Using Class Imbalance Learning for Software Defect Prediction

Shuo Wang, *Member, IEEE,* and Xin Yao, *Fellow, IEEE*

***Abstract*—To facilitate software testing and save testing cost, a wide range of machine learning methods have been studied to predict defects in software modules. Unfortunately, the imbalanced nature of this type of data increases the learning difficulty of such a task. Class imbalance learning specializes in tackling classification problems with imbalanced distributions, which could be helpful for defect prediction but has not been investigated in depth so far. In this paper, we study the issue of if and how class imbalance learning methods can benefit software defect prediction with the aim of finding better solutions. We investigate different types of class imbalance learning methods, including resampling techniques, threshold moving and ensemble algorithms. Among those we studied, AdaBoost.NC shows the best overall performance in terms of the measures including balance, G-mean and AUC. To further improve the performance of the algorithm and facilitate its use in software defect prediction, we propose a dynamic version of AdaBoost.NC, which adjusts its parameter automatically during training. Without the need to pre-define any parameters, it is shown to be more effective and efficient than the original AdaBoost.NC.**

***Index Terms*—Software defect prediction, class imbalance learning, ensemble learning, negative correlation learning.**

ACRONYMS

SDP	Software defect prediction
RUS	Random undersampling
RUS-bal	Balanced version of random undersampling
THM	Threshold-moving
BNC	AdaBoost.NC
SMOTE	Synthetic minority oversampling technique
SMB	SMOTEBoost
ROC	Receiver operating characteristic
AUC	Area under the curve
SVM	Support vector machine
NB	Naive Bayes with the log filter
RF	Random Forest
CV	Cross validation
PD	Probability of detection
PF	Probability of false alarm
DNC	Dynamic version of AdaBoost.NC

The authors are with the Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA), School of Computer Science, The University of Birmingham, Edgbaston, Birmingham B15 2TT, UK. E-mail: {S.Wang, X.Yao}@cs.bham.ac.uk

NOTATIONS

Z	Data set to be learnt
Z_{min}	Minority-class examples in Z (defect class)
Z_{maj}	Majority-class examples in Z (non-defect class)
N	Size of data set Z
N_{min}	Number of minority-class examples in Z
N_{maj}	Number of majority-class examples in Z
θ	Size ratio between majority and minority classes N_{maj}/N_{min}
O_{min}	Output of the classifier for the minority class
O_{maj}	Output of the classifier for the majority class
α	Decrement value used in the parameter searching strategy for RUS and RUS-bal
c	Cost coefficient used in the parameter searching strategy for THM
k	Number of nearest neighbors used in SMB
λ	Penalty strength for encouraging ensemble diversity in BNC and DNC
amb	Ambiguity term that assesses the ensemble diversity and is used in BNC and DNC
Acc	A chosen accuracy performance criterion for adjusting the parameter in DNC

I. INTRODUCTION

SOFTWARE DEFECT PREDICTION (SDP) can be formulated as a learning problem in software engineering, which has drawn growing interest from both academia and industry. Static code attributes are extracted from previous releases of software with the logs of defects and used to build models to predict defective modules for the next release. It helps to locate parts of software that are more likely to contain defects. This is particularly useful when the project budget is limited or the whole software system is too large to be tested exhaustively. A good defect predictor can guide software engineers to focus the testing on defect-prone parts of software.

For a high-performance defect predictor, researchers have been working on the choice of static attributes and effective learning algorithms since 1990s. The McCabe [1] and Halstead [2] metrics are widely used to describe the attributes of each software module (i.e. the unit of functionality of source code). In addition to seeking a better subset of attributes,

choosing a good learning algorithm was shown to be at least equally important to the final performance [3]. Various statistical and machine learning methods have been investigated for SDP, among which Naive Bayes [3] and Random Forest [4] [5] were shown to have relatively high and stable performance [6] [4]. AdaBoost based on C4.5 decision trees was also found to be effective in some studies [7] [8].

However, none of these studies have taken into consideration an important feature of the SDP problem, i.e., the highly imbalanced nature between the defect and non-defect classes of the data set. In most cases, the collected training data contains much more non-defective modules (majority) than defective ones (minority). The imbalanced distribution is a main factor accounting for the poor performance of certain machine learning methods especially on the minority class [6] [7]. Class imbalance learning is a growing research area in machine learning that aims to better deal with this kind of problems [9]. It includes a number of data-level and algorithm-level techniques. Several researchers noticed the negative effect of class imbalance on SDP and considered using class imbalance learning techniques to improve the performance of their predictors recently [10] [11] [12] [13] [8]. However, it is still unclear to what extent class imbalance learning can contribute to SDP and how to make better use of it to improve SDP.

As the first effort of an in-depth study of class imbalance learning methods in SDP, this paper explores their potential by focusing on the following research questions: can class imbalance learning methods be good solutions to SDP problems in comparison with the existing methods? What are their advantages and disadvantages? Can we make better use of them for different software projects efficiently? The answers will provide guidance and valuable information for choosing and designing good predictors for SDP.

For the first two questions, we conduct a systematic and comparative study of five class imbalance learning methods and two high-performance defect predictors on ten public data sets from real-world software projects. The five class imbalance learning methods are random undersampling (RUS), the balanced version of random undersampling (RUS-bal), threshold-moving (THM), AdaBoost.NC (BNC) [14] [15] and SMOTEBoost (SMB) [16], covering three major types of solutions to learning from imbalanced data. A parameter searching strategy is applied to these methods for deciding how much degree the minority class should be emphasized. They are then compared with Naive Bayes with the log filter and Random Forest, the two top-ranked methods in the SDP literature [3] [4]. Our results show that AdaBoost.NC and Naive Bayes are better choices among the seven algorithms. AdaBoost.NC has the best overall performance in terms of the measures – balance, G-mean and AUC. Naive Bayes has the best defect detection rate among all. Particularly, the performance evaluation metric "balance" is discussed in this paper as it is commonly used by software engineers in real SDP applications. One major challenge of using class imbalance learning methods is how to choose appropriate parameters, such as the sampling rate and misclassification cost of classes, which are crucial to their generalization on the minority class and can be time-consuming and problem-dependent to tune. Different SDP problems are shown to have their own best parameters.

To contribute to a wider range of SDP problems and simplify the training procedure, our next objective is to develop a better solution that combines the strength of AdaBoost.NC and Naive Bayes without the parameter setting issue as the answer to the third question. We propose a dynamic version of AdaBoost.NC that adjusts its parameter automatically during training based on a performance criterion. It is shown to be more effective and efficient than the original AdaBoost.NC in predicting defects and improving the overall performance. It offers the advantage of reduced training time, as no pre-defined parameters of emphasizing the minority class are required.

The rest of this paper is organized as follows. Section II gives the background knowledge about class imbalance learning and software defect prediction. Section III explains the experimental methodology, including the SDP data sets, the algorithms used in the experiments and our training strategy. Section IV discusses the experimental results, and addresses the parameter setting issue of class imbalance learning methods by proposing a better solution. Section V draws the conclusions and points out our future work.

II. Related Work

This section introduces the two focal points of this paper. First, we describe what problems class imbalance learning aims to solve and the state-of-the-art methods in this area. Subsequently, we briefly review the current research progress in software defect prediction.

