IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON SOFTWARE STANDARDS AND THEIR IMPACT IN REDUCING SOFTWARE FAILURES

Received January 30, 2018, accepted March 1, 2018, date of publication March 20, 2018, date of current version May 24, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2817572

An Ensemble Oversampling Model for Class Imbalance Problem in Software Defect Prediction

SHAMSUL HUDA[1], KEVIN LIU [1], MOHAMED ABDELRAZEK[1], AMANI IBRAHIM[1], SULTAN ALYAHYA[2], HMOOD AL-DOSSARI[2], AND SHAFIQ AHMAD[3]

[1]School of IT, Deakin University, Melbourne VIC 3125, Australia
[2]Information Systems Department, King Saud University, Riyadh 11451, Saudi Arabia
[3]Department of Industrial Engineering, College of Engineering, King Saud University, Riyadh 11451, Saudi Arabia

Corresponding author: Shamsul Huda (shamsul.huda@deakin.edu.au)

This work was supported by the Deanship of Scientific Research at King Saud University under Grant RGP-1436-039.

ABSTRACT Software systems are now ubiquitous and are used every day for automation purposes in personal and enterprise applications; they are also essential to many safety-critical and mission-critical systems, e.g., air traffic control systems, autonomous cars, and SCADA systems. With the availability of massive storage capabilities, high speed Internet, and the advent of Internet of Things devices, modern software systems are growing in both size and complexity. Maintaining a high quality of such complex systems while manually keeping the error rate at a minimum is a challenge. Therefore, automated detection of faulty components in a software system is important during software development and also post-delivery. Fault detection models usually needs to be trained on a labeled-balanced dataset with both faulty and non-faulty samples. Earlier work, e.g. Mohsin *et al.* (2016), showed that most real fault detection training dataset are imbalanced. Thereby, the trained model gets over-fitted and classifies faulty components as non-faulty components. The consequence of a high false negative rate is cumulative and results in generating more errors when using the model in other software systems –never seen before, which is very expensive. In this paper, we propose a software defect prediction ensemble model which considers the class imbalance problem in real software datasets. We use different oversampling techniques to build an ensemble classifier that can reduce the effect of low minority samples in the defective data. The proposed approach is verified using PROMISE software engineering datasets. The results show that our ensemble oversampling technique can more greatly reduce the false negative rate compared to the standard classification techniques and identify the faulty components more accurately resulting in a less expensive detection system (lowering the rate of non-faulty predictions of faulty modules).

INDEX TERMS Software quality and fault detection, imbalanced metric data, ensemble model of detection, oversampling, highly accurate detection.

I. INTRODUCTION

Software is currently being used to automate and operate most of the processes and tasks we have today, including many safety-critical and mission-critical systems, e.g., air traffic control systems, autonomous cars, SCADA systems, personal applications, and enterprise applications. Testing such software systems is very challenging and time consuming. Software spending worldwide amounted to $3.8 trillion in 2014 [1]–[3], with testing and other quality assurance activities accounting for 23% of the spending [1]–[3]. This percentage reflects the criticality and complexity of the software testing task as a part of the software development life cycle (SDLC) [3].

The challenge with modern software engineering is that systems have become more complex and dynamic with the widespread adoption of new continuous delivery models, where new features can be pushed into production faster. Hence, automated software testing and software defect detection is very critical to speeding up this process and cutting down the cost to locate and analyze for defects.

Most of the testing strategies currently available cannot guarantee better than 40% for statement coverage [3], [4]. Thus, there is always a need for more effective detection techniques to spot defect-prone components in the software. Locating and fixing a software defect after deployment in a production environment is far more expensive than during the

development process; it is estimated to be 100 times more expensive to fix a defect in production [3]–[5].

Deciding if a component is defective has been proved to have a strong relationship with some software metrics (feature vector), including McCabe metrics [5], [6], Halstead metrics [6], etc. Hence, automated prediction of defective components (defective vs. non-defective) from extracted software metrics is a very active research area [6]. Many approaches have used machine learning (ML) techniques such as Naive Bayes (NB) [7]–[9], support vector machines (SVM) [7], [9], and decision trees [8], and neural networks [9], [10] have been proposed for software defect detection based on the measure of internal metrics and defect data from similar projects or earlier releases to construct defect detection models.

However, Menzies *et al.* [11] proved that different datasets lead to different prediction accuracy [11], and often the accuracy depends on different metrics/features that change from one dataset to another. It is recommended to use all features when conducting defect prediction [11]. The traditional challenge in having a dataset with too many features and limited instances (data points) to use for training is that it becomes very challenging to achieve high accuracy. Moreover, in case of the defect prediction problem, the dataset usually has a class imbalance problem, i.e., the number of instances that represent the ''defective'' class is far less compared to the number of instances that represent the ''non-defective'' class. This usually causes the classification techniques to give less weight (importance) to the instances from the defective class.

The different strategies for solving the class imbalance problem include: 1) apply feature reduction/selection techniques [12], which affects accuracy; remove some instances from the non-defective class under-sampling, which affects the accuracy of the classification technique; or setting a higher cost for the misclassification of defective instances [12]; or 2) add more examples (instances) to fix the imbalance problem [11], [12] by over sampling.

Earlier work by Ali *et al.* [12], proposed an approach by developing a parallel framework-based significant metric selection and a fault identification technique using a hybrid wrapper-filter approach [12]. That work [12] developed a computationally efficient classification model, which achieved high accuracy to some extent. However, in that approach [12], the false negative rate was not significantly reduced [12].

In this paper, we introduce a novel hybrid ensemble of oversampling strategy to generate more pseudo instances from the defective classes. Our approach uses a combination of random oversampling [13], Majority Weighted Minority Oversampling Technique [13], and Fuzzy-Based Feature-Instance Recovery [14] to build an ensemble classifier. Our proposed approach can minimize the effect of imbalance distribution of classes in the training data to reduce false negative rate and improve cost sensitive classification performance.

The rest of the paper is presented as follows. The class imbalance problem and related research in software fault detection are discussed in Section II. In Section III, the proposed hybrid-ensemble oversampling approach is presented. The datasets used in this paper and the experimental results are explained in Section IV. Section V concludes.

II. CLASS IMBALANCE PROBLEM FOR SOFTWARE DEFECT PREDICTION

A class imbalance problem occurs when a class of data is highly under-represented (minority) compared to the other class (majority) in a given data set [15]. Class imbalance is a common problem when learning the behavior and attributes that characterize rare scenarios, e.g., security anomalies and risk management. Similarly, in software defect prediction, the defective class usually has fewer instances and are less likely to occur. However, this defective class is the most important class – it is the class which we want to learn and be able to predict. Due to the under-representation of the defective class, it is usually hard to learn with high accuracy and soundness [16]. The solutions to this problem are found in the literature as below:

The class imbalance problem was addressed in the literature by adjusting the learning algorithm to be more sensitive (appreciate) to the importance of instances from the minority class, including the one-class learning [15] and cost-sensitive learning algorithms [2], [14], [15]. The challenge with algorithm-based solutions, however, is that they require specific treatments/tuning, which complicates their application in most cases.

