MWMOTE—Majority Weighted Minority Oversampling Technique for Imbalanced Data Set Learning

Sukarna Barua, Md. Monirul Islam, Xin Yao, *Fellow*, *IEEE*, and Kazuyuki Murase

Abstract—Imbalanced learning problems contain an unequal distribution of data samples among different classes and pose a challenge to any classifier as it becomes hard to learn the minority class samples. Synthetic oversampling methods address this problem by generating the synthetic minority class samples to balance the distribution between the samples of the majority and minority classes. This paper identifies that most of the existing oversampling methods may generate the wrong synthetic minority samples in some scenarios and make learning tasks harder. To this end, a new method, called Majority Weighted Minority Oversampling TEchnique (MWMOTE), is presented for efficiently handling imbalanced learning problems. MWMOTE first identifies the hard-to-learn informative minority class samples and assigns them weights according to their euclidean distance from the nearest majority class samples. It then generates the synthetic samples from the weighted informative minority class samples using a clustering approach. This is done in such a way that all the generated samples lie inside some minority class cluster. MWMOTE has been evaluated extensively on four artificial and 20 real-world data sets. The simulation results show that our method is better than or comparable with some other existing methods in terms of various assessment metrics, such as geometric mean (G-mean) and area under the receiver operating curve (ROC), usually known as area under curve (AUC).

Index Terms—Imbalanced learning, undersampling, oversampling, synthetic sample generation, clustering

1 Introduction

Imbalanced learning problems contain unequal distribution of data samples among different classes, where most of the samples belong to some classes and rest to the other classes. If such samples come only from two classes, the class having most of the samples is called the majority class and other the minority class. Learning from the imbalance data is of utmost important to the research community as it is present in many vital real-world classification problems, such as medical diagnosis [1], information retrieval systems [2], detection of fraudulent telephone calls [3], detection of oil spills in radar images [4], data mining from direct marketing [5], and helicopter fault monitoring [6]. The imbalance ratio, the ratio between the number of the minority class and majority class samples, has been found to be as high as 1:100 in the fraud detection problem [3] and 1:100,000 in the classification problem of high-energy physics [7].

- *S. Barua and M.M. Islam are with the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka 1000, Bangladesh. E-mail: sukarna_barua@yahoo.com, mdmonirulislam@cse.buet.ac.bd.*
- *X. Yao is with Natural Computation Group, School of Computer Science, University of Birmingham, Edgbaston, Birmingham B15 2TT, United Kingdom. E-mail: x.yao@cs.bham.ac.uk.*
- *K. Murase is with the Department of Human and Artificial Intelligence Systems, University of Fukui, 3-9-1 Bunkyo, Fukui 910-8507, Japan. E-mail: murase@synapse.his.u-fukui.ac.jp.*

Manuscript received 24 Apr. 2012; revised 11 Sept. 2012; accepted 7 Nov. 2012; published online 16 Nov. 2012.
Recommended for acceptance by L. Khan.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2012-04-0282.
Digital Object Identifier no. 10.1109/TKDE.2012.232.

The primary goal of any classifier is to reduce its classification error, i.e., to maximize its overall accuracy. However, imbalance learning problems pose a great challenge to the classifier as it becomes very hard to learn the minority class samples [8], [9], [10]. It is because the classifier learned from the imbalanced data tends to favor the majority class samples, resulting in a large classification error over the minority class samples [11]. This becomes very costly when identification of the minority class samples is crucial [1], [2], [3], [4], [6], [7]. Thus, the classifier learned from the imbalanced data needs to perform equally well both on the minority class and the majority class samples.

Imbalance that exists between the samples of two classes is usually known as *between-class* imbalance. The actual cause for the bad performance of conventional classifiers on the minority class samples is not necessarily related to only on the between-class imbalance. Classifiers' performance have been found to depreciate in the presence of *within-class* imbalance and small disjuncts problems [12], [13], [14]. Besides, the complexity of data samples is another factor for the classifiers' poor performance [12]. If the samples of the majority and minority classes have more than one concepts in which some concepts are rarer than others and the regions between some concepts of different classes overlap, then the imbalance problem becomes very severe [13], [14].

Some of the most popular approaches to deal with imbalanced learning problems are based on the synthetic oversampling methods [15], [16], [17], [18]. In this paper, we illustrate that in some scenarios many of these methods become inappropriate and fail to generate the useful synthetic minority class samples. We also show scenarios

for which the methods will generate wrong and unnecessary synthetic samples, thus will make learning tasks difficult. In this respect, we propose a novel synthetic oversampling method, i.e., Majority Weighted Minority Oversampling TEchnique (MWMOTE), whose goal is to alleviate the problems of imbalanced learning and generate the useful synthetic minority class samples. The essences of the proposed method are: 1) selection of an appropriate subset of the original minority class samples, 2) assigning weights to the selected samples according to their importance in the data, and 3) using a clustering approach for generating the useful synthetic minority class samples.

The remainder of this paper is divided into six sections. In Section 2, we provide a brief review of existing works on imbalanced problem domain. Our motivations behind the proposed MWMOTE are presented in Section 3. In Section 4, we describe MWMOTE and its various components in detail. The experimental study and simulation results are presented in Section 5. Finally, in Section 6, we conclude the paper with some future research directions.

2 Related Works

Significant works have been done for handling the imbalanced learning problems. Most of these works fall under four different categories: sampling-based methods, cost-based methods, kernel-based methods, and active learning-based methods [19]. Although, there is no single best approach to deal with imbalance problems, sampling methods have been shown to be very successful in recent years. In this paper, we are interested in sampling methods and provided here a brief overview of the works performed in this category only. Details of work performed on the other categories can be found in [19].

In imbalanced learning, sampling methods focus on balancing the distribution between the majority class and the minority class samples. Although it is impossible to predict what true class distribution should be [20], [11] it is observed that classifiers learn well from a balanced distribution than from an imbalanced one [21], [22], [23]. Two different sampling methods exist in the literature. They are undersampling and oversampling. Undersampling methods work by reducing the majority class samples. This reduction can be done randomly in which case it is called random undersampling [24] or it can be done by using some statistical knowledge in which case it is called informed undersampling [25]. Some informed undersampling methods also apply data cleaning techniques [26] to further refine the majority class samples [13], [27].

Oversampling methods, on the other hand, add the minority class samples to an imbalanced data set. These methods can be categorized into random oversampling and synthetic oversampling. Random oversampling is a non-heuristic method that adds samples through the random replication of the minority class samples [14], [28]. This kind of oversampling sometimes creates very specific rules, leading to overfitting [9]. However, synthetic oversampling methods add samples by generating the synthetic minority class samples. The generated samples add essential information to the original data set that may help to improve classifiers' performance. Depending on the technique of how synthetic samples will be generated, various methods exist in the literature such as Synthetic Minority Oversampling TEchnique (SMOTE) [15], Borderline-SMOTE [16], and Adaptive Synthetic Sampling Technique (ADASYN) [17].

Some sampling methods first use clustering to partition the data set and then apply undersampling and/or oversampling on different partitions' data [29], [30], [31]. A cluster-based oversampling method was proposed in [29], which randomly oversampled both the minority class and majority class samples in such a way that all clusters became the same size. In [30], clustering was applied within each large class for producing subclasses with relatively balanced sizes and random oversampling was applied on the minority class samples. Cieslak and Chawla [31] proposed a cluster-based algorithm, called local sampling, in which the Hellinger distance measure [32] is used first for partitioning the original data set. A sampling method is then applied to each partition and finally data of all partitions are merged to create the new data set.

An oversampling method may create additional bias to classifiers, which may decrease classifiers' performance on the majority class samples. To prevent such degradation, an ensemble method, for example, boosting [33], [34], can be integrated with the oversampling method. While oversampling focuses on improving classifiers' performance on the minority class samples, boosting iteratively focuses on the hard-to-learn majority class samples. Thus, boosting and oversampling together provide a good option for efficiently learning imbalanced data [18], [35], [36], [37].

It has been shown that oversampling is lot more useful than undersampling and oversampling dramatically improves classifiers performance even for complex data [12]. However, both oversampling and undersampling are effective and it is still an arguable question whether one dominates the other. Even, other methods besides the sampling-based strategy also work comparably well and there is no single best method for all scenarios. While comparing oversampling and undersampling one natural observation favoring oversampling is that undersampling may remove essential information from the original data, while oversampling does not suffer from this problem. Regardless of the method used to counter the imbalance problem, factors such as the uncertainty of the true distribution between the samples of the minority class and majority class samples, complexity of data, and noise in data may pose a limit on the classifiers' performance [13], [14], [38].

3 Motivation

Synthetic oversampling methods have been shown to be very successful in dealing with imbalance data [15], [16], [17], [18]. However, our study finds out some insufficiencies and inappropriateness of the existing methods that may occur in many different scenarios. We describe them in this section.

One main objective of the synthetic oversampling methods, for example, Borderline-SMOTE [16], is to identify the border-line minority class samples (also called the seed samples). The oversampling method uses these seed samples for generating the synthetic samples because

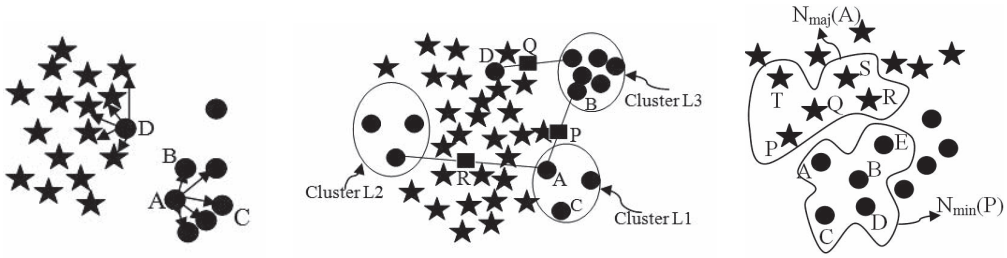


Fig. 1. (a) k-nearest neighbors of A and D are shown by arrow (for $k = 5$). (b) Several minority clusters in data set. Synthetic samples generated are shown by square. (c) Nearest majority neighbor set, $N_{maj}(A)$, and nearest minority neighbor set, $N_{min}(P)$ for $k2 = k3 = 5$.

they are most likely to be misclassified by a classifier. That is why Borderline-SMOTE generates the synthetic samples in the neighborhood of the border(s). According to [16], a minority class sample is identified as a borderline sample, if the number of the majority class samples, m, among its k-nearest neighbors satisfies

$$\frac{k}{2} \leq m < k, \tag{1}$$

where k is a user-specified parameter. The above criterion may fail to identify the borderline samples in some scenarios. Fig. 1a shows such a scenario, where the stars and circles represent the samples of the majority and minority classes, respectively. The minority class samples A and B of the figure have no majority class samples in their k-nearest neighborhood (assuming $k = 5$). It can be seen from Fig. 1a that the five-nearest neighbors of A (shown by arrows) are the minority class samples. Hence, m for both A and B would be 0. According to (1), these values, i.e., $m = 0$ and $k = 5$, relinquish the chances for A and B to be selected as the borderline samples, though they are the closest samples to the decision boundary (apparently borderline). The outcome is that later in the sample generation phase no synthetic samples will be generated from A and B. This situation will make learning tasks difficult due to insufficient information about the minority class samples near the decision boundary.

Oversampling methods ADASYN [17] and RAMOBoost [18] try to avoid the aforementioned problem by adaptively assigning weights to the minority class samples. A large weight helps in generating many synthetic samples from the corresponding minority class sample [17] or enhances the chance for the minority class sample as a participant in the synthetic sample generation process [18]. To assign the weight, both [17] and [18] use a parameter δ, defining the number of the majority class samples among the k-nearest neighbors of the minority class sample. A large δ increases the weight, while a small δ reduces it. However, the use of δ for assigning the weights may encounter the following problems:

1. *The parameter δ is inappropriate for assigning weights to the minority class samples located near the decision boundary.* This can be understood from Fig. 1a in which the minority class samples A and B have no majority class sample in their k-nearest neighborhood (assuming $k = 5$). Under this condition, δ for these samples will be 0. It means that A and B will be given the lowest weight, although seemingly they are the most important samples due to their position near the decision boundary.
2. *The parameter δ is insufficient to distinguish the minority class samples with regard to their importance in learning.* It can be seen from Fig. 1a that the minority class sample A is closer to the decision boundary than C. It is, therefore, reasonable that A should be given higher weight (importance) than C. However, if we compute δ with $k = 5$, δ would be 0 for both A and C, because there is no majority class sample in the neighborhood of these two samples (Fig. 1a). We may think that it is possible to avoid this problem and the one stated in problem 1 by increasing the value of k. The shortcomings with this type of solution are described as follows:

   First, the value of k will not be the same in all regions of the minority class samples. In a region, where the minority class samples are close to the majority class samples and the former ones are sparsely located, the low value of k may suffice to include the majority class samples. However, in some other regions, the value of k has to be large enough for including the majority class samples. Hence, it would be difficult to find an appropriate value of k a priori. It should be kept in mind that an appropriate k leads to a proper δ that is used for assigning the weights to the minority class samples. Second, even if we choose k to include few majority class samples, the δ value will still be lower. This means the borderline samples will not get a larger δ relative to the other samples, although they deserve it. It is now understood that δ cannot differentiate the minority class samples according to their importance in learning.
3. *The parameter δ may favor noisy samples.* Now, consider the minority class sample D as shown in Fig. 1a. Since this sample falls in the majority class region, it is likely to be noisy. It is seen that the five-nearest neighbors of D are the majority class samples. If we compute δ with $k = 5$, it would be 5. Thus, D is going to get the highest weight unexpectedly and consequently, later in the sample generation phase more noisy synthetic samples will be generated from this sample. This will eventually make learning tasks difficult because of the introduction of more noisy samples in the data set.

In the sample generation phase, existing synthetic oversampling methods (e.g., [15], [17], [18]) employ the k-nearest neighbor (also called k-NN)-based approach. To generate a synthetic sample from an original minority class sample, say x, the k-NN-based approach randomly selects another minority class sample, say y, from the k-nearest neighbors of x. The approach then generates a synthetic sample, g, by using the linear interpolation of x and y. This can be expressed as

$$g = x + (y - x) \times \alpha, \tag{2}$$

where α is a random number in the range $[0, 1]$. The above equation says that g will lie in the line segment between x and y. However, in many cases, the k-NN-based approach may generate wrong minority class samples. To show why, again consider Fig. 1b. Assume, we are going generate a synthetic sample from the minority class sample A and $k = 5$ in this case. The k-NN-based approach will randomly select another minority class sample from the five-nearest minority neighbors of A. Let the sample B is selected. According to (2), the linear interpolation of A and B may generate a synthetic sample, say P, shown by the square in Fig. 1b. It can be seen from Fig. 1b that P is clearly a wrong minority class sample because it overlaps with a majority class sample.

The above problem will be magnified when the small sized clusters are present in the minority class concept such as clusters $L1$ and $L2$ shown in Fig. 1b. If we generate a synthetic sample by using any member of the cluster $L2$, the k-NN-based approach may select a member from the other cluster (say, $L1$). According to (2), the generated synthetic sample, i.e., R will lie inside the majority class region situated between $L1$ and $L2$ (Fig. 1b). This overlap will definitely make learning tasks difficult. Let we like to generate a synthetic sample from the noisy sample D. In this scenario, the synthetic sample Q may be generated, which will supposed to be noisy and lie inside the majority class region (Fig. 1b). It is natural that in learning Q will induce more difficulty than R. Now, consider the sample B, a member of the dense minority cluster $L3$, is to be used for the synthetic sample generation (Fig. 1b). For $k = 5$, the generated samples will be very similar to the existing samples of $L3$. This is due to the fact that all the k-nearest neighbors of B are very close to each other. One such generation is shown in Fig. 1b. Generation of such a nearly duplicated sample is less useful because it does not add any new information to the imbalanced data.

