M.Sc. Engg. Thesis

MWMOTE-Boost: Majority Weighted Minority Over-sampling Technique Integrated with Boosting for Imbalanced Data Set Learning

by
Sukarna Barua

Submitted to

Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science and Engineering

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology (BUET)
Dhaka 1000, Bangladesh

October 2011

The thesis titled "**MWMOTE-Boost: Majority Weighted Minority Over-sampling Technique Integrated with Boosting for Imbalanced Data Set Learning**," submitted by Sukarna Barua, Roll No. 0409052061P, Session April 2009, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering and approved as to its style and contents. Examination held on October 1, 2011.

Board of Examiners

1. _____

Dr. Md. Monirul Islam — Chairman
Professor & Head — (Supervisor)
Department of Computer Science and Engineering
BUET, Dhaka 1000

2. _____

Dr. Md. Monirul Islam — Member
Professor & Head — (Ex-officio)
Department of Computer Science and Engineering
BUET, Dhaka 1000

3. _____

Dr. Md. Saidur Rahman — Member
Professor
Department of Computer Science and Engineering
BUET, Dhaka 1000

4. _____

Dr. Md. Monirul Islam — Member
Assistant Professor
Department of Computer Science and Engineering
BUET, Dhaka 1000

5. _____

Dr. Chowdhury Mofizur Rahman — Member
Professor — (External)
Department of Computer Science and Engineering
United International University (UIU), Dhaka 1000

Candidate's Declaration

It is hereby declared that this thesis or any part of it has not been submitted elsewhere for the award of any degree or diploma.

Sukarna Barua
Candidate

Contents

List of Figures

List of Tables

Acknowledgments

I express my heart-felt gratitude to my supervisor, Dr. Md. Monirul Islam for his constant supervision of this work. He helped me a lot in every aspect of this work and guided me with proper directions whenever I sought one. His patient hearing of my ideas, critical analysis of my observations and detecting flaws (and amending thereby) in my thinking and writing have made this thesis a success.

I would also want to thank the members of my thesis committee for their valuable suggestions. I thank Dr. Md. Saidur Rahman, Dr. Md. Monirul Islam, and specially the external member Dr. Chowdhury Mofizur Rahman.

In this regard, I remain ever grateful to my beloved parents, who always exists as sources of inspiration behind every success of mine I have ever made.

Abstract

Imbalanced data sets contain an unequal distribution of data samples among the classes and pose a challenge to the learning algorithms as it becomes hard to learn the minority class concepts. Synthetic oversampling techniques address this problem by generating synthetic minority samples to balance the distribution between the samples of the majority and minority classes. This thesis identifies that most of the existing synthetic oversampling techniques may generate wrong synthetic samples in some scenarios and make the learning task harder. To this end, the thesis presents a new synthetic oversampling method, called Majority Weighted Minority Oversampling Technique (MWMOTE), for handling imbalanced data sets efficiently. The term 'majority weighted minority oversampling' here means important minority samples for oversampling will be identified and weighted by the nearest majority samples and then will be used for oversampling. To do this, MWMOTE uses information from both the minority and majority samples in the data set. First, it identifies hard-to-learn informative minority samples and assigns them weights according to their importance using distance information from the nearest majority samples. MWMOTE then identifies the clusters in the minority data set and uses weighted informative minority samples to generate synthetic samples inside the clusters. This is done in order to ensure that generated samples always lie inside some minority cluster and do not overlap with majority regions.

The thesis finally presents a new stand-alone ensemble algorithm, called, MWMOTE-Boost, by integrating MWMOTE inside the famous AdaBoost.M2 boosting procedure. MWMOTE-Boost algorithm is obtained from MWMOTE oversampling algorithm by inserting it into the boosting iteration of classic AdaBoost.M2 ensemble algorithm. The manner in which MWMOTE and AdaBoost.M2 are integrated is similar to the recent state-of-the-art RAMOBoost algorithm except that in place of RAMOBoost's RAMO

oversampling procedure, MWMOTE oversampling procedure is used. The proposed methods, i.e., MWMOTE and MWMOTE-Boost have been evaluated extensively on four artificial and seventeen real-world data sets and using several classifier models such as neural network, decision tree, k-nearest neighbor and ensemble classifier. The simulation results show that our new methods MWMOTE and MWMOTE-Boost are better or comparable than some other existing methods in terms of various assessment metrics, such as precision, recall, F-measure, G-mean, and area under the receiver operating curve (ROC), usually known as area under curve (AUC).

Chapter 1

Introduction

The task of machine learning is to generate a model, also called a hypothesis, which best approximates an objective function. In classification problems, the output of the objective function is discrete in which the goal is to classify an example to one of a set of output classes. Each machine learning task is given a training data set consisting of a set of known examples and their corresponding target function outputs, for its learning. Training data is supposed to provide adequate information for the underlying function to be learned. However, in many practical learning problems, training datsets are imbalanced where most of the training examples belong to one class and rest of the examples belongs to other classes [1]. These datasets are known as imbalanced datasets. When the dataset involves only two classes, the class having most of the data samples is called the majority class, and the other, minority class. The ratio of the number of minority and majority class instances is defined as the degree of imbalance or imbalance ratio for the data set [1]. Imbalance data sets are of utmost importance to the research community as it is present in many vital real world classification problems such as medical diagnosis [2], information retrieval systems [3], detection of fraudulent telephone calls [4], detection of oil spills in radar images [5], data mining from direct marketing [6], helicopter fault monitoring [7] etc. Imbalance ratio has been found to be as high as 1:100 in fraud detection problems [4] to 1:100000 in high energy physics event classification [8].

The class imbalance can be categories in different ways. One categorization is *between-class* imbalance and *within-class* imbalance [1]. A *between-class* imbalance is the imbalance existing between classes of the data set. A *within-class* imbalance is an imbalance existing within the same class. This occurs due to unequal distribution of samples among the various sub-concepts presents in the same class. Another categorization is *relative* imbalance and *absolute* imbalance [1]. To illustrate the difference between *relative* and *absolute* imbalance, consider a training dataset that contains 260 minority and 10923 majority class samples, a total of 11183 samples. Assume that, we will double the data set size, by taking 11183 more data samples. Now, in the new data set, if we get approximately double number of minority samples, then we say that this type of imbalance is *relative*. Because, increasing the data set size also increases the number of minority samples, however, keeping the ratio of imbalance approximately constant. This says the minority classes are not rare in its own, but relatively outnumbered by majority due to the far greater existence of the majority samples in nature. *Absolute* imbalance occurs for rare cases, in which minority classes are very rare. Therefore, increasing data set size does not necessarily increase the number of minority samples.

Class imbalance pose a challenge to the learning algorithms as it becomes very hard to learn the minority class concepts [9]–[11]. Classifiers usually aim to reduce the global classification error and therefore any classifier, learned form an imbalanced dataset, tends to favor the majority class. The decision output of the classifier is always biased toward the majority class as it expects that the distribution of examples in the nature is imbalanced and dominated by majority. As a result, the classifiers show greater classification errors over the examples of minority class. They show very high accuracy on majority class samples, however, poor accuracy on minority class samples. This becomes very costly in problems where the identification of the minority is of utmost importance [2]–[8]. Therefore, it is very important that, the learned classifiers from an imbalanced data set perform well on the minority class samples as it performs on the majority.

The actual cause of the worse performance of conventional classifiers on minority class data is not necessarily related to only *between-class* imbalance problems. Classifiers' per-

formance have been found to be depreciated in the presence of *within-class* imbalance, and small disjuncts problem [12]–[14]. Due to *within-class* imbalance one or more sub-concepts become difficult to learn compared to other sub-concepts that are sufficiently represented. The main problem relates to the representation of the concepts. If a minority class concept is well represented, then the between class imbalance usually poses no problem. But, if the class concept is not represented well by the given samples, then imbalance creates problems.

Besides the class imbalance, complexity of data samples is another factor for poor performance [1]. If a classification problem is such that, it has a *between-class* imbalance of 1:100, however, the minority class is well represented and the data set is linearly separable, then the imbalance will not create any problem to learn the concepts. However, if, on the other hand, data samples are such that, the majority and minority have more than one sub-concepts, some sub-concepts are rare than others, and regions of some sub-concepts between majority and minority class overlaps, then, the imbalance becomes a severe problem [12, 14] and it becomes really hard for the classification algorithms to learn the concepts.

1.1 Our Contribution

Extensive research works exist in literature that deals with imbalanced learning problem of which synthetic oversampling methods are among the most popular and effective techniques [15]–[18]. These techniques handle the imbalance problem by generating new synthetic samples for the minority class in order to balance the data set before presenting it to the final classifier [15]. In this thesis, we illustrate that in some scenarios, most of these methods may become inappropriate and fail to generate useful synthetic minority samples. We also show scenarios where these methods may generate wrong and unnecessary synthetic samples, making the learning task difficult. In this respect, we propose a novel synthetic oversampling method, Majority Weighted Minority Oversampling TEchnique (MWMOTE), whose goal is to alleviate the problems of existing methods and

generate useful synthetic minority samples beneficial for learning. The essence of the proposed method is (i) selection of an appropriate subset of the original minority, (ii) assigning weights to the selected samples according to their importance in the data set, and (iii) using an unsupervised clustering approach for generating useful synthetic minority samples. Finally, We present an ensemble algorithm MWMOTE-Boost by integrating MWMOTE and classic AdaBoost.M2 boosting ensemble in order to obtain a stand-alone learning model for imbalanced learning problems.

1.2 Outline of the Thesis

The remainder of this thesis is divided into seven chapters. In Chapter 2, we provide a brief overview of existing works in imbalanced problem domain and present the problems of existing approaches. Chapter 3 describes the MWMOTE algorithm and its various components in detail. In Chapter 4, we present the experimental study and simulation results of MWMOTE. Chapter 5 presents the MWMOTE-Boost algorithm and its associated experimental results. Finally, in Chapter 6, we provide some future aspects of this research and conclude the thesis.

Chapter 2

Related Works

Significant works have been conducted to deal with the imbalanced learning problem. Most of these works fall into one of the four different categories: sampling based approaches, cost based approaches, kernel based approaches, and active learning based approaches [1]. In this chapter, we provide a brief overview of sampling based approaches and cost based approaches. Details of works performed in other categories can be found in [1].

2.1 Sampling Methods for Imbalanced Learning

In imbalanced learning sampling methods are used to modify an imbalanced data set to create a balanced distribution. Classifiers learn well from the balanced data set than from the imbalanced one [19]–[21]. This doesn't mean that classifiers can't learn from imbalanced datasets. In fact some studies show that the performance of classifiers induced from certain imbalanced data sets are almost same as classifiers induced from the same data set balanced by sampling techniques. But in most of the cases for imbalanced dataset the application of sampling techniques does improve classifier accuracy. There are two differnt types of sampling methods: undersampling and oversampling.

2.2 Undersampling Methods

Undersampling methods work by reducing the number of instances of the majority class to balance the class distribution. The reduction can be done randomly in which case it is called random undersampling or it can be done by using some statistical knowledge in which case it is called informed undersampling. Both techniques show improvements in classifier performance over the imbalanced data sets [22]–[24].

2.2.1 Random Undersampling

Random Undersampling first selects a set of samples E, at random from the original majority data set, S_{maj}. The size of the set E determines the degree of balance required. The samples of set E are then deleted from the original training set, $S = S_{min} \cup S_{maj}$. The new training set S' becomes:

$$S' = S_{min} \cup (S_{maj} - E) \tag{2.1}$$

Thus random undersampling randomly removes a set of samples from S_{maj} before providing it to the classifier. Random undersampling has been shown to give improvements in performance compared to purely imbalanced data [22]–[24]. However, a basic disadvantage of this method is that, it removes examples without any consideration. Therefore, important majority examples containing necessary information may be removed and those regions becomes hard to learn for the classifier.

2.2.2 Informed Undersampling

Informed undersampling removes samples from the original majority data set using some statistical information. Two different undersampling algorithms, *EasyEnsemble* and *BalancedCascade*, were shown to provide good results in informed undersampling [25]. These algorithms can overcome the deficiency of information loss introduced in the traditional random undersampling methods. The *EasyEnsemble* method develops an ensemble learn-

ing system by independently sampling several subsets from the majority class and developing multiple classifiers based on the combination of each subset with the minority class data. So we can consider *EasyEnsemble* as an unsupervised learning algorithm that explores the majority class data by using independent random sampling with replacement. On the contrary, *BalancedCascade* algorithm follows supervised learning approach. This method develops an ensemble of classifiers. These classifiers can systematically select the majority class examples to undersample.

Informed undersampling can also be done by the help of k-nearest neighbor (k-NN) classifier. Four k-NN undersampling methods were proposed in [22] which uses the characteristics of the training data distribution. They are NearMiss-1, NearMiss-2, NearMiss-3, and the "most distant" method. These methods calculate distances between majority and minority samples in different ways. NearMiss-1 method removes those majority examples from the data set whose average distance to the three closest minority class examples is smallest. NearMiss-2 method removes those majority class examples whose average distance to the three farthest minority class examples is smallest. NearMiss-3 method selects a given number of the closest majority examples for each minority example to guarantee that every minority example is surrounded by some majority examples. Lastly, the "most distance" method selects the majority class examples whose average distance to the three closest minority class examples is largest. Among all these methods, NearMiss-2 method was shown to provide competitive results [22].

2.2.3 Undersampling with Data Cleaning Techniques

Some undersampling methods apply data cleaning techniques to further refine the majority data set. One such method was one-sided selection (OSS) [24]. In OSS, those examples from the majority set are removed that are redundant, noisy and borderline. According to OSS, borderline samples are unreliable and create overlapping regions. This creates difficulty for the classifier. So, borderline and noisy samples are removed. They are identified using a procedure called tomek-link method [26]. To understand what tomek-link is,

let x and y are two examples belonging to majority and minority class respectively. The pair (x, y) is called a tomek-link, if no other example z exists such that $d(x, z) < d(x, y)$ or $d(y, z) < d(x, y)$. Examples belonging to tomek-links are said to be either noisy or borderline. All majority examples that forms a tomek-link with a minority are therefore removed from the original data set. Besides this, another set of examples are removed from the majority class that are redundant. The redundant samples are identified using a concept of consistent subset $C \subset S$. A subset C of S is called consistent, if using 1-NN rule, examples in C are able to correctly classify all examples in S. The consistent subset is found by first selecting a subset, C of S consisting of all minority examples and only one positive example. 1-NN rule is then used to classify examples in S using examples in C. Those examples in S that will be misclassified are put in C. The process has been shown to improve the accuracy on both minority and majority class in some real-world data sets [24].

In [12] various undersampling methods were compared like condensed nearest neighbor rule (CNN), addition of tomek links with condensed nearest neighbor rule (CNN+Tomek-link), tomek only method (Tomek), and neighborhood clearning rule (NCL) method. CNN rule is similar to finding a consistent subet C of S, that can correctly classify all examples of S using 1-NN rule. In CNN+Tomek, tomek-links are idenfied and removed after applying CNN. In NCL, a modified version of Wilson's edited nearest neighbor rule [27] is used to enforce the cleaning: For each example x of S, its three nearest neighbors are identified. If x is a majority example and classification assigned by its three nearest neighbors differs, then x is removed from S. On the other hand, if x is a minority example, and classification by its three nearest neighbors differ, then majority examples belonging to three nearest neighbors are removed. In this way, only majority examples are removed from the training data set.

2.3 Oversampling Methods

Oversampling methods add minority samples to the data set to balance the distribution of samples between majority and minority classes. Similar to undersampling methods, oversampling methods also fall in two broad categories: random oversampling and synthetic oversampling methods.

2.3.1 Random Oversampling

Random Oversampling is a very simple mechanism by which instances are added to the original minority data set S_{min} by random resampling. First, a set of samples, E is selected from the original minority samples S_{min}. Then samples in E are added with S_{min} to form the new minority data set S'_{min} ($S'_{min} = S_{min} \cup E$). In this way, the minority set is augmented to balance with the majority set. The size of the set E determines the degree of balance required. Due to resampling, some examples of the minority data set are duplicated. The new training set, S' becomes:

$$S' = S_{maj} \cup S'_{min} \tag{2.2}$$

Random oversampling adds data to the original minority set, thereby adding new information in the dataset. However, due to duplication, the additional information is overlapped or replicated. This may create small disjuncts in the minority data set and lead to overfitting. In rule-based learning, this type of small disjuncts creates very specific rules for some regions, which is good for fitting training data. However, the performance of such specific rules sometimes shows bad performance over unseen examples due to poor generalization [10].

Cluster Based Random Oversampling

A cluster-based random oversampling (CBO) technique was proposed in [28]. It focuses not only on *between-class* imbalance but also on *within-class* imbalance. The objective

is to remove both *within-class* imbalance and *between-class* imbalance problems. The technique randomly oversamples the minority and majority class in such a fashion that all clusters become of same size. The method starts with clustering majority and minority set using k-means algorithm. In k-means algorithm, initially k random examples are selected from the original set. Each of these k samples represents one of the k clusters. For each of the other examples in the set, the nearest of the k initial examples are identified and the sample is assigned to the cluster represented by the nearest example. In this way, all examples are assigned to one of k clusters. Then, k new cluster centers are calculated by taking average of all members of each cluster. Examples of all the clusters are then re-assigned to one of k clusters depending on its nearest cluster center. After clustering is done, CBO algorithm randomly oversamples each majority cluster except the largest one so that its size become equal to the size of the largest cluster in the majority set. In this way, original majority set S_{maj} is augmented to new majority S'_{maj}. The minority clusters are then oversampled such that each minority cluster has size equal to $|S'_{maj}|/k$.

2.3.2 Synthetic Oversampling

Synthetic oversampling, like random oversampling adds data to the minority data set. However, unlike random oversampling, the samples that will be added are not selected from the original minority data set S_{min}. Instead of duplicating existing minority samples, synthetic oversampling methods generate new minority examples. The generated examples provide new information to the original data set and can lead to good classifier performance. Depending on the technique of how synthetic samples will be generated there are many variants existing in literature such as SMOTE [15], Borderline-SMOTE [16], ADASYN [17].

SMOTE

SMOTE (Synthetic minority oversampling technique) [15] works by generating synthetic samples and adding those samples to the original minority data set S_{min}. The synthetic

samples are generated by combining pairs of samples from S_{min} in a linear combination. For each sample $x \in S_{min}$, its k nearest minority examples are identified. The value k is a parameter which determines the local neighborhood information used. The k nearest neighbors are found according to a distance metric such as Euclidean distance. After that, an example y is selected at random from these k nearest neighbors. One synthetic sample g is then generated from x and y as:

$$g = x + (y - x) \times \delta \tag{2.3}$$

where δ is a random number in the range $[0, 1]$. The generated synthetic sample will always lie on the line segment between x and y. Depending on the amount of oversampling required, one or more examples from the k neareset neighbors are selected. For example, if we require a 300% oversampling-rate, then three are selected at random from the k nearest neighbors of x and three new synthetic samples are generated by using (2.3). The SMOTE technique leads to larger and more generalized regions in the feature space. Unlike random oversampling, which leads to smaller and specific decision region, SMOTE creates larger and less specific regions.

Borderline-SMOTE

In classifier learning, it appears that, some decision regions are very well represented which can be learned easily, some regions are poorly represented which are harder to learn and some regions are overlapped which creates difficulty in learning. Regions that are near the decision boundary are usually overlapped between classes. Therefore, samples along the decision boundary are in the regions that are harder to learn for the classifier. So, it would be better, if more synthetic samples are generated for harder decision regions to make it more precise and represented and few synthetic samples are generated for the easier regions (that are far from the boundary and can be learned without any ambiguity). Borderline-SMOTE [16] tries to achieve this. In Borderline-SMOTE, synthetic samples are generated using (2.3). However, unlike SMOTE, which create synthetic samples from all minority examples without any consideration of their importance, Borderline-SMOTE

creates synthetic samples from borderline minority examples only. The technique first identifies all minority examples that are borderline. The borderline samples form a set, DANGER and synthetic samples are generated from only members of the $DANGER$ set. According to [16], a sample falls in the DANGER set, if at least half of its k nearest neighbors are majority examples. So, to find borderline minority samples, Borderline-SMOTE first finds, for each minority sample x, it's k nearest neighbors (considering all minority and majority). Then, the teqnique finds the number of majority samples in these k-neighbors and say, the number is m. Then sample x will be called a borderline sample if following equation holds:

$$\frac{k}{2} \leq m < k. \tag{2.4}$$

Equation (2.4) states that m must by greater than or equal to $\frac{k}{2}$ and strictly less that k. This says that, at least half of the k-neighobrs must be majority to call x a borderline sample. However, the value of m must be less than k. In case, $m = k$, the minority sample x is not called borderline and no-synthetic samples are generated from this. Rather it is identified as a potential noise sample and removed from the data set.