Ensemble learning [17] allows the application of different learning algorithms and combining their results using an ensemble. Ensemble learning was introduced in software defect prediction in [34] which focused mainly on error rate or accuracies. Since the PROMISE data set [11], [31] has data imbalance problem and classifying defect as non-defect (false negative) is the most expensive in software defect prediction, therefore this work [34] has limitations in analyzing cost-sensitive classification.

Adjustments in training datasets and hence resolving the class imbalance problem are used in traditional ML techniques [18]. Under-sampling techniques try to solve the class imbalance problem by removing instances from the majority class. Oversampling techniques try to solve the class imbalance problem by adding instances from the minority class either by duplicating or adding fake data. Hybrid techniques try to solve the problem using a combination of under-sampling and oversampling.

Random under-sampling and oversampling techniques have been widely used [14], [15]. However, their accuracy relies on the data in hand and the algorithm used for the classification [14]. Menzies *et al.* [11] and [11] proposed a class imbalance learning technique that under-samples the non-defective class to balance the training data. The authors reported that this did not affect the accuracy of the C4.5 or the Naïve Bayes classifier. However, the objective was to emphasize that we could achieve similar accuracy if we carefully sub-sampled instances, i.e., the goal was not to improve

the accuracy of the classification technique. Chen *et al.* [17] proposed a model to tackle the class overlap and imbalance problem in SDP by applying a neighbor cleaning method to remove non-defective class overlapping instances, followed by random under-sampling several times to generate a balanced subset that can be trained using traditional classifiers. Wang and Yao [19] introduced a detailed comparison between five class imbalance learners including, Random Under-sampling of the majority, Random Under-sampling of both classes, Threshold Moving (which applies a cost-sensitive method directly on the data, SMOTE Boost (which is a combination of the random oversampling technique plus Adaptive Boost [19], which was applied on UCI machine learning repository data set [33], but not on software fault detection. The authors showed that random oversampling followed by adaptive boosting achieved the best balance in 8 out of 10 benchmark datasets. Barandela *et al.* [20] conducted a comparative study of various samplings.

Both oversampling and under-sampling techniques are used in SDP according to the above review to address the class imbalance problem. These have improved the prediction performance over without sampling of original data. Barandela *et al.* [20] showed that in highly imbalanced datasets, oversampling of the minority class is more effective. They [20] also observed that under-sampling would be useful when the difference between the number of minority instances and majority instances is not too big. Literature review shows that sampling techniques for class imbalanced problem was used only for a single classifier which acquire bias-variance problem. Ensemble approach is a popular approach in machine learning domain to overcome the variance problem; which was not adopted in software fault detection. Literature gap motivates us to employ ensemble technique with oversampling approach for class imbalance problem.

III. PROPOSED APPROACH TO CLASS IMBALANCE PROBLEM: ENSEMBLE OVERSAMPLING LEARNING SCHEME

The proposed approach considers an ensemble of oversampling techniques to address the class imbalance problem in software fault detection. We propose a hybrid of oversampling techniques. Detailed steps are described in Table 1, Algorithm 1.

Step-1 (Multiple Oversampling and Generating Training Data for Ensemble Classifier): Three different kinds of oversampling methods were used, namely ROS, MWM, and FIDos.

Random Oversampling (ROS) [21]: ROS generates new samples based on the minority classes by randomly selecting training samples from the minority class, and then duplicating it. In doing so, the class distribution can be balanced, but this usually causes overfitting and longer training times during the imbalanced learning process.

Majority Weighted Minority (MWM) Oversampling Technique (MWMOTE) [13]: This technique involves three key steps: 1) identify the most important and hard-to-learn minority class samples, 2) calculate the selection weight S_w from each member of S_{imin} where S_{imin} is the informative minority set; 3) generate synthetic samples from S_{imin} using S_w; and 4) produce the output set S_{omin} by adding the new generated samples to the original minority class, S_{min}.

Precisely, there are three stages in constructing S_{imin}. In the first stage, MWMOTE filters the original minority class samples, S_{min}, in order to find a filtered minority set, S_{minf}. In this respect, the nearest neighbour of each sample x_i of S_{min} is calculated as $NN(x_i)$ which has k_1 neighbours. Then, x_i will be removed if its $NN(x_i)$ contains only the majority class samples. In the second stage, construct a nearest majority set, $N_{maj}(x_i)$, for each x_i with the number of majority neighbours used for constructing informative minority samples, k_2, as few as possible. A borderline majority set S_{bmaj} is obtained by combining all the $N_{maj}(x_i)$. For each $y_i \in S_{bmaj}$, and then we can obtain nearest minority $N_{min}(y_i)$. $N_{min}(y_i)$ consists of nearest k_3 minority examples from S_{bmaj} which is computed using Euclidean distance. Then we compute the informative minority set S_{imin} by the union of all $N_{min}(y_i)$. For each $y_i \in S_{bmaj}$, and for each $x_i \in S_{imin}$, we compute the information weight $I_w(y_i, x_i)$ For the selection weights, S_w is expressed as:

$$S_w(x_i) = \sum\nolimits_{y_i \in S_{bmaj}} I_w(y_i, x_i),$$

where $I_w(y_i, x_i)$ is the information weight and is computed as the product of the closeness factor, $C_f(y_i, x_i)$ and the density factor $D_f(y_i, x_i)$:

$$I_w(y_i, x_i) = C_f(y_i, x_i) \times D_f(y_i, x_i) \tag{1}$$

The closeness factor $C_f(y_i, x_i)$ is defined as:

$$C_f(y_i, x_i) = \frac{f(\frac{1}{d_n(y_i, x_i)})}{C_f(th)} \times CMAX \tag{2}$$

where $C_f(th)$ and $CMAX$ are the user-defined parameters, and f is a cut-off function, which is:

$$f(x) = \begin{cases} x & if\ x \le C_f(th) \\ C_f(th) & otherwise \end{cases} \tag{3}$$

Moreover, MWMOTE computes $D_f(y_i, x_i)$ by normalizing $C_f(y_i, x_i)$, which is:

$$D_f(y_i, x_i) = \frac{C_f(y_i, x_i)}{\sum_{q \in S_{imin}} C_f(y_i, x_i)} \tag{4}$$

MWMOTE first clusters S_{min} into M clusters, which can be denoted as $L_1, L_2, \cdots, L_M$. Then, a sample x is selected from S_{imin} following the probability distribution $\{S_p(x_i)\}$ (where $S_p(x_i) = \{S_w(x_i)\} / \sum_{z_i \in S_{imin}} S_w(z_i)$). Let's assume $x \in L_k$. After that, randomly choose another sample y from L_k and generate a synthetic sample s using the linear interpolation of x and y, which is:

$$s = x + \alpha \times (y - x) \tag{5}$$

where α is a random number of [0, 1].

The procedure can be repeated 'N' times to generate 'N' synthetic samples.