All these examples illustrate one point, i.e., the k-NN-based sample generation approach is inappropriate in some cases. This approach may generate duplicate and wrong synthetic minority class samples from the members of dense and small-sized clusters, respectively. It is important to note that when the k-NN-based approach generates the synthetic samples from the noisy samples, the generated samples will also be noisy. The aforementioned problems occur due to the fact that the approach uses all the k-nearest neighbors blindly without considering the position and distance of the neighbors from the minority class sample under consideration. Moreover, the appropriate value of k cannot be determined in advance.

4 The Proposed Method

Motivated by the problems stated in the previous section, we devise a novel minority oversampling method, MWMOTE. The objective of the method is twofold: to improve the sample selection scheme and to improve the synthetic sample generation scheme. Our MWMOTE involves three key phases. In the first phase, MWMOTE identifies the most important and hard-to-learn minority class samples from the original minority set, S_{min} and construct a set, S_{imin}, by the identified samples. In the second phase, each member of S_{imin} is given a selection weight, S_w, according to its importance in the data. In the third phase, MWMOTE generates the synthetic samples from S_{imin} using S_ws and produces the output set, S_{omin}, by adding the synthetic samples to S_{min} . The complete method is presented in [Algorithm 1]. Steps 1 to 6 and 7 to 9 comprise the first phase and the second phase, respectively. Finally, steps 10 to 13 complete our third phase. The main components of MWMOTE are described below.

Algorithm 1. $MWMOTE(S_{maj}, S_{min}, N, k1, k2, k3)$.
Input:

1) S_{maj}: Set of majority class samples
2) S_{min}: Set of minority class samples
3) N: Number of synthetic samples to be generated
4) $k1$: Number of neighbors used for predicting noisy minority class samples
5) $k2$: Number of majority neighbors used for constructing informative minority set
6) $k3$: Number of minority neighbors used for constructing informative minority set

Procedure Begin

1) For each minority example $x_i \in S_{min}$, compute the nearest neighbor set, $NN(x_i)$. $NN(x_i)$ consists of the nearest $k1$ neighbors of x_i according to euclidean distance.
2) Construct the filtered minority set, S_{minf} by removing those minority class samples which have no minority example in their neighborhood:
   $S_{minf} = S_{min} - \{x_i \in S_{min}: NN(x_i)$ contains no minority example$\}$
3) For each $x_i \in S_{minf}$, compute the nearest majority set, $N_{maj}(x_i)$. $N_{maj}(x_i)$ consists of the nearest $k2$ majority samples from x_i according to euclidean distance.
4) Find the borderline majority set, S_{bmaj}, as the union of all $N_{maj}(x_i)$s, i.e.,
   $S_{bmaj} = \bigcup_{x_i \in S_{minf}} N_{maj}(x_i)$
5) For each majority example $y_i \in S_{bmaj}$, compute the nearest minority set, $N_{min}(y_i)$. $N_{min}(y_i)$ consists of the nearest $k3$ minority examples from y_i according to euclidean distance.
6) Find the informative minority set, S_{imin}, as the union of all $N_{min}(y_i)$s, i.e., $S_{imin} = \bigcup_{y_i \in S_{bmaj}} N_{min}(y_i)$
7) For each $y_i \in S_{bmaj}$ and for each $x_i \in S_{imin}$, compute the information weight, $I_w(y_i, x_i)$.
8) For each $x_i \in S_{imin}$, compute the selection weight $S_w(x_i)$ as
   $S_w(x_i) = \sum_{y_i \in S_{bmaj}} I_w(y_i, x_i)$

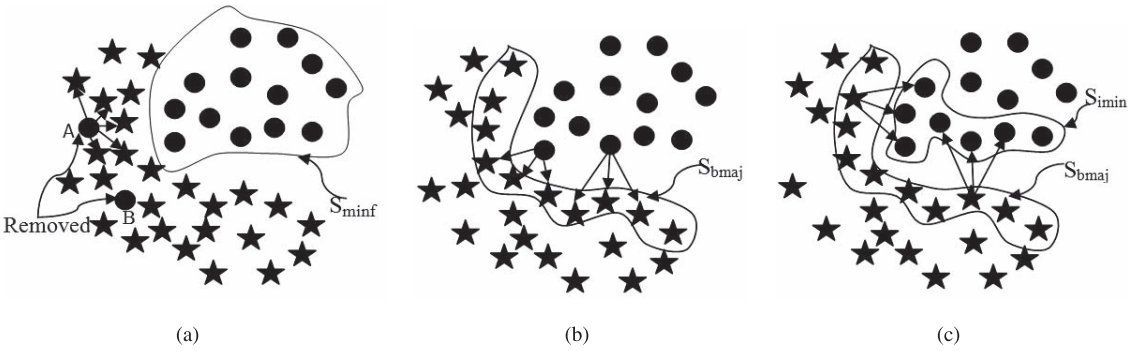


Fig. 2. Construction of the set S_{imin}: (a) S_{minf} is found after removing noisy minorities such as samples A and B. (b) S_{bmaj} is found. (c) S_{imin} is found.

9) Convert each $S_w(x_i)$ into selection probability $S_p(x_i)$ according to
$S_p(x_i) = S_w(x_i)/\sum_{z_i \in S_{imin}} S_w(z_i)$
10) Find the clusters of S_{min}. Let, M clusters are formed which are $L_1, L_2, \ldots, L_M$.
11) Initialize the set, $S_{omin} = S_{min}$.
12) Do for $j = 1 \ldots N$.
    a) Select a sample x from S_{imin} according to probability distribution $\{S_p(x_i)\}$. Let, x is a member of the cluster $L_k, 1 \leq k \leq M$.
    b) Select another sample y, at random, from the members of the cluster L_k.
    c) Generate one synthetic data, s, according to $s = x + \alpha \times (y - x)$, where α is a random number in the range $[0, 1]$.
    d) Add s to S_{omin}: $S_{omin} = S_{omin} \cup \{s\}$.
13) End Loop

End

Output: The oversampled minority set, S_{omin}.

4.1 Construction of the set S_{imin}

All the minority class samples might not be important for generating the synthetic samples because some of them might be very easy for learning. It is, therefore, necessary to identify a set of hard-to-learn samples to be used for generating the synthetic samples. These samples are those that are usually located near the decision boundary and/or belong to the small-sized cluster(s). Most of the existing oversampling methods (e.g., [15], [17], [18]) do not explicitly identify the hard-to-learn samples. Although, Borderline-SMOTE [16] tries to find the set of borderline minority (hard-to-learn) samples, we have shown that it fails to correctly identify all the borderline samples (Section 3). In MWMOTE, we adopt a new approach for identifying the important minority class samples and construct the informative minority set, S_{imin}.

Before going to the detail construction of S_{imin}, we here explain few terms for a sample, x. They are the nearest neighbor set ($NN(x)$), nearest majority neighbor set ($N_{maj}(x)$), and nearest minority neighbor set ($N_{min}(x)$). All these sets use the euclidean distance for choosing neighbors. Our $NN(x)$ contains the nearest $k1$ neighbors of x. Similarly, $N_{maj}(x)$ contains the nearest $k2$ majority neighbors and $N_{min}(x)$ contains the nearest $k3$ minority neighbors. The neighbors in $NN(x)$ can be the minority class samples, majority class samples or both. However, the nearest $k2$ neighbors in $(N_{maj}(x))$ and the nearest $k3$ neighbors in $(N_{min}(x))$ are only the majority class and minority class samples, respectively. To understand the fact clearly, consider Fig. 1c and assume $k1 = k2 = k3 = 5$. Our $NN(A)$ and $N_{maj}(A)$ for the minority class sample A are $\{P, Q, B, C, D\}$ and $\{P, Q, R, S, T\}$, respectively. However, for the majority class sample P, $N_{min}(P)$ is $\{A, B, C, D, E\}$.

The whole process of constructing S_{imin} (Steps 1-6 of [Algorithm 1]) can be described as follows:

1. Our MWMOTE first filters the original minority set, S_{min}, to find a filtered minority set, S_{minf}. To do this, we compute $NN(x_i)$ for each $x_i \in S_{min}$. We then remove each x_i if its $NN(x_i)$ contains only the majority class samples. The removed minority class sample is likely to be noisy because it is surrounded only by the majority class samples. This removal prohibits the noisy minority class sample to take part in the synthetic sample generation process. Thus, MWMOTE will be able to remove existing noisy samples from the given data and at the same time it will not add any new noise, i.e., noisy synthetic sample to the data. Fig. 2a shows this process for $k1 = 5$. It is clear from this figure that all the five-nearest neighbors of the minority class sample A (shown by arrow) are the majority class samples. Hence, the sample A is removed from S_{min} and similarly, the sample B. Finally, MWMOTE uses the rest of the samples for S_{minf}.
2. For each $x_i \in S_{minf}$, MWMOTE constructs $N_{maj}(x_i)$. The samples in $N_{maj}(x_i)$ will be the borderline majorities and expected to be located near the decision boundary when $k2$ is small. We combine all the $N_{maj}(x_i)$s to form the borderline majority set, S_{bmaj} (Steps 3-4 of [Algorithm 1]). In Fig. 2b, we show the construction of S_{bmaj} for $k2 = 3$, where the nearest majority class samples for some minorities are explicitly shown by arrows.
3. For each $y_i \in S_{bmaj}$, MWMOTE constructs $N_{min}(y_i)$ and combines all such $N_{min}(y_i)$s to form S_{imin} (Steps 5-6 of [Algorithm 1]). The parameter $k3$ used in $N_{min}(y_i)$ needs to be large enough for including all the hard-to-learn minority class samples required to generate the synthetic samples. A large $k3$ ensures the participation of many difficult samples in the sample generation process. The generated samples will likely add sufficient and essential information to learning. Fig. 2c shows this step for $k3 = 3$, where the nearest minority neighbors for some majority class samples are explicitly shown by arrows.

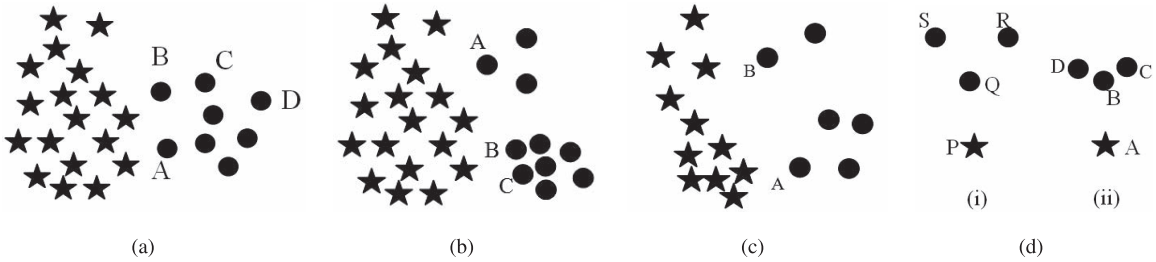


Fig. 3. (a)-(c) Figures illustrating the three observations on weighting minority class samples (d) Figure illustrating the computation of density factor (i) Minority class sample Q is in a sparse cluster, (ii) Minority sample B is in a dense cluster.

In Fig. 1a, we illustrated that δ can be zero due to no majority class samples in the k-nearest neighborhood of a minority class sample x_i. This can happen when the minority class samples are very close to each other or very far away from the majority class samples. One may relate this situation with the one presented in Fig. 2b and may think that there will be no majority class samples satisfying the $k2/k3$-nearest neighbors; hence, the boundary set S_{bmaj} and informative set S_{imin} will be empty. Even, he/she may think that we can have higher k value to get majority neighbors. However, these perceptions are incorrect. Our S_{bmaj} is the union of the nearest majority neighbors sets, i.e., $N_{maj}(x_i)$s (Step 4 of [Algorithm MWMOTE]) and each $N_{maj}(x_i)$ contains the nearest $k2$ (not k-nearest) majority samples from x_i. In Fig. 2b, $N_{maj}(x_i)$s are shown by arrows for some minority class samples (assume $k2 = 3$). It is clear from this figure that $N_{maj}(x_i)$ will never become empty due to minority samples' position or distribution as long as there are $k2$ majority samples in the data set. The same explanation applies to finding S_{imin} using the $k3$-nearest minority neighbor set.

4.2 Finding the Weights for the Members of S_{imin}

In the preceding section, we showed how MWMOTE constructs a set of the hard-to-learn minority class samples, S_{imin}, to be used for generating the synthetic samples. However, all the samples of this set may not be equally important. Some samples may provide more useful information to the data than the others. Hence, it is necessary for assigning weights to the samples according to their importance. A large weight implies that the sample requires many synthetic samples to be generated from and nearby it. This is due to the insufficiency of information in its minority concept.

Some oversampling methods (e.g., [17], [18]) use a different approach than the one used in this work for weighting the minority class samples. However, it is clear from Section 3 that the weighting mechanism used by the methods is inappropriate and insufficient in many scenarios. So, our MWMOTE uses a new mechanism for assigning weights properly. In steps 7 to 9 of [Algorithm 1], the weight is calculated for each minority class sample $x_i \in S_{imin}$ and it is called the selection weight, $S_w(x_i)$. Our MWMOTE computes this weight based on the following three important observations:

Observation 1: Samples close to the decision boundary contain more information than those of further. This observation indicates that the closer samples should be given a higher weight than the further ones. Consider Fig. 3a in which the samples A and B are closer to the decision boundary compared to those of C and D. Therefore, A and B are more informative than C and D. Similarly, C is more informative than D. It implies that A and B need to be given a higher S_w than C and D. At the same time, C is to be given a higher S_w than D.

Observation 2: The minority class samples in a sparse cluster are more important than those in a dense cluster. From the perspective of the synthetic sample generation, the members of a sparse cluster are more important than those of a dense cluster. This is due to the fact that the dense cluster contains more information than the sparse cluster. Thus, the sparse cluster requires more synthetic samples to increase its size for reducing within-class imbalance. This is why the members of the sparse cluster deserve higher S_ws than those of the dense cluster. Consider Fig. 3b, where the sample A is more important than B and C though all of them are close to the decision boundary. This is obvious because A is a member of the sparse cluster, while B and C are the members of the dense cluster.

Observation 3: The minority class samples near a dense majority class cluster are more important than those near a sparse majority class cluster. Compare samples A and B shown in Fig. 3c. Both of these samples are equally distant from the decision boundary and are the members of the equal-sized clusters. However, the density of the majority class neighbors near to A is higher than that to B. This relative imbalance will make difficulty for a classifier to correctly learn A. Because of this A is more important for the synthetic sample generation than B and its S_w will be higher than that of B.

It is now understood that S_w is to be computed by considering the aforementioned observations. Our MWMOTE considers them and employs the majority class set S_{bmaj} in computing S_w, which can be described as follows:

1. Each majority class sample $y_i \in S_{bmaj}$ gives a weight to each minority class sample $x_i \in S_{imin}$. This weight is called the information weight, $I_w(y_i, x_i)$.
2. For x_i, we sum up all the $I_w(y_i, x_i)$s to find its selection weight, $S_w(x_i)$. This can be expressed as

$$S_w(x_i) = \sum_{y_i \in S_{bmaj}} I_w(y_i, x_i). \tag{3}$$

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

$$I_w(y_i, x_i) = C_f(y_i, x_i) \times D_f(y_i, x_i). \quad (4)$$

The closeness factor assigns higher weights to the closer samples than the further ones, satisfying the goal of our observation 1. To achieve the goal of the observation 2, the density factor assigns higher weights to the members of the sparse cluster than those of the dense cluster when they are equidistant from the decision boundary. Finally, the summation used in (3) facilitates our observation 3, i.e., the minority class samples having more majority class neighbors in S_{bmaj} will get a higher selection weight. Below, we describe how MWMOTE computes the two factors.