A. Class Imbalance Learning

Class imbalance learning refers to learning from data sets that exhibit significant imbalance among or within classes. The common understanding about "imbalance" in the literature is concerned with the situation, in which some classes of data are highly under-represented compared to other classes [9]. By convention, we call the classes having more examples the majority classes, and the ones having fewer examples the minority classes. Misclassifying an example from the minority class is usually more costly. For SDP, due to the nature of the problem, the defect case is much less likely to happen than the non-defect case. The defect class is thus the minority. The recognition of this class is more important, because the failure of finding a defect could degrade software quality greatly.

The challenge of learning from imbalanced data is that the relatively or absolutely underrepresented class cannot draw equal attention to the learning algorithm compared to the majority class, which often leads to very specific classification rules or missing rules for the minority class without much generalization ability for future prediction [17]. How to better recognize data from the minority class is a major research question in class imbalance learning. Its learning objective can be generally described as "obtaining a classifier that will provide high accuracy for the minority class without severely jeopardizing the accuracy of the majority class" [9].

Numerous methods have been proposed to tackle class imbalance problems at data and algorithm levels. Data-level methods include a variety of resampling techniques, manipulating training data to rectify the skewed class distributions, such as random over/under-sampling and SMOTE [18]. They are simple and efficient, but their effectiveness depends greatly on the problem and training algorithms [19]. Algorithm-level methods address class imbalance by modifying their training mechanism directly with the goal of better accuracy on the minority class, including one-class learning [20] and cost-sensitive learning algorithms [21] [9]. Algorithm-level methods require specific treatments for different kinds of learning algorithms, which hinders their use in many applications, since we do not know in advance which algorithm would be the best choice in most cases. In addition to the aforementioned data-level and algorithm-level solutions, ensemble learning [22] [23] has become another major category of approaches to handling imbalanced data by combining multiple classifiers, such as SMOTEBoost [16] and AdaBoost.NC [14] [24]. Ensemble learning algorithms have been shown to be able to combine strength from individual learners and enhance the overall performance [25] [26]. They also offer additional opportunities to handle class imbalance at both the individual and ensemble levels.

This paper investigates two undersampling strategies [3] and two ensemble methods [16] [24] because of their simplicity, effectiveness and popularity in the literature. Threshold-moving is also considered in our study as a frequently used cost-sensitive technique [21]. Algorithm descriptions and settings will be given in the next section.

B. Software Defect Prediction

Defect predictors are expected to help to improve software quality and reduce costs of delivering those software systems. There is a rapid growth of SDP research after the PROMISE repository [27] was created in 2005. It includes a collection of defect prediction data sets from real-world projects for the public use and allows researchers to build repeatable and comparable models across studies. So far, great research efforts have been devoted to metrics describing code modules and learning algorithms to build predictive models for SDP.

For describing the attributes of a module, which is usually the smallest unit of functionality, static code metrics defined by McCabe [1] and Halstead [2] have been widely used and commonly accepted by most papers. McCabe metrics collect information of the complexity of pathways contained in the module through a flow graph. Halstead metrics estimate the reading complexity based on the number of operators and operands in the module. A more complex module is believed to be more likely to be fault-prone. Menzies et al. [3] showed the usefulness of these metrics for building defect predictors and suggested that the choice of learning method is far more important than seeking the best subsets of attributes for good performance.

A variety of machine learning methods have been proposed and compared for SDP problems, such as decision trees [28], neural networks [10] [29], Naive Bayes [30] [3], support vector machines [31] and Artificial Immune Systems [4]. It is discouraging but not surprising that no single method is found to be the best, due to different types of software projects, different algorithm settings and different performance evaluation criteria of assessing the models. Among all, Random Forest [32] appears to be a good choice for large data sets and Naive Bayes performs well for small data sets [33] [4]. However, they didn't consider the data characteristic of class imbalance.

Some researchers have noticed that the imbalanced distribution between defect and non-defect classes could degrade a predictor's performance greatly and attempted to use class imbalance learning techniques to reduce this negative effect. Menzies et al. [8] undersampled the non-defect class to balance training data and checked how little information was required to learn a defect predictor. They found that throwing away data does not degrade the performance of Naive Bayes and C4.5 decision trees and instead improves the performance of C4.5. Some other papers also showed the usefulness of resampling based on different learners [34] [13] [35]. Ensemble algorithms and their cost-sensitive variants were studied and shown to be beneficial if a proper cost ratio can be set [10] [36]. However, none of these studies have performed a comprehensive comparison among different class imbalance learning algorithms for SDP. It is still unclear in which aspect and to what extent class imbalance learning can benefit SDP problems, and which class imbalance learning methods are more effective. Such information would help us to understand the potential of class imbalance learning methods in this specific learning task and develop better solutions. Motivated by aforementioned studies, we will investigate class imbalance learning in terms of how it facilitates SDP and how it can be harnessed better to solve SDP more effectively through extensive experiments and comprehensive analyses next.

III. Experimental Methodology

This section describes the data sets, learning algorithms and evaluation criteria used in this study. The data sets we chose vary in imbalance rates, data sizes and programming languages. The chosen learning algorithms cover different types of methods in class imbalance learning.

A. Data Sets

All ten SDP data sets listed in Table I come from practical projects, which are available from the public PROMISE repository [27] to make sure that our predictive models are reproducible and verifiable and to provide an easy comparison to other papers. They are sorted in the order of the imbalance rate, i.e. the percentage of defective modules in the data set, varying from 6.94% to 32.29%. Each data sample describes the attributes of one "module/method", plus the class label of whether this module contains defects. The module attributes include McCabe metrics, Halstead metrics, lines of code, and some other attributes. It is worth mentioning that a repeated pattern of exponential distribution in the numeric attributes is observed in these data sets, formed by many small values and a few much larger values. Some work thus applied a logarithmic filter to all numeric values as a preprocessor, which appeared

to be useful for some types of learners [3]. For examples, the log filter was shown to improve the performance of Naive Bayes significantly but contribute very little to decision trees [37]. The data sets cover three programming languages. Data set jm1 contains a few missing values, which are removed before our experiment starts. Missing data handling techniques could be used instead in future work.

TABLE I: PROMISE data sets, sorted in order of the imbalance rate (defect%: the percentage of defective modules).

data	language	examples	attributes	defect%
mc2	C++	161	39	32.29
kc2	C++	522	21	20.49
jm1	C	10885	21	19.35
kc1	C++	2109	21	15.45
pc4	C	1458	37	12.20
pc3	C	1563	37	10.23
cm1	C	498	21	9.83
kc3	Java	458	39	9.38
mw1	C	403	37	7.69
pc1	C	1109	21	6.94

B. Learning Algorithms

In the following experiments, we will examine five class imbalance learning methods in comparison with two top-ranked learners in SDP. The two SDP predictors are Naive Bayes with the log filter (NB) [3] and Random Forest (RF) [5]. The five class imbalance learning methods are random undersampling (RUS), balanced random undersampling (RUS-bal, also called micro-sampling in [8]), threshold-moving (THM) [21] [38], SMOTEBoost (SMB) [16] and AdaBoost.NC (BNC) [14]. RUS and RUS-bal belong to data resampling techniques, shown to be effective in dealing with SDP [8] and outperform other resampling techniques such as SMOTE and random oversampling [34] [39]. RUS only undersamples the majority class, while RUS-bal undersamples both classes to keep them having the same size. THM is a simple and effective cost-sensitive method in class imbalance learning. It moves the output threshold of the classifier toward the inexpensive class based on the misclassification costs of classes such that defective modules become more costly to be misclassified. SMB is a popular ensemble learning method that integrates oversampling into Boosting [40]. It creates new minority-class examples by using SMOTE to emphasize the minority class at each round of training. Based on our previous finding that ensemble diversity (i.e. the disagreement degree among the learners in the ensemble) has a positive role in recognizing rare cases [41], BNC combined with random oversampling makes use of diversity to improve the generalization on the minority class successfully through a penalty term [24] [42].