Fuzzy-Based Feature and Instance Recovery Using Information Decomposition (FIDos) [14]: FIDos produces a function mapping which maps the input space to a discrete set. Let us consider the mapping as described below:

$$\mu : X \times U \rightarrow [0, 1], \tag{6}$$

$$(x_i, u_s) \rightarrow \mu\,(x_i, u_s), \tag{7}$$

Where vector X is an example of input feature which have dimension 'i'. The discrete set is denoted by U. The central interval points are denoted by u_s where $s = 1, \ldots, t$ where t=total number of values to be estimated for synthetic samples. A membership function $\mu\,(x_i, u_s)$ is considered which is constructed as below. Let us consider the required step length of interval is h_i

$$\mu\,(x_i, u_s) = \begin{cases} 1 - \dfrac{\|x_i - u_s\|}{h_i}, & if \;\; \|x_i - u_s\| \leq h_i \\ 0, & otherwise \end{cases} \tag{8}$$

$$\tilde{m}_s = \begin{cases} \bar{x}_i, & if \;\; \sum_i \mu\,(x_i, u_s) = 0 \\ \dfrac{\sum_{j=1}^{m} m_{is}}{\sum_{i=1}^{m} \mu\,(x_i, u_s)}, & otherwise \end{cases} \tag{9}$$

where $m_{is} = \mu\,(x_i, u_s) \times x_i$.
Now we use the membership function to generate the new instances in imbalanced data.

Let us assume that

$$A = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \ldots . \mathbf{x_n} \tag{10}$$

$$B = \{\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3 \ldots . \mathbf{y_n} \tag{11}$$

Where $\mathbf{x_n}$ and $\mathbf{y_n}$ are the feature vectors and A and B are the set of minority and majority class.

Let us assume we want to create 'l' synthetic instances, we compute the minimum and maximum for each feature vector from the set A and B as below.

$$\mathrm{a} = \min\{\mathbf{x_i}, \mathbf{y_i}\} \tag{12}$$

$$b = \max\{\mathbf{x_i}, \mathbf{y_i}\} \tag{13}$$

We compute the step length for each feature vector as below

$$h_i = \frac{b - a}{2l} \tag{14}$$

Here is.

I_s the intervals for each values to be estimated as follows:

$$I_s = \{\mathrm{a} + (\mathrm{s} - 1) \times h, a + s \times h\}$$

For each interval point 's' where $\mathrm{s} = 1, 2, 3, ..(2l - 1)$ we compute u_s (which is the center) as below

$$u_s = (a + (s - 1) \times h + a + s \times h)/2 \tag{15}$$

Then we compute weight $\mu\,(x_i, u_s)$ using equation-(8)
The information decomposition is computed as below

$$m_{is} = \mu\,(x_i, u_s) \times x_i. \tag{16}$$

Then the value for feature for the synthetic instances is computed using equation (9).

Step-2 (Training the Individual–Based Learner for Ensemble Classification): Ensemble classification is a popular approach in machine learning (ML) when datasets have class imbalance issues or when the data size is small. Ensemble classifiers are constructed from a set of classifiers. The samples are classified based on the individual classifiers and are combined using a voting or an average strategy. Several individual learning algorithms are trained for the same classification task, and the trained model of each learning algorithm is used to test the new samples to obtain better predictive performance.

An ensemble model is built in three steps [22], which are ensemble strategy, learners, and final results generation procedure. Many prevailing methods are from the Bagging [23] and Boosting [22], [23] families. The predicted results [22]–[25] show that the ensemble method can achieve better results than the single learner method [22]–[25].

Recently, there have been many other methods for obtaining the component learners. For example, Liu *et al.* [26] proposed sampling ensembles to improve the prediction accuracy for frequent patterns.

This paper introduces the novel approach of oversampling using an ensemble approach to empower the oversampling algorithms. This can help repair imbalanced data by integrating the random oversampling, Majority Weighed Minority Oversampling Technique, and Fuzzy-Based Feature and Instance Recovery approaches into one unified technique.

As shown in Figure 1, for a given observation dataset D, we randomly divide D into training data and test data. Then, we re-balance the training data on three subset and obtain three pre-processed datasets: $D_1 = (D_{ROS+} \cup D_-)$, $D_2 = (D_{MWM+} \cup D_-)$, and $D_3 = (D_{FIDos+} \cup D_-)$. For each re-balanced training dataset, Random Forest (RF) is used to create the model.

RF [27], [28] is a bagging ensemble approach which is constructed based on standard decision tree. The trees in the RF are constructed by applying bootstrapping technique on the training data and then train a decision tree from each bootstrapped sampled set. The set of decision trees in the RF can also be constructed by randomly generating subset of input features and then construct a decision tree from each subset. A majority vote is utilized for the final decision make for a given observation. During the training process, about $\frac{1}{3}$ of the training data are not used in bootstrapping technique which are known as out-of-bag (OOB) samples.

In recent studies, RF was extensively used for class imbalance learning [28]–[30]. In this paper, we have used RF for the classification. RF predicts an unseen test sample according to the following equation:

$$\begin{aligned} \hat{y} &= \frac{1}{m} \sum_{j=1}^{m} \sum_{i=1}^{n} W_j\,(x_i, x)\, y_i \\ &= \sum_{i=1}^{n} (\frac{1}{m} \sum_{j=1}^{m} W_j\,(x_i, x)) y_i. \end{aligned} \tag{17}$$

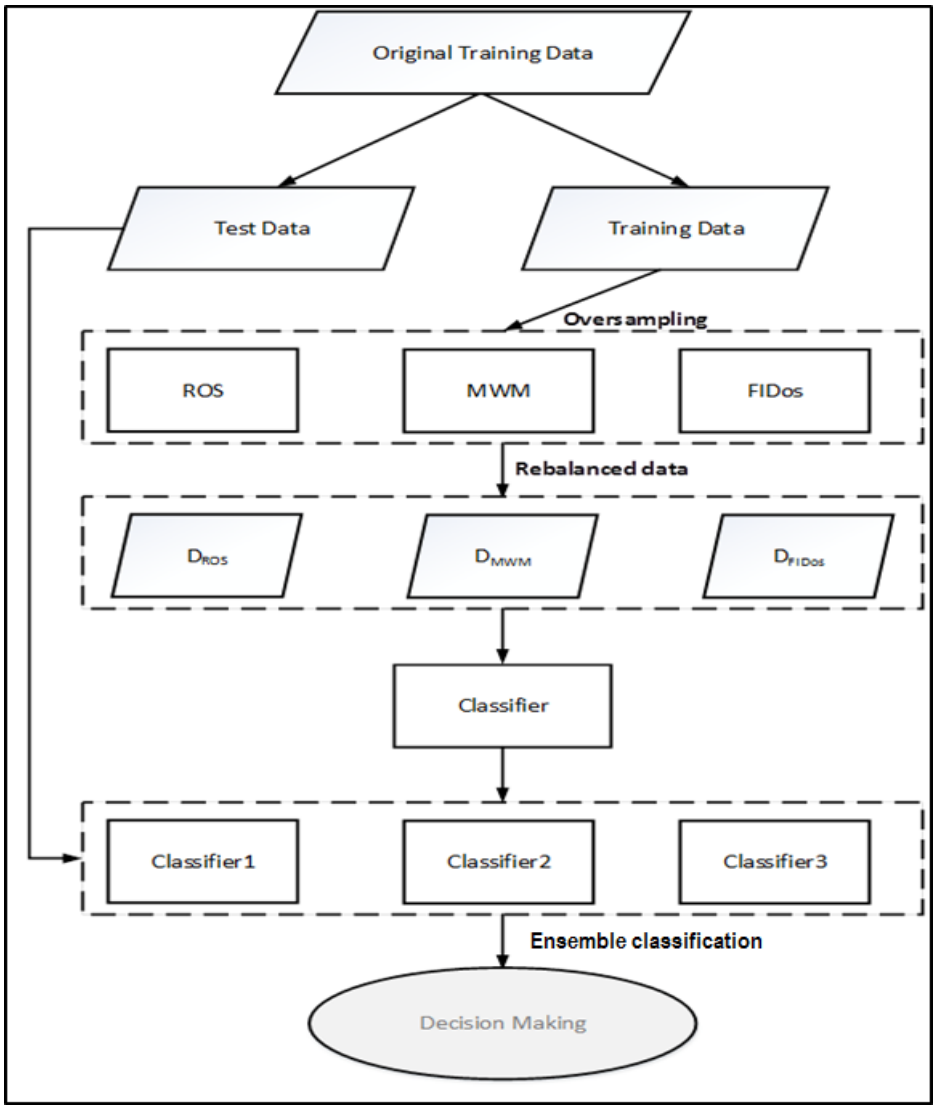

FIGURE 1. Framework of the detection strategy based on ensemble oversampling.