ADASYN

Borderline-SMOTE [16] is too aggressive in the sense that, it generates all synthetic samples from only DANGER set and no synthetic samples are generated from any other minority examples. Therefore, region near the decision boundary become too dense, while rest of the regions become sparse. ADASYN (adaptive synthetic sampling) [17] works in an adaptive fashion and tries to balance the distribution of synthetic among the minority regions depending on its importance. It finds the number of synthetic samples to be generated from each minority according to its importance in the learning. The importance of a minority sample in the learning phase is determined by number of majority neighbors it has in the neighborhood. ADASYN first calculates the total number of synthetic samples to be generated, G from the original minority data set, S_{min}:

$$G = (|S_{maj}| - |S_{min}|) \times \beta \tag{2.5}$$

Where, β is a balance parameter. The value of β determines the degree of balance required. Next, for each sample x_i, k nearest neighbors (considering both majority and minority) are identified using Euclidean distance as the distance measure. Then, a ratio r_i is calculated for each x_i as follows:

$$r_i = \frac{\delta_i}{K} \tag{2.6}$$

where, δ_i is the number of majority examples in the k nearest neighbors. The value r_i is then normalized according to the following equation:

$$\widehat{r_i} = \frac{r_i}{\sum_i r_i} \tag{2.7}$$

The normalized r_i value is the probability of importance and is used to find the number of synthetic samples to be generated, g_i from each x_i as:

$$g_i = \widehat{r_i} \times G \tag{2.8}$$

After that, g_i samples are generated from each x_i, using (2.3) similar to SMOTE and Borderline-SMOTE. The objective of ADASYN is to balance the distribution of synthetic samples among the regions depending on its difficulty for the classifier. Unlike, SMOTE, where number of synthetic samples that will be generated from each minority is prefixed, ADASYN finds it adaptively for each sample. The resulting data distribution is far focused on the harder regions and therefore expected to provide a better minority data set to the classifier.

2.4 Integration of Sampling and Boosting

Boosting is expected to improve the classifier performance by iteratively focusing on hard-to-learn examples in the training data set. The application of boosting [29, 30] can be integrated with sampling to provide a better classifier structure for imbalanced learning. Works in this category were proposed by SMOTE-Boost [31], DataBoost-IM [32], RAMOBoost [18], etc.

SMOTE-Boost

SMOTE-boost [31] integrates Adaboost.M2 [29] procedure with the SMOTE [15] oversampling technique to improve the prediction accuracy of the minority class as well as the majority class. In each iteration of standard AdaBoost.M2 boosting procedure, the weight distribution of the minority data set is updated by creating N synthetic samples using SMOTE. While SMOTE focuses on the predictive accuracy of the minority class, boosting focuses on the majority class so that total accuracy is not sacrificed for improving minority performance.

RAMOBoost

RAMOBoost [18] is obtained by modifying the ADASYN [17] algorithm and integrating the standard Adaboost.M2 boosting procedure. Similar to SMOTE-boost [31], RAMOBoost's objective is to improve accuracy on both minority and majority class by applying boosting and oversampling simultaneously. RAMOBoost's oversampling procedure is a slight modification of ADASYN which is describe below:

Similar to ADASYN, RAMOBoost finds a value, δ_i for each minority sample x_i. Unlike ADASYN, RAMOBoost then finds r_i as follows:

$$r_i = \frac{1}{1 + \exp^{-\alpha \times \delta_i}} \tag{2.9}$$

The value of r_i is then normalized to form a probability distribution $\widehat{r_i}$:

$$\widehat{r_i} = \frac{r_i}{\sum_i r_i} \tag{2.10}$$

An informative minority example set, S'_{min} is selected from S_{min} according to the probability distribution of $\widehat{r_i}$. Synthetic samples are then generated from all members of S'_{min} according to the similar procedure of SMOTE. The difference between ADASYN and RAMOBoost is in two-folds. First, ADASYN aggressively finds the number of synthetic samples to be generated for each x_i according to δ_i. But, RAMOBoost converts the δ_i value to a probability distribution $\widehat{r_i}$. The $\widehat{r_i}$ value in RAMOBoost determines the

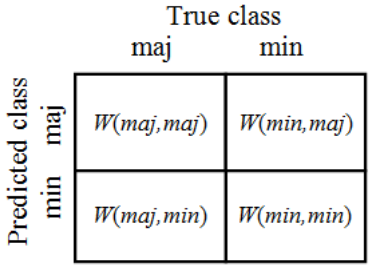


Figure 2.1: Cost of misclassifications.

probability of selection of a minority example for synthetic sample generation. Another difference is that, ADASYN generates no synthetic samples from minorities that have no majority in its k nearest neighborhood. However, in RAMOBoost, those samples may contribute to generation of synthtetic samples although they will have the smallest $\widehat{r}_i$ value, the probability that it will be selecting for synthetic sample generation.

2.5 Cost-sensitive Learning for Imbalance Data

Cost sensitive learning approaches assigns higher misclassification costs for the minority examples to bias the classifier to the minority class [1]. A cost matrix formally defines the costs of each misclassification. In Fig. 2.1, we show an example of a cost matrix for two classes. The term 'cost' can also be interpreted as a 'penalty' term given to the classifier for a misclassification. The cost $W(maj, min)$ determines the cost of misclassifying a majority object. Similarly, $W(min, maj)$ is the cost of misclassifying a minority object as a majority object. The other two costs $W(maj, maj)$ and $W(min, min)$ are usually given zero value to emphasize the fact that no cost is imposed when a correct classification occurs. For imbalanced learning problem $W(min, maj)$ is assigned a higher value than $W(maj, min)$. The classifiers are expected to minimize the overall cost of the classification. In Bayesian learning techniques, this term is also referred as risk minimization.

There are different ways on how to impose the cost to the learning algorithm. The first way, is to weight the individual data samples. This is usually referred to as data-space

weighting [1]. Classification algorithms are integrated with some form of ensemble learning methods, the most common and successful ensemble technique of which is boosting. In each iteration of ensemble methods, a new training set is chosen from the available data samples according to the weight of the samples. The costs are imposed on weight of each sample to increase its selection in training sets of ensemble algorithms. In this way, costly data samples are selected many times and classifier performance on those improves. The second way of applying cost sensitive learning is incorporate the cost into the objective function to be minimized. Minimizing the cost-sensitive objective function therefore introduces additional bias toward the minority class since their cost is high. The third way of cost-sensitive algorithms is to incorporate cost factors inside leaning algorithms in such a way that they are biased toward minority class.

2.5.1 Cost Sensitive Boosting

Various cost sensitive boosting algorithms based on famous Adaboost algorithm were developed in research community. In [33], Adaboost is modified to incorporate cost factors inside the algorithm. The classic Adaboost algorithm [29] employs a distribution function $D(t)$ in each iteration which is updated according to the classifier learned. For each training instance x_i, The distribution update function is stated as follows:

$$D(t+1) = D(t) * e^{-\alpha_t * h_t(x_i) * y_i} / Z_t \tag{2.11}$$

where $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$ is the weight update parameter, $h_t(x_i)$ is the hypothesis output on the training example x_i, y_i is the true output of x_i, and Z_t is a normalization constant so that $D(t+1)$ becomes a distribution. Equation 2.11 increases the weight of the sample x_i, if it is misclassified by the current classifier. Therefore, misclassified samples achieve higher weight in the next iteration of boosting. In [33], three new cost-sensitive algorithms were proposed by modifying the update equation (Eqn. 2.11) in three ways (Eqns. 2.12-2.14) to incorporate the cost factor:

$$D(t+1) = D(t) * e^{-C_i * \alpha_t * h_t(x_i) * y_i} / Z_t \tag{2.12}$$

$$D(t+1) = C_i * D(t) * e^{-\alpha_t * h_t(x_i) * y_i} / Z_t \tag{2.13}$$

$$D(t+1) = C_i * D(t) * e^{-C_i * \alpha_t * h_t(x_i) * y_i} / Z_t \tag{2.14}$$

In (2.12), cost is imposed inside the exponential, in (2.13), outside the exponential and in (2.14), both inside and outside exponential. In all equations C_i is the misclassification cost associated with example x_i. Classifiers corresponding to (2.12), (2.13), and (2.14) are called AdaC1, AdaC2, and AdaC3 respectively. The objective was to find the best cost-sensitive model. Various experiments were performed on UCI data and results were reported in [33]. The reported results says the AdaC2 and AdaC3 obtains a significantly improved performance than AdaC1 and other conventional boosting procedures.

Another cost-sensitive boosting technique was proposed in [34] and is called AdaCost. AdaCost employs a similar technique of $adaC1$ [33] by applying cost factors inside exponential. However, the cost factor is not applied directly. Rather a cost-adjustment function is used as follows:

$$D(t+1) = D(t) * e^{-\alpha_t * h_t(x_i) * y_i * \beta_i} / Z_t \tag{2.15}$$

where, β_i is a cost-adjustment function which aggressively increases the weights of misclassified examples that are costly and conservatively decreases the weight of costly examples that are correctly classified. The cost-adjustment function can take the following form:

$$\beta_i = \beta(sign(h(x), y), C_i) \tag{2.16}$$

The function $sign(h(x), y)$ gives positive for correct classification and negative for incorrect classification. The β function can also be stated separately as $\beta(+)$ and $\beta(-)$. $\beta(+)$ adjusts weight of correctly classified examples and $\beta(-)$ adjusts for misclassified samples. One suggestion on how $\beta(+)$ and $\beta(-)$ can be selected is given in [34] as:

$$\beta(+) = -0.5 * C_i + 0.5, \beta(-) = 0.5 * C_i + 0.5 \tag{2.17}$$

Using $\beta(+)$ and $\beta(-)$ as in (2.17) was shown to give good results in most applications. The advantage of such a cost-adjustment function is that, cost can be adjusted according to specific need of the problem at hand.

2.6 Comparison between Undersampling and Oversampling

Sampling algorithms and boosting algorithms integrated with sampling dominate among all the works performed in imbalanced learning. Although both types of sampling, i.e., undersampling and oversampling, have been shown to improve classifer performance over imbalanced data sets, in [13], it was shown that, oversampling is lot more useful than undersampling. The performance of oversampling algorithms was shown to improve dramatically even for complex data sets [13]. The reason is that, undersampling methods may remove important information from the data sets during their undersampling procedure which may cause the classifier to miss important sub-concepts of the majority class. Therefore, most of the recent works are based on synthetic oversampling algorithms [17, 18].

2.7 Problems of Existing Methods

Existing Synthetic oversampling techniques have been shown to be very successful in dealing with imbalance data sets [15]–[18]. However, our study finds some insufficiencies and inappropriatenesses of some of these existing techniques that may occur in many different scenarios of the data samples. In this section, we describe those inefficiencies observed by existing techniques in detail.

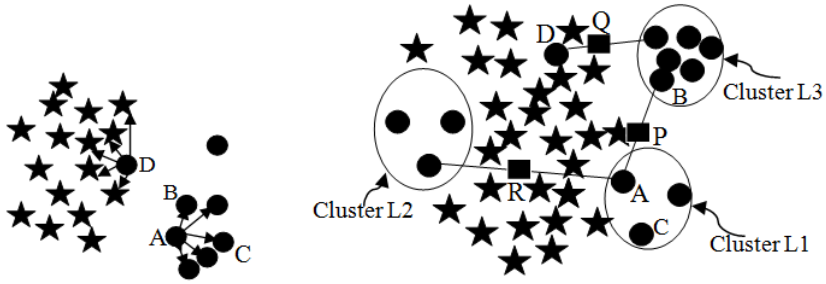
2.7.1 Problems in Minority Selection and Minority Weighting Mechanism

The main objective of some synthetic oversampling methods, e.g. Borderline-SMOTE [16], is to identify borderline minority samples (also called seed samples). These samples are then used for generation of synthetic samples. The intuition is that, borderline samples

are most likely to be misclassified and therefore synthetic samples should be generated from them in the neighborhood of the borders. According to [16], a minority sample is called borderline, if the number of majorities, m, among its k nearest neighbors satisfies the criterion (2.4). However, the criterion (2.4) may fail to identify borderline samples in some scenarios. Figure 2.2(a) shows such a scenario where stars and circles represent the samples of majority and minority classes, respectively. Minority samples A and B in this figure have no majority samples in their k-nearest neighborhood assuming the value of k is 5. In fig 2.2(a), 5 nearest neighbors of A are shown by arrow that are all minority samples. Hence, the value of m for these minority samples is 0. According to (2.4), these values ($m = 0$ and $k = 5$) imply that A and B will not be identified as borderline samples, though they are the closest samples to the decision boundary (apparently borderline). The outcome is the prohibition of synthetic sample generation from A and B later in the sample generation phase. This situation will make the learning task harder, because a classifer will get less information about minority samples near the decision boundary.

Some synthetic oversampling techniques such as ADASYN [17] and RAMOBoost [18] try to find the relative importance of different minority samples in the data set using weighting mechanisms. A higher weight for a minority sample results in higher number of synthetic samples to be created from that sample in ADASYN [17] or a higher chance for that sample to be selected for synthetic data generation in RAMOBoost [18]. To weight the minority samples, both ADASYN and RAMOBoost find a value δ, where δ is the number of majorities among the k nearest neighbors of a sample. In both techniques, a higher value of δ for a minority sample results in a higher weight for that sample. However, using the δ, number of majorities, to weight individual minority samples, may encounter follwoing problems.

1. *δ-value may be insufficient for assigning weights to minority samples located near the decision boundary.* This can be understood from Fig. 2.2(a) in which minority samples A and B have no majority samples in their k-nearest neighborhood (assuming $k = 5$). Under this condition, the value of δ for these samples will be 0. It



(a) (b)

Figure 2.2: Problems of existing approaches: (a) k nearest neighbors of A and D are shown by arrow (for $k = 5$). All neighbors for A are minority which will result in lowest weight. On the contrary, for the noisy sample D, all neighbors are majority, which will result in largest weight. (b) Some minority clusters where generated synthetic samples are shown by square.

means 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. *δ-value may be insufficient to discover the difference of minority samples w.r.t their importance in learning.* It can be seen from Fig. 2.2(a) that the minority sample A is closer to the decision boundary while C is quite further. It is, therefore, reasonable that A should be given higher weight (importance) than C. However, if we compute the δ-value with $k = 5$ for samples A and C, then it is evident from the figure that the δ-value is 0 for both samples (due to no majority samples in their neighborhood). This example illustrates that the δ-value cannot differentiate minority samples according to their importance in learning.

3. *δ-value may favor noisy samples.* Now, consider the minority sample D shown in Fig. 2.2(a). This sample falls in the majority region and is likely to be noisy. Assuming $k = 5$, the δ-value for D is 5, since all the 5 nearest neighbors of D are majority samples (shown by arrow in the figure). This is the highest among the δ values of all minority samples. At this point, we can say that D is unexpectedly getting the highest weight and consequently, later in the sample generation phase,

more noisy synthetic samples will be generated from D. This will eventually make the learning task harder due to introduction of more noise in the data set.

2.7.2 Problems in Synthetic Sample Generation Mechanism

In the sample generation phase, existing synthetic oversampling methods (e.g. [15]–[18]) employ a k-nearest neighbor (also called k-NN) based approach. To generate a synthetic sample from an existing minority sample, say x, the k-NN based approach randomly selects another minority sample, say y, from the k nearest neighbors of x. Here k is a user specified parameter. A synthetic sample, g, is then generated by linear interpolation of x and y using (2.3).

Equation 2.3 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 samples. To show why, again consider Fig. 2.2(b). Assume we are going to generate a synthetic sample from the minority sample A and the value of k is 5. The k-NN based approach will randomly select another minority sample from the 5 nearest minority neighbors of A, say B is selected. According to (2.3), linear interpolation of A and B may generate a synthetic sample like P, shown by a square in Fig. 2.2(b). We see that P is clearly a wrong minority sample because it overlaps with a majority sample (Fig. 2.2(b)).

The above problem is magnified when small sized clusters are present in the minority class concept due to rare or exceptional cases in the training data set such as clusters $L1$ and $L2$ of Fig. 2.2(b). If synthetic samples are generated from any member x of any of these clusters (say, $L2$), the k-NN based approach will likely select y of (2.3) from the other cluster (i.e., $L1$). This causes the generation of a synthetic sample in the majority region situated between $L1$ and $L2$ (e.g. R shown in Fig. 2.2(b)). The generated samples clearly produces overlapping minority and majority regions, which will make the learning task harder. The above problem is even worse when synthetic samples are generated from a noisy sample such as D of Fig. 2.2(b). In this scenario, a synthetic sample Q may be generated that falls inside majority regions (Fig. 2.2(b)).

Now, think what happens, when synthetic samples are generated from the members of a dense minority cluster, for example $L3$ of Fig. 2.2(b). For $k = 5$, generated samples will nearly be duplications of existing samples within the same cluster. This is due to the fact that all of the k nearest neighbors are very close to each other. Generation of dupicated samples is useless because they will not add any new information to the data set.

The above analyses show that k-NN based sample generation approach used may be inappropriate in some cases. This approach may generate unnecessary, nearly duplicated synthetic minority samples from members of dense clusters and wrong synthetic minority samples from members of small-sized clusters. It is important to note that when synthetic samples are generated from noisy samples, generated samples will also be noisy in most cases. All these problems occur due to the fact that the k-NN based sample generation approach uses all k nearest neighbors blindly without considering the position and distance of the neighbors from the minority sample under consideration. Moreover, the appropriate value of k cannot be determined, beforehand.

2.8 Summary

In summary, most of the existing approaches fail to identify required minority samples for synthetic sample generation. Some approaches use weights for the minority samples according to its importance, however the mechanisms used in these approaches are insufficient and can not appropriately weight the necessary samples. Besides, the k-NN based data generation approach used by these techniques may generate wrong synthetic samples making the learning task harder.

Chapter 3

The Proposed MWMOTE Technique

Motivated by the problems stated in previous chapter, we have devised a novel minority oversampling technique, Majority Weighted Minority Oversampling Technique, i.e., MWMOTE. In this chapter, we describe the details of MWMOTE algorithm. The objective of MWMOTE is twofold, to improve the minority selection and weighting mechanism, and to improve the data generation mechanism. MWMOTE technique employs a minority selection mechanism by which important minority samples in the learning are identified first. The selected minority samples are then given a weight distribution depending on the importance of the respective minority samples. MWMOTE then generate synthetic samples from these minorities using their weight distribution. MWMOTE uses a different data generation mechanism than existing techniques based on unsupervised clustering for avoiding wrong and noisy synthetic sample generation.

3.1 MWMOTE Technique

MWMOTE technique works in three key phases. In the first phase, MWMOTE identifies the most important and hard-to-learn minorities samples and combine them to form a set, S_{imin}. In the second phase, members of S_{imin} are assigned a selection weight, S_w, according to its importance in the data set. In the third phase of the algorithm, MWMOTE

creates synthetic minority samples from members of S_{imin} using the weight distribution found in the second phase. The complete MWMOTE algorithm is presented in [Algorithm 1]. Steps 1 to 3 comprises the first phase of MWMOTE where S_{imin} is constructed. Step 4 weights the members of S_{imin} to implement the second phase. Steps 5 to 9 completes the third phase of MWMOTE where synthetic samples are generated using samples from S_{imin} and weight distribution found in the second phase. The generated samples are added to the original minority set, S_{min} to form the output set, S_{omin}. The algorithm returns the final oversampled minority set, S_{omin}. The key components of this algorithm are described below.