Closeness factor, $C_f(y_i, x_i)$. The computation of the closeness factor is very straightforward. If $x_i \notin N_{min}(y_i)$, then $C_f(y_i, x_i)$ is 0, otherwise $C_f(y_i, x_i)$. We compute $C_f(y_i, x_i)$ using the following the two steps:

1. We first compute the normalized euclidean distance, $d_n(y_i, x_i)$, from y_i to x_i. This can be expressed as

$$d_n(y_i, x_i) = \frac{dist(y_i, x_i)}{l}, \quad (5)$$

where $dist(y_i, x_i)$ is the euclidean distance from y_i to x_i and l is the dimension of the feature space.

2. We then compute $C_f(y_i, x_i)$ in the following way:

$$C_f(y_i, x_i) = \frac{f\left(\frac{1}{d_n(y_i, x_i)}\right)}{C_f(th)} * CMAX, \quad (6)$$

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

In (6), the inverse of the normalized euclidean distance, $(1/d_n(y_i, x_i))$, is first applied to f. We do so for ignoring the values that are too high and for slicing them to the highest value $C_f(th)$. The value, thus, found would lie $[0, CMAX]$. We define f in the following way:

$$f(x) = \begin{cases} x & \text{if } x \leq C_f(th), \\ C_f(th) & \text{otherwise.} \end{cases} \quad (7)$$

Density factor, $D_f(y_i, x_i)$. The density factor facilitates the observation 2. This means that the sparse cluster should have more synthetic samples than the dense cluster, given that both the clusters are equally distant from the decision boundary. It should be kept in mind that the observation 1 cannot be violated in satisfying the condition of the observation 2. Hence, MWMOTE computes $D_f(y_i, x_i)$ by normalizing $C_f(y_i, x_i)$. That is,

$$D_f(y_i, x_i) = \frac{C_f(y_i, x_i)}{\sum_{q \in S_{imin}} C_f(y_i, q)}. \quad (8)$$

Consider Fig. 3d to show why (8) gives a large weight to the members of the sparse clusters. Assume that the minority class samples B and Q are equally distant from the majority class samples A and P, respectively. Hence, their euclidean distances are equal and consequently, $C_f(A, B)$ and $C_f(P, Q)$ will be equal. Visual inspection, however, indicates that $C_f(A, C)(=C_f(A, D)) > C_f(P, R)(=(C_f(P, S))$. Hence, the denominator in (8) would be larger for $D_f(A, B)$ than that for $D_f(P, Q)$. So, $D_f(P, Q)$ will be greater than $D_f(A, B)$. This implies that although the closeness factors for B and Q are same, Q will get a larger density factor than B due to its relative position in the sparse cluster.

4.3 Synthetic Sample Generation

In Section 3, we discussed the problems of the k-NN-based sample generation approach used by most of the existing oversampling methods. To alleviate those problems, MWMOTE adopts a different approach based on clustering (Steps 10-13 of [Algorithm 1]). First, in Step 10, MWMOTE finds the clusters of the minority data set, S_{min}, using a modified hierarchical clustering algorithm, which we describe in the next section. Then, in Step 11, the oversampled minority data set, S_{omin}, is initialized by S_{min}. Finally, in Step 12, the synthetic minority class samples are generated using (2) and they are added to S_{omin}.

Although the sample generation approach described here and the k-NN-based approach used in the existing methods employ the same equation, i.e., (2), our approach differs with the others in the way how y in (2) is chosen for each sample, x (Step 12(b)). MWMOTE chooses y from the members of the x's cluster (Step 10 of [Algorithm 1]), while the k-NN-based approach randomly selects it from the k-nearest neighbors. It is obvious that if x and y reside in the same cluster, then according to (2) the generated synthetic samples will also lie inside the same cluster. This is beneficial in the sense that the generated samples will never erroneously fall in the majority class region. Another benefit of our cluster-based approach lies when the synthetic samples are generated from the noisy minority class samples. If x is a noisy sample, then it will likely form an isolated cluster, consisting of only one member (itself). In this case, y selected by our approach will be the same as x and (2) will then generate a duplicate sample, i.e., x. This is much better than the k-NN-based approach that may generate the noisy minority class samples, which in turn will enlarge the minority class region. An erroneously enlarged region is likely to overlap with the majority class region, creating difficulty in learning.

Consider Figs. 4a, 4b, and 4c to show why our cluster-based sample generation approach is better than the k-NN-based approach. To generate a synthetic sample from the minority class sample A and with $k = 5$, the k-NN-based approach randomly chooses a sample (say, D) from the A's five-nearest neighbors. This selection may lead to the generation of a synthetic sample (shown by the square in Fig. 4a(i), residing in the majority class region. In contrast, our cluster-based approach selects the sample not randomly but from the same cluster of A. It ensures that D will not be selected, because A and D are not likely to be the members of the same cluster (Fig. 4a(ii)). Our approach may select B because A and B are likely to be the members of the same cluster.

Another problem of the k-NN-based approach is that it may produce nearly duplicate samples. It can be seen from Fig. 4b(i) that the synthetic samples generated from A by the k-NN-based approach (assuming $k = 3$) are nearly duplicate, while they are more informative when generated by our approach (Fig. 4b(ii)). It may seem that k-NN approach may avoid this problem by increasing value of k. However, making k too large can introduce the other problem, i.e., the generation of wrong synthetic samples. In general, finding

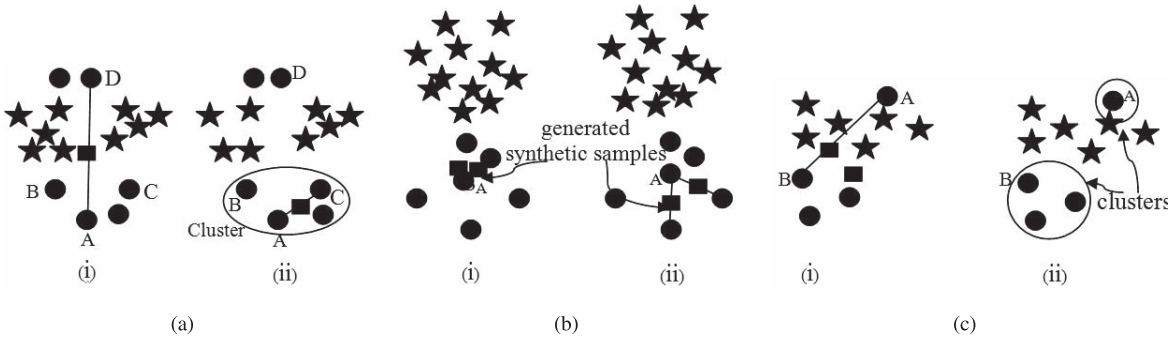


Fig. 4. Figure illustrating the difference between k-NN and proposed cluster-based data generation, where synthetic data are generated from a minority class sample A. (a) (i) Using k-nearest neighbor-based approach ($k = 5$) generates a synthetic sample (shown by square) which falls in the majority region. (ii) MWMOTE's cluster-based approach generates a sample which remains inside a minority cluster. (b)(i) Shows that k-nearest neighbor-based approach generates nearly duplicated samples from A. (ii) Cluster-based approach generates more informative samples from A. (c) (i) k-nearest neighbor-based approach generates more wrong samples (shown by square) from A. (ii) A forms an isolated cluster, so cluster-based approach generates a duplicate sample of A.

an appropriate k for all such cases is not easy. Clustering can likely avoid the problems by partitioning the data into different clusters and generating the synthetic samples inside the clusters. However, clustering may generate nearly duplicate sample from the samples of the dense cluster. We show that such scenarios are more likely to occur in the k-NN-based approach than in our approach (as in Fig. 4b(ii)). Furthermore, our approach mitigates such problems by assigning a small weight to the samples of the dense cluster (Observation 2 of Section 4.2).

Consider Fig. 4c to see what happens when the synthetic samples are generated from a noisy sample, say A. There is every possibility that the k-NN-based approach may generate a noisy sample from A ((shown by square in Fig. 4c(i)). However, in our cluster-based approach, A will form an individual cluster consisting of only one member (i.e., itself), and the generated sample will be just a duplication of it (Fig. 4c(ii)). This is much better than the k-NN approach that adds new noise to the data.

4.4 Clustering S_{min}

The success of MWMOTE is largely dependent on how we partition the set, S_{min} (Step 10 of [Algorithm 1]). For this purpose, MWMOTE uses average-linkage agglomerative clustering [39], [40], a hierarchical clustering process. Agglomerative clustering does not require the number of clusters to be fixed a priori. It generates clusters in a bottom-up fashion. The key steps of the algorithm are given below (assume, D data samples are given as input):

1. Assign each sample to a separate cluster, i.e., initially, there will be D clusters, each of size one.
2. Find the two closest clusters say, L_i and L_j.
3. Merge the clusters L_i and L_j into a single cluster, L_m. This will reduce the number of clusters by one.
4. Update the distance measures between the newly computed cluster and all the previous cluster(s).
5. Repeat Steps 2-4 until all data samples are merged into a single cluster of size D.

The basic algorithm described above produces one cluster of size D, which is not our goal at all. We can find more than one cluster, if we early stop the merging process in Step 3. For this purpose, MWMOTE uses a threshold, T_h and stops the merging process when the distance between closest pair exceeds T_h. Then, the output will be the set of clusters remaining at that point.

What should be value of T_h? Clearly, this value should not be constant because the distance measure will vary with dimension of the feature space. The same algorithm may produce a different number of clusters for the same type of data, where the only difference is the dimension of the feature space. The second problem of using a constant T_h lies in the fact that in some data sets samples are relatively sparse (average distance between the samples is large), while in other sets samples are relatively dense (average distance between the samples is small). A constant T_h will produce few clusters for those data sets where the average distance is small and many clusters where the average distance is large. The key point here is that T_h should be the data dependent and it should be calculated using some heuristics of the distance measures between samples of the data. In this work, we compute T_h as follows:

1. Find the average distance d_{avg} as

$$d_{avg} = \frac{1}{|S_{minf}|} \sum_{x \in S_{minf}} \min_{y \neq x, y \in S_{minf}} \{dist(x, y)\}. \quad (9)$$

2. Compute T_h by multiplying d_{avg} with a constant parameter, C_p as

$$T_h = d_{avg} \times C_p. \quad (10)$$

For each member of S_{minf} (rather than S_{min} to avoid the effect of noisy minority class samples), we find the minimum euclidean distance to any other member in the same set. We then compute the average of all these minimum distances to find d_{avg}. The parameter C_p is used to tune the output of the clustering algorithm. Increasing C_p will increase the cluster size, but reduces the number of clusters. The opposite scenario will happen if we decrease C_p.

5 Experimental Study

In this section, we evaluate the effectiveness of our proposed MWMOTE and compare its performance with

TABLE 1
Confusion Matrix

		True class	
		Positive	Negative
Classifier output	Positive	TP	FP
	Negative	FN	TN
	Rowsum	p	n

SMOTE [15], ADASYN [17], and RAMO [18]. The RAMO method is found from [18] by excluding its boosting part. This is done for a fair comparison. We use one artificial data set with different complexities and imbalance ratios. We also use 20 real-world data sets collected from the UCI machine learning repository [41]. Three different sets of experiments are performed on each category of data. These include the evaluation of a method using 1) a single classifier, 2) an ensemble of classifiers, and 3) a different classifier than the one used in steps 1 and 2.

5.1 Assessment Metric

When dealing with two class classification problems, we can always label one class as a positive and one class as a negative. Let the testing set consists of p positive and n negative samples, respectively. The task of any classifier is to assign a class to each of the samples, but some of the assignments might be wrong. To assess the classifier performance, we count the number of true positive (TP), true negative (TN), false positive (FP) (actually negative, but classified as positive), false negative (FN) (actually positive, but classified as negative) samples and form a confusion matrix (Table 1).

Using Table 1, several performance measures can be derived. The first and most common is overall accuracy (or simply accuracy) and can be defined as

$$accuracy = \frac{TP + TN}{p + n}. \tag{11}$$

The main problem associates with the accuracy measure is its dependence on the distribution of positive class and negative class samples in the data set, thus not suitable for imbalanced learning problems [19]. Some other performance measures derived from the Table 1 are commonly used in imbalanced learning. They are precision, recall, geometric-mean (G-mean), and F-measure [19].

G-mean is a good indicator for assessing the overall performance of a classifier, because it combines the classifier's accuracy on the positive class and negative class samples. Hence, a large value of this metric indicates that the classifier performs equally well on the samples of both classes. While G-mean focuses on both classes, F-measure combines precision and recall for maximizing the performance on a single class. Hence, it is used for measuring the performance of the classifier on the minority class samples. Another very popular measure is the area under the Receiver Operating Characteristics (ROC) graph [42], usually known as AUC. Unlike G-mean and F-measure, AUC is not sensitive to the distribution between the positive class and negative class samples, thus suitable for performance comparison of different classifiers [42], [43]. The ROC graph is obtained by plotting the false positive rate (FPR) on the X-axis and the true positive rate (TPR) on the Y-axis, where

$$FPR = \frac{FP}{n}, \quad TPR = \frac{TP}{p}. \tag{12}$$

We use the Wilcoxon signed-rank test [44] to statistically compare the performance of two different methods. The test is applied to compare MWMOTE with each of the other methods in a pairwise manner. Suppose μ pairs of results are obtained that are being tested. At first, we compute the difference for each pair. Let, the difference is d_i for $i = 1, \ldots, \mu$. According to the absolute value, all the μ differences are sorted from the smallest to the largest and assigned a rank to each of them, starting from rank 1 for the smallest and ending at rank μ for the largest. In case, if more than one differences are equal, an average rank is assigned to each of them [44]. The ranks are converted to signed-ranks by giving it the sign of the difference. All the positive and negative ranks are separately summed up and they are denoted as $R+$ and $R-$, respectively. The minimum between $R+$ and $R-$ is found and termed as T. At a specific significance level (usually a significance level 0.05 is used), the T value is compared against a critical value obtained from a table [45]. The null hypothesis is that all performance differences observed by two methods occur by chance. If the calculated T value is less than or equal to the critical value, then we reject the null hypothesis, indicating the difference is statistically significant and cannot occur by chance.

5.2 Selection of Classifiers and Parameter Settings

Since our main focus is evaluating the oversampling methods, we do not adhere to choose any specific classifier, rather we use several of them such as neural network, ensemble of neural networks, k-NN, and decision tree classifiers. As mentioned before, we run three sets of experiments. In the first and second sets of experiments, we use neural network and ensemble of neural networks, respectively. The AdaBoost.M2 is chosen for ensemble because of its better performance for imbalance problems [18], [36], [37]. We first apply an oversampling method to a given data set and then run AdaBoost.M2 on the oversampled data set with 20 boosting iterations [47]. To know whether the performance of the oversampling methods remains good or bad independent of the chosen classifier, we use the k-NN (with $k = 5$) and C4.5 decision tree classifiers [48] in the third set of experiments

We use back propagation learning algorithm in the first and second set of experiments. The number of hidden neurons is randomly set to 5, the number of input neurons is set to be equal to the number of features in the data set, and the number of output neurons is set to 2. We use the sigmoid function as an activation function. The number of training epochs is randomly set to 300 and learning rate to 0.1. For MWMOTE, the values for different parameters are: $k1 = 5, k2 = 3, k3 = |S_{min}|/2, C_p = 3, C_f(th) = 5$, and $CMAX = 2$. All these values are chosen after some preliminary runs and they are not meant to be optimal. For SMOTE and ADASYN, the value of the nearest neighbors, k, is set to 5 [15], [17]. The values of the nearest

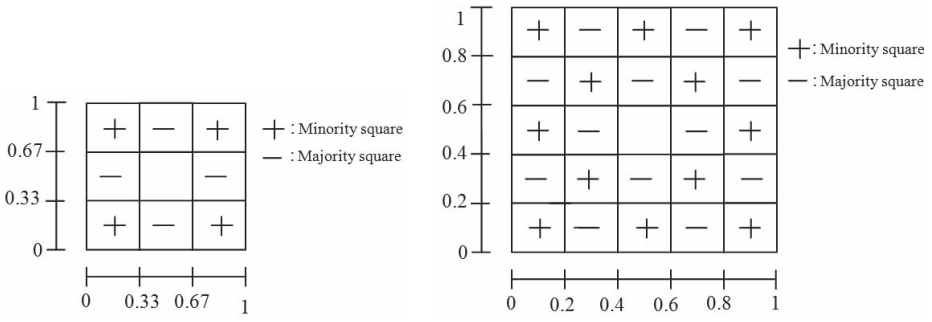


Fig. 5. Figure describing artificial problem domain for: (a) complexity level, 3×3 (b) complexity level, 5×5.