where W_j is the non-negative weight of the ith training sample for test sample x with respect to the jth tree. m is the number of trees. According to [28], an RF results in better performance with 50 trees. Therefore, in our study m is set to 50.

Finally, given a test sample, a majority voting scheme is introduced for decision making. Given a test sample, the final prediction is determined by the following equation. Our proposed ensemble approach framework is illustrated in Table 1.

$$C(x) = \begin{cases} positive\ class & if\ N(y_+) \geq 2 \\ negtive\ class & otherwise \end{cases} \quad (18)$$

$N(y_+)$ is the number of classifiers that predict x as a positive class sample, given as:

$$N(y_+) = \sum_{Cl=1}^{3} f(C_{Cl}(x) = y_+) \quad (19)$$

Cl=Classifier index, y_+ = Posiotive class, $f(.)$ is the boolean function. The function f(.) = 1 if the test criteria is true, otherwise f(.) = 0.

IV. DATASETS USED IN OUR EXPERIMENTS

The datasets used in the experiments herein were collected from the PROMISE repository software engineering databases [11], [31], and they are varied in their degree of complexity, number of instances, and imbalance ratio (IR), which means the size of the majority class to the size of minority class is as follows:

$$IR = \frac{\#Majority\ samples}{\#Minroity\ samples} \quad (20)$$

Precisely, Table 2 summarizes the 15 datasets utilized in this paper. We can see the imbalance ratio varies from 3.50 (only slightly imbalanced) to 45.56 (highly imbalanced). We also considered datasets with diversity in the number of instances; the smallest dataset has 36 samples, while the largest dataset contains 17,186 samples.

TABLE 1. The proposed ensemble oversampling algorithm-1.

Step-1: Oversampling and Training of Base Learners

INPUT: Training data set D ($D_+ \cup D_-$); Oversampling ratio α; Classification algorithm C;

OUTPUT: Multiple base classifiers C_1, C_2, and C_3

1: Redistribute D_+ with random oversampling method $D_{ROS+} = ROS(D_+, \alpha)$, $D_1 = (D_{ROS+} \cup D_-)$;

2: Redistribute D_+ with MWMOTE method $D_{MWMOTE+} = MWMOTE(D_+, \alpha)$, $D_2 = (D_{MWMOTE+} \cup D_-)$;

3: Redistribute D_+ with FIDos method $D_{FIDos+} = FIDos(D_+, \alpha)$, $D_3 = (D_{FIDos+} \cup D_-)$;

4: For Cl in 1, 2, and 3 do

5: Train classifier C_{Cl} using base classification algorithm: $C_{Cl} = C(D_i)$;

6: End for

7: Return C_1, C_2, and C_3.

Step-2: Build Ensemble Model

INPUT: Test sample x.

OUTPUT: Decision making for x.

1: For Cl in 1, 2, and 3 do

2: Calculate $Cl(x)$ as mentioned in equation (18)

3: End for

4: Calculate $N(y_{+})$ using equation (19)

5: $C(x) = \begin{cases} positive\ class & if\ N(y_{+}) \geq 2 \\ negtive\ class & otherwise \end{cases}$

A. RESULTS AND PERFORMANCE ANALYSIS

In the experiments, a 5-fold cross validation (CV) with 10 trial runs were applied to justify the performances of the proposed approach. An RF is used as a base learner in our experiments which was also tested on standard machine learning data sets in an earlier study [26] by one of the co-authors [26].

1) EXPERIMENTAL SETTINGS FOR THE PARAMETERS

The experimental settings for proposed approach and existing other approaches are described here.

For Random forest (RF), Tree size (m) =50 is considered. We considered high tree size to avoid overfitting which is determined from earlier experiments [12], [14].

For MWMOTE, the values for parameters have been chosen from our earlier experiments and best performance while using MWMOTE techniques for dealing with missing values in our earlier works [14], [26]. The values for MWMOTE are as below:

$K_1 = 5$, $K_2 = 3$ and $K_3 = S_{min}/2$, $C_p = 3$, $C_{f(th)} = 5$,

TABLE 2. Data set.

Dataset	# Instances	# Minority	# Majority	#attr	IR
CM1	327	42	285	37	6.79
JM1	7782	1672	6110	21	3.65
MW1	253	27	226	37	8.37
AR1	121	9	112	29	12.44
AR3	63	8	55	29	6.88
AR4	107	20	87	29	4.35
AR5	36	8	28	29	3.50
AR6	101	15	86	29	5.73
KC1	2109	326	1783	21	5.47
KC2	522	107	415	21	3.88
PC1	1109	77	1032	21	13.40
PC2	745	16	729	36	45.56
PC3	1077	134	943	37	7.04
PC4	1458	178	1280	37	7.19
PC5	17186	516	16670	38	32.31

and CMAX = 2, S_{min} is computed form the data set which is the number of original minority class samples. According to the earlier experiments [14], [26], it is seen that an oversample ratio of 200 percent of the original minority class samples provided significant classification performance.

Different performance metrics are used to verify the detection performance of the proposed approach. The performance metrics are: detection accuracy, true positive rate (TPR), false negative rate (FNR), area under the curve (AUC), F-measure. When computing the confusion matrix for performance analysis, we considered a faulty module as a positive class. In the training data sets, faulty modules are in the minority group and non-faulty modules are in the majority group.

We compared the proposed ensemble technique with existing approaches including original data sets without sampling (ORI), Random Over sampling (ROS) [21], MWMOTE/MWM [13], FIDos [14], [26] our earlier fuzzy information decomposition based sampling technique.

For a binary classification problem of software fault detection, a confusion matrix visualizes the performances of learning algorithm. Confusion matrix is shown in table 3.

In the confusion matrix in table 3, if a module is faulty and predicted as a positive, it is considered as a true positive (TP). However, if a module is faulty, and it is predicted as a negative, it is considered as false negative (FN). If a module is non-faulty and predicted as a positive, it is considered as a false positive (FP). However, if a module is non-faulty, and it is predicted as a negative, it is considered as true negative (TN).

