Class Imbalance Learning for Software Defect Prediction

Ashwini N
Dept. of CSE
AMC Engineering College
Bangaluru-560083
ashwini.jnr@gmial.com

Bharathi R
Asso. Prof., Dept. of CSE
AMC Engineering College
Bangaluru-560083
bharathi_saravanan@hotmail.com

Abstract- The objective of Software Defect Prediction is to find as many defective software modules as possible without hurting the overall performance of the constructed predictor. The imbalanced distribution of SDP data is a main cause of its learning difficulty, but has not received much attention.Class imbalance learning is an important research area in machine learning, where instances in some classes heavily outnumber the instances in other classes. This unbalanced class distribution causes performance degradation. In this paper, we study 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. Among those methods, AdaBoost.NC shows the best overall performance in terms of the measures including balance, G-mean, and Area Under the Curve (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.

Keywords–* *Software Defect Prediction, Class Imbalance Learning, Ensemble Learning, Sampling.

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 characterstic
AUC	Area under the curve

Corresponding authors Mrs.Bharathi R, and Ashwini N are with AMCEC, VTU, Belgaum, Karanataka, India.

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 log files of defects, and used to build models to predict defective modules for the next release. It helps to locate parts of the software that are more likely to contain defects. This effort 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 the software.

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, stable performance [4][6]. 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 contain 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 problem [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 [8][10]-[13]. However, it is still unclear what extent class imbalance learning can contribute to SDP, and how to make better use of it to improve SDP.

We conduct a systematic 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 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]. AdaBoost.NC has the best overall performance in terms of the three measures: *balance*, G-mean, and Area Under the Curve (AUC). Naive Bayes has the best defect detection rate among all.

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 Naïve 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 at predicting defects, and improving the overall performance. It can reduce training time, as no pre-defined parameters for 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.

II. BACKGORUND

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 a classification problem, where the data set presents skewed class distributions. For a typical imbalanced data set with two classes, one class is heavily underrepresented compared to the other class that contains a relatively large number of examples. Class imbalance pervasively exists in many real-world applications, such as medical diagnosis [1], fraud detection[2], risk management [3], text classification [4], etc. Rare cases in these domains suffer from higher misclassification costs than common cases. Finding minority class examples effectively and accurately without losing overall performance is the objective of class imbalance learning.

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 oversampling, random undersampling, 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[9][21]. Algorithm-level methods require specific treatments for different kinds of learning algorithms, which hinders their use in many applications, because 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.

B. Software Defect Prediction

Defect predictors are expected to help to improve software quality and reduce the 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 public use, and allows researchers to build repeatable, 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. McCabe metrics collect information about 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 to performance than seeking the best subsets of attributes.

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 [3][30], support vector machines, 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 for assessing the models. Among all, Random Forest appears to be a good choice for large data sets, and Naive Bayes performs well for small data sets [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 [13].

Ensemble algorithms and their cost-sensitive variants were studied, and shown to be beneficial if a proper cost ratio can be set [10]. 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 next 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.

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 Collection

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. The data sets are sorted in order of the imbalance rate, i.e. the percentage of defective modules in the data set, varying from 6.94% to 32.29%.

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
kc2	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

Each data sample describes the attributes of one module or method (hereafter referred to as module), plus the class label of whether this module contains defects. The module attributes include McCabe metrics, Halstead metrics, lines of code, and other attributes. It is worth mentioning that a repeated pattern of an exponential distribution in the numeric attributes is observed in these data sets, formed by many small values combined with 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 example, the log filter was shown to improve the performance of Naive Bayes significantly, but contributed very little to decision trees. 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.

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], SMOTEBoost (SMB) [16], and AdaBoost.NC (BNC) [14]. RUS and RUS-bal are data resampling techniques, shown to be effective in dealing with SDP [8], and they outperform other resampling techniques such as SMOTE and random oversampling. RUS only undersamples the majority class, while RUS-bal undersamples both classes to keep them having the same size. THM is a simple, 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.

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, BNC combined with random oversampling makes use of diversity to improve the generalization on the minority class successfully through a penalty term [24].

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 the 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. Because the best parameter is always problem- and algorithm- dependent, we apply a parameter searching strategy to each of the methods here, as shown in Fig. 1.

Concretely, we employ 10-fold cross-validation (CV). For each run, we use nine of the ten partitions to build models, which will then be tested on the remaining partition. Before the training starts, the nine data partitions are further split into

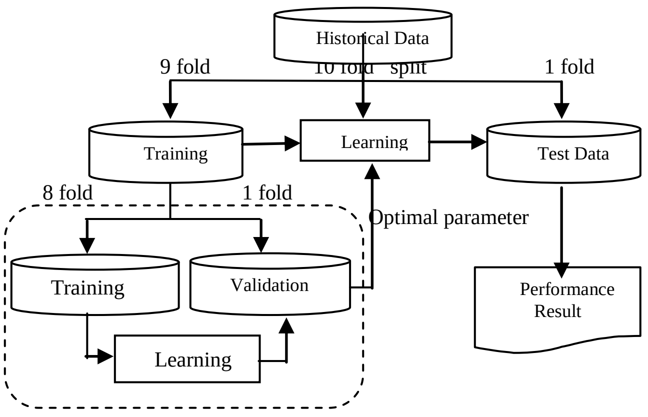

Fig. 1. 10-fold cross validation used in our experimental studies

a training set with eight partitions, and a validation set with the remaining partition. 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. 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 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 = 0,1,2,…..,20).