Most class imbalance learning methods require careful parameter settings to control the strength of emphasizing the minority class prior to learning. The undersampling rate needs to be set for RUS and RUS-bal. THM requires misclassification costs of both classes. SMB needs to set the amount of new generated data and the number of nearest neighbors. BNC needs to set the strength of encouraging the ensemble diversity. Since the best parameter is always problem- and algorithm-dependent [19] [43], we apply a parameter searching strategy to each of the methods here, as shown in Fig. 1.

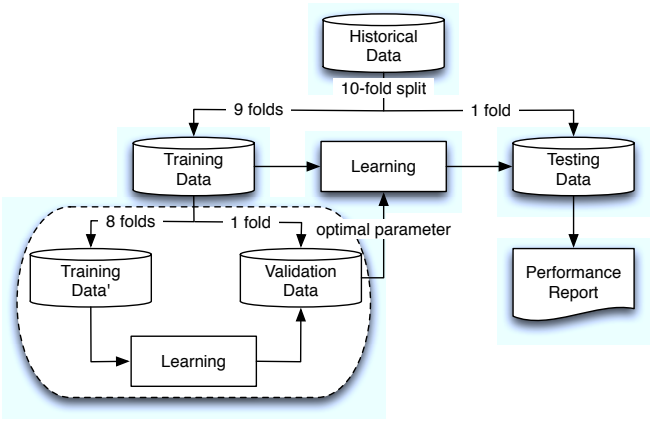


Fig. 1: Framework of our experimental studies.

Concretely, we employ 10-fold cross-validation (CV). At each time of building models using nine of the ten partitions, the nine data partitions are further split into a training set with eight partitions and a validation set with the remaining partition first. The learning method is repeated with different parameters on the same training set and evaluated on the validation set. The optimal parameter that results in the best performance on the validation set is then obtained. Using the best parameter, a model is trained based on data composed of the initial nine partitions and tested on the remaining one. The above procedure is repeated 100 times (10 folds * 10 independent runs) in total for each method. All methods and their parameter settings are described as follows.

Assuming a data set Z with N examples, Z_{min} is composed of the examples belonging to the defect class with size N_{min}, which is the minority. Likewise, Z_{maj} include examples belonging to the majority class with size N_{maj}. We define the size ratio between classes $\theta = N_{maj}/N_{min}$. Let O_{min} and O_{maj} denote the outputs of any classifier, which is capable of producing real-valued numbers as the estimation of the posterior probabilities of examples for the minority and majority classes respectively ($O_{min} + O_{maj} = 1$).

- RUS: it removes examples from Z_{maj} randomly. Concretely, we divide the difference between N_{maj} and N_{min} by 10 and use this value as a decrement α. After undersampling, the new size of the majority class $N'_{maj} = N_{maj} - n\alpha$ ($n = 0, 1, \ldots, 20$).
- RUS-bal: it removes examples from both classes randomly at different sampling rates until they reach the same pre-defined size. We choose the decrement of undersampling $\alpha = (N_{min} - 25)/10$ based on the findings in [8]. The new size of each class is $N_{min} - n\alpha$ ($n = 0, 1, \ldots, 10$) after undersampling.
- THM: we define a cost value c. The class returned by the classifier is the label with the larger output between

cO_{min} and O_{maj}. c is varied from 1 to 2θ with the increment of $\theta/10$.

- SMB: 51 classifiers are constructed based on the training data with SMOTE applied at each round of Boosting. The number of nearest neighbors k is 5 as recommended by the original paper [16]. The amount of new data at each round is set to $n \times N_{min}$ ($n = 1, 2, \ldots, 5$) respectively.
- BNC: as a prerequisite, random oversampling is applied to the minority class first to make sure both classes have the same size. Then 51 classifiers are constructed sequentially by AdaBoost.NC. The penalty strength λ for encouraging the ensemble diversity is varied from 1 to 20 with the increment of 1.

We use the well-known C4.5 decision tree learner [44] in the above methods in our experiment, as it is the most commonly discussed technique in the SDP literature. C4.5 models are built using the Weka software [45]. Default parameters are used except that we disable the tree pruning, because pruning may remove leaves describing the minority concept when data is imbalanced [46].

Regarding the two additional SDP techniques, the Naive Bayes classifier is built based on data preprocessed by the log filter. The Random Forest model is formed by 51 unpruned trees.

C. Evaluation Criteria

Due to the imbalanced distribution of SDP data sets and various requirements of software systems, multiple performance measures are usually adopted to evaluate different aspects of constructed predictors. There is a trade-off between the defect detection rate and the overall performance, and both are important.

To measure the performance on the defect class, the Probability of Detection (PD) and the Probability of False Alarm (PF) are usually used. PD, also called recall, is the percentage of defective modules that are classified correctly within the defect class. PF is the proportion of non-defective modules misclassified within the non-defect class. Menzies et al. claimed that a high-PD predictor is still useful in practice even if the other measures may not be good enough [47] [37].

For more comprehensive evaluation of predictors in the imbalanced context, G-mean [48] and AUC [49] are frequently used to measure how well the predictor can balance the performance between two classes. By convention, we treat the defect class as the positive class and the non-defect class as the negative class. A common form of G-mean is expressed as the geometric mean of recall values of the positive and negative classes. A good predictor should have high accuracies on both classes, and thus a high G-mean. In the SDP context, G-mean $= \sqrt{PD\,(1-PF)}$. It reflects the change in PD efficiently [50].

AUC estimates the area under the ROC curve, formed by a set of (PF, PD) pairs. The ROC curve illustrates the trade-off between detection and false alarm rates – the performance of a classifier across all possible decision thresholds. AUC provides a single numeric for performance comparison, varying in $[0, 1]$. A better classifier should produce a higher AUC. AUC is equivalent to the probability that a randomly chosen example of the positive class will have a smaller estimated probability of belonging to the negative class than a randomly chosen example of the negative class.

Since the point (PF=0, PD=1) is the ideal position on the ROC curve, where all defects are recognized without mistakes, the measure balance is introduced by calculating the Euclidean distance from the real (PF, PD) point to (0, 1) and frequently used by software engineers in practice [3]. By definition,

$$balance = 1 - \frac{\sqrt{(0-PF)^2 + (1-PD)^2}}{\sqrt{2}}.$$

In our experiment, we compute PD and PF for the defect class. Higher PDs and lower PFs are desired. We use AUC, G-mean and balance to assess the overall performance, which are expected to be high for a good predictor. The advantage of these five measures is their insensitivity to class distributions in data [51] [9].