AUC Results From Our Experiments: Figure II–Figure V present the False Positive Rate and AUC values of software

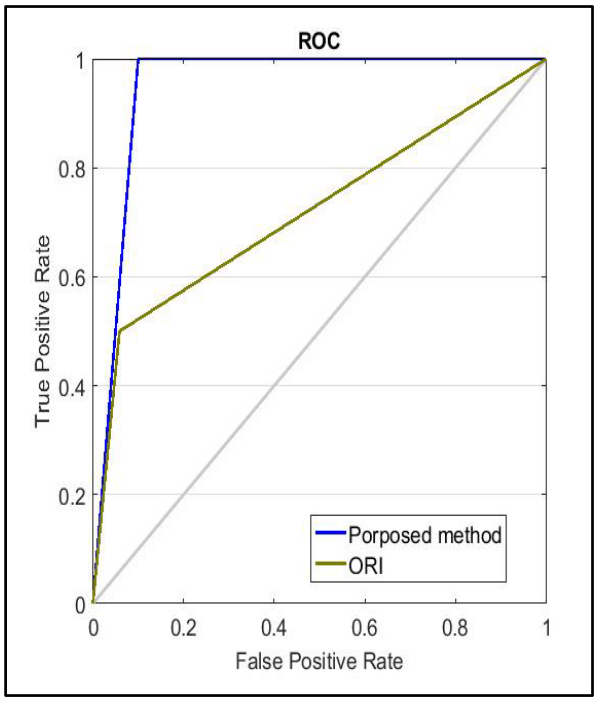

FIGURE 2. ROC curve based on AR6 dataset.

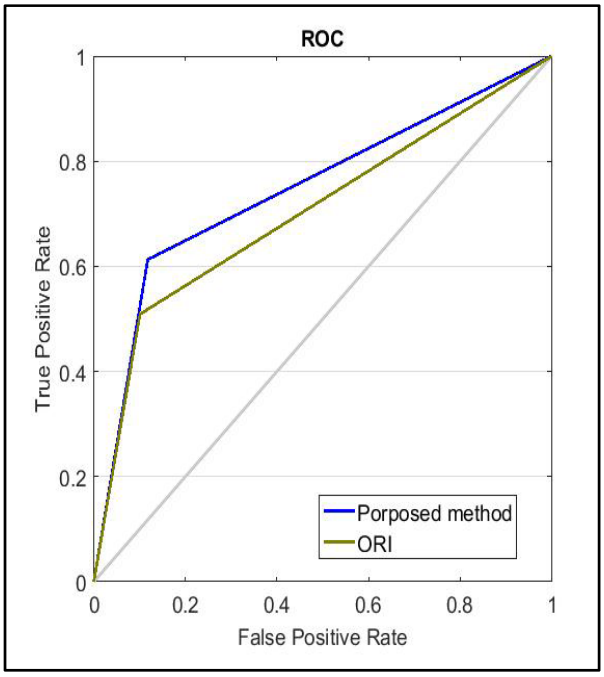

FIGURE 3. ROC curve based on kc1 dataset.

defects prediction results derived by the RF classification algorithm based on the 15 datasets. One can observe that the proposed ensemble learning method produces the lowest average false positive rate at 39%, which is much lower than the second best, MWM, with about 48.5%. When comparing with the original data, the proposed method can improve the false positive rate by at least 20%. Figure V also demonstrates

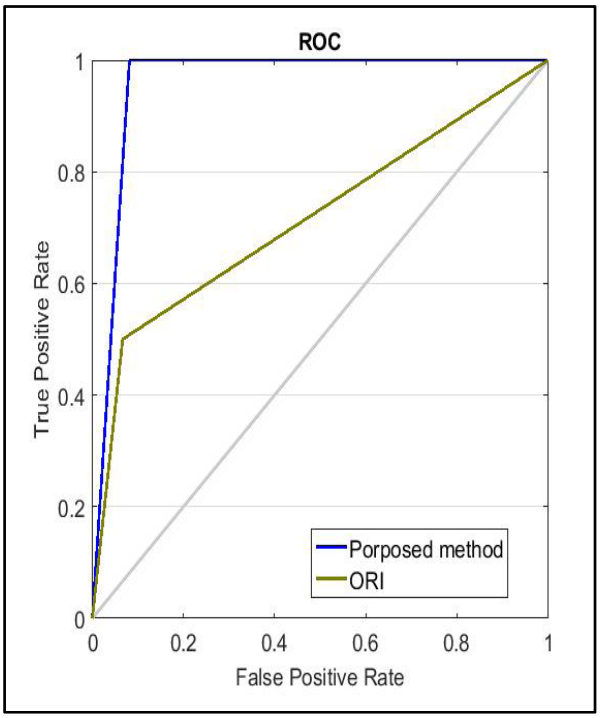

FIGURE 4. ROC curve based on MW1 dataset.

that our proposed ensemble method obtains the highest AUC values among all data sets and all approaches with about 68% compared with other approaches. ROS and FIDos also improved the performance to 66% and 66.5%, respectively, compared to original data sets without sampling (ORI) with 63.5%. MWM (which was 64.8%) performed only slightly better than ORI. This indicates that MWM is not stable in classification performance.

The experimental results for false positive rate and AUC for all data sets and all algorithms (ORI, ROS, MWM, FIDos and proposed ensemble) have been box plotted in Figure V. The performance of ORI indicates that directly learning from the original data yields poor classification performance. ROS, MWM, and FIDos can somehow improve the classification performance a bit but are not stable. The proposed algorithm can result in a low false positive rate and obtains the maximum AUC values and highest median values, which means it can significantly boost the software defects prediction performance to another level.

Tables 4 and 5 compare the accuracy and recall of the five approaches based on the 15 datasets. One can see that ORI and FIDos produced the highest accuracy values in 6 and 5 out of 15 datasets, respectively. ROS obtains highest accuracy values in 4 datasets. However, the proposed algorithm can result in very comparative accuracy values in some cases. For example, the FIDos performance on the PC4 dataset was 0.907, while the proposed method was 0.906, which is 0.001 lower. An earlier work [32] showed that the overall classification accuracy was not enough to justify the classifiers' performances when a dataset has 95% examples from