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

Input:

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

Procedure Begin

1. Compute filtered minority set, S_{minf} as
   $S_{minf} = S_{min} - \{x \in S_{min} : k1\text{-nearest neighbors}$
   of x contains no minority$\}$

2. Find the borderline majority set, S_{bmaj} as

   $S_{bmaj} = \{y \in S_{maj} : y$ is in $k2$-nearest majorities
   of some $x \in S_{minf}$ $\}$

3. Find the informative minority set, S_{imin} as

   $S_{imin} = \{y \in S_{minf} : y$ is in $k3$-nearest minorities of some $x \in S_{bmaj}$ $\}$

4. Compute selection weight $S_w(p)$ for each member p of S_{imin}.

5. Convert $S_w(p)$ into selection probability $S_p(p)$ according to:

   $S_p(p) = S_w(p) / \sum_{p \in S_{imin}} S_w(p)$

6. Find the clusters of S_{min}.

7. Initialize set, $S_{omin} = S_{min}$.

8. Do for $j = 1 \ldots N$.

   (a) Select a minority sample x from S_{imin} according to probability distribution $\{S_p(p)\}$.

   (b) Select another minority sample y, at random, from the members of x's cluster as found in step 6.

   (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 synthetic sample, s to the set, S_{omin}.

9. End Loop

End

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

3.1.1 Construction of the Set S_{imin}

It is natural that synthetic samples need not to be generated from all minority samples in a data set. There are some samples that are usually situated far from the decision

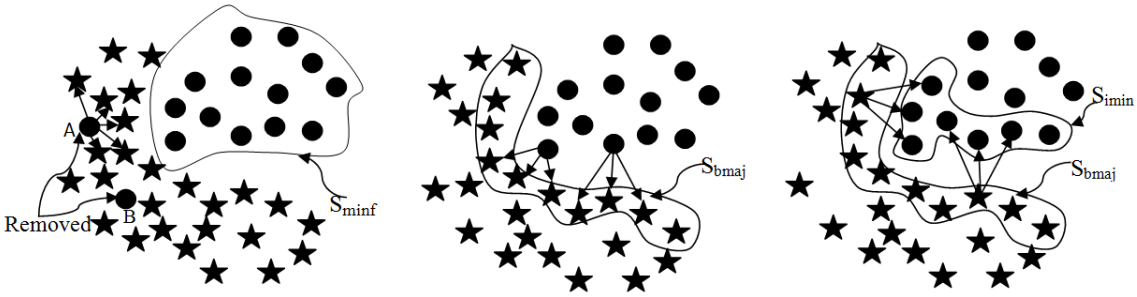
boundary and therefore synthetic samples generated from them will not add any new useful information. So, it is necessary to identify a subset of the minority samples which are important for synthetic sample generation. These samples are usually those that are located near the decision boundary and belong to small-sized clusters of the minority concept. However, most of the existing oversampling methods (e.g., [15, 17, 18]) do not explicitly construct such a subset of minority samples. Although, Borderline-SMOTE [16] method tries to find such a subset consisting of borderline minority samples, we have shown that the method fails to identify all border line samples correctly (chapter 2). In MWMOTE, we, therefore adopt a new approach for identifying these minority samples and use them to form a set, S_{imin}. The whole process of constructing S_{imin} (Steps 1–3 of [Algorithm 1]) can be described as follows.

1. Our method first filters the original minority set S_{min} to find a filtered minority set S_{minf}. This is done by removing those minority samples from S_{min} whose $k1$ nearest neighbors include no minority sample (Step 1 of [Algorithm 1]). These minority samples are expected to be noisy and removed to avoid synthetic sample generation from them in the later stage. Figure 3.1(a) shows this step where $k1 = 5$. It is clear from this figure that all 5 nearest neighbors of the minority sample A (shown by arrow) are majority samples. Hence, sample A is removed from S_{min}. Similarly, sample B is also removed and rest of the samples form the set, S_{minf}.

2. For all members of S_{minf}, MWMOTE finds their $k2$ nearest majority neighbors. These majorities are border-line majorities because they are expected to be located near the decision boundary when the value of $k2$ is small. The set of these majorities will be called border-line majority set, S_{bmaj} (Step 2 of [Algorithm 1]). The value of $k2$ does not need to be large in MWMOTE. In Fig. 3.1(b), we show the construction of S_{bmaj} for $k2 = 3$. In the figure, $k2$ (=3) nearest majority samples for some minorities are explicitly shown by arrow.

3. For all members of S_{bmaj}, MWMOTE finds their $k3$ nearest minority neighbors. These minority neighbors form the informative minority set, S_{imin} (Step 3 of [Algo-

rithm 1]). The value of $k3$ needs to be large enough to include all necessary minority samples required for synthetic sample generation. A large value of $k3$ ensures a sufficient number of minority samples that may add important information to the data set. Figure 3.1(c) shows this step for $k3 = 3$. In the figure, $k3$ (=3) nearest minority neighbors of some majority samples are explicitly shown by arrow.



(a) (b) (c)

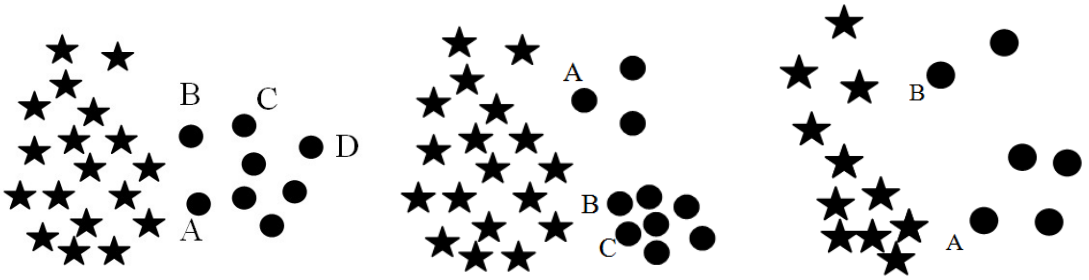
Figure 3.1: Construction of the set S_{imin}: (a) S_{minf} is found after removing noisy minorities such as samples A and B. (b) Set of borderline majority samples, S_{bmaj} is found. (c) Informative minority set, S_{imin} is found.

3.1.2 Finding Weights for Members of S_{imin}

In previous section, we showed how MWMOTE constructs a subset of minority samples, S_{imin}, that will be used for synthetic sample generation. However, all samples of this subset may not be equally important. Some samples may result in more useful information in the data set than other samples. So, it is necessary that minority samples are weighted according to their importance. A higher weight implies that the sample requires higher number of synthetic samples to be generated nearby due to its insufficiency of information in the minority concept.

Some of the existing oversampling methods (e.g., [17, 18]) use different approaches to achieve minority weighting. However, it is clear from chapter 2 that the weighting mechanism used by these methods are inappropriate and insufficient in many scenarios.

So, our MWMOTE uses a new mechanism for properly determining such weights. In step 4 of [Algorithm 1], the weight is calculated for each minority sample $p \in S_{imin}$, and it is called selection weight, $S_w(p)$. MWMOTE computes this weight based on the following three important observations.



(a) (b) (c)

Figure 3.2: Three observations for assessing the importance of a minority sample in learning: (a) Samples closer the decision boundary, e.g., A, are more important than samples that are far from the boundary, e.g., D. (b) Samples that reside in sparse cluster, e.g., A, are more important than samples that reside in dense cluster, e.g., B. (c) Samples that have many majority neighbors nearby, e.g., A, are more important than samples that have few, e.g., B.

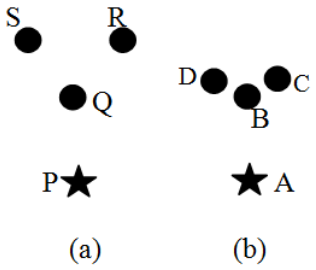


Figure 3.3: Computation of density factor: (a) Minority sample Q is in a sparse cluster. (b) Minority sample B is in a dense cluster.

Observation 1: Samples close to the decision boundary contain more information than those far from the boundary. This observation indicates that closer samples should be given higher weight than further samples. Consider Fig. 3.2(a) in

which samples A and B are closer to decision boundary compared to samples C and D. Therefore, A and B are more informative than C and D. Similarly, C is more informative than D. This signifies that A and B should be given higher selection weight than C and D. At the same time, C should be given higher selection weight than D.

Observation 2: Minority samples in sparse clusters are more important than those in dense clusters. From the perspective of synthetic sample generation, members of sparse clusters are more important than those of dense clusters. This is due to the fact that a dense cluster is already represented and does not need any new information. However, sparse clusters may require new synthetic sample to increase its size and to reduce within-class imbalance. This is why members of sparse clusters need to be given higher selection weight than those of dense clusters. Consider Fig. 3.2(b), where the sample A is more important than B and C though all of them are equally distant from the decision boundary. This is natural because A is a member of a sparse cluster, while B and C are members of a dense cluster.

Observation 3: Minority samples near dense majority clusters are more important than those near sparse majority clusters. Compare samples A and B shown in Fig. 3.2(c)). Both of these samples are equally distant from the decision boundary and are members of equal-sized clusters. However, the density of majority neighbors near to A is higher than that to B. This relative imbalance will make difficulty for a classifier to learn A correctly. Because of this A is more important for synthetic sample generation than B, and the selection weight of A needs to be given higher.

Considering all these, the selection weight S_w needs to be computed in such a way that it facilitates the aforementioned three observations. MWMOTE computes S_w using the majority set S_{bmaj} as follows.

1. Each majority sample $x \in S_{bmaj}$ gives a weight to each minority sample $p \in S_{imin}$. This weight is called information weight, $I_w(x, p)$.

2. For a minority sample p, all information weights, $I_w(x, p)$ are summed up to find its

selection weight, $S_w(p)$:

$$S_w(p) = \sum_{x \in S_{bmaj}} I_w(x, p) \tag{3.1}$$

In MWMOTE, $I_w(x, p)$ is computed as the product of two factors: closeness factor, $C_f(x, p)$ and density factor, $D_f(x, p)$:

$$I_w(x, p) = C_f(x, p) \times D_f(x, p). \tag{3.2}$$

The closeness factor facilitates our observation 1, giving higher weight to closer samples than further samples. The density factor facilitates observation 2, giving higher weight to members of sparse clusters than members of dense clusters. Finally, the summing up in (3.1) facilitates our observation 3, that is, minority sample having higher number of majority neighbors will get higher selection weight. Below, we describe how MWMOTE computes the two factors.

Closeness factor, $C_f(x, p)$: The computation of closeness factor is very straight forward. If p is not any of the $k3$ nearest neighbors of x, then $C_f(x, p)$ is 0, otherwise $C_f(x, p)$ is computed as follows.

1. Find the Euclidean distance from the majority sample x to the minority sample p and divide the Euclidean distance by the dimension of the feature space, call it $d_n(x, p)$.

2. Find $C_f(x, p)$ according to the following equation.

$$C_f(x, p) = \frac{f(\frac{1}{d_n(x,p)})}{C_f(th)} * C_f(max) \tag{3.3}$$

Where, $C_f(th)$ and $C_f(max)$ are user defined parameters and f is a cut-off function. In (3.3), $1/d_n(x, p)$ is first applied to f. The application of f is for ignoring values that are too high and slicing them to the highest value $C_f(th)$. The value thus found is rescaled to a range $[0, C_f(max)]$. We define f in the following way

$$f(x) = \begin{cases} x & \text{if } x \leq C_f(th), \\ C_f(th) & \text{otherwise.} \end{cases} \tag{3.4}$$

Density factor, $D_f(x, p)$: The density factor facilitates observation 2. It should be higher for members of sparse clusters and lower for members of dense clusters, given that they are equally distant from the decision boundary. However, at the same time, it should not violate observation 1. Our MWMOTE computes $D_f(x, p)$ by normalizing $C_f(x, p)$. This can be expressed as

$$D_f(x, p) = \frac{C_f(x, p)}{\sum_{j \in S_{imin}} C_f(x, j)} \tag{3.5}$$

To show why (3.5) gives higher weights to members of sparse clusters, consider Fig. 3.3. Assume that, minority samples B and Q are equally distant from majority samples A and P respectively (Figs. 3.3(a) and 3.3(b)). Hence, their Euclidean distances are equal and closeness factors will also be equal, i.e., $C_f(A, B) = C_f(P, Q)$. However, from the figures, it is visually apparent that, $C_f(A, C) = C_f(A, D) > C_f(P, R) = C_f(P, S)$. Hence, if we compute $D_f(A, B)$ and $D_f(P, Q)$ from these two scenarios, the denominator of (3.5) will be larger for $D_f(A, B)$ than for $D_f(P, Q)$. So, $D_f(P, Q)$ will be greater than $D_f(A, B)$. This means that, even though closeness factors for both of B and Q are same, Q gets larger density factor than B due to its relative position in a sparse cluster.

3.1.3 Synthetic Data Generation

In chapter 2, 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 sample generation mechanism based on unsupervised clustering (Steps 6–8 of [Algorithm 1]). First, in Step 6, MWMOTE finds the clusters of minority data set, S_{min} using an unsupervised clustering algorithm (discussed later). Then, in Step 7, oversampled minority data set, S_{omin} is initialized with input minority set, S_{min}. Finally, in Step 8, synthetic minority samples are generated and added to S_{omin} using the same linear interpolation technique (2.3) used by existing techniques. However, in Step 8(b), MWMOTE differs with the k-NN based approach in the way how y of (2.3) is chosen for each sample, x. Rather than choosing y at random from the k nearest neighbors of x (as used in the k-NN based approach), MWMOTE selects y as one from the members

of x's cluster (as found in Step 6 of [Algorithm 1]). The intuition is that if y is selected from the cluster of x, then synthetic samples that will be generated from x and y, according to (2.3), will also lie inside the same cluster whose member is x. So, the generated samples will rarely fall in majority regions (the k-NN based approach suffers from this problem). Another advantage of cluster-based approach is where synthetic samples are generated from noisy minority samples. If x is a noisy sample, then it is likely that, x will form an isolated cluster consisting of only one member (itself) during the clustering process. In this scenario, y to be selected (2.3) by our cluster-based approach will be the same as x and Eqn. 2.3 will then generate a duplicate sample (of x). This is much better than the k-NN based approach, which may create more noisy minority samples and broaden minority regions. An erroneously broadened minority region is likely to overlap with majority regions creating difficulty for a classifier.

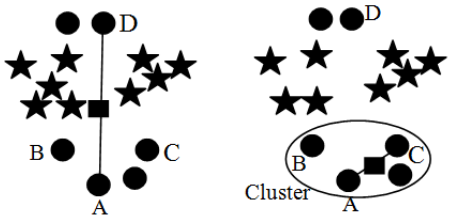


Figure 3.4: Difference of k-NN and proposed cluster based data generation from a minority sample: (a) Using k-nearest neighbor based approach (for $k = 5$) generates a synthetic sample (shown by square) which falls in the majority region. (b) MWMOTE's cluster based approach generates a sample which remains inside a minority cluster.

Consider Figs. 3.4-3.6 to show why our cluster based sample generation is better than the k-NN based sample generation. To generate a synthetic sample from the minority sample$A$, the k-NN based approach randomly selects a sample, say D, from A's 5-nearest neighbors, assuming $k = 5$ (Fig. 3.4(a)). The selection of D may lead to generate a synthetic sample (shown by square in Fig. 3.4(a)), that lies inside the majority regions. However, our cluster-based approach selects one from the members of the cluster of A. It ensures that D will not be selected, since it will not likely be a member of A's cluster

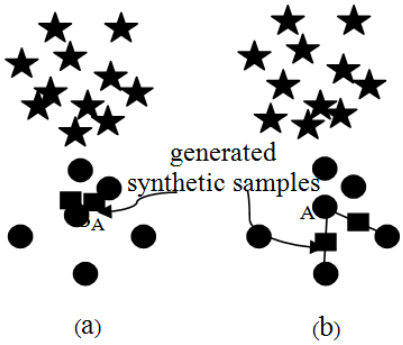


Figure 3.5: Difference of k-NN and proposed cluster based data generation from members of dense minority cluster e.g. A: (a) k-nearest neighbor based approach generates nearly duplicated samples from A. (b) Cluster based approach generates more informative samples from A.

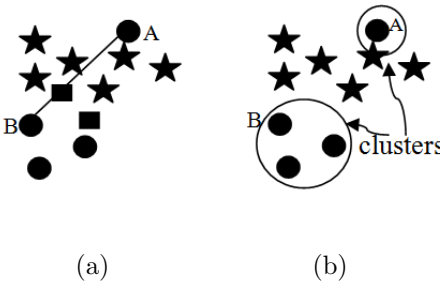


Figure 3.6: Difference of k-NN and proposed cluster based data generation from a noisy sample A. (a) k-NN based approach generates more wrong samples (shown by square) from A. (b) Cluster based approach generates a duplication of A.

(Fig. 3.4(b)). Our approach will select either B or C (considering that B and C forms cluster with A). In this condition, the generated synthetic sample will definitely lie inside A's cluster (Fig. 3.4(b)). It can be seen that synthetic samples generated from A via k-NN (assuming $k = 3$) approach are nearly duplicated samples (Fig. 3.5(a)), while they are more informative when generated via our cluster-based approach (Fig. 3.5(b)).

Consider Fig. 3.6 to see what happens when synthetic sample are generated from a noisy sample, say A. There is every possibility that the k-NN based approach may generate a noisy sample (shown by square) from A (Fig. 3.6(a)). 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 A (Fig. 3.6(b)). This is much better than the k-NN approach because, for noisy samples, our cluster-based approach does not generate any new noisy sample, while the k-NN based approach does it, thus, broadening minority regions erroneously.

3.1.4 Clustering S_{min}

The success of MWMOTE will largely depend on how we cluster the set, S_{min} in Step 6 of [Algorithm 1]. For this purpose, MWMOTE uses average-linkage agglomerative clustering, a hierarchical clustering algorithm [40, 41]. Agglomerative clustering does not require the number of clusters beforehand. The algorithm generates clusters in a bottom-up fashion. The key steps of this algorithm are given below (assume, M data samples are given as input):

1. Initially, each data sample is assigned to a separate cluster. So, initially there are M clusters, each of size one.

2. Find the two closest clusters cluster $L1$ and cluster $L2$.

3. Merge cluster $L1$ and cluster $L2$ into a single cluster, cluster $L3$. This merging reduces number of clusters by one.

4. Update distance measures between the newly computed cluster and all previous clusters.

5. Repeat step 2-4 until all data samples are merged into a single cluster of size M.

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

1. Initially, each item is assigned to a cluster. So, initially there are M clusters, each of size one.

2. Find the two closest clusters, cluster $L1$ and cluster $L2$.

3. If distance measure between cluster $L1$ and cluster $L2$ exceeds T_h, stop clustering.

4. Merge cluster $L1$ and cluster $L2$ into a single cluster, cluster $L3$. This merging reduces number of clusters by one.

5. Update distance measures between the new computed cluster, cluster $L3$ and all other clusters.

6. Repeat step 2 to 5 until there is one cluster of size M, remaining.

What should be value of T_h? Clearly, this value should not be constant, because, the distance measure varies with dimension of the feature space. So, the same algorithm will produce different number of clusters for the same types of data sets, where the only difference is in feature space dimension. The second problem of using a constant value for T_h lies in the fact that in some data sets samples are relatively sparse (average distance between samples is high), whereas in some other data sets, samples are relatively dense

(average distance between samples is low). So, using a constant T_h, will produce, fewer number of clusters for data sets where average distance between samples is low and larger number of clusters for data sets where average distance between samples is high. So, the intuition is that, the value of T_h should be data set dependent. It should be calculated using some heuristics of the distance measures between samples of the data set. For this purpose, we first find a value d_{avg} as follows:

$$d_{avg} = \frac{1}{|S_{minf}|} \sum_{x \in S_{minf}} \min_{y \neq x, y \in S_{minf}} \{dist(x, y)\} \tag{3.6}$$

We then compute T_h by multiplying d_{avg} with a constant parameter, C_p:

$$T_h = d_{avg} \times C_p \tag{3.7}$$

For each member of S_{minf} (rather than S_{min} to avoid the affect of noisy minority 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 the value of C_p, increases the sizes of clusters, thus reducing the number of clusters. On the other hand, decreasing the value of C_p, decreases the sizes of clusters, thus increasing the number of clusters.