IV. Class Imbalance Learning for SDP

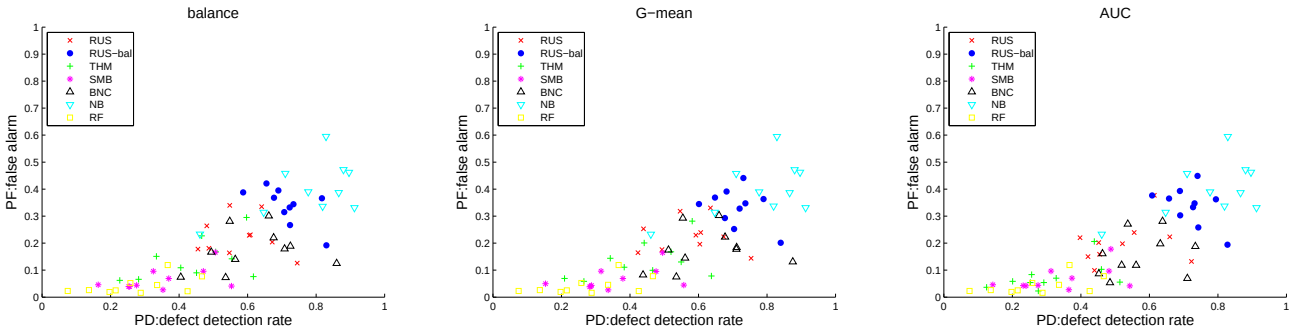
In this section, we first compare the performance of the five class imbalance learning methods based on the parameter searching strategy and the two existing SDP methods. The results will show their advantages and disadvantages. Based on the observations, we then improve them further.

A. Comparative Study

For each data set, we build seven predictive models following the algorithm settings described in the previous section. We use balance, G-mean and AUC, respectively, as the criterion for determining the best parameter of class imbalance learning methods under our training scheme. We use the Student T-test at confidence level of 95% for the statistical significance test.

For the defect class, Fig. 2 presents the scatter plots of (PD, PF) points from the seven training methods on the ten SDP data sets. Each plot has a different parameter searching criterion applied to the class imbalance learning methods. Each point is the average of 100 independent runs. Classifiers with more points distributed at the bottom right corner indicate better performance of higher defect detection rate and lower performance sacrifice on the non-defect class. We can gain the following results from Fig. 2. In terms of the defect detection rate (PD), NB outperforms all the five class imbalance learning models, which shows its effectiveness in finding defects. Although RUS-bal appears to be better at PD than other class imbalance learning models, it is still not as good as NB in most cases. In terms of the false alarm rate (PF), although RF is the best, it performs the worst in PD, which makes it hardly useful in practice. THM and SMB show better PD than RF, but their advantage is rather limited. BNC presents generally higher PD than RUS, THM and SMB, and lower PF than RUS-bal and NB.

To understand which measure is better to be the criterion for choosing parameters of class imbalance learning methods for SDP, we produce bar graphs in Fig. 3, displaying the average performance values of PD, balance, G-mean and AUC in the



(a) balance criterion (b) G-mean criterion (c) AUC criterion

Fig. 2: Scatter plots of (PD, PF) points of the seven training methods on the ten SDP data sets. Each point is the average from 100 independent runs. The five class imbalance learning methods use balance, G-mean and AUC as the parameter searching criterion respectively.

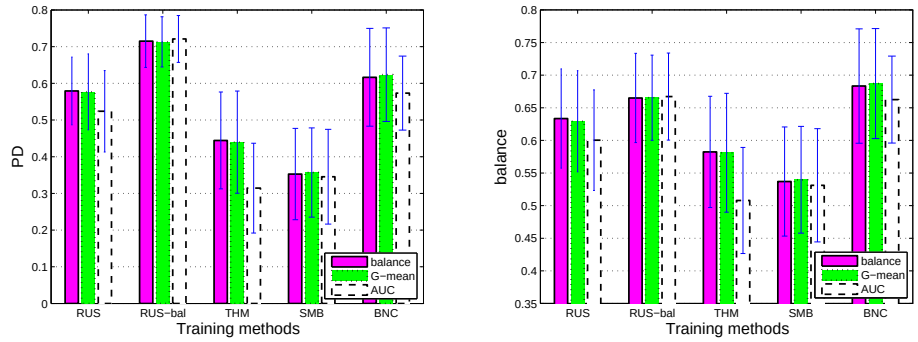
four sub-plots respectively (indicated on the Y-axis) over the ten data sets of each method with the corresponding standard deviation under the three chosen criteria. Every bar cluster in each plot compares the performance of the corresponding method (denoted on the X-axis) with different criteria. The results illustrate that balance and G-mean are better choices than AUC for deciding training parameters. In most cases, the models with the best parameters chosen based on balance and G-mean produce better PD, balance and G-mean than the ones trained based on AUC (Fig. 3(a)-(c)). Only the RUS-bal method seems to be robust to the chosen criterion. In its bar clusters, the three bars present very close results. Besides, AUC does not seem to be affected much by the chosen criterion. In Fig. 3(d), the three bars in all clusters have very similar height. There is evidence, showing that AUC is a more stable metric than the others [52]. Hence, different settings do not change AUC significantly. Using balance or G-mean would be more appropriate for the choice of training parameters.

Furthermore, among the five class imbalance learning methods, the bar plots show that BNC performs the best according to the three overall performance measures (Fig. 3(b)-(d)); RUS-bal performs the best and BNC comes to the second in terms of PD (Fig. 3(a)), which are consistent with our earlier observations from the scatter plots.

For more details about the overall performance, Tables II - IV show the mean and standard deviation values of balance, G-mean and AUC, respectively, produced by the seven training methods using balance as the parameter searching criterion. In each row, values in boldface are significantly better than the rest; there is no significant difference between the boldface ones. The significance test is based on Student T-test at confidence level of 95%. We can make following observations. BNC achieves the significantly best balance in 8 out of 10 cases, the significantly best G-mean in 6 cases and the significantly best AUC in 8 cases. More specifically, BNC achieves 3.2% improvement over NB and 19.7% improvement over RF in terms of balance on average. This is due to a 33.9% rise in PD over RF and a 22.3% reduction in PF over NB. Similar observations can be obtained for G-mean and AUC. Their improvement implies that BNC achieves a better balance between PD and PF. In other words, more defects are found without hurting the performance on the non-defect class much. For highly imbalanced data, such as kc3, mw1 and pc1, RUS-bal and NB present quite good balance and G-mean. It suggests that more aggressive techniques of emphasizing the minority class are more desirable for more imbalanced data and less likely to sacrifice the performance on the majority class, i.e. the non-defect class.