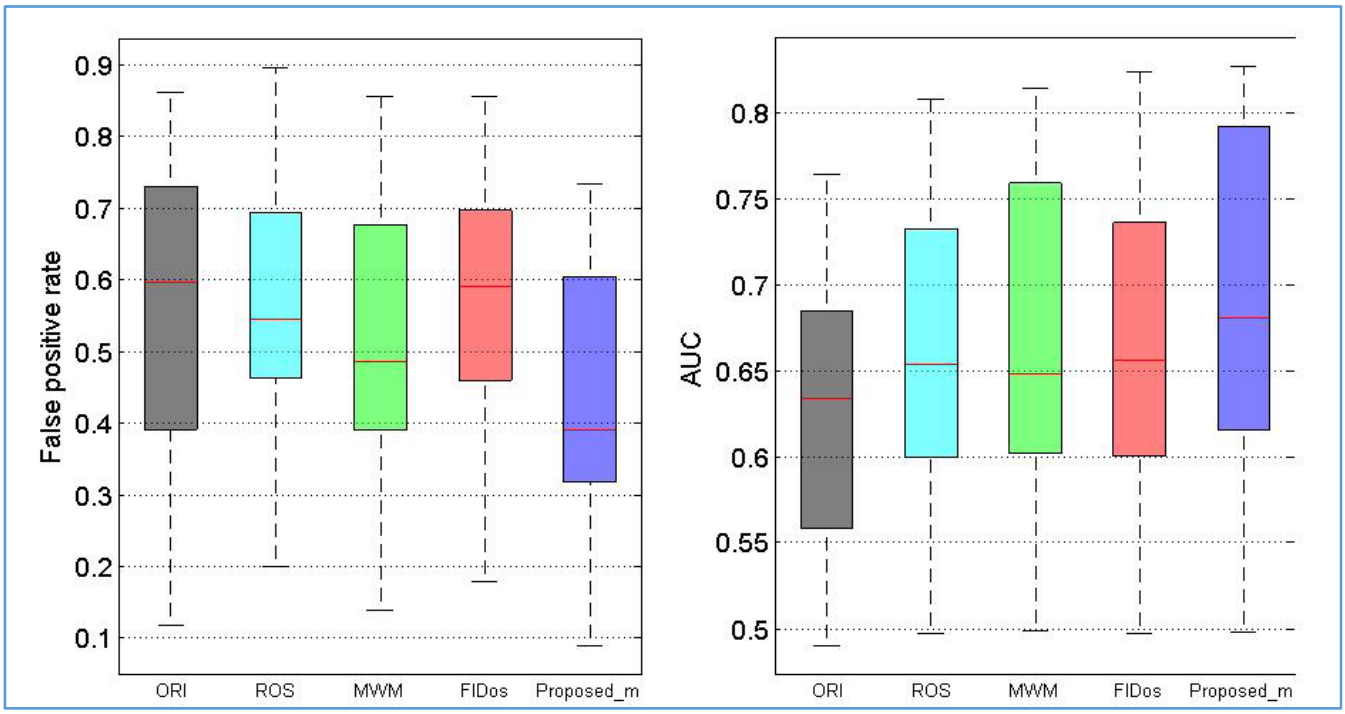

FIGURE 5. Averaged overall False Positive Rate and AUC results.

TABLE 3. Confusion matrix.

Confusion Matrix		
	Positive prediction	Negative prediction
Actually positive	True positive(TP)	False negative(FN)
Actually negative	False positive(FP)	True negative(TN)

majority group and rest of the examples are from minority group. In this case, the classifier can accurately identify all members of majority group in the dataset. Accordingly, classifier's accuracy performance becomes 95%. ORI, ROS, and FIDos raise the accuracy values at the cost of simultaneously raising the false positive rate. This concludes that ORI, ROS, and FIDos are not useful in practice. Comparatively, along with Figure V, we can see that our proposed algorithm is more practical and applicable to software defects prediction.

2) COMPARISON WITH OTHER WORKS IN THE LITERATURE

Earlier works on fault detection based on ensemble technique [34] mainly focused on defect detection based on overall accuracies which has limitation in cost sensitive classification (reducing false negative rate and increasing the recall). In [34], PROMISE data was used and only four data sets were tested. In this work [34], 21.3% error rate was achieved for PC1. 23.4% error rate was achieved for JM1 data set. 24.1% error rate was achieved for CM1 data set. 26.1% error rate was achieved in JP/NASDA data set.

However PC1 our proposed method achieved only 7% error rate. For JM1 proposed method achieved 24.1% error rate. For CM1, proposed method achieved 21.9% error rate. None of the data set was tested in [34] for recall or AUC which are the best performance metrics.

Recall values was tested in [35] with three PROMISE datasets (KC1, PC3 and PC4). In [35], for KC1 86.5% accuracy and 31.61 recall, for PC3 90.96% accuracy and 22.50 recall, for PC4 91.89% accuracy and 54.51% recall were achieved.

Table 4 and Table 5 also summarize the Recall and AUC values of the five methods based on the 15 datasets. For KC1 proposed method achieved 52.4% recall which is 60% times higher recall achieved in [35]. For PC3 and PC4 dataset, our proposed method achieved 41.8% and 68.3% recall values which are higher than recall achieved in [35]. The proposed algorithm produces 68.3% on the PC4 dataset, which is higher than ORI and FIDos (which achieved 15.5%). Precisely, the proposed method yields the best recall performance in all data sets as mentioned in Table 4 and Table 5.

Because of the in-class imbalance problem, we are most interested in the minority class samples. Especially in software defects prediction, it is very important to have better recall (true positive rate) performance because the cost of

TABLE 4. Accuracies.

Data	Imbalanced data redistribute algorithms									
	ORI		ROS		MWM		FIDos		Proposed	
	average	Std.	average	Std.	average	Std.	average	Std.	average	Std.
CM1	0.694	0.127	**0.826**	0.045	0.823	0.031	0.819	0.047	0.781	0.057
JM1	**0.791**	0.002	0.777	0.003	0.781	0.007	0.778	0.002	0.759	0.002
MW1	0.864	0.046	**0.868**	0.011	0.855	0.113	0.864	0.010	0.814	0.069
AR1	0.361	0.068	0.520	0.126	0.491	0.403	**0.573**	0.145	0.329	0.380
AR3	0.728	0.129	0.767	0.091	0.800	0.295	**0.857**	0.085	0.779	0.110
AR4	0.792	0.059	0.779	0.058	0.793	0.063	**0.813**	0.024	0.739	0.072
AR5	0.751	0.115	**0.809**	0.077	0.774	0.242	0.774	0.111	0.774	0.086
AR6	0.664	0.133	0.784	0.087	0.724	0.277	**0.801**	0.065	0.682	0.143
KC1	**0.858**	0.003	0.844	0.006	0.846	0.014	0.847	0.003	0.831	0.007
KC2	**0.826**	0.009	0.804	0.009	0.813	0.034	0.810	0.004	0.784	0.007
PC1	**0.938**	0.002	0.936	0.002	0.932	0.012	0.935	0.003	0.930	0.003
PC2	0.135	0.132	**0.438**	0.195	0.174	0.352	0.325	0.160	0.134	0.131
PC3	**0.871**	0.005	0.864	0.005	0.852	0.019	0.862	0.006	0.843	0.005
PC4	0.906	0.004	0.904	0.006	0.907	0.014	**0.907**	0.004	0.906	0.002
PC5	**0.976**	0.001	0.975	0.001	0.973	0.002	0.974	0.001	0.972	0.000

TABLE 5. Recall values.

Data	Imbalanced data redistribute algorithms name									
	ORI		ROS		MWM		FIDos		Proposed_m	
	average	Std.	average	Std.	average	Std.	average	Std.	average	Std.
CM1	0.263	0.171	0.106	0.083	0.146	0.113	0.146	0.062	**0.268**	0.114
JM1	0.201	0.008	0.295	0.008	0.273	0.024	0.292	0.004	**0.384**	0.008
MW1	0.221	0.093	0.267	0.040	0.345	0.203	0.294	0.085	**0.363**	0.074
AR1	0.660	0.084	0.470	0.149	0.550	0.476	0.410	0.185	**0.680**	0.047
AR3	0.760	0.108	0.750	0.085	0.770	0.338	0.770	0.067	**0.840**	0.108
AR4	0.405	0.117	0.455	0.101	0.515	0.269	0.420	0.075	**0.610**	0.084
AR5	0.770	0.125	0.800	0.105	0.860	0.227	0.820	0.114	**0.910**	0.074
AR6	0.420	0.191	0.353	0.157	0.473	0.324	0.340	0.142	**0.527**	0.155
KC1	0.310	0.018	0.394	0.023	0.437	0.069	0.394	0.018	**0.524**	0.017
KC2	0.455	0.035	0.541	0.032	0.619	0.095	0.544	0.023	**0.661**	0.034
PC1	0.294	0.031	0.348	0.027	0.318	0.118	0.333	0.037	**0.390**	0.025
PC2	**0.880**	0.140	0.560	0.207	0.840	0.370	0.680	0.169	**0.880**	0.140
PC3	0.141	0.023	0.246	0.029	0.310	0.094	0.252	0.030	**0.418**	0.024
PC4	0.367	0.014	0.524	0.031	0.575	0.086	0.528	0.017	**0.683**	0.019
PC5	0.427	0.011	0.520	0.016	0.578	0.048	0.524	0.011	**0.659**	0.011

TABLE 6. AUC values.