3.2 Summary

In this chapter, we describe our proposed oversampling technique MWMOTE. We illustrate the different components of MWMOTE and explain theoretically, with suitable example scenarios, how MWMOTE solve problems faced by existing oversampling techniques. These analyses are justified with the experimental results in the next chapter.

Chapter 4

Experimental Studies

In this chapter, we evaluate the effectiveness of our proposed oversampling technique MWMOTE and compare its performance with different methods existing in literature. We have selected three other competitive oversampling algorithms SMOTE [15], ADASYN [17], and RAMO [18]. The RAMO technique is found from RAMOBoost excluding the boosting portion and keeping only the oversampling portion [18]. We evaluate the performance of these four data generation algorithms in two different categories of datasets. Firstly, we use some artificially generated data sets to assess their performance. Secondly, we evaluate the performance on some real world data sets collected from UCI machine learning repository having different degrees of imbalance. Three different sets of experiments are performed in each category which are: evaluation using a single classifier, evaluation using an ensemble technique and evaluation using other types of classifiers than the one chosen in the first and second set of experiments.

4.1 Assessment Metrics

The performance of a machine learning model is usually evaluated by the correctness of the model. This measure is called overall accuracy or accuracy. For classification learning tasks, the accuracy is the percent of correct classifications made by the classifier over the

test set examples. In case of two-class classification problem, one class is the positive class and the other class is the negative class. Let, p, and n denote the number of positive and negative examples in a testing data set and P, and N be the number of positive and negative outputs made by a classifier over this test set. Then, the classification results of the classifier can be shown as Table 4.1.

Table 4.1: Confusion matrix.

		True class		
		Pos	Neg	Row sum
Classifier output	Pos	TP	FP	P
	Column sum	p	n	

In table 4.1, the cell, denoted by TP (true positive), is the number of positive examples that are correctly classified (classifier output is positive). FP counts those examples whose actual classes were negative; however, the classifier predicted them as positive. This is called false positive count. Similarly, FN is false negative, number of positive examples that are predicted as negative, and TN (true negative) is the number of negative examples that are correctly classified (labeled as N, by the classifier). Structure similar to Table 4.1 is generally known as a confusion matrix.

The cells of the confusion matrix is filled up after evaluation the classifier with the testing set. For example, consider a medical diagnosis problem, where, a classifier, have to identify the cancerous and non-cancerous patients. The cancerous patients belong to the positive class and non-cancerous belongs to the negative class. Suppose, a testing data set contains 100 cancerous (positive) patients and 100 non-cancerous (negative) cases. Assume that, of the 100 positive examples, a learned classifier correctly classifies 90 examples, while incorrectly classifies 10 of them as non-cancerous patients (negative). Similarly, of the 100 non-cancerous patients, the classifier classifies 85 correctly, while rest 15 incorrectly. The confusion matrix can then be formed which is shown in Table 4.2.

Using the confusion matrix (Table 4.1), several performance measures can be derived.

Table 4.2: Confusion matrix for the cancer diagnosis problem.

		True class		
		Pos	Neg	Row sum
Classifier output	Pos	90	15	$P = 105$
	Neg	10	85	$N = 95$
	Column sum	$p = 100$	$n = 100$	

The first one, most common in evaluating any classifier, is the overall accuracy, which can be defined based as follows:

$$accuracy = \frac{TP + TN}{p + n} \tag{4.1}$$

Sometimes, error rate or overall error rate is used, which is simply found as:

$$errorrate = 1 - accuracy \tag{4.2}$$

Accuracy is very well known and popular performance measure in research community. However, the main problem of using accuracy as the performance measure is that, the term depends on the distribution of positive and negative examples in the test data set. To see why, let us first measure the accuracy of the classifier using the information from Table 4.2:

$$accuracy = \frac{90 + 85}{100 + 100} = \frac{175}{200} = 0.875$$

Now, consider the case, when the above testing data set contained no negative examples at all ($n = 0$ and $p = 100$). In that case, Table 4.2 would be represented as Table 4.3: Now, again, if we calculate accuracy from Table 4.3, we find:

$$accuracy = \frac{90 + 0}{100 + 0} = 0.90$$

We see from above calculations that, accuracy varies when distribution of the positive and negative examples vary in the testing data set, although classifier remains unchanged. For the same classifier, we find that, accuracy changed from 87.5% to 90%, when it is tested on a different data set. This implies that accuracy cannot measure the actual performance

Table 4.3: Confusion matrix when data set has no negative examples.

		True class		
		Pos	Neg	Row sum
Classifier output	Pos	90	0	$P = 90$
	Neg	10	0	$N = 10$
	Column sum	$p = 100$	$n = 0$	

of a classifier. In fact, any performance measure which uses information from both columns of the confusion matrix (Table 4.1 will be sensitive to class distribution in testing set. Since, an imbalanced data set has a skewed distribution of data samples among the classes, accuracy cannot be used to evaluate a classifier's effectiveness in imbalanced data sets. To make it clear, consider the case when the medical diagnosis data set explained before is imbalanced. Suppose, non-cancerous patients are rare in the world and testing data set contains 190 cancerous and only 10 non-cancerous cases. Now, if a blind classifier, which classifies all examples as non-cancerous (hence, all non-cancerous patients in the testing set will be classified correctly and all cancerous patients will be classified incorrectly), is evaluated on this testing set, the confusion matrix would be formed as Table 4.4:

Table 4.4: Confusion matrix for a classifier which classifies all examples as positive.

		True class		
		Pos	Neg	Row sum
Classifier output	Pos	190	10	$P = 200$
	Neg	0	0	$N = 0$
	Column sum	$p = 190$	$n = 10$	

From Table 4.4, we observe that, all positive examples are correctly classified (true positive is 190). This is obvious, since, the classifier, blindly says all examples positive. Due to similar reason, none of the 10 negative examples are correctly classified (TN is 0). In fact, the classifier cannot classify any negative example correctly. However, the overall

accuracy of the classifier is $\frac{190+0}{190+10} = 0.95$, which is 95%! A very high accuracy indeed! This is the reason for which accuracy is not usually used in research community as the performance measure in the imbalanced learning problems.

Some other performance measures which can be derived from the same confusion matrix (Table 4.1) and are suitable for imbalanced learning, are *precision* and *recall*. These two terms are formally defined as:

$$precision = \frac{TP}{P} \tag{4.3}$$

$$recall = \frac{TP}{p} \tag{4.4}$$

Precision is the number of true positives, divided by the number of positive outputs given by the classifier. *Precision* measures the correctness of the classifier's positive output, that is, what fraction of the positive outputs of the classifier, are actually positive. A precision of 1.0 means that, every result, given as positive, by the classifier, is correct. *Recall* is the number of true positives, divided by the total number of positive examples in the testing data set. This measures the performance of the classifier over the positive examples only. A *recall* of 1.0 means that, all positive examples are correctly classified. *Precision* is the probability that, a predicted positive output is correct while *Recall* is the probability that, a positive example is predicted correctly.

Using, either *precision* or *recall*, as a single classification measure, poses similar problems described with accuracy. *Precision* measures the correctness of the classifier's positive output. However, *precision* does not say anything about the correctness of the classifier's negative output. Similarly, *recall* finds the performance of the classifier over the positive examples, however, it does not say, anything about the classifier's performance over negative examples. Therefore, two other measures combining the *precision* and *recall* are used in research community which can give a thorough insight of the classifier's overall performance. These two metrics are known as *F-measure* and *G-mean* and are defined as:

$$F - measure = \frac{(1+\beta^2) * (precision * recall)}{\beta^2 * precision + recall} \tag{4.5}$$

$$\begin{aligned} G - mean &= \sqrt{\text{positive accuracy} \times \text{negative accuracy}} \\ &= \sqrt{\frac{TP}{p} \times \frac{TN}{n}} \end{aligned} \tag{4.6}$$

F-measure combines *precision* and *recall* and β determines the relative importance of the two. If β is 1, then *precision* and *recall* are given equal importance and *F-measure* becomes the harmonic mean of *precision* and *recall*. *G-mean* is the geometric mean of positive accuracy and negative accuracy.

Although, *F-measure* and *G-mean* are well enough for analyzing a classifier's performance, still they are sensitive to class distributions and therefore are unsuitable for comparing several classifiers. In this respect, the most and accepted measure for assessing classifiers performance and comparing various classifier models is Receiver Operating Characteristics (ROC) Curve which is described below.

4.1.1 Receiver Operating Characteristics Curve

A receiver operating characteristics curve (ROC) curve is used to visualize and compare classifiers based on their performance [35, 36]. A ROC graph (Fig. 4.1 is obtained by plotting false positive rate (FPR) on the X-axis, and true positive rate (TPR) on the Y-axis where FPR and TPR are defined as:

$$TPR = \frac{TP}{p} \tag{4.7}$$

$$FPR = \frac{FP}{n} \tag{4.8}$$

Classifiers that output only class labels (discrete classifiers), then its output on the testing data set can be used to calculate the FPR and TPR values. This will correspond to a point in ROC graph. There are several points that are worthy of noting here.

The upper left corner point (0,1), shown in Fig. 4.1 by D, corresponds to $FPR = 0$ and $TPR = 1$. This correspond to a perfect classifier. The lower right corner (1,0) corresponds to a FPR of 1 and TPR of 0, which is the worst classifier. If a point on the ROC graph is northwest than another point, then corresponding first classifier is better than the second classifier. Hence, ROC point B is a better classifier than the ROC point C. Any point on the ROC graph that lies on the diagonal line segment $y = x$, corresponds to a classifier equivalent to random guess, such as point F. This is because, a random guess classifier, which guesses the positive class half the time, is expected to classify 50% positive examples correctly (having 0.5 TPR) and 50% negative examples correctly (having 0.5 FPR). Therefore, its ROC point is (0.5,0.5) which lies on the diagonal line. Similarly, a random classifier, guessing positive class 90% cases, will correctly classifier 90% positives (0.9 TPR), however, at the same time, will misclassify 90% negatives (0.9 FPR), leading to the point (0.9,0.9) on the diagonal line of the ROC graph. Any point below the diagonal line such as point E corresponds to a classifier, whose performance is worse than a random guess classifier. However, such a classifier can be made a good classifier by inverting all of its classification decisions. In figure, point E corresponds to a bad classifier and its reflected point A corresponds to the good classifier found after inverting all decisions made by classifier E.

Some classifiers (e.g. neural network) produce continuous values such as probability estimates as its output. These continuous outputs can be converted to discrete classification decisions by selecting an appropriate threshold and labeling outputs above threshold as positive, while output below threshold as negative. An ROC point can then be calculated and plotted on the ROC graph. By varying the threshold from 0 to ∞, many such points can be calculated and plotted on the ROC graph. These points constitute a ROC curve such as curve A, and curve B in Fig. 4.1. The ROC curve is, therefore composed of a series of ROC points, found by selecting different thresholds in the range $[0, \infty]$.

One important characteristic of ROC graph is that, they are insensitive to class skew. Both of TPR and FPR do not depend on class distributions and remain unchanged when class distribution changes, because both of them uses information from a single column of

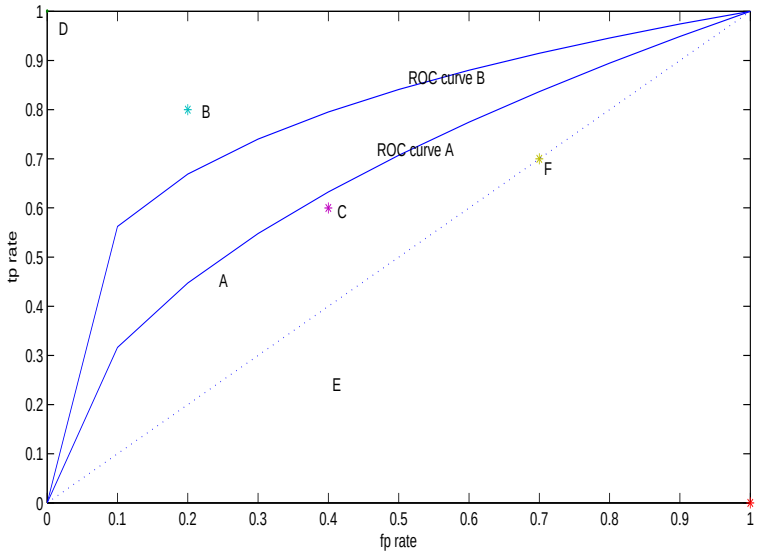


Figure 4.1: Different points on the ROC graph, e.g., B, C, and E, are shown representing different discrete output classifiers. Two ROC curves, e.g., ROC curve A, and ROC curve B, are shown representing two continuous-output classifiers.

the confusion matrix (Table 4.1). Therefore, the ROC point does not change. However, ROC graph is a two-dimensional plot and difference of classifiers performance can only be visualized from the ROC graph. To compare classifiers, as a singular assessment metric, area under the ROC curve, usually known as AUC is used. The AUC of a classifier is the expected performance of that classifier. In Fig. 4.1, ROC curve B has higher AUC then ROC curve A. However, it does not necessarily mean that, curve B is everywhere better than curve A. It is possible that, a curve, having higher AUC, can perform worse in some regions of the graph, than another, which have lower AUC.

4.1.2 Test of significance

Although values of AUC and other metrics can be used to compare several classifiers, a test of significance is usually required to determine whether the observed performance difference between two learning models is really significant to be accepted. The significance test will evaluate whether the difference in performance of two methods is statistically

significant. There are many significance tests available such as t-tests [37] and Wilcoxon-signed-rank test [38]. In this research work, we use the Wilcoxon signed-rank test. The advantage of Wilcoxon signed-rank test is that it does not assume any prior distribution of the data samples. So, it can be applied where original distribution of data samples is unknown. Moreover, outliers can be handled better by this test than t-tests [37].

Wilcoxon signed-rank test is performed as follows. Suppose, n pair of results are obtained which are being tested. At first, differences are computed for each pair. Let, the differences be d_i for pair $i = 1..n$. These differences are then sorted from smallest to largest according to their absolute value. A rank is then assigned to each difference starting from rank 1 for the smallest and ending at rank n for the largest. In case, more than one differences are equal, each difference is given an average rank. The ranks are then converted to signed-ranks by giving it the sign of the difference. All the positive ranks are then summed and called $R+$. Similarly, all the negative ranks are summed to form $R-$. The minimum of $R+$, and $R-$ are found. This value is called T. The value of T is then compared against a critical value found from a table [39] at a specific significance level (usually 0.05 significance level is widely used). If the T value is less then or equal to the critical value, then we say that, the difference is statistically significant.

4.2 Selection of Classifiers and Parameter Settings

We use single neural network and ensemble of neural networks as classifiers in our first and second set of experiments, respectively. The AdaBoost.M2 is chosen for ensemble because of its better performance for dealing with imbalance data sets [18, 31, 32]. For the third set of experiments, we use k-nearest neighbor and decision tree classifier to evaluate the effectiveness of MWMTOE using classifiers other than the neural network chosen for first and second set of experiments.

For the neural network classifier used in first and second set of experiments, the well known back propagation learning algorithm is used for training. 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 sigmoid function as an activation function. The number of training epochs is randomly set to 300 and learning constant having a learning rate of 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 $C_f(max) = 2$. All these values are chosen after some preliminary experiments and they are not meant to be optimal. For SMOTE and ADASYN the value of nearest neighbors, K, is set to 5 [15, 17]. For RAMO method, values of nearest neighbors are: $k1 = 5$ and $k2 = 10$ [18]. The scaling coefficient, α for RAMO is set to 0.3 [18]. The number of synthetic samples generated was set to 200% of the original minority samples for all the simulation runs [42]. *F-measure* is computed with $\beta = 1$. To compute the average ROC graph from the ROC graphs of multiple simulation runs, we use the vertical averaging approach [35]. Average AUC is computed by averaging the AUC values of all ROC graphs obtained from multiple simulation runs.

4.3 Simulation on Artificial Problem Domain

We have generated an artificial two-dimensional two-class problem domain motivated from [13]. The data generation process is as follows. First, we take a unit square in a two-dimensional feature plane. We then equally divide the unit square into $S \times S$ smaller squares to form a $S \times S$ squared grid, where the value of S indicates the complexity of the domain. The larger the value of S the higher the complexity of the domain. The size of each grid-cell is $\frac{1}{S} \times \frac{1}{S}$. Each of the $S \times S$ grid-cells is assigned to a different class (i.e., minority or majority), except one at the middle of the grid, which is kept blank to equally distribute both classes to the grid-cells. Thus, each grid-cell denotes either a minority or a majority region. Classes are assigned in such a way that no two adjacent cells belong to the same class. After assigning classes to grid-cells, points are sampled from each cell at random. These points form the training data set and testing data set. It is to be noted that data points in the training and testing sets are different.

We use two different complexity levels, which are $S = 3$, and $S = 5$. Figures 4.2 and 4.3 show problem domain structure and class regions for these two complexity levels. Besides two different complexity levels, we also use two different imbalance ratios to understand its effect on classifiers' performance. For each complexity level, we use imbalance ratio 1:3 (moderate) and 1:10 (high).

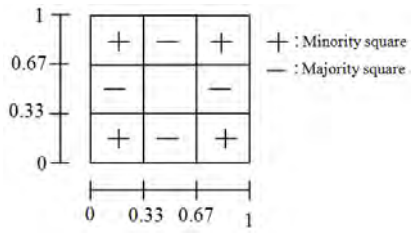


Figure 4.2: Artificial problem domain for complexity level, 3×3

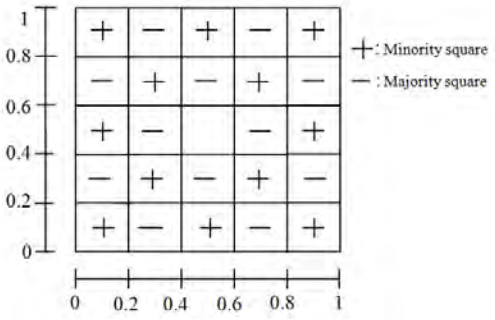


Figure 4.3: Artificial problem domain for complexity level, 5×5

4.3.1 Simulation 1

In this simulation, we perform the first and second set of experiments on real world data sets. We run single neural network classifier and AdaBoost.M2 ensemble technique with neural network as base classifier on four different artificial data sets. For AdaBoost.M2 ensemble, each dataset is first oversampled using the oversampling algorithms. Then, AdaBoost.M2 is run on this oversampled dataset with 20 boosting iterations [43]. Tables 4.5 and 4.6 summarize the results of applying SMOTE, ADASYN, RAMO, and MWMOTE on artificial data sets. Each result is the average of 20 simulation runs. The best results are highlighted in bold-face type. Figure 4.4 and 4.5 show the ROC graphs.

For single neural network classifier, Table 4.5 shows that, MWMOTE outperforms SMOTE, ADASYN, and RAMO for all problem sizes, and imbalance ratios in terms of accuracy, precision, F-measure, G-mean, and AUC. However, MWMOTE cannot outperform other methods in terms of recall. This is because, as described in chapter 2, SMOTE, ADASYN and RAMO techniques erroneously generates synthetic minority samples, which enlarges the minority region erroneously, falling inside the majority region. Therefore, neural network classifier shift the decision boundary more toward majority, which increases the number of positive outcomes from the classifier. Therefore, recall performance is increased. At the same time, due to erroneous over-generalization of minority region, SMOTE, ADASYN, and RAMO methods misclassify many majority samples as minority, decreasing the precision of the classifier. Therefore, in spite of improvement in recall, overall measures such as accuracy, F-measure, G-mean, and AUC of SMOTE, ADASYN, and RAMO methods are reduced as is evident from Table 4.5.

From Table 4.5, we see that, the recall performance of all four oversampling methods is better when imbalance ratio is 1:3 and recall reduces when imbalance ratio becomes high (1:10). This is natural, because, it becomes much harder to classify minority samples, when imbalance ratio increases. At the same time, we see that, the precision performance of all four oversampling methods increases as imbalance ratio increases. The reason behind this is that, when imbalance ratio increases, number of majority instances also increase. Therefore, decision boundary is shifted more toward the minority region resulting in decreased positive outcomes from the classifiers, which increases the precision of the classifier. Similar to precision, other performance measures such as accuracy, precision, AUC does not decrease as imbalance ratio increases. So increase in imbalance ratio may make it harder to classify minority samples, but it may not necessarily degrade overall performance of the classifier such as F-measure, G-mean, and AUC. On the other hand, as complexity level increases from 3×3 to 5×5, we see that the performance of all four oversampling methods degrades (Table 4.5).