TABLE 2
Accuracy, Precision, and Recall Performance Values of SMOTE [15], ADASYN [17], RAMO [18], and MWMOTE on Artificial Domains Using Single Neural Network Classifier, and AdaBoost.M2 Ensemble of Neural Network Classifier

		Single Neural Network			AdaBoost.M2 Ensemble of Neural Network		
Data set	Methods	Accuracy	Precision	Recall	Accuracy	Precision	Recall
Complexity 3X3, Imbalance ratio 1:3	SMOTE	0.75625	0.51958	0.7	0.76875	0.53487	0.85833
	ADASYN	0.74167	0.51612	0.6875	0.76771	0.52755	0.87917
	RAMO	0.75521	0.52506	0.68333	0.76354	0.52324	0.875
	MWMOTE	**0.80937**	**0.64072**	**0.6875**	**0.9**	**0.76165**	**0.89583**
Complexity 3X3, Imbalance ratio 1:10	SMOTE	0.88955	0.46875	0.2193	0.90909	0.52024	0.62917
	ADASYN	0.89553	0.46151	0.23246	0.90568	0.5038	0.59167
	RAMO	0.8982	0.48992	0.23958	0.90795	0.53235	0.53333
	MWMOTE	**0.90829**	**0.60369**	**0.26754**	**0.95682**	**0.98786**	**0.6375**
Complexity 5X5, Imbalance ratio 1:3	SMOTE	0.55174	0.2758	0.48333	0.65451	0.38094	0.58611
	ADASYN	0.5375	0.26886	0.49306	0.62917	0.36147	0.60417
	RAMO	0.52882	0.26615	**0.49583**	**0.69375**	**0.41835**	0.54583
	MWMOTE	**0.56424**	**0.28124**	0.475	0.65799	0.38973	**0.61528**
Complexity 5X5, Imbalance ratio 1:10	SMOTE	0.76024	0.46143	0.23457	0.89926	0.414	**0.097953**
	ADASYN	0.78313	0.43821	0.21732	0.9	0.38935	0.080556
	RAMO	0.75505	0.42935	**0.23704**	0.90265	0.46988	0.084795
	MWMOTE	**0.87374**	**0.63631**	0.10526	**0.90457**	**0.49378**	0.093056

Best results are highlighted in bold-face type.

TABLE 3
F-Measure, G-Mean, and AUC Performance Values of SMOTE [15], ADASYN [17], RAMO [18], and MWMOTE on Artificial Domains Using Single Neural Network Classifier, and AdaBoost.M2 Ensemble of Neural Network Classifier

		Single Neural Network				AdaBoost.M2 Ensemble of Neural Network			
Data set	Methods	F-measure	G-mean	AUC	Std of AUC	F-measure	G-mean	AUC	Std of AUC
Complexity 3X3, Imbalance ratio 1:3	SMOTE	0.58768	0.72963	0.78119	0.091193	0.65504	0.79439	0.86924	0.041639
	ADASYN	0.58102	0.71935	0.77609	**0.085673**	0.66189	0.805	0.88421	0.02833
	RAMO	0.58129	0.71997	0.76172	0.093432	0.65216	0.79536	0.88641	0.049514
	MWMOTE	**0.64897**	**0.75855**	**0.8059**	0.092268	**0.81481**	**0.89239**	**0.92833**	**0.024689**
Complexity 3X3, Imbalance ratio 1:10	SMOTE	0.27038	0.43235	0.68262	0.05142	0.56147	0.76664	0.82383	**0.032819**
	ADASYN	0.29623	0.46111	0.69803	0.09207	0.53783	0.74231	0.8239	0.061257
	RAMO	0.28445	0.45764	0.67894	**0.05123**	0.56829	**0.76991**	0.83843	0.058356
	MWMOTE	**0.34939**	**0.49556**	**0.80497**	0.096432	**0.68886**	0.7282	**0.89572**	0.044637
Complexity 5X5, Imbalance ratio 1:3	SMOTE	0.34862	0.52101	0.51085	0.044897	0.45977	0.62825	0.6739	**0.036993**
	ADASYN	0.34312	0.50921	0.51455	0.042833	0.45052	0.61863	0.65094	0.040935
	RAMO	0.3408	0.501	0.50611	0.042264	0.47181	0.63508	0.67106	0.045278
	MWMOTE	**0.34979**	**0.52347**	**0.52903**	**0.040807**	**0.47464**	**0.6409**	**0.68896**	0.041026
Complexity 5X5, Imbalance ratio 1:10	SMOTE	**0.14274**	**0.34423**	0.51114	0.046001	**0.14997**	**0.30492**	0.62887	0.051011
	ADASYN	0.12983	0.32446	0.51309	**0.03125**	0.12471	0.27255	0.62497	0.043673
	RAMO	0.13914	0.34136	0.50811	0.034106	0.13325	0.2752	0.63006	0.053353
	MWMOTE	0.1201	0.27677	**0.52819**	0.0341	0.14341	0.29097	**0.64329**	**0.043318**

Best results are highlighted in bold-face type.

neighbors, i.e., $k1$ and $k2$, of RAMO are chosen as 5 and 10, respectively [18]. The scaling coefficient, α, for RAMO is set to 0.3 [18]. According to [46], the number of synthetic samples to be generated is set to 200 percent of the original minority class samples. We obtain the F-measure with $\beta = 1$ and the average ROC graph of multiple runs using the

TABLE 4
Performance of SMOTE [15], ADASYN [17], RAMO [18], and MWMOTE on Artificial Domains Using k-Nearest Neighbor, and Decision Tree

		k-nearest neighbor					Decision tree				
Data set	Methods	Accuracy	Precision	Recall	F-measure	G-mean	Accuracy	Precision	Recall	F-measure	G-mean
Complexity 3X3, Imbalance ratio 1:3	SMOTE	0.78021	0.53694	0.9375	0.68141	0.82484	0.77292	0.53748	**0.825**	0.64576	0.78548
	ADASYN	0.77292	0.52757	0.94583	0.67619	0.8215	0.74896	0.50377	0.78333	0.60753	0.75389
	RAMO	0.77187	0.52983	0.93333	0.67361	0.81697	0.77396	0.55281	0.7875	0.64041	0.7734
	MWMOTE	**0.81458**	**0.58023**	**0.9625**	**0.72226**	**0.85705**	**0.88646**	**0.82354**	0.70417	**0.74831**	**0.81072**
Complexity 3X3, Imbalance ratio 1:10	SMOTE	0.92045	0.5955	0.39167	0.46922	0.61217	0.89924	0.48443	0.64583	0.54382	0.76611
	ADASYN	0.92614	0.65937	0.40417	0.49809	0.62524	0.91742	0.55534	0.6625	0.59768	0.78587
	RAMO	0.92765	0.66714	0.41667	0.5088	0.63438	0.91894	0.55049	**0.75833**	0.6337	**0.84**
	MWMOTE	**0.93712**	**0.7625**	**0.44167**	**0.55654**	**0.65608**	**0.9572**	**1**	0.52917	**0.68281**	0.72137
Complexity 5X5, Imbalance ratio 1:3	SMOTE	0.76563	0.52394	0.71389	0.60383	0.74725	0.68576	0.41885	0.62083	0.49893	0.66119
	ADASYN	0.76319	0.51994	0.72222	0.60401	0.74859	0.69826	0.4363	**0.63472**	0.51508	0.67413
	RAMO	0.77083	0.53216	0.71944	0.61127	0.75265	0.68889	0.42279	0.62778	0.50421	0.66597
	MWMOTE	**0.79306**	**0.56732**	**0.72778**	**0.6372**	**0.76971**	**0.76042**	**0.52185**	0.56389	**0.54039**	**0.68036**
Complexity 5X5, Imbalance ratio 1:10	SMOTE	0.94697	0.75485	0.62778	0.68256	0.78308	0.88977	0.43136	**0.63472**	0.5127	0.76058
	ADASYN	0.94785	0.75038	0.64444	0.69119	0.793	0.88977	0.43053	0.62639	0.50839	0.75589
	RAMO	0.94571	0.7413	0.62778	0.67812	0.78258	0.88838	0.42899	0.61389	0.5033	0.7486
	MWMOTE	**0.95518**	**0.79581**	**0.68333**	**0.73482**	**0.81905**	**0.93283**	**0.64085**	0.61806	**0.62727**	**0.77129**

Best results are highlighted in bold-face type.

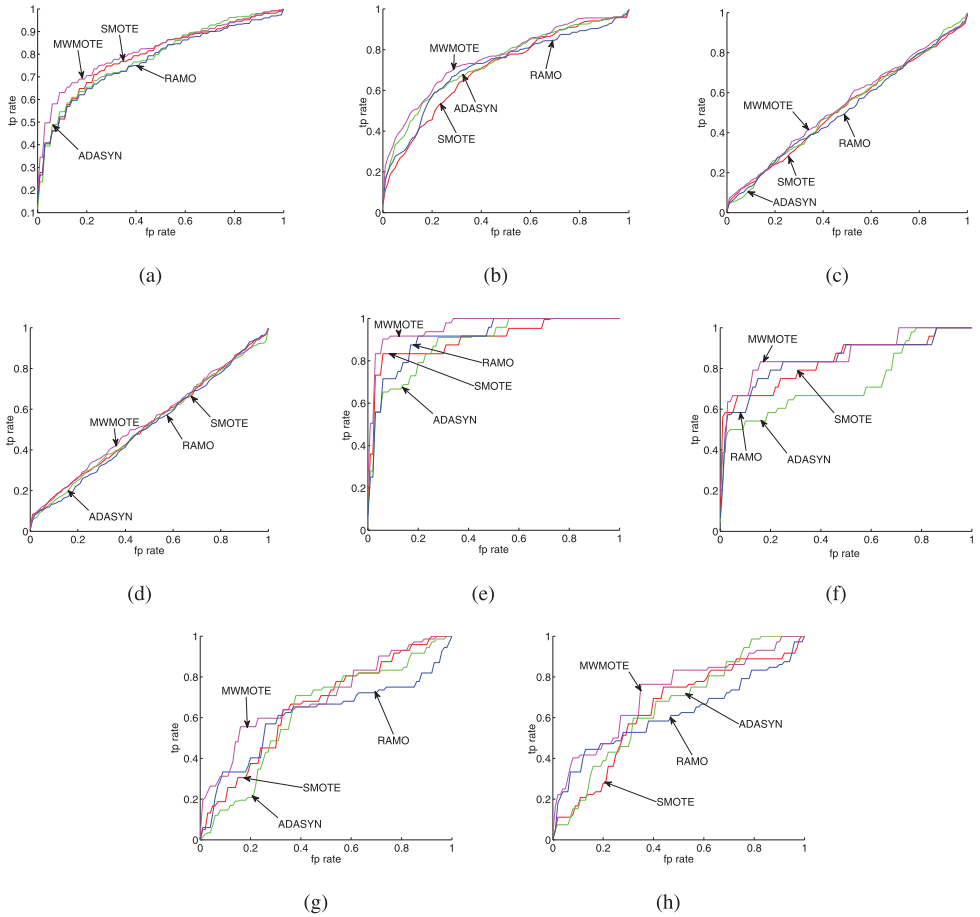


Fig. 6. Averaged ROC curves of SMOTE [15], ADASYN [17], RAMO [18], and MWMOTE for single neural network classifier ((a)-(d)) and for AdaBoost.M2 ensemble of neural network ((e)-(f)): (a) Data set with complexity level of 3×3, imbalance ratio 1:3, (b) Complexity level of 3×3, imbalance ratio 1:10, (c) Complexity level of 5×5, imbalance ratio 1:3, (d) Complexity level of 5×5, imbalance ratio 1:10, (e) Data set with complexity level of 3×3, imbalance ratio 1:3, (f) Data set with complexity level of 3×3, imbalance ratio 1:10, (g) Data set with complexity level of 5×5, imbalance ratio 1:3, (h) Data set with complexity level of 5×5, imbalance ratio 1:10.

vertical averaging approach [42]. We also obtain the average AUC by averaging the AUC values of the multiple ROC graphs.

5.3 Experiments on Artificial Problem Domain

We use an artificial two-class problem domain motivated from [12]. First, we take a unit square in a two-dimensional

TABLE 5
Description of Minority Class, Majority Class, and Characteristics of Real-World Data Sets

Data set	Minority Class	Majority Class	Features	Instances	Minority	Majority	%Minority	Imbalance Ratio
Pageblocks	Class of 'Graphic', 'Vert.line', 'Picture'	All other classes	10	5476	231	5245	4%	0.04:0.96
Abalone	Class of '18'	Class of '9'	7	731	42	689	6%	0.06:0.94
CTG	Class of '3', '4'	All other classes	21	2126	134	1992	6.3%	0.06:0.94
OCR	Class of '0'	All other classes	64	3826	376	3447	9%	0.09:0.91
Statlandsat	Class of '4'	All other classes	37	4435	415	4435	9.36%	0.09:0.91
Semeion	Class of '4'	All other classes	265	1593	161	1432	10.11%	0.11:0.89
Segment	Class of 'Grass'	All other classes	19	2310	330	1980	14%	0.14:0.86
Libra	Class of '1', '2', '3'	All other classes	90	360	72	288	20%	0.20:0.80
Yeast	Class of ME3', 'ME2', 'EXC', 'VAC', 'POX', 'ERL'	All other classes	8	1484	304	1180	21%	0.20:0.80
Robot	Class of 'slight-left-turn', 'slight-right-turn'	All other classes	24	5456	1154	4302	22%	0.21:0.79
Ecoli	Class of 'im'	All other classes	7	336	77	259	23%	0.23:0.77
Glass	Class of '5', '6', '7'	All other classes	9	214	51	163	24%	0.24:0.76
Vehicle	Class of '1'	All other classes	18	940	219	721	24%	0.23:0.77
Wine	Class of '3'	All other classes	13	178	48	130	27%	0.27:0.73
Texture	Class of '2', '3', '4'	All other classes	40	5477	1500	3977	28%	0.27:0.73
Phoneme	Class of '1'	All other classes	5	5227	1520	3707	30%	0.29:0.71
Satimage	Class of '2', '4', '5'	All other classes	36	6435	2036	4399	32%	0.32:0.68
Breast-tissue	Class of 'CAR', 'FAD'	All other classes	9	106	36	70	34%	0.34:0.66
Breast-cancer-original	Class of 'Malignant'	Class of 'Benign'	9	683	239	444	35%	0.35:0.65
Pima	Class of '1'	Class of '0'	8	768	268	500	35%	0.35:0.65

feature plane. We then equally divide the unit square to form $C \times C$ squared grids, each of size $\frac{1}{C} \times \frac{1}{C}$. Here, C is a parameter specified by the user and indicates the complexity of the problem. The larger the value of C is, the higher the complexity is.

Each of the $C \times C$ grids is assigned to a different class, i.e., either minority or majority, but the middle grid is kept blank for equal distribution of the classes. We assign the classes in such a way that no two adjacent grids are assigned to the same class. After assigning the classes, points are sampled from each grid at random and different points are taken for constructing the training and testing sets. Two different complexity levels are used in our simulation and they are $C = 3$ and $C = 5$. Figs. 5a and 5b show the problem domain structure and the class regions with two complexity levels. Besides, we use two different imbalance ratios, i.e., 1:3 (moderate) and 1:10 (high) for each complexity level.