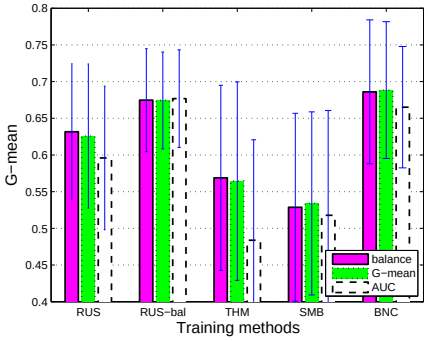
Finally, we show the optimal parameters obtained for the class imbalance learning methods based on the balance measure in Table V. The numbers in the "RUS" column indicate the percentage of the majority class that remains for training. Similarly, the numbers in brackets in the "RUS-bal" column indicate the percentages of the minority and majority classes kept for training after undersampling. The "THM" column includes the best misclassification cost for the minority class, when the cost for the majority class is fixed to one. The "SMB" column includes the best ratio of size of newly generated examples to the minority class size. The "BNC" column includes the best λ. We can see that the optimal values for each method varies greatly among different data sets, as expected. For the best performance, it is necessary to seek the optimal setting for every data domain. Our parameter searching strategy, as described in Fig. 1, can be adopted for tackling different problems. A general trend is that more imbalanced data sets need the algorithm to focus on the minority class more aggressively. For instance, in RUS, the percentage of the majority-class examples left for training in pc1 (the most imbalanced data) is much lower than that in mc2 (the least imbalanced data); in THM, the best misclassification cost of the minority class in pc1 is much higher than that in mc2.

To sum up, among the seven SDP and class imbalance learning methods, Naive Bayes is the winner according to PD and AdaBoost.NC is the winner based on the overall performance. From the viewpoint of the problem nature, the robustness of Naive Bayes to class imbalance in SDP implies that the extracted features are appropriate for describing the attributes of software code modules. Although the prior probability of the defect class is low, the statistical distribution of this class

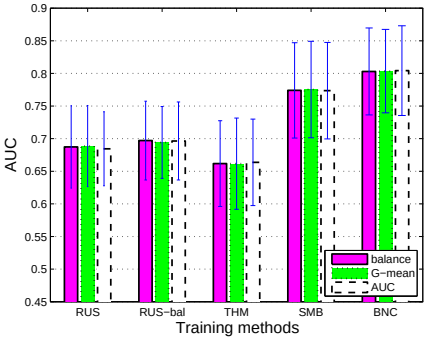


(a) mean PD

(b) mean balance



(c) mean G-mean



(d) mean AUC

Fig. 3: Bar plots of average PD, balance, G-mean and AUC with the corresponding standard deviation over the ten SDP data sets of the five class imbalance learning methods based on the parameter searching criteria of balance, G-mean and AUC.

TABLE II: Means and standard deviations of *balance* on the ten SDP data sets. Values in boldface are significantly better than the rest; there is no significant difference between the boldface ones from the same row.

balance	RUS	RUS-bal	THM	SMB	BNC	NB	RF
mc2	0.571±0.125	0.574±0.122	0.570±0.135	0.620±0.146	0.588±0.128	0.595±0.128	0.538±0.130
kc2	0.705±0.086	0.709±0.078	0.660±0.117	0.618±0.101	**0.753±0.084**	0.719±0.062	0.617±0.101
jm1	0.646±0.023	0.642±0.019	0.643±0.020	0.518±0.021	**0.678±0.017**	0.584±0.020	0.474±0.019
kc1	0.659±0.073	0.677±0.052	0.627±0.067	0.551±0.063	**0.718±0.050**	0.663±0.026	0.529±0.060
pc4	0.784±0.086	0.808±0.050	0.721±0.104	0.682±0.080	**0.854±0.049**	0.753±0.031	0.593±0.083
pc3	0.659±0.093	0.683±0.052	0.571±0.100	0.487±0.071	**0.749±0.078**	0.651±0.031	0.432±0.064
cm1	0.526±0.146	0.577±0.112	0.502±0.145	0.407±0.122	**0.607±0.118**	**0.663±0.284**	0.345±0.225
kc3	0.581±0.164	**0.669±0.114**	0.483±0.151	0.472±0.132	**0.655±0.144**	**0.693±0.129**	0.389±0.110
mw1	0.566±0.161	**0.619±0.126**	0.450±0.165	0.470±0.159	0.567±0.193	**0.636±0.137**	0.444±0.157
pc1	0.636±0.085	**0.688±0.078**	0.596±0.149	0.541±0.128	**0.665±0.126**	0.553±0.037	0.496±0.108

TABLE III: Means and standard deviations of *G-mean* on the ten SDP data sets. Values in boldface are significantly better than the rest; there is no significant difference between the boldface ones from the same row.

G-mean	RUS	RUS-bal	THM	SMB	BNC	NB	RF
mc2	0.571±0.150	0.578±0.140	0.566±0.173	0.627±0.169	0.597±0.135	0.607±0.137	0.535±0.191
kc2	0.720±0.081	0.720±0.080	0.675±0.120	0.642±0.108	**0.762±0.083**	0.734±0.070	0.647±0.105
jm1	0.649±0.022	0.645±0.018	0.647±0.019	0.541±0.025	**0.679±0.016**	0.594±0.020	0.494±0.025
kc1	0.670±0.071	0.686±0.050	0.643±0.064	0.582±0.072	**0.723±0.046**	0.694±0.031	0.562±0.072
pc4	0.799±0.076	0.816±0.053	0.747±0.097	0.723±0.077	**0.865±0.048**	0.780±0.036	0.638±0.089
pc3	0.669±0.095	0.690±0.057	0.588±0.110	0.504±0.099	**0.757±0.073**	0.680±0.039	0.423±0.122
cm1	0.485±0.254	0.584±0.143	0.452±0.250	0.301±0.257	0.597±0.216	**0.681±0.084**	0.152±0.225
kc3	0.563±0.247	0.682±0.123	0.430±0.268	0.422±0.262	0.665±0.175	**0.724±0.082**	0.252±0.265
mw1	0.540±0.253	**0.631±0.160**	0.336±0.315	0.388±0.304	0.525±0.306	**0.637±0.191**	0.334±0.313
pc1	0.645±0.157	**0.711±0.089**	0.601±0.195	0.553±0.191	**0.691±0.130**	0.576±0.050	0.505±0.167

TABLE IV: Means and standard deviations of *AUC* on the ten SDP data sets. Values in boldface are significantly better than the rest; there is no significant difference between the boldface ones from the same row.

AUC	RUS	RUS-bal	THM	SMB	BNC	NB	RF
mc2	0.615±0.133	0.623±0.135	0.639±0.116	**0.750±0.128**	0.690±0.130	0.700±0.150	0.722±0.118
kc2	0.730±0.087	0.726±0.092	0.687±0.092	0.742±0.092	**0.803±0.085**	**0.820±0.071**	**0.823±0.077**
jm1	0.665±0.023	0.658±0.020	0.661±0.024	0.691±0.017	0.733±0.017	0.679±0.019	**0.748±0.019**
kc1	0.710±0.063	0.713±0.059	0.670±0.058	0.755±0.046	**0.802±0.035**	0.785±0.038	**0.802±0.035**
pc4	0.823±0.069	0.827±0.063	0.791±0.068	0.923±0.023	**0.938±0.025**	0.863±0.035	**0.937±0.019**
pc3	0.689±0.087	0.694±0.073	0.664±0.071	0.813±0.052	**0.836±0.050**	0.777±0.054	**0.848±0.045**
cm1	0.622±0.126	0.622±0.129	0.595±0.139	0.704±0.112	**0.785±0.080**	0.748±0.091	0.742±0.105
kc3	0.643±0.183	0.686±0.157	0.590±0.151	0.740±0.133	**0.806±0.104**	**0.815±0.082**	**0.827±0.105**
mw1	0.647±0.145	0.681±0.131	0.584±0.142	**0.758±0.145**	**0.777±0.137**	**0.780±0.142**	**0.756±0.138**
pc1	0.726±0.129	0.739±0.098	0.735±0.116	0.851±0.080	**0.871±0.057**	0.700±0.096	0.847±0.067

TABLE V: The optimal parameter obtained from the training scheme based on balance on the ten SDP data sets, sorted in order of data imbalance rate.