From the ROC graphs (Fig. 4.4), we can visually determine the dominance of MWMOTE over other three methods. For all the four data sets, we see that, ROC graph of

MWMOTE is above the ROC graphs of other three methods. This is even justified by the better AUC values of MWMOTE in table 4.5, where AUC values are the average area of the respective ROC graphs.

In case of Adaboost.M2 ensemble classifier, the results in Table 4.6 show that, MWMOTE shows better performance than each of SMOTE, ADASYN, and RAMO techniques in terms of performance measures: accuracy, precision, recall, F-measure, G-mean, and AUC. Comparing the performance values of single neural network classifier and boosting ensemble from Table 4.6, we see that, boosting significantly improves performance of all four oversampling methods. The improvement is somewhat better for MWMOTE than other three methods. Interestingly, boosting has improved the recall performance of MWMOTE to beat other three techniques in three of the four problem domains. Even for the most complex problem domain considered in this experiment, which has a complexity level 5×5, with high imbalance ratio of 1:10, boosting shows improvement at a greater level. These improved results indicate that, boosting can be used as a handy tool for dealing with imbalanced data sets, even with complex sub-concepts. ROC graphs for this simulation run are shown in Fig. 4.5. From the ROC graphs, we see that, ROC graph of MWMOTE is above the ROC graphs of other three methods in most of the portion of the graph and justifies MWMOTE's superiority over other three methods.

4.3.2 Simulation 2

In this simulation we study whether performance of MWMOTE remain better when classifier is not neural network. For this purpose, we select k-nearest neighbor classifier (with $k = 5$) and C4.5 decision tree classifier [44] without pruning. We run simulation experiments SMOTE, ADASYN, RAMO, and MWMOTE using these classifiers on the same artificial data sets. The results are shown in Table 4.7 and 4.8.

Results in Tables 4.7 and 4.8 show that, MWMOTE can really outperform SMOTE, ADASYN, and RAMO techniques even if we change classifiers. For both of k-nearest neighbor and decision tree classifiers, MWMOTE shows better performance in terms of

Table 4.5: Performance of SMOTE [15], ADASYN [17], RAMO [18], and MWMOTE on four artificial data sets using single neural network classifier.

Dataset	Methods	Accuracy	Precision	Recall	F-measure	G-mean	AUC
Complexity 3×3, Imbalance ratio 1 : 3	SMOTE	0.75625	0.51958	**0.7**	0.58768	0.72963	0.78119
	ADASYN	0.74167	0.51612	0.6875	0.58102	0.71935	0.77609
	RAMO	0.75521	0.52506	0.68333	0.58129	0.71997	0.76172
	MWMOTE	**0.80937**	**0.64072**	0.6875	**0.64897**	**0.75855**	**0.8059**
Complexity 3×3, Imbalance ratio 1 : 10	SMOTE	0.88756	0.41696	0.19737	0.24839	0.40701	0.69153
	ADASYN	0.89941	0.4392	0.24537	0.30451	0.45165	0.72365
	RAMO	0.89015	0.49592	0.21875	0.27025	0.44211	0.70416
	MWMOTE	**0.91667**	**0.71592**	**0.26563**	**0.36861**	**0.50272**	**0.74614**
Complexity 5×5, Imbalance ratio 1 : 3	SMOTE	0.55174	0.2758	0.48333	0.34862	0.52101	0.51085
	ADASYN	0.5375	0.26886	0.49306	0.34312	0.50921	0.51455
	RAMO	0.52882	0.26615	**0.49583**	0.3408	0.501	0.50611
	MWMOTE	**0.56424**	**0.28124**	0.475	**0.34979**	**0.52347**	**0.52903**
Complexity 5×5, Imbalance ratio 1 : 10	SMOTE	0.75539	0.58453	**0.24116**	0.13228	**0.30405**	0.5151
	ADASYN	0.79944	0.44931	0.18275	0.117	0.29767	0.50879
	RAMO	0.79441	0.43086	0.18301	0.1254	0.32022	0.50148
	MWMOTE	**0.87926**	**0.62519**	0.10764	**0.13675**	0.29926	**0.5253**

Table 4.6: Performance of SMOTE [15], ADASYN [17], RAMO [18], and MWMOTE on four artificial data sets using AdaBoost.M2 ensemble of neural network classifiers.

Dataset	Methods	Accuracy	Precision	Recall	F-measure	G-mean	AUC
Complexity 3×3, Imbalance ratio 1 : 3	SMOTE	0.79167	0.55556	0.83333	0.66667	0.80508	0.89289
	ADASYN	0.77083	0.52381	**0.91667**	0.66667	0.81366	0.87723
	RAMO	0.70833	0.45833	**0.91667**	0.61111	0.76528	0.90082
	MWMOTE	**0.89583**	**0.73333**	**0.91667**	**0.81481**	**0.90267**	**0.95104**
Complexity 3×3, Imbalance ratio 1 : 10	SMOTE	0.90909	0.5	**0.66667**	0.57143	**0.78881**	0.8256
	ADASYN	0.90152	0.46154	0.5	0.48	0.68617	0.72245
	RAMO	0.93939	0.7	0.58333	0.63636	0.75416	0.83321
	MWMOTE	**0.95455**	**1**	0.5	**0.66667**	0.70711	**0.858**
Complexity 5×5, Imbalance ratio 1 : 3	SMOTE	0.63889	0.36667	0.61111	0.45833	0.62936	0.68578
	ADASYN	0.66667	0.375	0.5	0.42857	0.60093	0.67677
	RAMO	0.625	0.35	0.58333	0.4375	0.61048	0.64075
	MWMOTE	**0.68056**	**0.40741**	**0.61111**	**0.48889**	**0.65578**	**0.69332**
Complexity 5×5, Imbalance ratio 1 : 10	SMOTE	0.85101	0.12903	0.11111	0.1194	0.32059	0.62673
	ADASYN	0.90404	0.33333	0.055556	0.095238	0.23439	0.64314
	RAMO	0.90404	0.33333	0.055556	0.095238	0.23439	0.60508
	MWMOTE	**0.91667**	**0.61538**	**0.22222**	**0.32653**	**0.46812**	**0.71013**

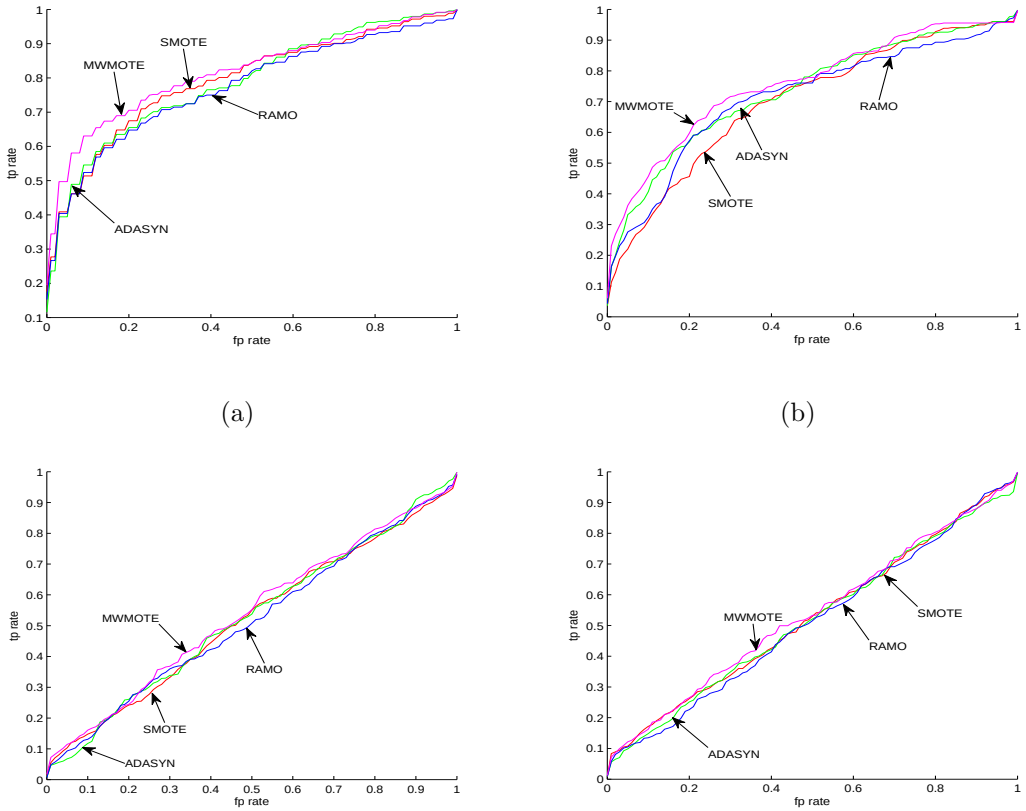


Figure 4.4: Averaged ROC curves of SMOTE [15], ADASYN [17], RAMO [18], and MWMOTE for single neural network classifier on four artificial data sets: (a) Complexity Level of 3×3 and imbalance ratio $1 : 3$. (b) Complexity level of 3×3 and imbalance ratio $1 : 10$. (c) Complexity level of 5×5 and imbalance ratio $1 : 3$. (d) Complexity level of 5×5 and imbalance ratio $1 : 10$.

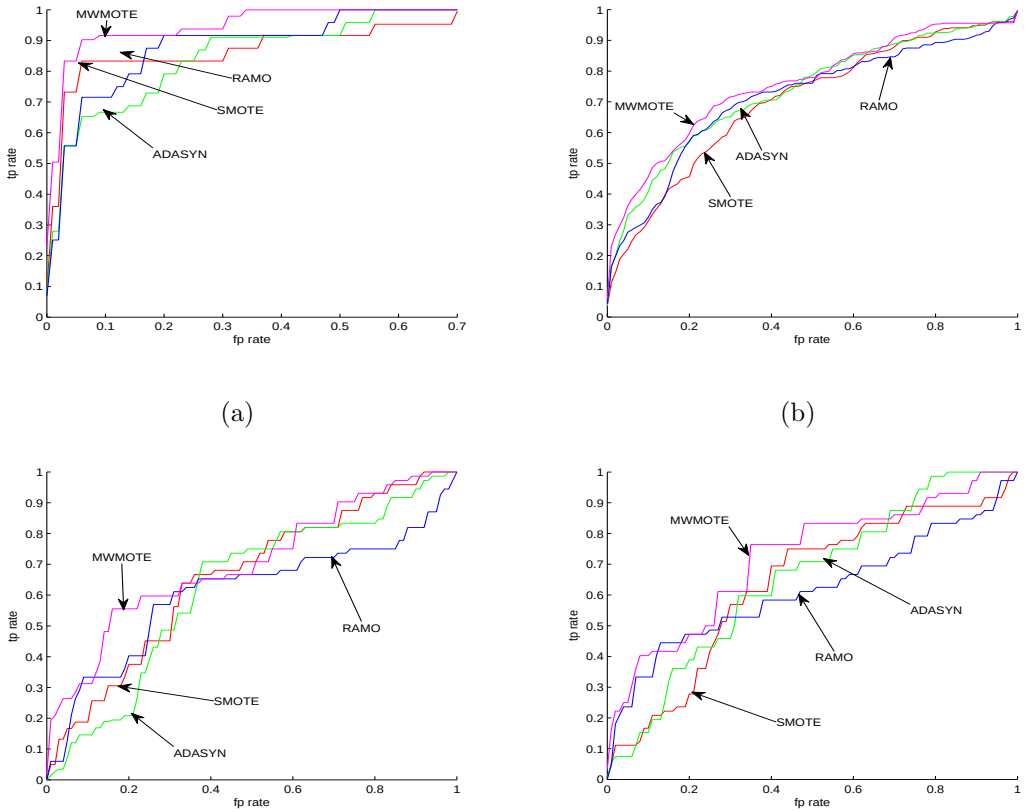


Figure 4.5: Averaged ROC curves of SMOTE [15], ADASYN [17], RAMO [18], and MWMOTE for Adaboost.M2 ensemble of neural network classifiers on artificial data sets: (a) Complexity Level of 3×3 and imbalance ratio $1 : 3$. (b) Complexity level of 3×3 and imbalance ratio $1 : 10$. (c) Complexity level of 5×5 and imbalance ratio $1 : 3$. (d) Complexity level of 5×5 and imbalance ratio $1 : 10$.

Table 4.7: Performance of SMOTE [15], ADASYN [17], RAMO [18], and MWMOTE on four artificial data sets using k-NN classifier (with $k = 5$).

Dataset	Methods	Accuracy	Precision	Recall	F-measure	G-mean
Complexity 3×3, Imbalance ratio $1:3$	SMOTE	0.78021	0.53694	0.9375	0.68141	0.82484
	ADASYN	0.77292	0.52757	0.94583	0.67619	0.8215
	RAMO	0.77187	0.52983	0.93333	0.67361	0.81697
	MWMOTE	**0.81458**	**0.58023**	**0.9625**	**0.72226**	**0.85705**
Complexity 3×3, Imbalance ratio $1:10$	SMOTE	0.92045	0.5955	0.39167	0.46922	0.61217
	ADASYN	0.92614	0.65937	0.40417	0.49809	0.62524
	RAMO	0.92765	0.66714	0.41667	0.5088	0.63438
	MWMOTE	**0.93712**	**0.7625**	**0.44167**	**0.55654**	**0.65608**
Complexity 5×5, Imbalance ratio $1:3$	SMOTE	0.76563	0.52394	0.71389	0.60383	0.74725
	ADASYN	0.76319	0.51994	0.72222	0.60401	0.74859
	RAMO	0.77083	0.53216	0.71944	0.61127	0.75265
	MWMOTE	**0.79306**	**0.56732**	**0.72778**	**0.6372**	**0.76971**
Complexity 5×5, Imbalance ratio $1:10$	SMOTE	0.94697	0.75485	0.62778	0.68256	0.78308
	ADASYN	0.94785	0.75038	0.64444	0.69119	0.793
	RAMO	0.94571	0.7413	0.62778	0.67812	0.78258
	MWMOTE	**0.95518**	**0.79581**	**0.68333**	**0.73482**	**0.81905**

Table 4.8: Performance of SMOTE [15], ADASYN [17], RAMO [18], and MWMOTE on four artificial data sets using decision tree classifier.

Dataset	Methods	Accuracy	Precision	Recall	F-measure	G-mean
Complexity 3×3, Imbalance ratio $1:3$	SMOTE	0.77292	0.53748	**0.825**	0.64576	0.78548
	ADASYN	0.74896	0.50377	0.78333	0.60753	0.75389
	RAMO	0.77396	0.55281	**0.7875**	0.64041	0.7734
	MWMOTE	**0.88646**	**0.82354**	0.70417	**0.74831**	**0.81072**
Complexity 3×3, Imbalance ratio $1:10$	SMOTE	0.89924	0.48443	0.64583	0.54382	0.76611
	ADASYN	0.91742	0.55534	0.6625	0.59768	0.78587
	RAMO	0.91894	0.55049	**0.75833**	0.6337	**0.84**
	MWMOTE	**0.9572**	**1**	0.52917	**0.68281**	0.72137
Complexity 5×5, Imbalance ratio $1:3$	SMOTE	0.68576	0.41885	0.62083	0.49893	0.66119
	ADASYN	0.69826	0.4363	**0.63472**	0.51508	0.67413
	RAMO	0.68889	0.42279	0.62778	0.50421	0.66597
	MWMOTE	**0.76042**	**0.52185**	0.56389	**0.54039**	**0.68036**
Complexity 5×5, Imbalance ratio $1:10$	SMOTE	0.88977	0.43136	**0.63472**	0.5127	0.76058
	ADASYN	0.88977	0.43053	0.62639	0.50839	0.75589
	RAMO	0.88838	0.42899	0.61389	0.5033	0.7486
	MWMOTE	**0.93283**	**0.64085**	0.61806	**0.62727**	**0.77129**

accuracy, precision, F-measure, and G-mean. As explained before, erroneous data samples generated by SMOTE, ADASYN, and RAMO techniques enlarges minority region, leading to better recall values than MWMOTE. This is justified from the recall values of single neural network classifier (Table 4.5), AdaBoost.M2 ensemble classifier (Table 4.6) and decision tree classifier (Table 4.8). However, recall values of k-nearest neighbor classifier (Table 4.7) show that, the same thing does not happen for k-nearest neighbor classifier, in case of which, MWMOTE beats other three in terms of recall also. The reason is that, k-nearest neighbor is an instance based classifier, and it does not construct any explicit decision boundary. Therefore, effect of erroneous synthetic data samples does not necessarily enlarge decision region of minority class.

4.4 Simulation on Real Problem Domain

In this subsection, we evaluate the performance of MWMOTE on real world datasets. For this purpose, we use 17 datasets collected from UCI machine learning repository [45]. The data sets were chosen in such a way that, they contained a varied level of imbalanced distribution of instances. Some of these original data sets were multi-class data. Since, we are only interested in two-class classification problem, these data sets were transformed to form two-class data sets. Table 4.9 shows the majority and minority class found after these modifications. Table 4.10 shows the detailed characteristics of these data sets.

4.4.1 Simulation 1

In this simulation, we perform the first and second set of experiments on real world data sets. We run the single neural network classifier and AdaBoost.M2 ensemble of neural network classifiers on the seventeen datasets shown in Table 4.10. For AdaBoost.M2 ensemble, 20 boosting iterations are performed on the oversampled data sets. The results of the four oversampling techniques are summarized in Tables 4.11-4.14. Results are found after a 10-fold cross-validation on each dataset. Best results are highlighted in bold-face

Table 4.9: Characteristics of Real World Datasets. All the datasets were obtained from UCI machine learning repository [45].

SL.	Dataset	Features	Instances	Minority	Majority	%Minority	Imbalance Ratio
1	Abalone	7	731	42	689	6%	0.06:0.94
2	Libra	90	360	72	288	20%	0.2:0.8
3	Yeast	8	1484	304	1180	21%	0.2:0.8
4	Robot	24	5456	1154	4302	22%	0.21:0.79
5	Ecoli	7	336	77	259	23%	0.23:0.77
6	Glass	9	214	51	163	24%	0.24:0.76
7	Vehicle	18	940	219	721	24%	0.23:0.77
8	Wine	13	178	48	130	27%	0.27:0.73
9	Texture	40	5477	1500	3977	28%	0.27:0.73
10	Phoneme	5	5227	1520	3707	30%	0.29:0.71
11	Satimage	36	6435	2036	4399	32%	0.32:0.68
12	Breast-tissue	9	106	36	70	34%	0.34:0.66
13	Breast-cancer-original	9	683	239	444	35%	0.35:0.65
14	Pima	8	768	268	500	35%	0.35:0.65
15	Ionosphere	34	351	126	225	36%	0.36:0.64
16	Breast-cancer-diagnostic	30	569	212	357	38%	0.37:0.63
17	Spambase	57	4601	1813	2788	40%	0.39:0.61

Table 4.10: Description of Minority and Majority class in datasets.

Dataset	Minority Class	Majority Class
Abalone	Class of 18	Class of 9
Libra	Class of '1', '2', '3'	All other classes
Yeast	Class of ME3', 'ME2','EXC','VAC','POX','ERL'	All other classes
Robot	Class of 'slight-left-trun','slight-right-turn'	All other classes
Ecoli	Class of 'im'	All other classes
Glass	Class of '5','6','7'	All other classes
Vehicle	Class of '1'	All other classes
Wine	Class of '3'	All other classes
Texture	Class of '2','3','4'	All other classes
Phoneme	Class of '1'	All other classes
Satimage	Class of '2','4','5'	All other classes
Breast-tissue	Class of 'CAR', 'FAD'	All other classes
Breast-cancer-original	Class of 'Malignant'	Class of 'Benign'
Pima	Class of '1'	Class of '0'
Ionosphere	Class of 'Bad'	Class of 'Good'
Breast-cancer-diagnostic	Class of 'Malignant'	Class of 'Benign'
Spambase	Class of 'spam'	Class of 'Not spam'

type.