5.3.1 Simulation Result

Tables 2, 3, and 4 summarize the results of SMOTE [15], ADASYN [17], RAMO [18], and MWMOTE on four different artificial data sets. Each result is the average of 20 independent runs and the best result is highlighted with bold-face type. The ROC graphs for some simulation are shown in Fig. 6.

For the single neural network classifier, MWMOTE performs better than SMOTE, ADASYN, and RAMO in terms of accuracy, precision, F-measure, G-mean, and AUC for all the problem sizes and imbalance ratios (Tables 2 and 3). The other methods, however, perform better than MWMOTE in terms of recall (Table 2). As described in Section 3, SMOTE, ADASYN, and RAMO erroneously generate the synthetic minority class samples that enlarge the minority class region wrongly. The outcome is that the minority class region falls inside the majority class region. This improves recall, i.e., classification accuracy on the minority class samples. At the same time, due to erroneous enlargement of the minority class region, SMOTE, ADASYN, and RAMO methods misclassify many majority class samples as minority. Hence, in spite of improved recall, SMOTE, ADASYN, and RAMO reduce F-measure, G-mean, and AUC (Table 3).

It can be seen that the recall performance for all the four oversampling methods we compare here is better with the imbalance ratio 1:3 and worse with the imbalance ratio 1:10 (Table 2). An opposite scenario is observed in case of precision. It is here worth mentioning that the dominance of the majority class samples near the decision boundary enhances with the increase of the imbalance ratio. This causes shifting the decision boundary more toward the minority class region. Because of this, the number of positive (minority) outputs from a classifier decreases with increasing imbalance ratio. The probability of an incorrect positive output thus decreases, resulting better precision and worse recall. Unlike precision, the overall performance measures such as F-measure, G-mean, and AUC do not decrease with the increase of imbalance ratio (Table 3). It indicates that a high imbalance ratio may enhance difficulty for classifiers to correctly classify the minority class samples, but it may not necessarily degrade the overall performance of classifiers. Contrary to this fact, when the complexity level increases from 3×3 to 5×5, all the four oversampling methods exhibit degraded performance in all performance metrics (Tables 2 and 3). This illustrates the fact that, whatever the imbalance ratio is, if data are highly

TABLE 6
Performance of SMOTE [15], ADASYN [17], RAMO [18], and MWMOTE on Real-World Data Sets Using Single Neural Network Classifier, and AdaBoost.M2 Ensemble of Neural Network Classifier

		Single Neural Network				AdaBoost.M2 Ensemble of Neural Network			
Data set	Methods	F-measure	G-mean	AUC	Std of AUC	F-measure	G-mean	AUC	Std of AUC
Abalone	SMOTE	**0.44167**	**0.62356**	0.87288	0.05505	0.54435	**0.77019**	0.89285	0.094805
	ADASYN	0.29806	0.44381	0.85144	**0.05029**	0.47472	0.69207	0.8899	0.085462
	RAMO	0.43962	0.58385	**0.89149**	0.08836	0.55304	0.7845	**0.90447**	**0.06685**
	MWMOTE	0.39497	0.51451	0.87745	0.093925	**0.54451**	0.74307	0.90024	0.07218
CTG	SMOTE	0.55373	0.70765	0.89328	0.038801	0.63623	0.80861	0.92909	0.083407
	ADASYN	0.5483	0.70256	0.91345	**0.034718**	0.64418	0.81255	0.92703	**0.059517**
	RAMO	0.46915	0.67389	0.90961	0.037897	0.59935	0.79819	0.9202	0.07193
	MWMOTE	**0.55378**	**0.7232**	**0.91479**	0.039941	**0.63973**	**0.81427**	**0.93843**	0.059522
Semeion	SMOTE	0.8636	0.91312	0.97867	0.010405	0.88655	0.92811	**0.98029**	**0.010369**
	ADASYN	0.84508	0.92025	0.96413	0.03218	0.88026	0.92121	0.97359	0.026031
	RAMO	0.84618	0.89846	0.97644	0.012777	**0.89997**	**0.92938**	0.97921	0.017679
	MWMOTE	**0.87694**	**0.92444**	**0.98094**	**0.0062194**	0.8915	0.93457	0.97833	0.018349
Statlandsat	SMOTE	0.48018	0.68017	0.8467	0.03165	0.53225	0.77091	0.89616	0.039478
	ADASYN	0.43299	0.69035	0.87853	0.024914	0.53304	0.77738	0.89788	0.032568
	RAMO	0.38348	0.59787	0.87734	0.027656	0.51357	**0.78263**	0.88958	0.036774
	MWMOTE	**0.50238**	**0.73219**	**0.89288**	**0.019968**	**0.54374**	0.7764	**0.90342**	**0.030274**
Bcancerorig	SMOTE	0.95949	0.97438	**0.97692**	0.0047089	0.94629	0.96142	0.97749	**0.0044664**
	ADASYN	**0.96119**	0.97454	0.97499	0.0040282	0.94307	0.95977	0.97516	0.0054823
	RAMO	0.95787	**0.97509**	0.97532	0.0051791	0.95147	0.967	0.97601	0.0063545
	MWMOTE	0.9594	0.97432	0.97629	**0.0038563**	**0.95839**	**0.96867**	**0.97969**	0.0049668
Btissue	SMOTE	0.67953	0.7156	0.84299	**0.077278**	0.64841	0.71844	0.81589	0.031100
	ADASYN	0.6972	0.73618	**0.8561**	0.078381	0.74381	0.79645	0.81829	0.03722
	RAMO	0.68376	0.71236	0.79948	0.098437	0.71468	0.78076	0.85525	0.091122
	MWMOTE	**0.73604**	**0.77634**	0.84189	0.085073	**0.76238**	**0.81106**	**0.86185**	**0.03102**
Ecoli	SMOTE	**0.76277**	0.87041	**0.94622**	**0.030888**	**0.77142**	0.85253	0.92811	0.070908
	ADASYN	0.75505	**0.87213**	0.94222	0.031349	0.75272	0.84561	0.91267	**0.053571**
	RAMO	0.7384	0.86449	0.93304	0.034294	0.76273	**0.85973**	0.92565	0.064785
	MWMOTE	0.73988	0.85931	0.93705	0.045388	0.76215	0.85731	**0.93045**	0.072141
Glass	SMOTE	0.82579	0.87774	0.93227	0.078684	0.86864	0.90903	0.94797	0.040366
	ADASYN	0.87148	0.92462	0.95363	0.053239	0.86995	**0.90926**	**0.96893**	**0.015261**
	RAMO	0.82138	0.90413	0.94235	0.085072	0.8651	0.90787	0.95717	0.044093
	MWMOTE	**0.87968**	**0.93611**	**0.95831**	**0.0472**	**0.87162**	0.90347	0.95524	0.038276

complex, then the performance of classifiers will be poor and it cannot be improved over a certain limit.

Similar to the single classifier, MWMOTE performs better than SMOTE, ADASYN, and RAMO in case of Adaboost.M2 ensemble (Tables 2 and 3). Interestingly, boosting improves the recall performance of MWMOTE and beating the other methods in three out of the four artificial problems. It is seen that boosting significantly improves the performance of all four oversampling methods. The improvement is somewhat better for MWMOTE than the others. Even for the most complex problem considered in this experiment, which has a complexity level of 5×5 and an imbalance ratio 1:10, boosting shows improvement at a greater level. This indicates that boosting can be used as a handy tool for dealing with imbalanced data sets, even with complex subconcepts. The superiority of MWMOTE over other the three methods is evident from the ROC graphs also (Fig. 6). This is true for both single neural network and ensemble of neural networks. For all the four data sets, the ROC graphs of MWMOTE are above from those of the other methods in most of the area of the graphs. This is even justified by the better AUC values of MWMOTE (Table 3).

Our MWMOTE with the k-NN and C4.5 decision tree classifiers also exhibits better performance than the other methods in terms of various metrics except recall (Table 4). As explained before, the erroneous samples generated by SMOTE, ADASYN, and RAMO cause their better recall compared to MWMOTE. This observation is still valid for the decision tree classifier but not for the k-NN classifier (Table 4), where the recall performance of MWMOTE is better than the other three methods. The k-NN is an instance-based classifier and does not construct any explicit decision boundary. Hence, the generated erroneous samples do not necessarily enlarge the minority class region. This is the main reason for getting an opposite scenario, i.e., better recall, by MWMOTE. All these results indicate that irrespective of the classifier type MWMOTE can perform better in comparison with other methods.

5.4 Experiments on Real-World Data Sets

This section presents the performance of MWMOTE and other methods on 20 real-world data sets. These sets are chosen in such a way that they have a different number of samples, features, classes, and imbalanced ratios. Some of the data sets have samples of more than two classes. We transform them to two-class problems because of our interest. Table 5 shows detailed characteristics of the data sets and the majority and minority classes found after the transformations.

TABLE 6
(Continued)

Libra	SMOTE	0.84458	0.88706	0.91719	**0.02274**	0.9554	0.96148	0.97544	0.01933
	ADASYN	0.87407	0.91104	0.89716	0.07303	0.947	0.95454	0.97214	0.029726
	RAMO	0.86324	0.88728	0.91022	0.082376	0.95469	0.96196	0.97214	0.022448
	MWMOTE	**0.91568**	**0.95059**	**0.96955**	0.02643	**0.96136**	**0.9637**	**0.97638**	**0.016343**
Phoneme	SMOTE	0.64755	0.75935	0.82397	0.024579	0.6916	0.79936	0.86299	0.012844
	ADASYN	0.62964	0.74213	0.81273	**0.022224**	0.69181	0.79981	0.85982	0.026848
	RAMO	0.62717	0.73113	**0.82494**	0.030001	0.68074	0.79053	0.86066	0.022231
	MWMOTE	**0.65563**	**0.76618**	0.82019	0.026922	**0.70557**	**0.81074**	**0.87389**	**0.0075304**
Pima	SMOTE	0.66811	0.73159	0.81443	0.038935	0.66121	0.72695	0.78641	0.044054
	ADASYN	0.68335	0.74335	0.82319	0.025653	0.66959	0.73393	0.79871	0.046455
	RAMO	0.66074	0.70707	0.82157	0.028351	0.66823	0.72662	0.7849	**0.030021**
	MWMOTE	**0.67917**	**0.74088**	**0.83026**	**0.025115**	**0.68906**	**0.75308**	**0.81326**	0.040964
Robot	SMOTE	0.63498	**0.80532**	**0.86939**	0.02672	0.71974	0.83432	0.9146	0.014192
	ADASYN	0.61757	0.79437	0.85526	0.033866	0.7434	**0.86236**	0.92285	**0.0060973**
	RAMO	0.54393	0.73985	0.80044	0.065839	0.72177	0.84722	0.91841	0.028429
	MWMOTE	**0.63647**	0.80073	0.86586	**0.023286**	**0.74775**	0.85517	**0.9275**	0.022375
Satimage	SMOTE	0.73049	0.81077	0.89643	0.019875	0.77739	0.84816	**0.92695**	0.032711
	ADASYN	0.71189	0.79614	0.88869	0.019328	**0.78348**	**0.85375**	0.92669	0.024701
	RAMO	0.64285	0.7104	0.88335	**0.015224**	0.75601	0.83745	0.92573	0.028452
	MWMOTE	**0.73935**	**0.818**	**0.90156**	0.015749	0.78103	0.85144	0.92679	**0.022892**
Vehicle	SMOTE	0.91769	0.96321	0.97428	0.020316	0.95072	0.97179	0.98157	0.0059574
	ADASYN	0.85784	0.937	0.94907	0.069568	0.95674	0.97509	0.98238	**0.0020709**
	RAMO	0.91613	**0.96702**	0.97826	0.0070189	0.94378	0.96772	0.98091	0.0066995
	MWMOTE	**0.92081**	0.9661	**0.97949**	**0.0053514**	**0.95965**	**0.97776**	**0.98316**	0.0026613
Wine	SMOTE	0.95844	0.97483	0.98182	0.0032953	0.9798	0.99215	**0.98218**	0.0040587
	ADASYN	0.9798	0.99215	0.9816	0.0036872	0.96869	0.98823	0.98151	**0.0021655**
	RAMO	0.96234	0.96928	0.97429	0.033165	0.95758	0.97767	0.98157	0.0048628
	MWMOTE	**0.98182**	**0.99215**	**0.98276**	**0.0021407**	**0.98889**	**0.99608**	0.98162	0.0024908
Yeast	SMOTE	0.68509	0.82486	0.88801	0.025871	0.71457	0.84453	0.89058	0.036351
	ADASYN	0.68609	0.83142	0.88972	**0.023166**	0.70341	0.84214	0.88823	**0.023248**
	RAMO	0.68774	**0.84089**	0.88935	0.02855	0.66724	0.82671	0.88644	0.03167
	MWMOTE	**0.68953**	0.83093	**0.89101**	0.025102	**0.73647**	**0.8573**	**0.90218**	0.023963
Texture	SMOTE	0.8696	0.93126	0.96672	**0.014346**	0.99135	0.99445	**0.98702**	0.0014264
	ADASYN	0.85257	0.92088	0.95273	0.044391	0.99002	0.99373	0.98682	0.0038179
	RAMO	0.82752	0.91116	0.94989	0.035368	**0.99335**	**0.99603**	0.98658	**0.0014037**
	MWMOTE	**0.88108**	**0.93872**	**0.96809**	0.024788	0.99068	0.99419	0.98689	0.0015757
Segment	SMOTE	0.58299	0.77089	0.90016	0.02899	**0.70495**	0.86105	0.96002	0.017385
	ADASYN	0.5386	0.74559	0.9102	**0.01451**	0.69155	0.87028	0.95937	0.014308
	RAMO	0.54843	0.76765	0.87816	0.03961	0.6815	**0.87076**	**0.96122**	**0.013325**
	MWMOTE	**0.59392**	**0.79108**	**0.94269**	0.019839	0.69052	0.85643	0.96043	0.017987
OCR	SMOTE	**0.98666**	**0.99255**	0.98639	0.0074111	0.99103	**0.99369**	**0.98966**	0.000168
	ADASYN	0.98262	0.98968	0.98776	0.0020983	0.98824	0.99268	0.98958	0.000226
	RAMO	0.97469	0.98641	0.98813	0.0013547	0.98819	0.9912	0.98961	0.000224
	MWMOTE	0.98378	0.98861	**0.98855**	**0.00061808**	**0.99148**	0.99304	0.98962	**0.000153**
Page-blocks	SMOTE	**0.97481**	**0.99032**	**0.98609**	**0.0052576**	0.98797	0.99416	0.98803	**0.00084593**
	ADASYN	0.95289	0.98545	0.98432	0.0076192	**0.99086**	**0.99464**	0.988	0.0014125
	RAMO	0.89444	0.97554	0.98367	0.0098392	0.9848	0.98982	0.98792	0.0010508
	MWMOTE	0.93337	0.97259	0.98312	0.0099119	0.98932	0.99312	**0.98834**	0.00089741

Best results are highlighted in bold-face type.

5.4.1 Simulation Result

Tables 6 and 9 show the average results of MWMOTE and three other methods, which are found for 10-fold cross validation on each data set. The best results are highlighted in bold-face type. Some of the representative ROC graphs of different simulations are presented in Fig. 7.

Similar to the artificial problems, MWMOTE exhibits better performance than SMOTE, ADASYN, and RAMO for most of the real-world problems we compare here (Tables 6 and 9). It is not surprise that, comparing to single neural network, ensemble of neural networks improves the performance of all the four methods significantly. Visual inspection of Fig. 7 reveals that the ROC graphs of MWMOTE is above (indicating better performance) compared to those of other three methods in most of the portion of the ROC space.