Optimal	RUS(%)	RUS-bal(%)	THM	SMB	BNC
mc2	68	(71,34)	1.2	2.7	9
kc2	48	(55,14)	2.1	2.8	10
jm1	29	(62,15)	5.7	3.1	15
kc1	31	(52,10)	5.7	3.0	14
pc4	40	(56,8)	2.2	2.9	10
pc3	25	(54,6)	4.7	3.0	15
cm1	31	(63,8)	4.8	2.6	11
kc3	40	(75,8)	2.2	2.5	10
mw1	43	(80,7)	2.2	2.4	7
pc1	28	(63,5)	7.7	2.7	13

can be represented quite well by those features. Its posterior probability is thus rectified by summing the information from multiple features. Moreover, as claimed in [3], the defects may be actually associated in some log-normal way to the features. AdaBoost.NC is less aggressive in finding defects, as it tries to maintain the performance balance between classes. For highly imbalanced data, RUS-bal and Naive Bayes tend to be good choices. Random Forest is shown to be ineffective, as the bootstrapping training strategy and the tree learner are sensitive to class imbalance [53] [9]. The other class imbalance learning methods, RUS, THM and SMB, are all better than Random Forest. Based on the above results, Naive Bayes is recommended when a high hit rate of defects is more important (even at the cost of higher false alarm rates); otherwise, AdaBoost.NC could be a good choice.

B. Dynamic AdaBoost.NC

Given complementary strength of Naive Bayes and AdaBoost.NC, it would be ideal if we can find a predictor that combines their advantages. For practical algorithm application, it is also desirable to reduce the number of pre-defined parameters. In section IV-A, we applied a parameter searching strategy to each method in order to obtain an appropriate setting. In this section, we propose a dynamic version of AdaBoost.NC that can adjust its parameter automatically during training, with the goal of improving or at least maintaining the effectiveness of AdaBoost.NC without the exhaustive search for the best parameter.

Similarly to the parameter searching strategy we described before, we still split data into three parts: a training set, a validation set and a testing set. We make use of the sequential training framework of AdaBoost.NC to adjust its main parameter λ at each time of building the individual classifier, based on a chosen accuracy performance criterion (we use the "balance" measure in this section). We set an initial value of λ, such as 9, before the training starts. If the next classifier has a better balance on the validation set, we increase λ by 1 to emphasize the minority class further. Otherwise, we reduce λ by 1. By doing so, the minority class performance can be boosted as much as possible without hurting the overall performance.

TABLE VI: Dynamic AdaBoost.NC algorithm for binary classification.

Given data set $\{(x_1, y_1), \ldots, (x_i, y_i), \ldots, (x_m, y_m)\}$ and a chosen performance criterion Acc, initialize data weights $D_1(x_i) = 1/m$; penalty term $p_1(x_i) = 1$; penalty strength λ.

For training epoch $t = 1, 2, \ldots, T$:

Step 1. Train weak classifier f_t using distribution D_t.

Step 2. Get weak classifier f_t: $X \rightarrow R$.

Step 3. Calculate the penalty value for every example x_i:
$p_t(x_i) = 1 - |amb_t(x_i)|$.

Step 4. Calculate f_t's weight α_t by error and penalty
$$\alpha_t = \frac{1}{2}\log\frac{\sum_i D_t(x_i)(p_t(x_i))^{\lambda}(1+h_t(x_i)y_i)}{\sum_i D_t(x_i)(p_t(x_i))^{\lambda}(1-h_t(x_i)y_i)},$$
which is equivalent to
$$\alpha_t = \frac{1}{2}\log\left(\frac{\sum_{i,y_i=h_t(x_i)} D_t(x_i)(p_t(x_i))^{\lambda}}{\sum_{i,y_i\neq h_t(x_i)} D_t(x_i)(p_t(x_i))^{\lambda}}\right)$$
for discrete label outcome.

Step 5. If $Acc(f_t) \geq Acc(f_{t-1})$, then $\lambda = \lambda + 1$; else $\lambda = \lambda - 1$.

Step 6. Update data weights D_t and obtain new weights D_{t+1} by error and penalty:
$$D_{t+1}(x_i) = \frac{(p_t(x_i))^{\lambda} D_t(x_i) exp(-\alpha_t f_t(x_i) y_i)}{Z_t},$$
where Z_t is a normalization factor.

Output the final ensemble:
$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t f_t(x)\right)$.

The algorithm is described in Table VI. An "ambiguity" term (amb in step 3) decomposed from the classification error function is introduced into the weight-updating rule of

TABLE VII: Means and standard deviations of the five performance measures produced by DNC on the ten SDP data sets. Significantly better/worse values than BNC and NB are denoted by 'b' and 'n' superscripts/subscripts.

data	PD	PF	balance	G-mean	AUC
mc2	$0.521\pm0.222_{n}$	0.313 ± 0.175^{n}	0.569 ± 0.134	0.571 ± 0.152	$0.649\pm0.142_{b,n}$
kc2	$0.771\pm0.124_{n}^{b}$	$0.216\pm0.079_{b}^{n}$	$0.777\pm0.071^{b,n}$	0.777 ± 0.071^{n}	0.828 ± 0.074^{b}
jm1	0.660 ± 0.034^{n}	$0.314\pm0.018_{b,n}$	0.672 ± 0.027^{n}	$0.672\pm0.017_{b}^{n}$	$0.766\pm0.016^{b,n}$
kc1	$0.710\pm0.083_{n}^{b}$	0.241 ± 0.032	$0.733\pm0.042^{b,n}$	0.734 ± 0.040^{n}	$0.818\pm0.034^{b,n}$
pc4	$0.887\pm0.087_{n}^{b}$	$0.165\pm0.036_{b}^{n}$	0.850 ± 0.043^{n}	0.859 ± 0.047^{n}	$0.917\pm0.031_{b}^{n}$
pc3	$0.703\pm0.113_{n}$	$0.205\pm0.037_{b}^{n}$	0.739 ± 0.064^{n}	0.745 ± 0.061^{n}	$0.816\pm0.056_{b}^{n}$
cm1	$0.590\pm0.210_{n}^{b}$	$0.226\pm0.066_{b}^{n}$	0.653 ± 0.117^{b}	0.659 ± 0.133^{b}	0.787 ± 0.097^{n}
kc3	$0.579\pm0.237_{n}$	$0.197\pm0.082_{b}^{n}$	0.655 ± 0.143	$0.662\pm0.160_{n}$	0.797 ± 0.102
mw1	$0.486\pm0.273_{n}^{b}$	$0.138\pm0.057_{b}^{n}$	0.623 ± 0.177^{b}	0.647 ± 0.272^{b}	$0.714\pm0.139_{b,n}$
pc1	$0.570\pm0.201_{n}$	$0.107\pm0.041_{b}^{n}$	0.682 ± 0.133^{n}	0.698 ± 0.145^{n}	0.866 ± 0.081^{n}

Boosting (in step 6). It assesses the average difference between ensemble and its individuals, and is used to penalize training examples causing low diversity. The parameter λ controls the strength of applying the penalty, which is given an initial value before the algorithm start and then adjusted according to the measure "Acc" (step 5). Both accuracy and ensemble diversity are taken into account through the sequential training.