For single neural network classifier, from Tables 4.11 and 4.12, we find that, similar to artificial data sets, MWMOTE shows comparatively better performance than SMOTE, ADASYN, and RAMO techniques in terms of accuracy, precision, F-measure, G-mean, and AUC. Due to reasons explained before, MWMOTE fails to perform better in terms of recall values. Some representative ROC graphs are shown in Fig. 4.6 for this simulation run. The graphs clearly show that, ROC graph of MWMOTE is better (above) than the ROC graphs of other three methods.

To test whether MWMOTE can statistically outperform other three techniques, we apply the Wilcoxon signed-rank test to compare the AUC values of Tables 4.11 and 4.12. Averaged AUC performance values are used for the significance test following the suggestions in [46]. AUC values of MWMOTE is compared in a pairwise manner against each of SMOTE, ADASYN, and RAMO techniques. The test results are summarized in Tables 4.15, 4.16, and 4.17. For MWMOTE vs SMOTE, we see that, in 5 datasets, the

Table 4.11: Performance of SMOTE [15], ADASYN [17], RAMO [18], and MWMOTE using single neural network classifier on real world data sets: Abalone, Breast-cancer-diagnostic, Breast-cancer-original, Breast-tissue, Ecoli, Glass, Ionosphere, and Libra.

Dataset	Methods	Accuracy	Precision	Recall	F-measure	G-mean	AUC
Abalone	SMOTE	0.9398	0.53786	**0.42**	**0.44167**	**0.62356**	0.87288
	ADASYN	0.93177	0.35926	0.26667	0.29806	0.44381	0.85144
	RAMO	**0.94233**	**0.6119**	0.41111	0.43962	0.58385	**0.89149**
	MWMOTE	0.94082	0.53095	0.36111	0.39497	0.51451	0.87745
Breast-cancer-diagnostic	SMOTE	0.97014	0.96026	0.96277	0.96031	0.96835	0.98022
	ADASYN	0.96491	0.943	0.96753	0.95399	0.96519	0.98002
	RAMO	0.9263	0.86004	0.96277	0.90743	0.93294	0.97546
	MWMOTE	**0.97713**	**0.96795**	**0.97186**	**0.96919**	**0.97584**	**0.98086**
Breast-cancer-original	SMOTE	0.97069	0.93375	0.9875	0.95949	0.97438	**0.97692**
	ADASYN	**0.97216**	**0.94082**	0.98315	**0.96119**	0.97454	0.97499
	RAMO	0.9692	0.92316	**0.99583**	0.95787	**0.97509**	0.97532
	MWMOTE	0.97067	0.93359	0.98732	0.9594	0.97432	0.97629
Breast-tissue	SMOTE	0.69091	0.53611	0.94167	0.67953	0.7156	0.84299
	ADASYN	0.70909	0.55944	0.95	0.6972	0.73618	**0.8561**
	RAMO	0.68818	0.53623	0.96667	0.68376	0.71236	0.79948
	MWMOTE	**0.74727**	**0.58706**	**1**	**0.73604**	**0.77634**	0.84189
Ecoli	SMOTE	**0.87834**	**0.69084**	0.86964	**0.76277**	0.87041	**0.94622**
	ADASYN	0.86638	0.66685	**0.89643**	0.75505	**0.87213**	0.94222
	RAMO	0.85416	0.63947	**0.89643**	0.7384	0.86449	0.93304
	MWMOTE	0.8605	0.65582	0.87143	0.73988	0.85931	0.93705
Glass	SMOTE	0.92466	0.85405	0.82	0.82579	0.87774	0.93227
	ADASYN	0.93459	0.85893	0.92	0.87148	0.92462	0.95363
	RAMO	0.90624	0.75952	0.92	0.82138	0.90413	0.94235
	MWMOTE	**0.93894**	0.84167	**0.94**	**0.87968**	**0.93611**	**0.95831**
Ionosphere	SMOTE	0.86007	0.8061	0.81026	0.80542	0.8469	0.90296
	ADASYN	0.87722	0.8195	**0.84808**	0.83205	0.8696	**0.92885**
	RAMO	0.86015	0.80753	0.80962	0.80597	0.84696	0.88042
	MWMOTE	**0.88866**	**0.86549**	0.82564	**0.83823**	**0.87052**	0.91353
Libra	SMOTE	0.9442	0.88631	0.81786	0.84458	0.88706	0.91719
	ADASYN	0.95539	0.91548	0.85893	0.87407	0.91104	0.89716
	RAMO	0.953	**0.94643**	0.80536	0.86324	0.88728	0.91022
	MWMOTE	**0.9638**	0.91006	**0.93214**	**0.91568**	**0.95059**	**0.96955**

Table 4.12: Performance of SMOTE [15], ADASYN [17], RAMO [18], and MWMOTE using single neural network classifier on real world datasets: Phoneme, Pima, Robot, Satimage, Vehicle, Wine, Yeast, Spambase, and texture.

Dataset	Methods	Accuracy	Precision	Recall	F-measure	G-mean	AUC
Phoneme	SMOTE	0.72221	0.51453	0.875	0.64755	0.75935	0.82397
	ADASYN	0.70422	0.49617	0.86316	0.62964	0.74213	0.81273
	RAMO	0.67973	0.47512	**0.92434**	0.62717	0.73113	**0.82494**
	MWMOTE	**0.73294**	**0.52747**	0.86974	**0.65563**	**0.76618**	0.82019
Pima	SMOTE	0.711	0.55863	0.83618	0.66811	0.73159	0.81443
	ADASYN	0.71873	0.56578	0.86567	0.68335	0.74335	0.82319
	RAMO	0.67973	0.5282	**0.88832**	0.66074	0.70707	0.82157
	MWMOTE	**0.72271**	**0.57508**	0.83989	**0.67917**	**0.74088**	**0.83026**
Robot	SMOTE	0.80187	0.52214	**0.81271**	0.63498	**0.80532**	**0.86939**
	ADASYN	0.78793	0.50103	0.80676	0.61757	0.79437	0.85526
	RAMO	0.71992	0.41884	0.77982	0.54393	0.73985	0.80044
	MWMOTE	**0.80882**	**0.53573**	0.79097	**0.63647**	0.80073	0.86586
Satimage	SMOTE	0.80948	0.66244	0.8168	0.73049	0.81077	0.89643
	ADASYN	0.79487	0.64123	0.80203	0.71189	0.79614	0.88869
	RAMO	0.66542	0.48767	**0.94647**	0.64285	0.7104	0.88335
	MWMOTE	**0.81508**	**0.66938**	0.82815	**0.73935**	**0.818**	**0.90156**
Vehicle	SMOTE	0.95845	0.87097	0.97251	0.91769	0.96321	0.97428
	ADASYN	0.91381	0.76882	**0.99069**	0.85784	0.937	0.94907
	RAMO	0.95743	0.85777	0.98615	0.91613	**0.96702**	0.97826
	MWMOTE	**0.96061**	**0.87218**	0.97706	**0.92081**	0.9661	**0.97949**
Wine	SMOTE	0.97745	0.95	0.975	0.95844	0.97483	0.98182
	ADASYN	0.98856	0.96333	1	0.9798	0.99215	0.9816
	RAMO	0.98268	**0.98333**	0.95	0.96234	0.96928	0.97429
	MWMOTE	**0.98889**	0.96667	**1**	**0.98182**	**0.99215**	**0.98276**
Yeast	SMOTE	0.85176	**0.61218**	0.78581	0.68509	0.82486	0.88801
	ADASYN	0.84905	0.60127	0.8057	0.68609	0.83142	0.88972
	RAMO	0.84433	0.5857	**0.8386**	0.68774	**0.84089**	0.88935
	MWMOTE	**0.85312**	0.60962	0.79903	**0.68953**	0.83093	**0.89101**
Spambase	SMOTE	0.8668	0.78499	0.92117	0.84658	0.87463	0.92774
	ADASYN	0.87894	0.79702	0.93272	0.85911	0.88703	0.93818
	RAMO	0.78875	0.65745	**0.97187**	0.78408	0.80646	0.94503
	MWMOTE	**0.8998**	**0.83544**	0.92996	**0.87988**	**0.90461**	**0.94644**
Texture	SMOTE	0.91912	0.79784	0.96067	0.8696	0.93126	0.96672
	ADASYN	0.90521	0.77243	0.96	0.85257	0.92088	0.95273
	RAMO	0.885	0.7215	**0.978**	0.82752	0.91116	0.94989
	MWMOTE	**0.92497**	**0.81064**	0.972	**0.88108**	**0.93872**	**0.96809**

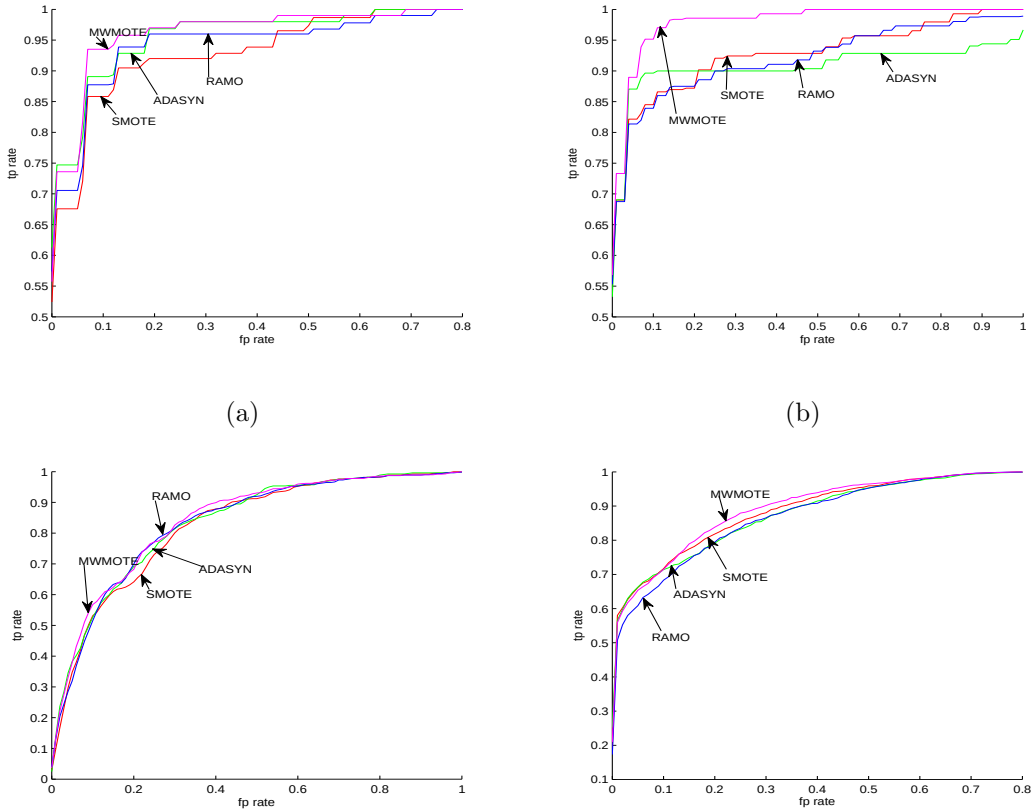


Figure 4.6: Averaged ROC curves of SMOTE [15], ADASYN [17], RAMO [18], and MWMOTE oversampling techniques using single neural network classifier on real world data sets: (a) Glass. (b) Libra. (c) Pima. (d) Satimage.

Table 4.13: Performance of SMOTE [15], ADASYN [17], RAMO [18], and MWMOTE using Adaboost.M2 ensemble of neural network classifiers on real world datasets: Abalone, Breast-cancer-diagnostic, Breast-cancer-original, Breast-tissue, Ecoli, Glass, Ionosphere, and Libra.

Dataset	Methods	Accuracy	Precision	Recall	F-measure	G-mean	AUC
Abalone	SMOTE	0.93979	0.4869	0.61905	0.54435	**0.77019**	0.89285
	ADASYN	0.93569	0.45273	0.5	0.47472	0.69207	0.8899
	RAMO	0.9398	0.48958	**0.64286**	0.55304	0.7845	0.90447
	MWMOTE	**0.94528**	**0.5219**	0.57143	**0.54451**	0.74307	**0.90024**
Breast-cancer-diagnostic	SMOTE	0.9666	0.94925	**0.96212**	0.95523	0.96543	0.98105
	ADASYN	0.96134	0.94208	0.95736	0.9486	0.95999	0.9818
	RAMO	0.96845	0.95638	**0.96212**	0.95815	0.96668	**0.98328**
	MWMOTE	**0.97541**	**0.97593**	0.95758	**0.96632**	**0.97149**	0.98189
Breast-cancer-original	SMOTE	0.96195	0.93385	0.96196	0.94629	0.96142	0.97749
	ADASYN	0.95912	0.92544	0.9625	0.94307	0.95977	0.97516
	RAMO	0.96496	0.93063	**0.97482**	0.95147	0.967	0.97601
	MWMOTE	**0.97078**	**0.9553**	0.96232	**0.95839**	**0.96867**	**0.97969**
Breast-tissue	SMOTE	0.73545	0.61333	0.71667	0.64841	0.71844	0.81589
	ADASYN	0.81091	0.73333	0.8	0.74381	0.79645	0.81829
	RAMO	0.77455	0.64167	**0.83333**	0.71468	0.78076	0.85525
	MWMOTE	**0.83182**	**0.77167**	0.775	**0.76238**	**0.81106**	**0.86185**
Ecoli	SMOTE	**0.88968**	**0.76409**	0.79251	**0.77142**	0.85253	0.92811
	ADASYN	0.87774	0.72504	0.79217	0.75272	0.84561	0.91267
	RAMO	0.87774	0.71892	**0.83165**	0.76273	**0.85973**	0.92565
	MWMOTE	0.88077	0.72162	0.81849	0.76215	0.85731	**0.93045**
Glass	SMOTE	0.93977	0.88976	0.86667	0.86864	0.90903	0.94797
	ADASYN	0.93997	0.89167	**0.86667**	0.86995	**0.90926**	**0.96893**
	RAMO	0.93564	0.88	0.86667	0.8651	0.90787	0.95717
	MWMOTE	**0.94412**	**0.93**	0.84667	**0.87162**	0.90347	0.95524
Ionosphere	SMOTE	0.89396	0.87629	0.825	0.84635	0.87549	0.92728
	ADASYN	0.90548	0.88623	**0.84936**	0.86325	**0.89006**	**0.93009**
	RAMO	0.90311	0.9091	0.81795	0.85524	0.87911	0.92134
	MWMOTE	**0.91136**	**0.92121**	0.82564	**0.86702**	0.88809	0.9278
Libra	SMOTE	0.98348	0.98889	0.93036	0.9554	0.96148	0.97544
	ADASYN	0.98071	0.9875	0.91786	0.947	0.95454	0.97214
	RAMO	0.98348	0.9875	0.93214	0.95469	0.96196	0.97214
	MWMOTE	**0.98626**	**1**	**0.93214**	**0.96136**	**0.9637**	**0.97638**

Table 4.14: Performance of SMOTE [15], ADASYN [17], RAMO [18], and MWMOTE using Adaboost.M2 ensemble of neural network classifiers on real world datasets: Phoneme, Pima, Robot, Satimage, Vehicle, Wine, Yeast, Spambase, and texture.

Dataset	Methods	Accuracy	Precision	Recall	F-measure	G-mean	AUC
Phoneme	SMOTE	0.77693	0.57816	0.86053	0.6916	0.79936	0.86299
	ADASYN	0.77291	0.57208	0.87566	0.69181	0.79981	0.85982
	RAMO	0.75435	0.54752	**0.90066**	0.68074	0.79053	0.86066
	MWMOTE	**0.78726**	**0.5913**	0.87566	**0.70557**	**0.81074**	**0.87389**
Pima	SMOTE	0.70969	0.55953	0.81353	0.66121	0.72695	0.78641
	ADASYN	0.71487	0.56438	0.82821	0.66959	0.73393	0.79871
	RAMO	0.70441	0.55528	0.84687	0.66823	0.72662	0.7849
	MWMOTE	**0.73566**	**0.58916**	**0.83575**	**0.68906**	**0.75308**	**0.81326**
Robot	SMOTE	0.87243	0.67224	0.7747	0.71974	0.83432	0.9146
	ADASYN	0.87775	0.66918	**0.83709**	0.7434	**0.86236**	0.92285
	RAMO	0.86675	0.64844	0.81542	0.72177	0.84722	0.91841
	MWMOTE	**0.88453**	**0.69669**	0.80849	**0.74775**	0.85517	**0.9275**
Satimage	SMOTE	0.84429	0.71004	0.85904	0.77739	0.84816	**0.92695**
	ADASYN	**0.84771**	**0.71206**	0.87083	**0.78348**	**0.85375**	0.92669
	RAMO	0.81103	0.6394	**0.92485**	0.75601	0.83745	0.92573
	MWMOTE	0.84646	0.71163	0.86542	0.78103	0.85144	0.92679
Vehicle	SMOTE	0.97665	0.94047	0.96364	0.95072	0.97179	0.98157
	ADASYN	0.97976	0.94809	0.96753	0.95674	0.97509	0.98238
	RAMO	0.9734	0.93186	0.95844	0.94378	0.96772	0.98091
	MWMOTE	**0.98086**	**0.9484**	**0.97251**	**0.95965**	**0.97776**	**0.98316**
Wine	SMOTE	0.98856	0.96333	1	0.9798	0.99215	**0.98218**
	ADASYN	0.98268	0.94333	1	0.96869	0.98823	0.98151
	RAMO	0.97712	0.94333	0.98	0.95758	0.97767	0.98157
	MWMOTE	**0.99412**	**0.98**	**1**	**0.98889**	**0.99608**	0.98162
Yeast	SMOTE	0.86731	0.64046	0.80968	0.71457	0.84453	0.89058
	ADASYN	0.85779	0.61978	0.81871	0.70341	0.84214	0.88823
	RAMO	0.83155	0.56446	0.82194	0.66724	0.82671	0.88644
	MWMOTE	**0.87871**	**0.66907**	**0.8257**	**0.73647**	**0.8573**	**0.90218**
Spambase	SMOTE	0.90503	0.84607	0.92994	0.88554	0.90896	0.95377
	ADASYN	0.90067	0.83865	0.92827	0.88069	0.90499	0.95374
	RAMO	0.84721	0.73367	**0.96359**	0.83278	0.86203	0.94719
	MWMOTE	**0.90893**	**0.86334**	0.91505	**0.88799**	**0.9098**	**0.95412**
Texture	SMOTE	0.99525	0.99007	0.99267	0.99135	0.99445	**0.98702**
	ADASYN	0.99452	0.98807	0.992	0.99002	0.99373	0.98682
	RAMO	**0.99635**	**0.99137**	**0.99533**	**0.99335**	**0.99603**	0.98658
	MWMOTE	0.99489	0.98871	0.99267	0.99068	0.99419	0.98689

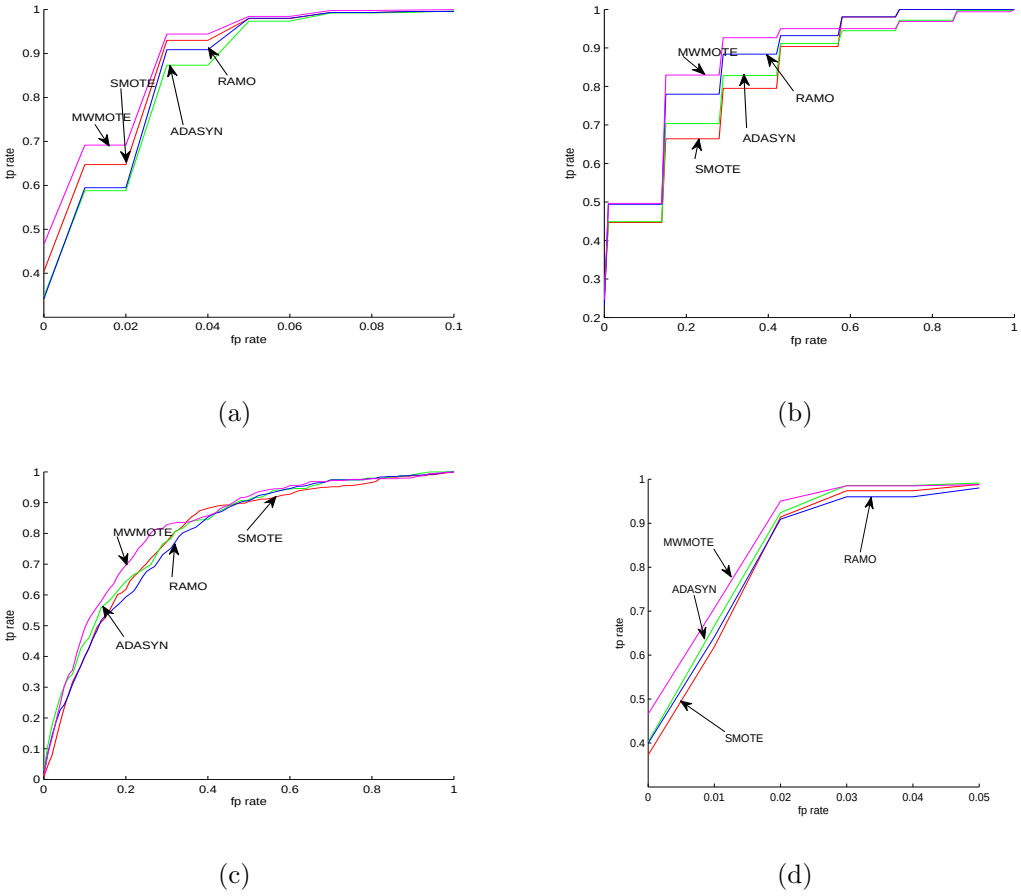


Figure 4.7: Averaged ROC curves of SMOTE [15], ADASYN [17], RAMO [18], and MWMOTE oversampling techniques using AdaBoost.M2 ensemble of neural network classifiers on real world data sets: (a) Breast Cancer. (b) Breast Tissue. (c) Pima. (d) Vehicle.