Following the suggestions in [49], we apply the Wilcoxon signed-rank test [44] on the average AUC values. The test results are summarized in Tables 7 and 8 for single neural network and ensemble of neural networks, respectively. It can be observed that with single neural network or ensemble of neural networks, the performance of MWMOTE is statistically better than that

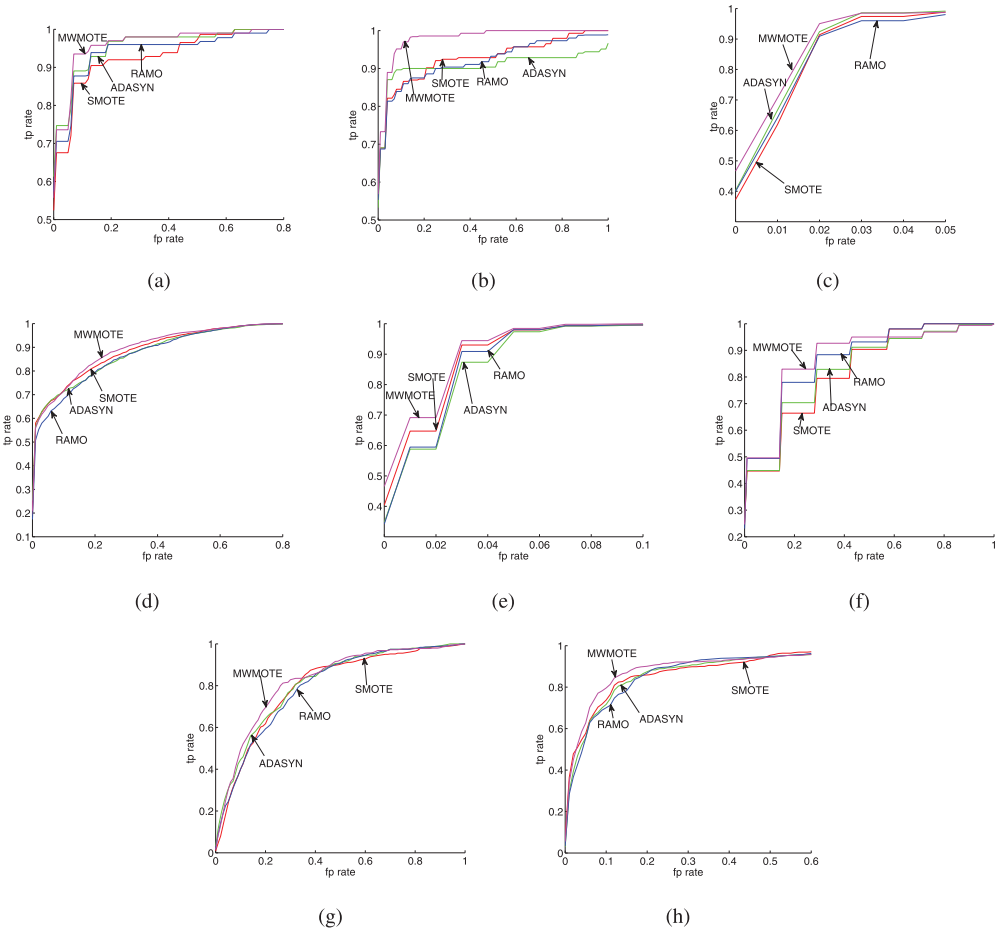


Fig. 7. Averaged ROC curves of SMOTE [15], ADASYN [17], RAMO [18], and MWMOTE for Neural Network classifier ((a)-(d)) and for ensemble of Neural Network classifiers ((e)-(h)): (a) Glass data set, (b) Libra data set, (c) Vehicle data set, (d) Satimage data set, (e) Breast Cancer Original data set, (f) Breast Tissue data set, (g) Pima data set, (h) Yeast data set.

with SMOTE, ADASYN, and RAMO. For example, in case of MWMOTE versus SMOTE with single neural network, SMOTE is better (negative difference) than MWMOTE for six data sets, while MWMOTE is better (positive difference) than SMOTE for 14 data sets (Table 7). We sum all the positive ranks under the *rank* column to find $R+$, which is 166 and similarly, all the negative ranks to find $R-$, which is 44. The minimum of $R+$ and $R-$ is the T value and it is 44. Since there are 20 data sets, the T value at a significance level of 0.05 should be less than or equal to 52 (the critical value) to reject the null hypothesis [44]. That is, MWMOTE is better than SMOTE.

5.5 Comparison with Cluster-Based Method

In the preceding sections, we compared our MWMOTE with some popular oversampling methods. However, none of those methods use partitioning/clustering in their data generation scheme. In this section, we compare MWMOTE with two cluster-based oversampling methods, namely clustering using local sampling (COGOS) [30] and local sampling (LS) [31]. The main differences of MWMOTE with these methods are as follows: First, both COGOS and LS divide a given data set into a predefined number of partitions, whereas MWMOTE adaptively finds this number. Second, COGOS applies partitioning on the majority class samples but oversampling on the minority class samples. Our MWMOTE, however, applies both partitioning and oversampling only on the minority class samples. Third, neither COGOS nor LS uses any weight for oversampling, whereas MWMOTE uses weights to discover the most important minority class samples for oversampling.

For brevity, we here use a subset of data sets and single and ensemble of neural network classifiers for simulations. We also employ here the back-propagation learning algorithm for training classifiers. All parameters for COGOS and LS were set according to [30], [31]. Other parameters such as parameters for neural network, ensemble of network, and MWMOTE were set same to our previous simulations. Table 10 shows the average results of MWMOTE, COGOS and LS for 10-fold cross validation on each data set. It is found that MWMOTE performs better than COGOS and LS for most of the data sets in terms of various performance metrics. The Wilcoxon signed-rank test confirms MWMOTE's superiority over COGOS and LS (Table 11).

5.6 Choosing Appropriate Values for MWMOTE Parameters

MWMOTE involves six parameters: $k1$, $k2$, $k3$, Cp, $C_f(th)$, and $CMAX$. As mentioned earlier, we chose these parameters values after some initial runs and they were not meant to be optimal. In this section, we provide a brief

TABLE 7
Simulation 1 on Real-World Data Sets Using Single Neural Network: Significance Test of Averaged AUC between MWMOTE versus SMOTE [15], ADASYN [17], and RAMO [18]

	MWMOTE vs. SMOTE				MWMOTE vs. ADASYN				MWMOTE vs. RAMO			
Data set	MWMOTE	SMOTE	Difference	Rank	MWMOTE	ADASYN	Difference	Rank	MWMOTE	RAMO	Difference	Rank
Abalone	0.87745	0.87288	0.00457	+11	0.87745	0.85144	0.02601	+17	0.87745	0.89149	-0.01404	-11
CTG	0.91479	0.89328	0.02151	+16	0.91479	0.90961	0.00518	+9	0.91479	0.91345	0.00134	+5
Bcancerorig	0.97629	0.97692	-0.00063	-1	0.97629	0.97499	0.0013	+5	0.97629	0.97532	0.00097	+3
Btissue	0.84189	0.84299	-0.0011	-3	0.84189	0.8561	-0.01421	-14	0.84189	0.79948	0.04241	+17
Ecoli	0.93705	0.94622	-0.00917	-14	0.93705	0.94222	-0.00517	-8	0.93705	0.93304	0.00401	+7
Glass	0.95831	0.93227	0.02604	+17	0.95831	0.95363	0.00468	+7	0.95831	0.94235	0.01596	+13
Semeion	0.91479	0.89328	0.00227	+6	0.98094	0.97644	0.0045	+6	0.98094	0.96413	0.01681	+14
Libra	0.96955	0.91719	0.05236	+20	0.96955	0.89716	0.07239	+20	0.96955	0.91022	0.05933	+18
Phoneme	0.82019	0.82397	-0.00378	-10	0.82019	0.81273	0.00746	+11	0.82019	0.82494	-0.00475	-8
Pima	0.83026	0.81443	0.01583	+15	0.83026	0.82319	0.00707	+10	0.83026	0.82157	0.00869	+10
Robot	0.86586	0.86939	-0.00353	-9	0.86586	0.85526	0.0106	+12	0.86586	0.80044	0.06542	+20
Satimage	0.90156	0.89643	0.00513	+12	0.90156	0.88869	0.01287	+13	0.90156	0.88335	0.01821	+16
Vehicle	0.97949	0.97428	0.00521	+13	0.97949	0.94907	0.03042	+18	0.97949	0.97826	0.00123	+4
Wine	0.98276	0.98182	0.00094	+2	0.98276	0.9816	0.00116	+2	0.98276	0.97429	0.00847	+9
Yeast	0.89101	0.88801	0.003	+8	0.89101	0.88972	0.00129	+4	0.89101	0.88935	0.00166	+6
Statlandsat	0.89288	0.8467	0.04618	+19	0.89288	0.87734	0.01554	+16	0.89288	0.87853	0.01435	+12
texture	0.96809	0.96672	0.00137	+4	0.96809	0.95273	0.01536	+15	0.96809	0.94989	0.0182	+15
Segment	0.94269	0.90016	0.04253	+18	0.94269	0.9102	0.03249	+19	0.94269	0.87816	0.06453	+19
OCR	0.98855	0.98639	0.00216	+5	0.98855	0.98776	0.00079	+1	0.98855	0.98813	0.00042	+1
Pageblocks	0.98312	0.98609	-0.00297	-7	0.98312	0.98432	-0.0012	-3	0.98312	0.98367	-0.00055	-2
	$T = min\{166, 44\} = 44$				$T = min\{185, 25\} = 25$				$T = min\{189, 21\} = 21$			

TABLE 8
Simulation 1 on Real-World Data Sets Using Adaboost.M2 Ensemble of Neural Network: Significance Test of Averaged AUC of MWMOTE versus SMOTE [15], ADASYN [17], and RAMO [18]

	MWMOTE vs. SMOTE				MWMOTE vs. ADASYN				MWMOTE vs. RAMO			
Data set	MWMOTE	SMOTE	Difference	Rank	MWMOTE	ADASYN	Difference	Rank	MWMOTE	RAMO	Difference	Rank
Abalone	0.90024	0.89285	0.00739	+14	0.90024	0.8899	0.01034	+12	0.90024	0.90447	-0.00423	-10
CTG	0.93843	0.92909	0.00934	+15	0.93843	0.9202	0.01823	+19	0.93843	0.92703	0.0114	+17
Bcancerorig	0.97969	0.97749	0.0022	+10	0.97969	0.97516	0.00453	+10	0.97969	0.97601	0.00368	+9
Btissue	0.86185	0.81589	0.04596	+20	0.86185	0.81829	0.04356	+20	0.86185	0.85525	0.0066	+15
Ecoli	0.93045	0.92811	0.00234	+11	0.93045	0.91267	0.01778	+18	0.93045	0.92565	0.0048	+13
Glass	0.95524	0.94797	0.00727	+13	0.95524	0.96893	-0.01369	-13	0.95524	0.95717	-0.00193	-7
Semeion	0.97833	0.98029	-0.00196	-9	0.97833	0.97921	-0.00088	-7	0.97833	0.97359	0.00474	+12
Libra	0.97638	0.97544	0.00094	+7	0.97638	0.97214	0.00424	+9	0.97638	0.97214	0.00424	+11
Phoneme	0.87389	0.86299	0.0109	+16	0.87389	0.85982	0.01407	+16	0.87389	0.86066	0.01323	+18
Pima	0.81326	0.78641	0.02685	+19	0.81326	0.79871	0.01455	+17	0.81326	0.7849	0.02836	+20
Robot	0.9275	0.9146	0.0129	+18	0.9275	0.92285	0.00465	+11	0.9275	0.91841	0.00909	+16
Satimage	0.92679	0.92695	-0.00016	-3	0.92679	0.92669	0.0001	+3	0.92679	0.92573	0.00106	+6
Vehicle	0.98316	0.98157	0.00159	+8	0.98316	0.98238	0.00078	+6	0.98316	0.98091	0.00225	+8
Wine	0.98162	0.98218	-0.00056	-6	0.98162	0.98151	0.00011	+4	0.98162	0.98157	0.00005	+2
Yeast	0.90218	0.89058	0.0116	+17	0.90218	0.88823	0.01395	+15	0.90218	0.88644	0.01574	+19
Statlandsat	0.90342	0.89616	0.00726	+12	0.90342	0.88958	0.01384	+14	0.90342	0.89788	0.00554	+14
texture	0.98689	0.98702	-0.00013	-2	0.98689	0.98682	0.00007	+2	0.98689	0.98658	0.00031	+3
Segment	0.96043	0.96002	0.00041	+5	0.96043	0.95937	0.00106	+8	0.96043	0.96122	-0.00079	-5
OCR	0.98962	0.98966	-0.000040	-1	0.98962	0.98958	0.00004	+1	0.98962	0.98961	0.00001	+1
Pageblocks	0.98834	0.98803	0.00031	+4	0.98834	0.988	0.00034	+5	0.98834	0.98792	0.00042	+4
	$T = min\{189, 21\} = 21$				$T = min\{190, 20\} = 20$				$T = min\{188, 22\} = 22$			

explanation on how to choose the appropriate values of the parameters. We also provide some simulation results in Table 12 to supplement our explanations:

1. $k1$: Our MWMOTE uses $k1$ for the finding noisy minority class samples. If all the $k1$ neighbors are the majority class samples, MWMOTE then assumes the sample in question as noise and thus removes it from the data set. We use $k1 = 5$ in our previous simulations. Other reasonable values could be 3, 4, 10, and so on. However, a very large value such as 20 is not realistic, because the noisy minority sample will not have all its 20 neighbors as the majority class samples. Similarly, a small $k1$, for example, 3 might be too small and can erroneously remove the valid minority class sample. So, any value between 5 and 10 would be reasonable for $k1$.

