than 80 referred papers in the areas of machine learning and software engineering. He is a board member of the Open Software Engineering Journal. His current research interests include data mining/machine learning, empirical software engineering, and trustworthy software.

Xiaoyan Zhu received her MS degree in computer science and technology from Xi'an Jiaotong University, China, in 2008. She is currently a Ph.D. candidate at Xi'an Jiaotong University, Xi'an, China. Her research interests include software engineering and data mining.