The improved AdaBoost.NC (denoted by "DNC") is compared to the original AdaBoost.NC (BNC) and Naive Bayes (NB) based on T-test at confidence level of 95%. Its performance outputs and the significance test results are shown in Table VII. The superscripts (subscripts) of 'b' and 'n' indicate that DNC's performance is significantly better (worse) than BNC and NB respectively.

In terms of the defect detection rate (PD), although DNC is still worse than Naive Bayes in 9 out of 10 cases, it outperforms BNC in 5 cases significantly and is competitive in the rest. It shows that AdaBoost.NC can find more defects by changing its parameter dynamically during training than that with a fixed parameter.

In terms of the overall performance, DNC performs significantly better than or at least comparably to BNC and Naive Bayes in all cases according to balance. According to G-mean, DNC performs significantly better than or comparably to BNC and Naive Bayes in 9 out of 10 cases. According to AUC, DNC outperforms BNC in 3 cases, is outperformed by BNC in 4 cases and is comparable to BNC in 3 remaining cases; DNC performs significantly better than or comparably to Naive Bayes in 8 cases. These results show that in general DNC has better or at least comparable overall performance to BNC and Naive Bayes. It can improve PD and overall performance without fixing the best parameter prior to learning.

V. Conclusions

The objective of SDP is to find defective software modules as many as possible without hurting the overall performance of the constructed predictor (e.g. without increasing the false alarm rate). The imbalanced distribution between classes in SDP data is a main cause of its learning difficulty but has not received much attention. This paper studied whether and how class imbalance learning can facilitate SDP. We investigated five class imbalance learning methods, covering three types (undersampling, threshold-moving, Boosting-based ensembles), in comparison with two top-ranked predictors (Naive Bayes and Random Forest) in the SDP literature. They were evaluated on ten real-world SDP data sets with a wide range of data sizes and imbalance rates. To ensure that the results presented in this paper are of practical value, five performance measures were considered, including PD, PF, balance, G-mean and AUC.

To fully discover the potential of using class imbalance learning methods to tackle SDP problems, we first searched for the best parameter setting for each method based on the balance, G-mean and AUC measures, since determining how much degree the defect class should be emphasized is crucial to their final performance. Then, random undersampling, the balanced random undersampling, threshold-moving, AdaBoost.NC and SMOTEBoost were compared with Naive Bayes with the log filter and Random Forest. The results show that AdaBoost.NC presents the best overall performance among all in terms of balance, G-mean and AUC. The balanced random undersampling has better defect detection rate (PD) than the other class imbalance learning methods, but it is still not as good as Naive Bayes. The balance and G-mean measures are shown to be better performance criteria than AUC for deciding algorithm parameters.

To further improve AdaBoost.NC and overcome the parameter setting issue, we proposed a dynamic version of AdaBoost.NC that can adjust its parameter automatically. It shows better PD and overall performance than the original AdaBoost.NC. It offers the advantage of reduced training time and more practical use, as no pre-defined parameters of emphasizing the minority class are required.

Future work of this paper includes the investigation of other base classifiers. Currently, this paper only considered C4.5 decision trees. In addition, it is important to look into more practical scenarios in SDP, such as learning from data with very limited defective modules and many unlabeled modules (semi-supervised), and defect isolation to determine the type of defects (multi-class imbalance).

Acknowledgment

This work was supported by EPSRC (Grant Nos. EP/D052785/1 and EP/J017515/1) on "SEBASE: Software Engineering By Automated SEarch" and "DAASE: Dynamic Adaptive Automated Software Engineering". Part of writing was completed while the first author was visiting Xidian University, China, supported by an EU FP7 IRSES grant on

"NICaiA: Nature Inspired Computation and its Applications" (Grant No. 247619).

REFERENCES

[1] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308–320, 1976.

[2] M. H. Halstead, *Elements of Software Science*. Elsevier, 1977.

[3] 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.

[4] C. Catal and B. Diri, "Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem," *Information Sciences*, vol. 179, no. 8, pp. 1040–1058, 2009.

[5] Y. Ma, L. Guo, and B. Cukic, "A statistical framework for the prediction of fault-proneness," *Advances in Machine Learning Applications in Softwre Engineering*, pp. 237–265, 2006.

[6] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic review of fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, 2011 (DOI: 10.1109/TSE.2011.103).

[7] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *Journal of Systems and Software*, vol. 83, no. 1, pp. 2–17, 2010.

[8] T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic, and Y. Jiang, "Implications of ceiling effects in defect predictors," in *The 4th International Workshop on Predictor Models in Software Engineering (PROMISE 08)*, 2008, pp. 47–54.

[9] 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.

[10] J. Zheng, "Cost-sensitive boosting neural networks for software defect prediction," *Expert Systems with Applications*, vol. 37, no. 6, pp. 4537–4543, 2010.

[11] Y. Kamei, A. Monden, S. Matsumoto, T. Kakimoto, and K. ichi Matsumoto, "The effects of over and under sampling on fault-prone module detection," in *International Symposium on Empirical Software Engineering and Measurement*, 2007, pp. 196–204.

[12] T. M. Khoshgoftaar, K. Gao, and N. Seliya, "Attribute selection and imbalanced data: Problems in software defect prediction," in *22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 2010, pp. 137–144.

[13] J. C. Riquelme, R. Ruiz, D. Rodriguez, and J. Moreno, "Finding defective modules from highly unbalanced datasets," *Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos*, vol. 2, no. 1, pp. 67–74, 2008.

[14] S. Wang, H. Chen, and X. Yao, "Negative correlation learning for classification ensembles," in *International Joint Conference on Neural Networks, WCCI*. IEEE Press, 2010, pp. 2893–2900.

[15] S. Wang and X. Yao, "The effectiveness of a new negative correlation learning algorithm for classification ensembles," in *IEEE International Conference on Data Mining Workshops*, 2010, pp. 1013–1020.

[16] N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer, "Smoteboost: Improving prediction of the minority class in boosting," in *Knowledge Discovery in Databases: PKDD 2003*, vol. 2838, 2003, pp. 107–119.

[17] G. M. Weiss, "Mining with rarity: a unifying framework," *SIGKDD Explor. Newsl.*, vol. 6, no. 1, pp. 7–19, 2004.

[18] 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. 341–378, 2002.