Table 4.15: Simulation 1 on real world datasets using single neural network classifier: Significance test of averaged AUC between MWMOTE, and SMOTE [15].

Dataset	MWMOTE	SMOTE	Difference	Rank
Abalone	0.87745	0.87288	0.00457	+8
Breast-cancer-diagnostic	0.98086	0.87288	0.10798	+17
Breast-cancer-original	0.97629	0.97692	-0.00063	-1
Breast-tissue	0.84189	0.84299	-0.0011	-3
Ecoli	0.93705	0.94622	-0.00917	-11
Glass	0.95831	0.93227	0.02604	+15
Ionosphere	0.91353	0.90296	0.01057	+12
Libra	0.96955	0.91719	0.05236	+16
Phoneme	0.82019	0.82397	-0.00378	-7
Pima	0.83026	0.81443	0.01583	+13
Robot	0.86586	0.86939	-0.00353	-6
Satimage	0.90156	0.89643	0.00513	+9
Vehicle	0.97949	0.97428	0.00521	+10
Wine	0.98276	0.98182	0.00094	+2
Yeast	0.89101	0.88801	0.003	+5
Spambase	0.94644	0.92774	0.0187	+14
Texture	0.96809	0.96672	0.00137	+4
$T = min\{R^+, R^-\} = min\{125, 28\} = 28$				

Table 4.16: Simulation 1 on real world datasets using single neural network classifier: Significance test of averaged AUC between MWMOTE, and ADASYN [17].

Dataset	MWMOTE	ADASYN	Difference	Rank
Abalone	0.87745	0.85144	0.02601	+15
Breast-cancer-diagnostic	0.98086	0.98002	0.00084	+1
Breast-cancer-original	0.97629	0.97499	0.0013	+4
Breast-tissue	0.84189	0.8561	-0.01421	-12
Ecoli	0.93705	0.94222	-0.00517	-6
Glass	0.95831	0.95363	0.00468	+5
Ionosphere	0.91353	0.92885	-0.01532	-13
Libra	0.96955	0.89716	0.07239	+17
Phoneme	0.82019	0.81273	0.00746	+8
Pima	0.83026	0.82319	0.00707	+7
Robot	0.86586	0.85526	0.0106	+10
Satimage	0.90156	0.88869	0.01287	+11
Vehicle	0.97949	0.94907	0.03042	+16
Wine	0.98276	0.9816	0.00116	+2
Yeast	0.89101	0.88972	0.00129	+3
Spambase	0.94644	0.93818	0.00826	+9
Texture	0.96809	0.95273	0.01536	+14
$T = min\{R^+, R^-\} = min\{122, 31\} = 31$				

Table 4.17: Simulation 1 on real world datasets using single neural network classifier: Significance test of averaged AUC between MWMOTE, and RAMO [18].

Dataset	MWMOTE	RAMO	Difference	Rank
Abalone	0.87745	0.89149	-0.01404	-10
Breast-cancer-diagnostic	0.98086	0.97546	0.0054	+7
Breast-cancer-original	0.97629	0.97532	0.00097	+1
Breast-tissue	0.84189	0.79948	0.04241	+15
Ecoli	0.93705	0.93304	0.00401	+5
Glass	0.95831	0.94235	0.01596	+11
Ionosphere	0.91353	0.88042	0.03311	+14
Libra	0.96955	0.91022	0.05933	+16
Phoneme	0.82019	0.82494	-0.00475	-6
Pima	0.83026	0.82157	0.00869	+9
Robot	0.86586	0.80044	0.06542	+17
Satimage	0.90156	0.88335	0.01821	+13
Vehicle	0.97949	0.97826	0.00123	+2
Wine	0.98276	0.97429	0.00847	+8
Yeast	0.89101	0.88935	0.00166	+4
Spambase	0.94644	0.94503	0.00141	+3
Texture	0.96809	0.94989	0.0182	+12
$T = min\{R^+, R^-\} = min\{137, 16\} = 16$				

Table 4.18: Simulation 1 on real world datasets using AdaBoost.M2 ensemble of neural network classifiers: Significance test of averaged AUC between MWMOTE, and SMOTE [15].

Dataset	MWMOTE	SMOTE	Difference	Rank
Abalone	0.90024	0.89285	0.00739	+12
Breast-cancer-diagnostic	0.98189	0.98105	0.00084	+6
Breast-cancer-original	0.97969	0.97749	0.0022	+9
Breast-tissue	0.86185	0.81589	0.04596	+17
Ecoli	0.93045	0.92811	0.00234	+10
Glass	0.95524	0.94797	0.00727	+11
Ionosphere	0.9278	0.92728	0.00052	+4
Libra	0.97638	0.97544	0.00094	+7
Phoneme	0.87389	0.86299	0.0109	+13
Pima	0.81326	0.78641	0.02685	+16
Robot	0.9275	0.9146	0.0129	+15
Satimage	0.92679	0.92695	-0.00016	-2
Vehicle	0.98316	0.98157	0.00159	+8
Wine	0.98162	0.98218	-0.00056	-5
Yeast	0.90218	0.89058	0.0116	+14
Spambase	0.95412	0.95377	0.00035	+3
Texture	0.98689	0.98702	-0.00013	-1
$T = min\{R^+, R^-\} = min\{145, 8\} = 8$				

Table 4.19: Simulation 1 on real world datasets using AdaBoost.M2 ensemble of neural network classifiers: Significance test of averaged AUC between MWMOTE, and ADASYN [17].

Dataset	MWMOTE	ADASYN	Difference	Rank
Abalone	0.90024	0.8899	0.01034	+11
Breast-cancer-diagnostic	0.98189	0.9818	0.00009	+2
Breast-cancer-original	0.97969	0.97516	0.00453	+9
Breast-tissue	0.86185	0.81829	0.04356	+17
Ecoli	0.93045	0.91267	0.01778	+16
Glass	0.95524	0.96893	-0.01369	-12
Ionosphere	0.9278	0.93009	-0.00229	-7
Libra	0.97638	0.97214	0.00424	+8
Phoneme	0.87389	0.85982	0.01407	+14
Pima	0.81326	0.79871	0.01455	+15
Robot	0.9275	0.92285	0.00465	+10
Satimage	0.92679	0.92669	0.0001	+3
Vehicle	0.98316	0.98238	0.00078	+6
Wine	0.98162	0.98151	0.00011	+4
Yeast	0.90218	0.88823	0.01395	+13
Spambase	0.95412	0.95374	0.00038	+5
texture	0.98689	0.98682	0.00007	+1
$T = min\{R^+, R^-\} = min\{134, 19\} = 19$				

Table 4.20: Simulation 1 on real world datasets using AdaBoost.M2 ensemble of neural network classifiers: Significance test of averaged AUC between MWMOTE, and RAMO [18].

Dataset	MWMOTE	RAMO	Difference	Rank
Abalone	0.90024	0.90447	-0.00423	-8
Breast-cancer-diagnostic	0.98189	0.98328	-0.00139	-4
Breast-cancer-original	0.97969	0.97601	0.00368	+7
Breast-tissue	0.86185	0.85525	0.0066	+12
Ecoli	0.93045	0.92565	0.0048	+10
Glass	0.95524	0.95717	-0.00193	-5
Ionosphere	0.9278	0.92134	0.00646	+11
Libra	0.97638	0.97214	0.00424	+9
Phoneme	0.87389	0.86066	0.01323	+15
Pima	0.81326	0.7849	0.02836	+16
Robot	0.9275	0.91841	0.00909	+14
Satimage	0.92679	0.92573	0.00106	+3
Vehicle	0.98316	0.98091	0.00225	+6
Wine	0.98162	0.98157	0.00005	+1
Yeast	0.90218	0.88644	0.01574	+16
Spambase	0.95412	0.94719	0.00693	+13
Texture	0.98689	0.98658	0.00031	+2
$T = min\{R^+, R^-\} = min\{136, 17\} = 17$				

difference is negative (SMOTE is better) and in rest 14 datasets, difference is positive (MWMOTE is better). We sum all the positive ranks under the *rank* column, to find $R+$, which is 128. Similarly, we sum all the negative ranks to find, $R-$, which is 28. The minimum of $R+$, and $R-$, is the value of T, which is $min\{128, 28\} = 28$. Since there are 17 datasets, the T value should be less than or equal to 35 at a significance level of 0.05 to reject the null hypothesis that, the difference is not significant [38]. So, in this case of MWMOTE vs SMOTE, T value (28) is less than 35, which means that, the difference is statistically significant. Similarly, for the other comparisons between MWMOTE vs ADASYN, and MWMTOE vs RAMO, we find that, the computed T values are 31, and 16 respectively, both of which are less than critical value, 35. These results show that, MWMOTE can really statistically outperform SMOTE, ADASYN, and RAMO techniques.

In case of AdaBoost.M2 ensemble classifier, results in Tables 4.13 and 4.14 show that, MWMOTE still outperforms other three oversampling techniques except the recall parameter as expected from discussions before. Comparing performance measures of single neural network classifier and boosting ensemble of neural network, we find that, boosting improves performance of all four oversampling techniques significantly. In particular, we observe that, recall performances are nearly same, while precision performance is much better for the boosting ensemble than single classifier. The reason is that, in case of single classifier, improvement of minority prediction accuracy (due to oversampling) is sacrificed by reduction in majority prediction accuracy (justified by low precision performance). However, boosting ensures that this majority prediction accuracy is not sacrificed (justified by better precision values), while keeping the minority prediction accuracy at the same level. So, boosting and oversampling together can be able to improve both minority and majority classification accuracy, which is justified by overall performance measures of boosting ensemble: F-measure, G-mean, and AUC values of Tables 4.13 and 4.14. Some representative ROC graphs are shown in Fig. 4.7 for this simulation run. From these graphs, we can visually inspect the better performance of MWMOTE compared to other three methods, since ROC graph of MWMOTE is above the ROC graphs of other three

methods in most of the portion of the ROC space.

The wilcoxon signed rank test is performed on AUC values of Tables 4.13 and 4.14 for comparing MWMOTE against each of the other three techniques. The test results are shown in Tables 4.18, 4.19, and 4.20,. The T values found for MWMOTE vs SMOTE, ADASYN, and RAMO techniques are 8, 19, and 17 respectively. At the significance level of 0.05, all these values are less than 35 (critical value), which says that the differences are statistically significant. Therefore, in case of boosting also, MWMOTE can statistically outporfoms each of SMOTE, ADASYN, and RAMO techniques.

4.4.2 Simualtion 2

In this simulation, we test the performance of MWMOTE on real world datasets when the single classifier is not neural network. Similar to the experiments performed for artificial data sets, we run k-nearest neighbor (with $k = 5$) and C4.5 decision tree classifier without pruning on the 17 real world datasets. We summarize the results of these two classifiers with the four oversampling techniques SMOTE, ADASYN, RAMO, and MWMOTE in Tables 4.21-4.24 . The results are found after a 10-fold cross validation on the datasets. From these results, we see that, even for both of k-nearest neighbor and decision tree classifiers, MWMOTE shows comparatively better performance in terms of accuracy, precision, F-measure, G-mean except the recall parameter.

4.5 Summary

In this chapter, we performed extensive experiments to evaluate the effectiveness of propose oversample technique MWMOTE. We generated some artificial data sets of different complexities and imbalances. We evaluated MWMOTE and three other existing oversampling techniques SMOTE, ADASYN, and RAMO using neural network classifier, Adaboost.M2 ensemble of neural network classifier, k-nearest neighbor classifier and decision tree classifier. Experimental results show that MWMOTE dominates all the other three

Table 4.21: Performance of SMOTE [15], ADASYN [17], RAMO [18], and MWMOTE using k-NN as a base classifier (with $k = 5$) on Real World Datasets: Abalone, Breast-cancer-diagnostic, Breast-cancer-original, Breast-tissue, Ecoli, Glass, Ionosphere, and Libra.

Dataset	Methods	Accuracy	Precision	Recall	F-measure	G-mean
Abalone	SMOTE	0.94235	0.51481	0.5	0.50188	0.64738
	ADASYN	0.93017	0.41481	0.5	0.44095	0.67309
	RAMO	0.94228	0.54471	0.47222	0.47735	0.66253
	MWMOTE	0.94078	0.47222	0.44444	0.44782	0.60836
Breast cancer diagonostic	SMOTE	0.94914	0.91588	0.95758	0.93414	0.95015
	ADASYN	0.93502	0.88086	0.95758	0.91652	0.939
	RAMO	0.93505	0.86624	0.98139	0.91945	0.94356
	MWMOTE	0.95269	0.91441	0.9671	0.93901	0.95529
Breast-cancer-original	SMOTE	0.96927	0.92838	0.99167	0.95833	0.97413
	ADASYN	0.96927	0.92838	0.99167	0.95833	0.97413
	RAMO	0.96635	0.91817	0.99583	0.95479	0.97276
	MWMOTE	0.97511	0.94221	0.99167	0.96586	0.97874
Breast-tissue	SMOTE	0.77455	0.63048	0.875	0.72328	0.77943
	ADASYN	0.76545	0.63	0.85	0.71402	0.7678
	RAMO	0.76545	0.63778	0.9	0.72987	0.76471
	MWMOTE	0.75545	0.61333	0.86667	0.70603	0.75379
Ecoli	SMOTE	0.88366	0.70838	0.88214	0.78133	0.88149
	ADASYN	0.89543	0.7301	0.89643	0.8004	0.89475
	RAMO	0.87521	0.69308	0.88393	0.77286	0.87766
	MWMOTE	0.90784	0.76389	0.86964	0.81196	0.89302
Glass	SMOTE	0.94349	0.86714	0.92	0.88817	0.93314
	ADASYN	0.93873	0.85048	0.92	0.87908	0.92996
	RAMO	0.93873	0.84333	0.94	0.88211	0.93655
	MWMOTE	0.95777	0.88714	0.96	0.91928	0.95784
Ionosphere	SMOTE	0.91169	0.91378	0.83462	0.86972	0.89153
	ADASYN	0.90549	0.9164	0.81667	0.85701	0.88031
	RAMO	0.91136	0.90772	0.84167	0.87076	0.8934
	MWMOTE	0.91464	0.95758	0.80256	0.86976	0.88452
Libra	SMOTE	0.99437	1	0.97143	0.98462	0.98516
	ADASYN	0.99437	1	0.97143	0.98462	0.98516
	RAMO	0.99159	0.9875	0.97143	0.97795	0.98342
	MWMOTE	0.99437	1	0.97143	0.98462	0.98516

Table 4.22: Performance of SMOTE [15], ADASYN [17], RAMO [18], and MWMOTE using k-NN as a base classifier (with $k = 5$) on Real World Datasets: Phoneme, Pima, Robot, Satimage, Vehicle, Wine, Yeast, Spambase, and texture.

Dataset	Methods	Accuracy	Precision	Recall	F-measure	G-mean
Phoneme	SMOTE	0.86704	0.72236	0.88487	0.79492	0.87203
	ADASYN	0.86838	0.72181	0.89211	0.79762	0.87503
	RAMO	0.85116	0.68451	0.90921	0.78062	0.86715
	MWMOTE	0.86876	0.72505	0.88618	0.79717	0.87365
Pima	SMOTE	0.68628	0.53585	0.78775	0.63697	0.70424
	ADASYN	0.66936	0.51967	0.76168	0.61664	0.68534
	RAMO	0.66668	0.51537	0.80627	0.62805	0.68986
	MWMOTE	0.67454	0.52369	0.7688	0.62194	0.69101
Robot	SMOTE	0.91788	0.75107	0.91942	0.8261	0.91832
	ADASYN	0.91642	0.74363	0.92809	0.82511	0.92061
	RAMO	0.90854	0.71746	0.94109	0.81368	0.92014
	MWMOTE	0.91257	0.74051	0.90729	0.81483	0.9105
Satimage	SMOTE	0.89122	0.76241	0.95383	0.84735	0.90685
	ADASYN	0.88827	0.75508	0.95826	0.84453	0.90559
	RAMO	0.86076	0.70096	0.97839	0.81661	0.88815
	MWMOTE	0.89215	0.76235	0.95923	0.84934	0.90879
Vehicle	SMOTE	0.92761	0.77738	0.97727	0.86446	0.94411
	ADASYN	0.92334	0.76913	0.97273	0.85694	0.93956
	RAMO	0.92122	0.76553	0.97273	0.85435	0.93815
	MWMOTE	0.92972	0.7852	0.97251	0.86721	0.94384
Wine	SMOTE	0.96634	0.90143	1	0.94495	0.97629
	ADASYN	0.96078	0.88476	1	0.93586	0.97237
	RAMO	0.95523	0.8681	1	0.92677	0.96845
	MWMOTE	0.9719	0.9181	1	0.95404	0.98022
Yeast	SMOTE	0.84166	0.59178	0.75269	0.65902	0.80426
	ADASYN	0.82212	0.55322	0.75591	0.63447	0.79413
	RAMO	0.80729	0.52412	0.78237	0.62475	0.79612
	MWMOTE	0.85176	0.61539	0.77258	0.67901	0.81796
Spambase	SMOTE	0.87939	0.79324	0.93987	0.86017	0.88847
	ADASYN	0.86917	0.77614	0.93987	0.85006	0.87952
	RAMO	0.84265	0.72973	0.95587	0.82747	0.85726
	MWMOTE	0.88308	0.80005	0.93878	0.86374	0.89158
Texture	SMOTE	0.99361	0.98228	0.99467	0.98842	0.99394
	ADASYN	0.99324	0.98226	0.99333	0.98775	0.99327
	RAMO	0.99124	0.97215	0.99667	0.98423	0.99292
	MWMOTE	0.99434	0.98425	0.99533	0.98974	0.99465

Table 4.23: Performance of SMOTE [15], ADASYN [17], RAMO [18], and MWMOTE using decision tree as a base classifier on Real World Datasets: Abalone, Breast-cancer-diagnostic, Breast-cancer-original, Breast-tissue, Ecoli, Glass, Ionosphere, and Libra.