Data	Imbalanced data redistribute algorithms name									
	ORI		ROS		MWM		FIDos		Proposed_m	
	average	Std.	average	Std.	average	Std.	average	Std.	average	Std.
CM1	0.511	0.011	0.519	0.020	0.534	0.052	0.532	0.021	**0.563**	0.034
JM1	0.576	0.004	0.602	0.004	0.597	0.011	0.601	0.003	**0.623**	0.004
MW1	0.581	0.028	0.604	0.020	**0.630**	0.089	0.613	0.042	0.615	0.032
AR1	0.490	0.020	0.498	0.032	**0.517**	0.097	0.498	0.026	0.516	0.012
AR3	0.746	0.041	0.765	0.033	0.790	0.210	**0.822**	0.049	0.808	0.049
AR4	0.643	0.028	0.654	0.025	0.686	0.132	0.662	0.042	**0.689**	0.043
AR5	0.764	0.054	0.807	0.063	0.813	0.163	0.794	0.047	**0.826**	0.045
AR6	0.563	0.017	0.606	0.043	**0.621**	0.121	0.611	0.035	0.618	0.048
KC1	0.634	0.009	0.660	0.012	0.679	0.031	0.662	0.009	0.**706**	0.010
KC2	0.688	0.018	0.706	0.015	**0.741**	0.050	0.711	0.009	0.739	0.016
PC1	0.640	0.015	0.664	0.014	0.648	0.058	0.656	0.019	**0.680**	0.013
PC2	**0.500**	0.001	0.498	0.002	0.499	0.002	0.499	0.001	0.499	0.002
PC3	0.558	0.012	0.599	0.014	0.619	0.046	0.601	0.015	**0.661**	0.013
PC4	0.674	0.007	0.741	0.016	0.764	0.043	0.744	0.008	**0.810**	0.009
PC5	0.710	0.006	0.754	0.008	0.782	0.024	0.756	0.006	**0.820**	0.005

false negatives (i.e., actually defective modules predicted as non-defective) is usually several times higher than the cost of false positives (i.e., actually non-defective but predicted as defective).

Moreover, one can see from Table 5 that our proposed algorithm results in the best performance in 9 out of 15 datasets. Although MWM obtained the highest AUC values in 4 out of 15 datasets, the proposed methods showed very comparative performance in these datasets. For example, the AUC values of MWM on the MW1 and AR1 datasets were 0.630 and 0.517; however, the related AUC values of the proposed method are 0.615 and 0.516, which are only 0.015 and 0.001 lower. Since AUC is not sensitive to the distribution between the majority and minority classes, it can sort models by overall performance. We can conclude that our proposed algorithm can build a more robust classifier for software defect prediction.

V. CONCLUSION

For high accuracy and a low false negative rate, balanced defect data is the most important criteria in automated software fault detection. However, in practical situations, due to limited facilities to collect software metric dataset with many faulty modules, most available datasets are imbalanced. Earlier work by Ali *et al.* [12] showed that when imbalanced data are used for training an automated detection system, it cannot reduce the false negative rate, even with a well-trained classification model [12]. The cost due to the high false negative rate can grow very rapidly if not detected at the early stages in the software development life cycle. Therefore, there is an urgent need to address the imbalanced data in software metrics.

In this paper we proposed an ensemble model by building a set of individual base-learners from different oversampling techniques. Three different oversampling techniques were used. Using multiple oversampling techniques mitigates the bias of sampling approaches, while the ensemble technique can take advantage of many classifiers and increases the correct classification rate and reduces the false negative rate. Experimental results show that proposed approach outperforms the standard fault detection techniques. Developing ensemble oversampling classifier is a complex task with a selected base classifier. The proposed method is based on random forest base classifier. One of the limitations in the proposed method is that it may carry the variance problem of individual classifier itself. This can be avoided by testing the proposed ensemble method with other base classifiers including support vector machine, neural networks and decision trees. Then select the best performing classifier from their comparative results. This can be done in a future extension of this work. Future work can also include a combination of our earlier work [12] with the current work of ensemble oversampling techniques for identifying significant metrics while also developing a fast and accurate fault detection system.

REFERENCES

[1] *Gartner Says Worldwide it Spending on Pace to Reach 3.8 Trillion in 2014*. Accessed: Feb. 21, 2016. [Online]. Available: http://www.gartner.com/newsroom/id/2643919