[19] A. Estabrooks, T. Jo, and N. Japkowicz, "A multiple resampling method for learning from imbalanced data sets," in *Computational Intelligence 20*, vol. 20, no. 1, 2004, pp. 18–36.

[20] N. Japkowicz, C. Myers, and M. A. Gluck, "A novelty detection approach to classification," in *IJCAI*, 1995, pp. 518–523.

[21] Z.-H. Zhou and X.-Y. Liu, "Training cost-sensitive neural networks with methods addressing the class imbalance problem," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 1, 2006, pp. 63– 77.

[22] T. K. Ho, J. J. Hull, and S. N. Srihari, "Decision combination in multiple classifier systems," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 1, pp. 66–75, 1994.

[23] L. Rokach, "Ensemble-based classifiers," *Artificial Intelligence Review*, vol. 33, no. 1-2, pp. 1–39, 2010.

[24] S. Wang and X. Yao, "Negative correlation learning for class imbalance problems," School of Computer Science, University of Birmingham, Tech. Rep., 2012.

[25] G. Brown, J. L. Wyatt, and P. Tino, "Managing diversity in regression ensembles," *Journal of Machine Learning Research*, vol. 6, pp. 1621–1650, 2005.

[26] K. Tang, P. N. Suganthan, and X. Yao, "An analysis of diversity measures," *Machine Learning*, vol. 65, pp. 247–271, 2006.

[27] G. Boetticher, T. Menzies, and T. J. Ostrand. (2007) Promise repository of empirical software engineering data. [Online]. Available: http://promisedata.org/repository

[28] T. M. Khoshgoftaar and N. Seliya, "Tree-based software quality estimation models for fault prediction," in *Proceedings of 8th IEEE Symposium on Software Metrics*, 2002, pp. 203–214.

[29] M. M. T. Thwin and T.-S. 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.

[30] B. Turhan and A. Bener, "Analysis of naive bayes' assumptions on software fault data: An empirical study," *Data and Knowledge Engineering*, vol. 68, no. 2, pp. 278–290, 2009.

[31] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "Using the support vector machine as a classification method for software defect prediction with static code metrics," *Engineering Applications of Neural Networks*, vol. 43, pp. 223–234, 2009.

[32] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, October 2001.

[33] C. Catal, "Software fault prediction: A literature review and current trends," *Expert Systems with Applications*, vol. 38, no. 4, pp. 4626–4636, 2010.

[34] L. Pelayo and S. Dick, "Evaluating stratification alternatives to improve software defect prediction," *IEEE Transactions on Reliability*, 2012 (DOI: 10.1109/TR.2012.2183912).

[35] ——, "Applying novel resampling strategies to software defect prediction," in *Annual Meeting of the North American Fuzzy Information Processing Society*, 2007, pp. 69–72.

[36] T. M. Khoshgoftaar, E. Geleyn, L. Nguyen, and L. Bullard, "Cost-sensitive boosting in software quality modeling," in *Proceedings of 7th IEEE International Symposium on High Assurance Systems Engineering*, 2002, pp. 51–60.

[37] 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.

[38] K. M. Ting, "An instance-weighting method to induce cost-sensitive trees," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 3, pp. 659–665, 2002.

[39] C. Seiffert, T. M. Khoshgoftaar, and J. V. 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.

[40] R. E. Schapire, "The boosting approach to machine learning: An overview," in *MSRI Workshop on Nonlinear Estimation and Classification*, 2002, pp. 1–23.

[41] S. Wang and X. Yao, "Relationships between diversity of classification ensembles and single-class performance measures," *IEEE Transactions on Knowledge and Data Engineering*, 2011 (DOI: 10.1109/TKDE.2011.207).

[42] ——, "Multi-class imbalance problems: Analysis and potential solutions," *IEEE Transactions on Systems, Man and Cybernetics, PartB: Cybernetics*, vol. 42, no. 4, pp. 1119–1130, 2012.

[43] J. V. Hulse, T. M. Khoshgoftaar, and A. Napolitano, "Experimental perspectives on learning from imbalanced data," in *ICML '07: Proceedings of the 24th international conference on Machine learning*, 2007, pp. 935–942.

[44] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[45] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. San Francisco, CA: Morgan Kaufmann., 2005.

[46] N. V. Chawla, "C4.5 and imbalanced data sets: Investigating the effect of sampling method, probabilistic estimate, and decision tree structure," in *Workshop on Learning from Imbalanced Datasets II, ICML, Washington DC, 2003.*, 2003, pp. 1–8.

[47] 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.

[48] M. Kubat and S. Matwin, "Addressing the curse of imbalanced training sets: One-sided selection," in *Proc. 14th International Conference on Machine Learning*, 1997, pp. 179–186.
[49] A. P. Bradley, "The use of the area under the roc curve in the evaluation of machine learning algorithms," *Pattern Recognition*, vol. 30, no. 7, pp. 1145–1159, 1997.
[50] 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. 2-3, pp. 195–215, 1998.
[51] T. Fawcett, "Roc graphs: Notes and practical considerations for researchers," *HP Labs, Palo Alto, CA, Technical Report HPL-2003-4*, 2003.
[52] S. Wang, "Ensemble diversity for class imbalance learning," Ph.D. dissertation, School of Computer Science, The University of Birmingham, 2011.
[53] X. Zhu, "Lazy bagging for classifying imbalanced data," in *Seventh IEEE International Conference on Data Mining*, 2007, pp. 763–768.

Shuo Wang received the B.Sc. degree in Computer Science from the Beijing University of Technology (BJUT), China, in 2006, and was a member of Embedded Software and System Institute in BJUT in 2007. She received the Ph.D. degree in Computer Science from the University of Birmingham, U.K., in 2011, sponsored by the Overseas Research Students Award (ORSAS) from the British Government (2007). She is currently a post-doctoral Research Fellow at the Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA) in the School of Computer Science, the University of Birmingham. Her research interests include class imbalance learning, ensemble learning, machine learning and data mining.

Xin Yao (M'91-SM'96-F'03) is a Chair (Professor) of Computer Science at the University of Birmingham, UK. He is the Director of CERCIA (the Centre of Excellence for Research in Computational Intelligence and Applications), University of Birmingham, UK, and of the Joint USTC-Birmingham Research Institute of Intelligent Computation and Its Applications. He is an IEEE Fellow and a Distinguished Lecturer of IEEE Computational Intelligence Society (CIS). He won the 2001 IEEE Donald G. Fink Prize Paper Award, 2010 IEEE Transactions on Evolutionary Computation Outstanding Paper Award, 2010 BT Gordon Radley Award for Best Author of Innovation (Finalist), 2011 IEEE Transactions on Neural Networks Outstanding Paper Award, and many other best paper awards at conferences. He won the prestigious Royal Society Wolfson Research Merit Award in 2012 and was selected to receive the 2013 IEEE CIS Evolutionary Computation Pioneer Award. He was the Editor-in-Chief (2003-08) of IEEE Transactions on Evolutionary Computation. He has been invited to give more than 65 keynote/plenary speeches at international conferences in many different countries. His major research interests include evolutionary computation and neural network ensembles. He has more than 400 refereed publications in international journals and conferences. According to Google Scholar, his H-index is 57 and total citations more than 17,000.