TABLE 9
Performance of SMOTE [15], ADASYN [17], RAMO [18], and MWMOTE on Real-World Data Sets Using k-NN, and Decision Tree as Base Classifiers

		k-Nearest Neighbor			Decision tree		
Data set	Methods	Accuracy	F-measure	G-mean	Accuracy	F-measure	G-mean
Abalone	SMOTE	**0.94235**	**0.50188**	0.64738	0.90159	0.30677	0.56252
	ADASYN	0.93017	0.44095	**0.67309**	0.90966	**0.36025**	**0.61153**
	RAMO	0.94228	0.47735	0.66253	0.89748	0.28168	0.53446
	MWMOTE	0.94078	0.44782	0.60836	**0.91801**	0.35762	0.59794
Bcancerorig	SMOTE	0.96927	0.95833	0.97413	**0.94891**	**0.92868**	**0.94776**
	ADASYN	0.96927	0.95833	0.97413	0.94588	0.92125	0.93715
	RAMO	0.96635	0.95479	0.97276	0.94296	0.91842	0.93606
	MWMOTE	**0.97511**	**0.96586**	**0.97874**	0.9488	0.92662	0.94449
Btissue	SMOTE	**0.77455**	0.72328	**0.77943**	0.76545	0.6659	0.7367
	ADASYN	0.76545	0.71402	0.7678	0.78455	**0.72582**	0.78109
	RAMO	0.76545	**0.72987**	0.76471	0.75545	0.68063	0.74722
	MWMOTE	0.75545	0.70603	0.75379	**0.79545**	0.71611	**0.78195**
Ecoli	SMOTE	0.88366	0.78133	0.88149	0.86648	0.7167	0.81219
	ADASYN	0.89543	0.8004	**0.89475**	0.86905	0.72472	0.82128
	RAMO	0.87521	0.77286	0.87766	0.8777	**0.74646**	**0.83959**
	MWMOTE	**0.90784**	**0.81196**	0.89302	**0.88118**	0.7452	0.83513
Glass	SMOTE	0.94349	0.88817	0.93314	0.8926	0.7782	0.85501
	ADASYN	0.93873	0.87908	0.92996	0.90152	0.79277	0.85936
	RAMO	0.93873	0.88211	0.93655	0.90669	0.81505	0.87458
	MWMOTE	**0.95777**	**0.91928**	**0.95784**	**0.91991**	**0.83313**	**0.88786**
Libra	SMOTE	0.99437	0.98462	0.98516	0.86955	0.63475	0.72467
	ADASYN	0.99437	0.98462	0.98516	0.90862	**0.76718**	**0.85059**
	RAMO	0.99159	0.97795	0.98342	**0.91108**	0.76357	0.83058
	MWMOTE	**0.99437**	**0.98462**	**0.98516**	0.90244	0.73482	0.81296
Phoneme	SMOTE	0.86704	0.79492	0.87203	0.87717	0.79721	0.86259
	ADASYN	0.86838	**0.79762**	**0.87503**	0.8766	0.7981	**0.86504**
	RAMO	0.85116	0.78062	0.86715	0.87067	0.79198	0.86308
	MWMOTE	**0.86876**	0.79717	0.87365	**0.88062**	**0.80172**	0.86446
Pima	SMOTE	**0.68628**	**0.63697**	**0.70424**	0.68886	0.58212	0.67011
	ADASYN	0.66936	0.61664	0.68534	0.67329	0.56164	0.65259
	RAMO	0.66668	0.62805	0.68986	0.67059	0.57864	0.66293
	MWMOTE	0.67454	0.62194	0.69101	**0.69405**	**0.60011**	**0.68405**
Robot	SMOTE	**0.91788**	**0.8261**	**0.91832**	0.98717	0.96993	0.98355
	ADASYN	0.91642	0.82511	0.92061	0.99175	0.9806	0.98934
	RAMO	0.90854	0.81368	0.92014	0.9912	0.97944	0.98996
	MWMOTE	0.91257	0.81483	0.9105	**0.99395**	**0.98562**	**0.98944**
Satimage	SMOTE	0.89122	0.84735	0.90685	**0.8864**	**0.82451**	**0.87424**
	ADASYN	0.88827	0.84453	0.90559	0.88392	0.82199	0.87333
	RAMO	0.86076	0.81661	0.88815	0.86931	0.80224	0.86054
	MWMOTE	**0.89215**	**0.84934**	**0.90879**	0.88066	0.81696	0.86942
Vehicle	SMOTE	0.92761	0.86446	**0.94411**	0.9489	0.89059	0.92885
	ADASYN	0.92334	0.85694	0.93956	0.95099	0.89603	0.93376
	RAMO	0.92122	0.85435	0.93815	0.94465	0.88345	0.9292
	MWMOTE	**0.92972**	**0.86721**	0.94384	**0.9521**	**0.89864**	**0.93458**
Wine	SMOTE	0.96634	0.94495	0.97629	0.97745	0.95758	0.97104
	ADASYN	0.96078	0.93586	0.97237	0.98301	0.96667	0.97496
	RAMO	0.95523	0.92677	0.96845	0.98856	0.97778	**0.98552**
	MWMOTE	**0.9719**	**0.95404**	**0.98022**	**0.98889**	**0.97778**	0.97889
Yeast	SMOTE	0.84166	0.65902	0.80426	0.84906	0.65709	0.7918
	ADASYN	0.82212	0.63447	0.79413	0.84708	**0.6662**	**0.80473**
	RAMO	0.80729	0.62475	0.79612	0.83898	0.63741	0.77879
	MWMOTE	**0.85176**	**0.67901**	**0.81796**	**0.85445**	0.65944	0.78618
texture	SMOTE	0.99361	0.98842	0.99394	0.97407	0.9525	0.96599
	ADASYN	0.99324	0.98775	0.99327	0.97572	0.95597	0.97117
	RAMO	0.99124	0.98423	0.99292	0.97645	0.95729	0.97231
	MWMOTE	**0.99434**	**0.98974**	**0.99465**	**0.979**	**0.96178**	**0.97446**
Cardiotocography	SMOTE	0.95814	0.70069	0.86335	0.9539	0.64531	0.79939
	ADASYN	0.95768	0.69745	0.86702	0.95248	0.65031	0.82259
	RAMO	0.9558	0.69695	0.88154	0.952	0.62357	0.78567
	MWMOTE	**0.96097**	**0.7249**	**0.8843**	**0.95861**	**0.68257**	**0.82625**
semeion	SMOTE	**0.94222**	**0.77035**	**0.95328**	0.92906	0.63946	0.78167
	ADASYN	0.93972	0.76155	0.9491	0.92904	0.64296	0.79027
	RAMO	0.93343	0.74355	0.94555	0.93411	0.64882	0.77013
	MWMOTE	0.92843	0.73458	0.9509	**0.93474**	**0.67492**	**0.8115**
statlandsat	SMOTE	**0.90891**	0.63423	0.8772	**0.90711**	**0.54412**	**0.74499**
	ADASYN	0.90711	**0.63964**	**0.89188**	0.89922	0.51231	0.72548
	RAMO	0.89358	0.61007	0.88894	0.90642	0.53911	0.73768
	MWMOTE	0.90147	0.62115	0.88242	0.90326	0.52647	0.73217
OCR	SMOTE	0.99895	0.99467	0.99703	0.99084	0.9532	0.97279
	ADASYN	**0.99921**	**0.99596**	**0.99717**	0.99058	0.95232	0.97095
	RAMO	**0.99921**	**0.99596**	**0.99717**	0.99111	0.95556	0.97815
	MWMOTE	**0.99921**	**0.99596**	**0.99717**	**0.99136**	**0.95541**	**0.97423**
PageBlocks	SMOTE	0.99351	0.9776	0.99237	0.99437	0.98018	0.98649
	ADASYN	0.99351	0.97774	0.99366	0.99437	0.98036	0.98906
	RAMO	0.99307	0.97629	0.99342	**0.99654**	**0.98778**	**0.99159**
	MWMOTE	**0.99437**	**0.98068**	**0.99417**	0.99567	0.9848	0.98981
Segmentation	SMOTE	0.96893	0.65946	0.83189	0.97552	0.71439	0.85076
	ADASYN	0.96802	0.65848	0.84225	**0.97808**	**0.74536**	0.86583
	RAMO	0.9629	0.6248	0.83041	0.97332	0.70979	**0.87259**
	MWMOTE	**0.96985**	**0.66394**	**0.84179**	0.97643	0.72597	0.85246

Best results are highlighted in bold-face type.

2. $k2$: This parameter is employed for finding the borderline majority class samples, which are used for finding the hard-to-learn minority class samples and assigning weights to them. In our previous simulation, we use $k2 = 3$. We observe that the value of $k2$ should be chosen in such way that we are able to find a sufficient number of majority class samples. In fact, any reasonable value, such as 5, 10, and 20, works fine. However, increasing the value of $k2$ will select many borderline majority class samples, some of which may be unnecessary. We provide simulation results to show how the performance of MWMOTE varies with different $k2$ (Table 12).
3. $k3$: This parameter assists our MWMOTE to select the minority class samples. A large $k3$ indicates our intention to use many minority class samples for generating the synthetic samples. Our observation is that $k3$ should be large enough so that a sufficient number of the minority class samples can be used in generating the synthetic samples. A reverse situation will appear with a small $k3$ for which all the generated samples will be located very close to the decision boundary. This will make the distribution of generated samples skewed, resulting an aggressive overbias. This suggests us a sufficiently large $k3$. We use $k3 = |S_{min}|/2$, i.e., approximately 50 percent of the original minority class samples in generating synthetic samples. From Table 12, it is seen that when $k3$ becomes too small (less than $|S_{min}|/4$), the performance of MWMOTE degrades from the best one (when $k3$ is around $|S_{min}|/2$ or $|S_{min}|$). Obviously, even if $k3$ is 100 percent, it does not mean that MWMOTE will generate the synthetic samples from all the minority class samples. This is because the weight of the minority class samples located far from the decision boundary will be very low, thus their selection probability for generating the synthetic samples will be very small.
4. C_p: This parameter controls the number of clusters considered in MWMOTE. A large C_p reduces the number of clusters but increases their size, while the small C_p induces the opposite scenario. We used $C_p = 3$ in our previous experiments. The results with different C_ps indicate that MWMOTE usually performs better when C_p is less than 10 (Table 12). A large C_p (e.g., 25 or 50) produces the large sized clusters. Thus, it increases the chance for generating the synthetic samples far from the decision boundary, leading to poor performance.
5. $C_f(th)$ and $CMAX$: Our MWMOTE uses these two parameters for smoothing and rescaling the values of different scaling factors used in weighting the minority class samples. Our observation is that setting $C_f(th)$ to 5 and $CMAX$ to 2 are good. However, any other reasonable values can be used and we refrain from presenting such results for brevity.

6 Conclusion

Many oversampling methods exist in the literature for imbalanced learning problems. These methods generate the synthetic minority class samples from the hard-to-learn minority class samples with an aim to balance the distribution between the samples of the majority and minority classes. However, in many conditions, existing methods are not able to select the hard-to-learn minority

TABLE 10
Performance of LS [30], COGOS [31], and MWMOTE on Real-World Data Sets Using Single Neural Network and Adaboost.M2 Ensemble of Neural Networks

		Single Neural Network				Adaboost.M2 ensemble of Neural Network			
Data set	Methods	F-measure	G-mean	AUC	Std of AUC	F-measure	G-mean	AUC	Std of AUC
Abalone	LS	**0.42054**	0.53786	0.86068	0.091106	0.53178	0.65402	0.86526	0.098365
	COGOS	0.35774	**0.77174**	0.8722	**0.0868**	0.48445	**0.82078**	0.89721	0.083659
	MWMOTE	0.39497	0.51451	**0.87745**	0.093925	**0.54451**	0.74307	**0.90024**	**0.02218**
CTG	LS	0.27892	0.45499	0.76899	0.04660	0.54491	0.66858	0.93105	**0.033878**
	COGOS	0.39147	**0.83849**	0.90243	0.051307	0.62276	**0.87594**	0.93806	0.038651
	MWMOTE	**0.55378**	0.7232	**0.91479**	**0.039941**	**0.63973**	0.81427	**0.93843**	0.059522
Semeion	LS	0.73819	0.90291	0.95734	0.023971	0.83917	0.92115	0.97677	**0.010615**
	COGOS	0.87017	**0.92691**	0.96054	0.028298	**0.89612**	0.92626	0.97816	0.013687
	MWMOTE	**0.87694**	0.92444	**0.98094**	**0.0062194**	0.8915	**0.93457**	**0.97833**	0.018349
Statlandsat	LS	0.39042	0.61672	0.84295	0.0297	**0.55063**	0.72204	**0.9146**	**0.017196**
	COGOS	0.40648	**0.79822**	0.8801	0.052008	0.50746	**0.8423**	0.90321	0.021864
	MWMOTE	**0.50238**	0.73219	**0.89288**	**0.019968**	0.54374	0.7764	0.90342	0.030274
Ecoli	LS	**0.76892**	**0.88785**	0.93743	**0.026215**	0.61996	0.71419	0.8816	0.065745
	COGOS	0.75376	0.87045	**0.94791**	0.027006	0.75374	0.85279	0.91972	**0.048999**
	MWMOTE	0.73988	0.85931	0.93705	0.045388	**0.76215**	**0.85731**	**0.93045**	0.072141
Glass	LS	0.81278	0.87009	0.94591	0.054413	0.83819	0.86701	0.91828	**0.02975**
	COGOS	0.8501	0.91366	0.95485	**0.03738**	0.86485	**0.90758**	**0.9556**	0.037927
	MWMOTE	**0.87968**	**0.93611**	**0.95831**	0.0472	**0.87162**	0.90347	0.95524	0.038276
Libra	LS	**0.92509**	0.94778	0.96321	0.038279	0.81244	0.84155	0.93836	0.06777
	COGOS	0.92321	0.94767	0.96703	0.047472	0.957	0.96322	**0.97652**	**0.01332**
	MWMOTE	0.91568	**0.95059**	**0.96955**	**0.02643**	**0.96136**	**0.9637**	0.97638	0.016343
Robot	LS	0.56496	0.69171	0.84586	0.031835	0.71114	0.80146	0.91858	0.013913
	COGOS	0.62103	**0.80209**	0.85207	0.047104	0.73647	**0.8657**	0.92618	**0.010886**
	MWMOTE	**0.63647**	0.80073	**0.86586**	**0.023286**	**0.74775**	0.85517	**0.9275**	0.022375
Vehicle	LS	**0.92328**	0.9572	0.97866	0.0088794	0.95723	0.97553	0.98232	0.003221
	COGOS	0.91153	0.95944	0.97691	0.011563	0.9478	0.96622	0.98216	**0.0025193**
	MWMOTE	0.92081	**0.9661**	**0.97949**	**0.0053514**	**0.95965**	**0.97776**	**0.98316**	0.0026613
Yeast	LS	0.65628	0.76674	0.8799	0.034702	0.6641	0.77556	0.8761	0.034158
	COGOS	0.66331	0.81682	0.87048	0.060892	0.70225	0.84721	0.89662	0.031869
	MWMOTE	**0.68953**	**0.83093**	**0.89101**	**0.025102**	**0.73647**	0.8573	**0.90218**	**0.023963**
Texture	LS	**0.93275**	**0.95393**	**0.98003**	**0.0044825**	**0.99567**	**0.99733**	0.98783	0.00035457
	COGOS	0.85042	0.91611	0.95188	0.040686	0.99304	0.99632	**0.98747**	**0.00032394**
	MWMOTE	0.88108	0.93872	0.96809	0.024788	0.99068	0.99419	0.98689	0.0015757
Page-blocks	LS	0.89387	**0.97382**	0.97752	0.01679	0.90122	0.97761	0.98622	0.0037745
	COGOS	0.92799	0.97372	0.9822	0.013166	**0.9924**	**0.99616**	0.98805	0.00093899
	MWMOTE	**0.93337**	0.97259	**0.98312**	**0.0099119**	0.98932	0.99312	**0.98834**	**0.00089741**

Best results are highlighted in bold-face type.

class samples effectively, assign relative weights to the selected samples appropriately, and generate synthetic samples correctly (Section 2). Based on these observations, we propose a new method, i.e., MWMOTE for imbalance learning problems. The method not only selects the hard-to-learn minority class samples effectively but also assigns them weights appropriately. Furthermore, it is able to generate correct synthetic samples.

Our MWMOTE uses the majority class samples near the decision boundary to effectively select the hard-to-learn minority class samples. It then adaptively assigns the weights to the selected samples according to their importance in learning. The samples closer to decision boundary are given higher weights than others. Similarly, the samples of the small-sized clusters are given higher weights for reducing within-class imbalance. The synthetic sample generation technique of MWMOTE uses a clustering approach. The aim of using clustering is to ensure that the generated samples must reside inside the minority class area for avoiding any wrong or noisy synthetic sample generation.

TABLE 11
Significance Test of Averaged AUC between MWMOTE, versus LS [31] and COGOS [30]

	Single Neural Network		Adaboost.M2 ensemble of Neural Network	
	MWMOTE vs. LS	MWMOTE vs. COGOS	MWMOTE vs. LS	MWMOTE vs. COGOS
$R+$	71	72	72	65
$R-$	7	6	6	13
T	7	6	6	13

Obtained T values are less than the critical value (14), which proves that MWMOTE is statistically better than the other two methods.