[2] Ö. F. Arar and K. Ayan, "Software defect prediction using cost-sensitive neural network," *Appl. Soft Comput.*, vol. 33, pp. 263–277, Aug. 2015.
[3] B. S. Ainapure, *Software Testing and Quality Assurance*, 1st ed. New Delhi, India: Technical Publications, 2014.
[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] L. Pelayo and S. Dick, "Applying novel resampling strategies to software defect prediction," in *Proc. Annu. Meeting North Amer. Fuzzy Inf. Process. Soc. (NAFIPS)*, Jun. 2007, pp. 69–72.
[6] M. Zhao, C. Wohlin, N. Ohlsson, and M. Xie, "A comparison between software design and code metrics for the prediction of software fault content," *Inf. Softw. Technol.*, vol. 40, no. 14, pp. 801–809, 1998.
[7] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. New York, NY, USA: Cambridge Univ. Press, 2000.
[8] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.
[9] B. Kröse and P. V. D. Smagt, *An Introduction to Neural Networks*. Amsterdam, The Netherlands: Univ. Amsterdam, 1993.
[10] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Appl. Soft Comput.*, vol. 27, pp. 504–518, Feb. 2015.
[11] T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic, and Y. Jiang, "Implications of ceiling effects in defect predictors," in *Proc. 4th Int. Workshop Predictor Models Softw. Eng. (PROMISE)*, 2008, pp. 47–54.
[12] M. M. Ali, S. Huda, J. Abawajy, S. Alyahya, H. Al-Dossari, and J. Yearwood, "A parallel framework for software defect detection and metric selection on cloud computing," *Cluster Comput.*, vol. 20, no. 3, pp. 2267–2281, 2017.
[13] S. Barua, M. M. Islam, X. Yao, and K. Murase, "MWMOTE–majority weighted minority oversampling technique for imbalanced data set learning," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 2, pp. 405–425, Feb. 2014.
[14] S. Liu, J. Zhang, Y. Wang, and Y. Xiang, "Fuzzy-based feature and instance recovery," in *Proc. Asian Conf. Intell. Inf. Database Syst.*, 2016, pp. 605–615.
[15] 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, no. 1, pp. 321–357, 2002.
[16] A. Estabrooks, T. H. Jo, and N. Japkowicz, "A multiple resampling method for learning from imbalanced data sets," *Comput. Intell.*, vol. 20, no. 1, pp. 18–36, 2004.
[17] L. Chen, B. Fang, Z. Shang, and Y. Tang, "Tackling class overlap and imbalance problems in software defect prediction," *Softw. Quality J.*, vol. 26, no. 1, pp. 97–125, 2016.
[18] K. O. Elish and M. O. Elish, "Predicting defect-prone software modules using support vector machines," *J. Syst. Softw.*, vol. 81, no. 5, pp. 649–660, 2008.
[19] S. Wang and X. Yao, "Multiclass imbalance problems: Analysis and potential solutions," *IEEE Trans. Syst., Man, Cybern., B, Cybern.*, vol. 42, no 4, pp. 1119–1130, Aug. 2012.
[20] R. Barandela, R. M. Valdovinos, J. S. Sánchez, and F. J. Ferri, "The imbalanced training sample problem: Under or over sampling?" in *Proc. Joint IAPR Int. Workshops Stat. Techn. Pattern Recognit. (SPR) Structural Syntactic Pattern Recognit. (SSPR)*, 2004, pp. 806–814.
[21] R. T. Hadke and P. Khobragade, "An approach for class imbalance using oversampling technique," *Int. J. Innov. Res. Comput. Commun. Eng.*, vol. 3, no. 11, pp. 11451–11455, 2015.
[22] S. Huda, J. Yearwood, H. F. Jelinek, M. M. Hassan, G. Fortino, and M. Buckland, "A hybrid feature selection with ensemble classification for imbalanced healthcare data: A case study for brain tumor diagnosis," *IEEE Access*, vol. 4, pp. 9145–9154, 2016.
[23] X. Wang and H. Wang, "Classification by evolutionary ensembles," *Pattern Recognit.*, vol. 39, no. 4, pp. 595–607, 2006.
[24] A. Awad, M. Bader-El-Den, J. McNicholas, and J. Briggs, "Early hospital mortality prediction of intensive care unit patients using an ensemble learning approach," *Int. J. Med. Inf.*, vol. 108, pp. 185–195, Dec. 2017.
[25] M. A. King, A. S. Abrahams, and C. T. Ragsdale, "Ensemble methods for advanced skier days prediction," *Expert Syst. Appl.*, vol. 41, no. 4, pp. 1176–1188, 2014.
[26] S. Liu, J. Zhang, Y. Xiang, and W. Zhou, "Fuzzy-based information decomposition for incomplete and imbalanced data learning," *IEEE Trans. Fuzzy Syst.*, vol. 25, no. 6, pp. 1476–1490, Dec. 2017, doi: 10.1109/TFUZZ.2017.2754998.
[27] F. Li *et al.*, "Cost-sensitive and hybrid-attribute measure multi-decision tree over imbalanced data sets," *Inf. Sci.*, vol. 422, pp. 242–256, Jan. 2018.
[28] A. Mellor, S. Boukir, A. Haywood, and S. Jones, "Exploring issues of training data imbalance and mislabelling on random forest performance for large area land cover classification using the ensemble margin," *ISPRS J. Photogramm. Remote Sens.*, vol. 105, pp. 155–168, Jul. 2015.
[29] A. Liaw and M. Wiener, "Classification and regression by randomforest," *R News*, vol. 2, no. 3, pp. 18–22, 2002.
[30] S. Huda, M. Abdollahian, M. Mammadov, J. Yearwood, S. Ahmed, and I. Sultan, "A hybrid wrapper–filter approach to detect the source(s) of out-of-control signals in multivariate manufacturing process," *Eur. J. Oper. Res.*, vol. 237, no. 3, pp. 857–870, 2014.
[31] J. S. Shirabad and T. J. Menzies, "The PROMISE repository of software engineering databases," School Inf. Technol. Eng., Univ. Ottawa, Ottawa, ON, Canada, Tech. Rep., 2005, accessed: Apr. 12, 2018. [Online]. Available: http://promise.site.uottawa.ca/SERepository/
[32] M. Lin, K. Tang, and X. Yao, "Dynamic sampling approach to training neural networks for multiclass imbalance classification," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 4, pp. 647–660, Apr. 2013.
[33] A. Frank and A. Asuncion. (2010). *UCI Machine Learning Repository*. [Online]. Available: http://archive.ics.uci.edu/ml
[34] B. Twala, "Predicting software faults in large space systems using machine learning techniques," *Defence Sci. J.*, vol. 61, no. 4, pp. 306–316, Jul. 2011.
[35] J. Moeyersoms, E. J. de Fortuny, K. Dejaeger, B. Baesens, and D. Martens, "Comprehensible software fault and effort prediction: A data miningapproachJulie Moeyersomsa," *J. Syst. Softw.*, vol. 100, pp. 80–90, Feb. 2015.

SHAMSUL HUDA received the Ph.D. degree in computer science. He was a Research Fellow with Federation University. He was an Assistant Professor with the Computer Science Department, Khulna University of Engineering and Technology, Bangladesh. He is currently a Lecturer with the School of Information Technology, Deakin University, Australia. He has published over 50 journal and conference papers in well reputed journals, including the IEEE Transactions. His main research interests include information security, cyber physical systems, computational intelligence, and machine learning.

KEVIN LIU (SHIGANG LIU) received the Ph.D. degree in computer science. He is currently a Research Associate with the School of Information Technology, Deakin University, Australia.

MOHAMED ABDELRAZEK received the Ph.D. degree from Swinburne University in 2014. He is currently an Associate Professor in software engineering and IoT with the School of Information Technology, Deakin University, Australia. He has over 10 years' experience in building software solutions. His research interests include software engineering, security, and artificial intelligence.

AMANI IBRAHIM is currently a Senior Lecturer in cyber security with the School of Information Technology, Deakin University. He is a cybersecurity professional with over a decade of experience across academia and industry. He is also the Cybersecurity Research Discipline Lead with the Deakin Software and Technology Innovation Laboratory.

SULTAN ALYAHYA received the B.Sc. degree (Hons.) in information systems from King Saud University, and the M.Sc. degree in information systems engineering and the Ph.D. degree in computer science from Cardiff University, U.K., in 2007 and 2013, respectively. He is currently an Assistant Professor with the College of Computer and Information Sciences, King Saud University. His main research interests include software project management, agile development, and computer supported co-operative work.

HMOOD AL-DOSSARI received the M.S. degree in computer science from King Saud University and the Ph.D. degree from Cardiff University. He is currently an Assistant Professor with the College of Computer and Information Sciences, King Saud University. His research interests include quality of service assessment, trust and reputation management systems, human and computer interaction, sentiment analysis, and social mining. He has several publications in international journals and conferences. He has attended various conferences and presented many seminars.

SHAFIQ AHMAD received the Ph.D. degree from RMIT University, Melbourne, Australia. He is currently an Assistant Professor with the College of Engineering, King Saud University. His research interests include smart manufacturing, performance analysis, and bibliometric. He has published a research book as well as a number of refereed research articles in international journals and conferences. He has over two decades experience both in industry and academia in Australia, Europe, and Asia.