- RUS-bal removes examples from both classes randomly at different sampling rates until they reach the same predefined 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,…,10) after undersampling.

- THM uses 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 proceeds with 51 classifiers 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 [16]. The amount of new data at each round is set to $n \times N_{min}$ (n = 1,2,……,5) respectively.
- BNC has, random oversampling 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 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 . Default parameters are used, except that we disable the tree pruning, because pruning may remove leaves describing the minority concept when data are imbalanced.

Regarding the two additional SDP techniques, the Naïve 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.

For more comprehensive evaluation of predictors in the imbalanced context, G-mean, and AUC 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 = $\sqrt{PD(1\text{-}PF)}$. It reflects the change in PD efficiently. 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, which serves as the performance of a classifier across all possible decision thresholds. AUC provides a single number 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.

Because 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 is frequently used by software engineers in practice [3]. By definition,

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

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. All of them are expected to be high for a good predictor. The advantage of these five measures is their insensitivity to class distributions in data [9].

IV. Conclusions

To facilitate software testing, and save testing costs, a wide range of machine learning methods have been studied to predict defects in software modules. 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. 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, because determining how much degree the defect class should be emphasized is crucial to their final performance. Then, random undersampling, 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 a better defect detection rate (PD) than the other class imbalance learning methods, but it is still not as good as Naïve 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 from 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).

REFERENCES

[1] T. J. McCabe, “A complexity measure,” *IEEE Trans. Software Eng.*,vol. 2, no. 4, pp. 308–320, Feb. 1976.

[2] M. H. Halstead, *Elements of Software Science*.. NewYork,NY,USA:Elsevier, 1977.

[3] T. Menzies, J. Greenwald, and A. Frank, “Data mining static code attributes to learn defect predictors,” *IEEE Trans. Software Eng.*, vol. 33, no. 1, pp. 2–13, Jan. 2007.

[4] C. Catal and B. Diri, “Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem,” *Inf. Sci.*, 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,” *Adv. Mach. Learn. Appl. Softwre Eng.*, 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 Trans. Software Eng.*, vol. 38, no. 6, pp. 1276–1304, Nov.-Dec. 2012.

[7] E. Arisholm, L. C. Briand, and E. B. Johannessen, “A systematic and comprehensive investigation ofmethods to build and evaluate fault prediction models,” *J. Syst. 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 *Proc. 4th Int.Workshop PredictorModels Software Eng. (PROMISE 08)*, 2008, pp. 47–54.

[9] H. He and E. A. Garcia, “Learning fromimbalanced data,” *IEEE Trans. Knowledge Data Eng.*, vol. 21, no. 9, pp. 1263–1284, Sep. 2009.

[10] J. Zheng, “Cost-sensitive boosting neural networks for software defect prediction,” *Expert Syst. Appl.*, 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-pronemodule detection,” in *Proc. Int. Symp. Empirical Software Eng.Measur.*, 2007, pp. 196–204.

[12] T.M.Khoshgoftaar,K.Gao, andN. Seliya, “Attribute selection and imbalanced data: Problems in software defect prediction,” in *Proc. 22nd IEEE Int. Conf. Tools Artif. Intell. (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 *Proc. Int. Joint Conf. Neural Netw., WCCI*, 2010, pp. 2893–2900.

[15] S. Wang and X. Yao, “The effectiveness of a new negative correlation learning algorithm for classification ensembles,” in *Proc. IEEE Int. Conf. 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,” *Knowledge Discovery in Databases: PKDD 2003*, vol. 2838, pp. 107–119,
2003.

[17] G. M. Weiss, “Mining with rarity: A unifying framework,” *ACM SIGKDD Explor. Newslett.*, 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,” *J. Artif. Intell. Res.*, vol. 16, pp. 341–378, 2002.

[19] A. Estabrooks, T. Jo, and N. Japkowicz, “A multiple resampling method for learning from imbalanced data sets,” *Comput. Intell. 20*, vol. 20, no. 1, pp. 18–36, 2004.

[20] N. Japkowicz, C. Myers, and M. A. Gluck, “A novelty detection approach to classification,” in *Proc. 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,” *IEEE Trans. Knowledge, Data Eng.*, vol. 18, no. 1, pp. 63–77, Jan. 2006.

[22] T. K. Ho, J. J. Hull, and S. N. Srihari, “Decision combination in multiple classifier systems,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 1, pp. 66–75, Jan. 1994.

[23] L. Rokach, “Ensemble-based classifiers,” *Artif. Intell. Rev.*, 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, 2012, Tech. Rep..

[25] G. Brown, J. L. Wyatt, and P. Tino, “Managing diversity in regression ensembles,” *J. Mach. Learn. Res.*, vol. 6, pp. 1621–1650, 2005.

[26] K. Tang, P. N. Suganthan, and X. Yao, “An analysis of diversity measures,” *Mach. Learn.*, 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 *Proc. 8th IEEE Symp. 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-orientedmetrics,” *J. Syst. 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 Knowledge Eng.*, vol. 68, no. 2, pp. 278–290, 2009.