Extensive experiments have been carried out to evaluate how well MWMOTE performs on different data sets in comparison with other methods. We use neural networks, ensemble of neural networks, k-nearest neighbor, and decision tree as classifiers. We evaluate the performance of MWMOTE and some other methods on the artificial and well real-world data sets. In almost all cases, MWMOTE outperformed the others in terms of several performance measures such as accuracy, precision, F-measure, G-mean,

TABLE 12
Simulation on MWMOTE Parameters: Averaged AUC of MWMOTE for Varying Parameters $k2$, $k3$, and C_p

	Averaged AUC for different $k2$ values		Averaged AUC for different $k3$ values		Averaged AUC for different C_p values	
Data set	k2	AUC	k3	AUC	Cp	AUC
Btissue	1	0.85754	$\lvert S_{min}\rvert$	0.85754	1	0.84846
	3	0.86485	$\lvert S_{min}\rvert/2$	**0.86688**	3	0.85138
	5	0.85791	$\lvert S_{min}\rvert/3$	0.86087	10	**0.85664**
	10	**0.87042**	$\lvert S_{min}\rvert/4$	0.86162	25	0.84607
	20	0.86659	$\lvert S_{min}\rvert/10$	0.85827	50	0.84975
Libra	1	0.94793	$\lvert S_{min}\rvert$	**0.9437**	1	0.94508
	3	**0.95355**	$\lvert S_{min}\rvert/2$	0.94135	3	**0.94892**
	5	0.94418	$\lvert S_{min}\rvert/3$	0.92376	10	0.94022
	10	0.9523	$\lvert S_{min}\rvert/4$	0.9326	25	0.94846
	20	0.95167	$\lvert S_{min}\rvert/10$	0.91793	50	0.93879
Yeast	1	0.91008	$\lvert S_{min}\rvert$	**0.9108**	1	0.90892
	3	**0.91261**	$\lvert S_{min}\rvert/2$	0.91067	3	0.90098
	5	0.91038	$\lvert S_{min}\rvert/3$	0.90879	10	**0.90643**
	10	0.90658	$\lvert S_{min}\rvert/4$	0.90477	25	0.89855
	20	0.91133	$\lvert S_{min}\rvert/10$	0.90555	50	0.89782

Classifier used is neural network. Results are shown for three data sets.

and AUC (Tables 2, 3, and 4 and Tables 6, 7, 8, 9, 10, and 11). However, our MWMOTE was beaten by others in terms of recall. This is due to the fact that other methods erroneously increase the region of the minority class, resulting in better recall performance.

Several other research issues of MWMOTE are left to be considered. First, the application of MWMOTE in multiclass (more than two class) problems. The objective will be how efficiently MWMOTE can be applied to multiclass problems. Second, the consideration of nominal features with MWMOTE. In this paper, we use data sets with continuous features only. So, MWMOTE can be generalized to handle features of any type. Third, it can be investigated whether some other clustering mechanism can give better performance for MWMOTE. Since, clustering is the key step of the data generation of MWMOTE, finding a better clustering scheme may give better performance. Fourth, MWMOTE can be integrated with some other undersampling and boundary estimation methods to investigate whether they together can give better results than single MWMOTE oversampling procedure. Fifth, MWMOTE involves a number of parameters, which can be optimized for the best performance depending on the specific problem at hand.

ACKNOWLEDGMENTS

The first and second authors would like to thank the Department of Computer Science and Engineering of Bangladesh University of Engineering and Technology for its generous support. The third author was supported by the iSense grant from EU FP7 (Grant No. 270428). The authors would like to thank the associate editor and anonymous reviewers for their constructive comments.

REFERENCES

[1] P.M. Murphy and D.W. Aha, "UCI Repository of Machine Learning Databases," Dept. of Information and Computer Science, Univ. of California, Irvine, CA, 1994.

[2] D. Lewis and J. Catlett, "Heterogeneous Uncertainty Sampling for Supervised Learning," *Proc. Int'l Conf. Machine Learning,* pp. 148-156, 1994.

[3] T.E. Fawcett and F. Provost, "Adaptive Fraud Detection," *Data Mining and Knowledge Discovery,* vol. 3, no. 1, pp. 291-316, 1997.

[4] 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.

[5] C.X. Ling and C. Li, "Data Mining for Direct Marketing: Problems and Solutions," *Proc. Int'l Conf. Knowledge Discovery and Data Mining,* pp. 73-79, 1998.

[6] N. Japkowicz, C. Myers, and M. Gluck, "A Novelty Detection Approach to Classification," *Proc. 14th Joint Conf. Artificial Intelligence,* pp. 518-523, 1995.

[7] S. Clearwater and E. Stern, "A Rule-Learning Program in High Energy Physics Event Classification," *Computer Physics Comm.,* vol. 67, no. 2, pp. 159-182, 1991.

[8] G.M. Weiss, "Mining with Rarity: A Unifying Framework," *ACM SIGKDD Explorations Newsletter,* vol. 6, no. 1, pp. 7-19, 2004.

[9] R.C. Holte, L. Acker, and B.W. Porter, "Concept Learning and the Problem of Small Disjuncts," *Proc. Int'l Joint Conf. Artificial Intelligence,* pp. 813-818, 1989.

[10] J.R. Quinlan, "Induction of Decision Trees," *Machine Learning,* vol. 1, no. 1, pp. 81-106, 1986.

[11] F. Provost, "Machine Learning from Imbalanced Data Sets 101," *Proc. Learning from Imbalanced Data Sets: Papers from the Am. Assoc. Artificial Intelligence Workshop,* 2000.

[12] N. Japkowicz and S. Stephen, "The Class Imbalance Problem: A Systematic Study," *Intelligent Data Analysis,* vol. 6, no. 5, pp. 429-449, 2002.

[13] R.C. Prati, G.E.A.P.A. Batista, and M.C. Monard, "Class Imbalances versus Class Overlapping: An Analysis of a Learning System Behavior," *Proc. Mexican Int'l Conf. Artificial Intelligence,* pp. 312-321, 2004.

[14] T. Jo and N. Japkowicz, "Class Imbalances versus Small Disjuncts," *ACM SIGKDD Exploration Newsletter,* vol. 6, no. 1, pp. 40-49, 2004.

[15] N.V. Chawla, K.W. Bowyer, L.O. Hall, and W.P. Kegelmeyer, "SMOTE: Synthetic Minority oversampling Technique," *J. Artificial Intelligence Research,* vol. 16, pp. 321-357, 2002.

[16] H. Han, W.Y. Wang, and B.H. Mao, "Borderline-SMOTE: A New Oversampling Method in Imbalanced Data Sets Learning," *Proc. Int'l Conf. Intelligent Computing,* pp. 878-887, 2005.

[17] H. He, Y. Bai, E.A. Garcia, and S. Li, "ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning," *Proc. Int'l Joint Conf. Neural Networks,* pp. 1322-1328, 2008.

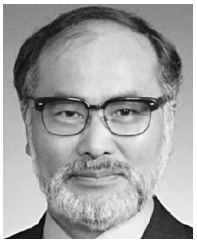
[18] S. Chen, H. He, and E.A. Garcia, "RAMOBoost: Ranked Minority Oversampling in Boosting," *IEEE Trans. Neural Networks,* vol. 21, no. 20, pp. 1624-1642, Oct. 2010.

[19] H. He and E.A. Garcia, "Learning from Imbalanced Data," *IEEE Trans. Knowledge Data Eng.,* vol. 21, no. 9, pp. 1263-1284, Sept. 2009.
[20] N.V. Chawla, D.A. Cieslak, L.O. Hall, and A. Joshi, "Automatically Countering Imbalance and Its Empirical Relationship to Cost," *Data Mining and Knowledge Discovery,* vol. 17, no. 2, pp. 225-252, 2008.
[21] G.M. Weiss and F. Provost, "The Effect of Class Distribution on Classifier Learning: An Empirical Study," Technical Report ML-TR-43, Dept. of Computer Science, Rutgers Univ., 2001.
[22] J. Laurikkala, "Improving Identification of Difficult Small Classes by Balancing Class Distribution," *Proc. Conf. AI in Medicine in Europe: Artificial Intelligence Medicine,* pp. 63-66, 2001.
[23] A. Estabrooks, T. Jo, and N. Japkowicz, "A Multiple Resampling Method for Learning from Imbalanced Data Sets," *Computational Intelligence,* vol. 20, pp. 18-36, 2004.
[24] J. Zhang and I. Mani, "KNN Approach to Unbalanced Data Distributions: A Case Study Involving Information Extraction," *Proc. Int'l Conf. Machine Learning, Workshop Learning from Imbalanced Data Sets,* 2003.
[25] X.Y. Liu, J. Wu, and Z.H. Zhou, "Exploratory Under Sampling for Class Imbalance Learning," *Proc. Int'l Conf. Data Mining,* pp. 965-969, 2006.
[26] I. Tomek, "Two Modifications of CNN," *IEEE Trans. System, Man, Cybernetics,* vol. SMC-6, no. 11, pp. 769-772, Nov. 1976.
[27] M. Kubat and S. Matwin, "Addressing the Curse of Imbalanced Training Sets: One-Sided Selection," *Proc. Int'l Conf. Machine Learning,* pp. 179-186, 1997.
[28] D. Mease, A.J. Wyner, and A. Buja, "Boosted Classification Trees and Class Probability/Quantile Estimation," *J. Machine Learning Research,* vol. 8, pp. 409-439, 2007.
[29] G.E.A.P.A. Batista, R.C. Prati, and M.C. Monard, "A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data," *ACM SIGKDD Explorations Newsletter,* vol. 6, no. 1, pp. 20-29, 2004.
[30] J. Wu, H. Xiong, P. Wu, and J. Chen, "Local Decomposition for Rare Class Analysis," *Proc. Int'l Conf. Know. Discovery and Data Mining (KDD),* pp. 814-823, 2007.
[31] D.A. Cieslak and N.V. Chawla, "Start Globally, Optimize Locally, Predict Globally: Improving Performance on Imbalanced Data," *Proc. IEEE Int'l Conf. Data Mining,* pp. 143-152, 2008.
[32] C. Rao, "A Review of Canonical Coordinates and an Alternative to Correspondence Analysis Using Hellinger Distance," *Questiió: Quaderns d'Estadística, Sistemes, Informatica i Investigació Operativa,* vol. 19, pp. 23-63, 1995.
[33] Y. Freund and R.E. Schapire, "Experiments with a New Boosting Algorithm," *Proc. Int'l Conf. Machine Learning,* pp. 148-156, 1996.
[34] Y. Freund and R.E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," *J. Computer and System Sciences,* vol. 55, no. 1, pp. 119-139, 1997.
[35] S. Wang and X. Yao, "Multi-Class Imbalance Problems: Analysis and Potential Solutions," *IEEE Trans. Systems, Man and Cybernetics,* Part B, vol. 42, no. 4, pp. 1119-1130, Aug. 2012.
[36] N.V. Chawla, A. Lazarevic, L.O. Hall, and K.W. Bowyer, "SMOTEBoost: Improving Prediction of the Minority Class in Boosting," *Proc. 17th European Conf. Principles and Practice of Knowledge Discovery in Databases,* pp. 107-119, 2003.
[37] H. Guo and H.L. Viktor, "Learning from Imbalanced Data Sets with Boosting and Data Generation: The DataBoost IM Approach," *ACM SIGKDD Explorations Newsletter,* vol. 6, no. 1, pp. 30-39, 2004.
[38] C. Drummond and R.C. Holte, "Severe Class Imbalance: Why Better Algorithms Aren't the Answer," *Proc. 16th European Conf. Machine Learning,* pp. 539-546, 2005.
[39] E.M. Voorhees, "Implementing Agglomerative Hierarchic Clustering Algorithms for Use in Document Retrieval," *Information Processing and Management,* vol. 22, no. 6, pp. 465-476, 1986.
[40] H. Schutze and C. Silverstein, "Projections for Efficient Document Clustering," *Proc. 20th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval,* pp. 74-81, 1997.
[41] UC Irvine Machine Learning Repository, http://archive.ics.uci.edu/ml/, 2009.
[42] T. Fawcett, "ROC Graphs: Notes and Practical Considerations for Data Mining Researchers," Technical Report HPL-2003-4, HP Labs, 2003.
[43] T. Fawcett, "An Introduction to ROC Analysis," *Pattern Recognition Letters,* vol. 27, no. 8, pp. 861-874, 2006.
[44] G.W. Corder and D.I. Foreman, *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach.* John Wiley & Sons, 2009.
[45] Critical Value Table of Wilcoxon Signed-Ranks Test, http://www.euronet.nl/users/warnar/demostatistiek/tables/WILCOXONTABEL.htm, 2013.
[46] N. Japkowicz, "Learning from Imbalanced Data Sets: A Comparison of Various Strategies," *Proc. Learning Imbalanced Data Sets, Papers from the AAAI Workshop,* pp. 10-15, 2000.
[47] D. Opitz and R. Maclin, "Popular Ensemble Methods: An Empirical Study," *J. Artificial Intelligence Research,* vol. 11, pp. 169-198, 1999.
[48] J.R. Quinlan, *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers Inc., 1993.
[49] J. Demsar, "Statistical Comparisons of Classifiers over Multiple Data Sets," *J. Machine Learning Research,* vol. 7, no. 7, pp. 1-30, 2006.

Sukarna Barua received the BSc Engg and MSc Engg degrees in computer science and engineering from the Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh, in 2009 and 2011, respectively. He is currently a lecturer with the Department of Computer Science and Engineering, BUET. His research interests include artificial neural networks, data mining, and pattern recognition.

Md. Monirul Islam received the BE degree in electrical and electronic engineering from the Bangladesh Institute of Technology (BIT), Khulna, Bangladesh, in 1989, the ME degree in computer science and engineering from Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh, in 1996, and the PhD degree in evolutionary robotics from Fukui University, Fukui, Japan, in 2002. From 1989 to 2002, he was a lecturer and an assistant professor with the Department of Electrical and Electronic Engineering, BIT, Khulna. In 2003, he moved to BUET as an assistant professor in the Department of Computer Science and Engineering, where he is currently a professor. His research interests include evolutionary robotics, evolutionary computation, neural networks, and pattern recognition.

Xin Yao (M'91-SM'96-F'03) received the BSc degree from the University of Science and Technology of China (USTC), Hefei, China, in 1982, the MSc degree from the North China Institute of Computing Technology, Beijing, China, in 1985, and the PhD degree from USTC in 1990. He was an associate lecturer and a lecturer with USTC, Hefei, China, in 1985-1990, a postdoctoral fellow at the Australian National University (Canberra) and CSIRO (Melbourne) in Australia, in 1990-92, and a lecturer, senior lecturer, and an associate professor at UNSW@ADFA, Canberra, Australia, in 1992-1999. Since April 1999, he has been a professor (chair) of Computer Science at the University of Birmingham, United Kingdom, where he is currently the director of the Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA). He is also a distinguished visiting professor (Grand Master Professorship) of USTC. His major research interests include evolutionary computation and neural network ensembles. He has more than 350 refereed publications. He received the 2001 IEEE Donald G. Fink Prize Paper Award, *IEEE Transactions on Evolutionary Computation*'s Outstanding 2008 Paper Award, *IEEE Transactions on Neural Networks*' Outstanding 2009 Paper Award, and several other best paper awards. He is a fellow of the IEEE, a distinguished lecturer of IEEE Computational Intelligence Society, an invited keynote/plenary speaker of 60 international conferences, and a former (2003-2008) editor-in-chief of the *IEEE Transactions on Evolutionary Computation*.



Kazuyuki Murase received the ME degree in electrical engineering from Nagoya University, Nagoya, Japan, in 1978 and the PhD degree in biomedical engineering from Iowa State University, Ames, in 1983. He has been a professor with the Department of Human and Artificial Intelligence Systems, Graduate School of Engineering, University of Fukui, Fukui, Japan, since 1999, where he was an associate professor with the Department of Information Science in 1988 and a professor in 1992. He was a research associate with the Department of Information Science, Toyohashi University of Technology, Toyohashi, Japan, in 1984. He is a member of the Institute of Electronics, Information and Communication Engineers, the Japanese Society for Medical and Biological Engineering, the Japan Neuroscience Society, the International Neural Network Society, and the Society for Neuroscience. He serves on the Board of Directors of the Japan Neural Network Society, is a councilor of the Physiological Society of Japan, and is a councilor of the Japanese Association for the Study of Pain.