Dataset	Methods	Accuracy	Precision	Recall	F-measure	G-mean
Abalone	SMOTE	0.90159	0.25422	0.4	0.30677	0.56252
	ADASYN	0.90966	0.31918	0.465	0.36025	0.61153
	RAMO	0.89748	0.24286	0.355	0.28168	0.53446
	MWMOTE	0.91801	0.36786	0.395	0.35762	0.59794
Breast cancer diagonostic	SMOTE	0.93311	0.89854	0.92922	0.91199	0.93173
	ADASYN	0.92969	0.89585	0.92446	0.90857	0.9282
	RAMO	0.92098	0.87276	0.92965	0.89876	0.92223
	MWMOTE	0.93317	0.90425	0.92489	0.91217	0.93076
Breast-cancer-original	SMOTE	0.94891	0.91495	0.94565	0.92868	0.94776
	ADASYN	0.94588	0.93354	0.91214	0.92125	0.93715
	RAMO	0.94296	0.92385	0.91612	0.91842	0.93606
	MWMOTE	0.9488	0.92286	0.93297	0.92662	0.94449
Breast-tissue	SMOTE	0.76545	0.64714	0.725	0.6659	0.7367
	ADASYN	0.78455	0.68452	0.80833	0.72582	0.78109
	RAMO	0.75545	0.64667	0.75833	0.68063	0.74722
	MWMOTE	0.79545	0.69167	0.76667	0.71611	0.78195
Ecoli	SMOTE	0.86648	0.71762	0.73929	0.7167	0.81219
	ADASYN	0.86905	0.71006	0.75179	0.72472	0.82128
	RAMO	0.8777	0.73168	0.7875	0.74646	0.83959
	MWMOTE	0.88118	0.74619	0.78214	0.7452	0.83513
Glass	SMOTE	0.8926	0.75111	0.82333	0.7782	0.85501
	ADASYN	0.90152	0.78083	0.82	0.79277	0.85936
	RAMO	0.90669	0.81952	0.82333	0.81505	0.87458
	MWMOTE	0.91991	0.83548	0.84	0.83313	0.88786
Ionosphere	SMOTE	0.85711	0.77784	0.8641	0.81388	0.85612
	ADASYN	0.84552	0.77246	0.82436	0.79147	0.83738
	RAMO	0.82583	0.72753	0.84038	0.77747	0.82791
	MWMOTE	0.87517	0.82165	0.86603	0.83633	0.87032
Libra	SMOTE	0.86955	0.75627	0.575	0.63475	0.72467
	ADASYN	0.90862	0.77202	0.78036	0.76718	0.85059
	RAMO	0.91108	0.82679	0.725	0.76357	0.83058
	MWMOTE	0.90244	0.8081	0.70714	0.73482	0.81296

Table 4.24: Performance of SMOTE [15], ADASYN [17], RAMO [18], and MWMOTE using decision tree as a base classifier on Real World Datasets: Phoneme, Pima, Robot, Satimage, Vehicle, Wine, Yeast, Spambase, and texture.

Dataset	Methods	Accuracy	Precision	Recall	F-measure	G-mean
Phoneme	SMOTE	0.87717	0.76706	0.83026	0.79721	0.86259
	ADASYN	0.8766	0.76124	0.83947	0.7981	0.86504
	RAMO	0.87067	0.7451	0.84605	0.79198	0.86308
	MWMOTE	0.88062	0.77709	0.82895	0.80172	0.86446
Pima	SMOTE	0.68886	0.54961	0.6235	0.58212	0.67011
	ADASYN	0.67329	0.53105	0.59744	0.56164	0.65259
	RAMO	0.67059	0.52767	0.64573	0.57864	0.66293
	MWMOTE	0.69405	0.55049	0.66382	0.60011	0.68405
Robot	SMOTE	0.98717	0.96282	0.97744	0.96993	0.98355
	ADASYN	0.99175	0.97627	0.98525	0.9806	0.98934
	RAMO	0.9912	0.97135	0.98786	0.97944	0.98996
	MWMOTE	0.99395	0.98965	0.98178	0.98562	0.98944
Satimage	SMOTE	0.8864	0.80661	0.84333	0.82451	0.87424
	ADASYN	0.88392	0.79942	0.84627	0.82199	0.87333
	RAMO	0.86931	0.76956	0.83794	0.80224	0.86054
	MWMOTE	0.88066	0.79485	0.84087	0.81696	0.86942
Vehicle	SMOTE	0.9489	0.88898	0.89459	0.89059	0.92885
	ADASYN	0.95099	0.88989	0.90368	0.89603	0.93376
	RAMO	0.94465	0.86826	0.90368	0.88345	0.9292
	MWMOTE	0.9521	0.89746	0.90433	0.89864	0.93458
Wine	SMOTE	0.97745	0.96333	0.96	0.95758	0.97104
	ADASYN	0.98301	0.98	0.96	0.96667	0.97496
	RAMO	0.98856	0.98	0.98	0.97778	0.98552
	MWMOTE	0.98889	1	0.96	0.97778	0.97889
Yeast	SMOTE	0.84906	0.61453	0.71355	0.65709	0.7918
	ADASYN	0.84708	0.6079	0.74355	0.6662	0.80473
	RAMO	0.83898	0.59164	0.69409	0.63741	0.77879
	MWMOTE	0.85445	0.63337	0.69118	0.65944	0.78618
Spambase	SMOTE	0.89502	0.85051	0.89025	0.86981	0.89412
	ADASYN	0.89415	0.85076	0.88693	0.86846	0.89286
	RAMO	0.88654	0.83627	0.88637	0.8604	0.88644
	MWMOTE	0.88828	0.84692	0.8748	0.86059	0.88584
Texture	SMOTE	0.979	0.95907	0.96467	0.96178	0.97446
	ADASYN	0.97572	0.95084	0.96133	0.95597	0.97117
	RAMO	0.97645	0.95139	0.96333	0.95729	0.97231
	MWMOTE	0.97407	0.95647	0.94867	0.9525	0.96599

methods and outperforms them in several useful performance metrics such as accuracy, precision, recall, AUC, and ROC graphs. Similar experiments were carried out on 17 real world data sets which gave similar results. Wilcoxon signed-rank significances are also performed which statistically justify the performance dominance of MWMOTE with other three methods.

Chapter 5

MWMOTE-Boost: Integration of MWMOTE with Boosting

In chapter 3, we presented our proposed oversampling technique MWMOTE. In chapter 4, we showed its effectiveness through experimental results. It was seen that, performance of MWMOTE is better than the existing oversampling algorithms used in experiments. Although, oversampling improves minority performance, sometimes it may lead to degradation of majority performance if excessive and inappropriate synthetic minority samples are generated near the decision boundary. The reduced majority performance may result in reduced overall performance measures such as accuracy, F-measure and G-mean are reduced. To this respect, some stand-alone algorithms exist in literature such as SMOTE-Boost [31] and RAMOBoost [18] which integrates Adaboost.M2 with their oversampling procedure. The goal of boosting is to improve the majority performance which are affected by oversampling. In this chapter, we propose a similar type of algorithm MWMOTE-Boost based on the MWMOTE oversampling procedure. MWMOTE-Boost integrates MWMOTE oversampling technique inside Adaboost.M2 ensemble in a manner similar to the recent RAMOBoost technique [18]. The simulation results at the end of this chapter shows that, the performance of the new stand-alone algorithm MWMOTE-Boost supersedes the performance of two other similar boosting algorithms existing in literature.

5.1 Integration of Boosting with Oversampling

Oversampling creates a bias for the classifier toward the minority class. The bias is obtained by creating synthetic samples near the decision boundary. However, the added minority samples introduced at the boundary may sometimes tend to over-bias the classifier's decision. The majority region near the over-sampled minority regions may become hard-to-learn for the classifier due to the dominance of the minority samples. This may lead to worse performance on the majority class in those regions of the over-sampled data set. So, oversampling introduces a trade off which sacrifices the performance on the majority class for the improved performance on the minority class. Sometimes, the reduced majority performance may not lead to any improvement in overall classifier performance. Therefore, it would have been better, if we could improve the minority performance and at the same time keep the majority performance unaltered. Boosting is introduced to solve this problem. The effect of boosting on the classifier is to iteratively focusing on the hard-to-learn examples of the data set and improving decision on those instances. So, if we add boosting on the over-sampled data set, then boosting may reduce the over-bias effect by focusing on those hard-to-learn majority samples iteratively. The application of boosting on oversampled data set has been studied in literature and shown to be very much successful [18, 31, 32]. So, motivated by those studies, we present a similar algorithm MMWOTE-Boost which MWMOTE oversampling technique inside Adaboost.M2 ensemble.

5.2 MWMOTE-Boost Algorithm

MWMOTE-Boost algorithm adopts the idea of the recent RAMOBoost [18] algorithm. RAMOBoost's performance has been shown to beat other stand-alone algorithms for imbalanced learning problems [18]. So, we used the same idea of RAMOBoost and replaced RAMOBoost's oversampling procedure (which is called RAMO) with our MWMOTE. Since, MWMOTE has outperformed RAMO technique, it is expected that MWMOTE-

Boost will also outperform RAMOBoost as a stand-alone algorithm. The complete algorithm is shown in [Algorithm 2]

Algorithm 2: MWMOTE-Boost($S_{min}, S_{maj}, N, k1, k2, k3$)

Input:

1. Training dataset with m class examples $(x_1, y_1), ..., (x_m, y_m)$, where $x_i (i = 1, ..., m)$ is an instance of the n dimensional feature space X and $y_i \in Y = \{\text{major,minor}\}$ is the class identity label associated with x_i
2. N: Number of synthetic samples to be generated
3. $k1$: Number of neighbors to consider for predicting noisy minority samples
4. $k2$: Number of majority neighbors to consider for constructing informative minority set
5. $k3$: Number of minority neighbors to consider for constructing informative minority set
   Let, $B = \{(i, y) : i \in \{1...m\}, y \neq y_i\}$

Initialize: $D_1(i, y) = 1/|B|$ for $(i, y) \in B$ (for two class problems $|B| = m$)
Procedure Begin Do for $t = 1, 2, ..., T$.

1. Sample the training data set D_t and get back a sampled data set S_e of identical size. Slice S_e into majority subset S_{maj} and minority subset S_{min}.
2. Generate N synthetic samples using MWMOTE(S_{min},S_{maj},N,$k1$,$k2$,$k3$) oversampling procedure.
3. Provide the base classifier with the Sampled dataset S_e and N synthetic samples.
4. Get back a hypothesis, h_t: $X \times Y \rightarrow \{0, 1\}$

5. Calculate the pseudo-loss of h_t

$$e_t = \frac{1}{2} \sum_{(i,y) \in B} D_t(i,y)(1 - h_t(x_i, y_i) + h_t(x_i, y))$$

6. Set $\beta_t = e_t/(1 - e_t)$

7. Update D_t

$$D_{t+1}(i,y) = \frac{D_t(i,y)}{Z_t} \beta_t^{1+h_t(x_i,y_i)-h_t(x_i,y)}$$

where, Z_t is a normalization constant.

8. End Loop

End

Output: The output hypothesis, $h_{final}(x)$ is calculated as follows:

$$h_{final}(x) = arg\ max_{y \in Y} \sum_{t=1...T} log(\frac{1}{\beta_t}) * h_t(x,y)$$

5.3 Description of MWMOTE-Boost

MWMOTE-Boost algorithm is constructed by modifying the classic adaptive boosting algorithm AdaBoost.M2 ensemble procedure [29]. The AdaBoost.M2 algorithm is the multi-class extension of AdaBoost algorithm using the pseudo-loss rather than error of the hypothesis [29]. The modification that MWMOTE-Boost incorporates in AdaBoost.M2 algorithm is by inserting synthetic oversampling method inside the boosting iteration (Step 2 of [Algorithm 2]). The algorithm is called adaptive boosting because the weight distribution inside the boosting is adaptively updated using error/loss of the learned hypothesis. In [Algorithm 2], Step 1 samples the training data set using weight distribution D_t to obtain a data set, S_e of identical size. Therefore, at each iteration a different data set is used for learning the base learning algorithm (Step 5 of [Algorithm 2]). This is the key step of the adaptive boosting algorithms where hard-to-learn examples are kept in and already learned examples are discarded from the data set that will be used for

training the next base learning algorithm. Due to this, classifier that will be learned next can focus only on examples that have not already learned correctly.

In Step 2, N new synthetic minority samples are generated from the examples in S_e using our MWMOTE oversampling procedure. This is the step where MWMOtE-Boost is different from RAMOBoost algorithm, replacing RAMOBoost's RAMO oversampling procedure with out MWMOTE method. In this step, over-sampling S_e rather than the original data set ensures that more synthetic examples are generated for hard-to-learn minority examples compared to other examples which have already been learned correctly by the ensemble. The sampled data set S_e and newly generated N synthetic samples are then presented to a base learning algorithm (Step 3). The learning algorithm returns a new hypothesis, h_t (Step 4). In Step 5, the pseudo-loss [29] of hypothesis h_t is computed. Then, the weight distribution D_t is updated (Step 7) to be used in next iteration.

The parameter β_t is a function pseudo-loss e_t which is used to update the weight distribution. This update rule reduces the weight corresponding those samples which are correctly learned (in case of two-class problems) or differentiated with other training classes (in case of multi-class problems) [29]. As a result, the probability of those examples on which classifier's decision is poor is increased and probability of rest examples is decreased. This ensures that at each iteration the algorithm focuses only on remaining hard-to-learn examples (examples having high probability because they were not learned correctly in previous iterations) and tries to learn them correctly by the base learning algorithm.

5.4 Experimental Results of MWMOTE-Boost

In this section, we evaluate performance of MWMOTE-Boost algorithm. We compare its performance with two other boosting algorithms SMOTEBoost and RAMOBoost. The experiments are performed using the same real world datasets used previously for MWMOTE's experiments (chapter 4). neural network is used as the base classifier for

Table 5.1: Performance of SMOTEBoost [31], RAMOBoost [18], and MWMOTE-Boost on 17 on Real World Datasets: Abalone, Breast-cancer-diagnostic, Breast-cancer-original, Breast-tissue, Ecoli, Glass, Ionosphere, Libra, Phoneme, Pima, Robot, Satimage, and Vehicle.

Dataset	Methods	Accuracy	Precision	Recall	F-measure	G-mean	AUC
Abalone	SMOTEBoost	0.95076	0.59	0.52	0.54304	0.70277	0.87004
	RAMOBoost	0.9453	0.51833	0.43	0.45214	0.62893	0.86828
	MWMOTEBoost	0.95215	0.63333	0.45	0.515	0.65449	0.85124
Breast-cancer-diagnostic	SMOTEBoost	0.96826	0.96245	0.9526	0.95676	0.96467	0.98019
	RAMOBoost	0.96128	0.95835	0.93853	0.9468	0.95581	0.97957
	MWMOTEBoost	0.97014	0.96488	0.95736	0.95979	0.96699	0.98192
Breast-cancer-original	SMOTEBoost	0.97063	0.94017	0.9788	0.9588	0.97234	0.97706
	RAMOBoost	0.96775	0.93676	0.97482	0.95489	0.96918	0.97562
	MWMOTEBoost	0.96494	0.94336	0.95833	0.95021	0.96317	0.97742
Breast-tissue	SMOTEBoost	0.72636	0.57667	0.8	0.65738	0.72173	0.82539
	RAMOBoost	0.75455	0.63952	0.75833	0.66629	0.7388	0.85603
	MWMOTEBoost	0.81091	0.71548	0.8	0.73193	0.78971	0.87313
Ecoli	SMOTEBoost	0.87833	0.72899	0.78214	0.74566	0.83675	0.92228
	RAMOBoost	0.89018	0.73296	0.84464	0.77928	0.87002	0.91258
	MWMOTEBoost	0.89321	0.74949	0.81964	0.77488	0.85988	0.92313
Glass	SMOTEBoost	0.94847	0.90905	0.88	0.88711	0.9198	0.95498
	RAMOBoost	0.94825	0.90571	0.88	0.89009	0.92156	0.95226
	MWMOTEBoost	0.9437	0.89571	0.88	0.88185	0.9175	0.96636
Ionosphere	SMOTEBoost	0.90344	0.90467	0.81731	0.85655	0.8801	0.93401
	RAMOBoost	0.9	0.92239	0.80128	0.8469	0.87061	0.92074
	MWMOTEBoost	0.90042	0.90707	0.80962	0.85243	0.87622	0.9333
Libra	SMOTEBoost	0.98063	1	0.90357	0.94597	0.94886	0.96471
	RAMOBoost	0.97499	0.97083	0.90357	0.93161	0.94494	0.97222
	MWMOTEBoost	0.98333	1	0.91607	0.95359	0.9558	0.97421
Phoneme	SMOTEBoost	0.75933	0.55713	0.86053	0.67577	0.78558	0.81615
	RAMOBoost	0.73178	0.5252	0.86513	0.65255	0.76438	0.80491
	MWMOTEBoost	0.74326	0.53969	0.85592	0.65983	0.77078	0.82678
Pima	SMOTEBoost	0.70048	0.55347	0.84359	0.6646	0.72182	0.76728
	RAMOBoost	0.70579	0.55406	0.84758	0.66808	0.72842	0.7877
	MWMOTEBoost	0.70702	0.55723	0.8433	0.66784	0.72736	0.77364
Robot	SMOTEBoost	0.82588	0.5645	0.7721	0.65211	0.80537	0.87886
	RAMOBoost	0.84916	0.62393	0.73484	0.67435	0.80407	0.88785
	MWMOTEBoost	0.84366	0.60021	0.78076	0.67861	0.8196	0.88622
Satimage	SMOTEBoost	0.80591	0.66017	0.80157	0.72107	0.80246	0.87751
	RAMOBoost	0.82937	0.69083	0.83202	0.75452	0.82966	0.90408
	MWMOTEBoost	0.8272	0.6745	0.87672	0.76241	0.83967	0.9061
Vehicle	SMOTEBoost	0.97768	0.93554	0.97273	0.95319	0.97577	0.98169
	RAMOBoost	0.98193	0.94824	0.97727	0.9621	0.98022	0.98042
	MWMOTEBoost	0.98086	0.94429	0.97727	0.96008	0.97954	0.98314

Table 5.2: Performance of SMOTEBoost [31], RAMOBoost [18], and MWMOTE-Boost on 17 Real World Data sets: Wine, Yeast, Spambase, and texture.

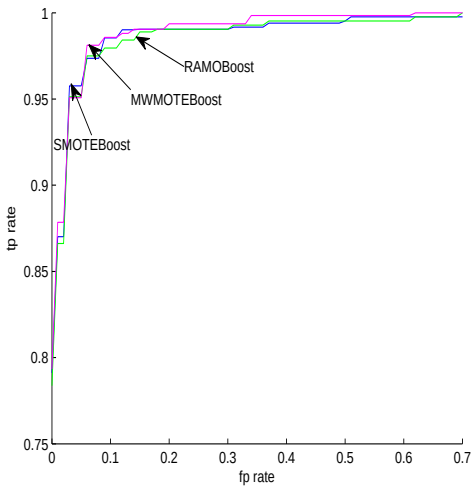
Dataset	Methods	Accuracy	Precision	Recall	F-measure	G-mean	AUC
Wine	SMOTEBoost	0.97191	0.92857	0.97917	0.9509	0.97384	0.98819
	RAMOBoost	0.97753	1	0.91667	0.95652	0.95743	0.98514
	MWMOTEBoost	0.98876	0.96	1	0.97959	0.99228	0.98863
Yeast	SMOTEBoost	0.85378	0.61619	0.76677	0.67974	0.81717	0.86797
	RAMOBoost	0.84975	0.60329	0.78312	0.68029	0.82299	0.87641
	MWMOTEBoost	0.86457	0.63762	0.80237	0.70816	0.83934	0.88123
Spambase	SMOTEBoost	0.9011	0.83395	0.93658	0.88207	0.90675	0.94425
	RAMOBoost	0.89436	0.82307	0.93438	0.87472	0.90052	0.94129
	MWMOTEBoost	0.89675	0.82555	0.93768	0.87771	0.90314	0.94709
Texture	SMOTEBoost	0.98594	0.96363	0.986	0.97466	0.98596	0.98585
	RAMOBoost	0.98138	0.94993	0.984	0.96665	0.98219	0.98544
	MWMOTEBoost	0.98594	0.96737	0.982	0.97455	0.9847	0.98535

Table 5.3: Simulation on real world datasets: Significance test of averaged AUC between MWMOTEBoost, and SMOTEBoost [31].

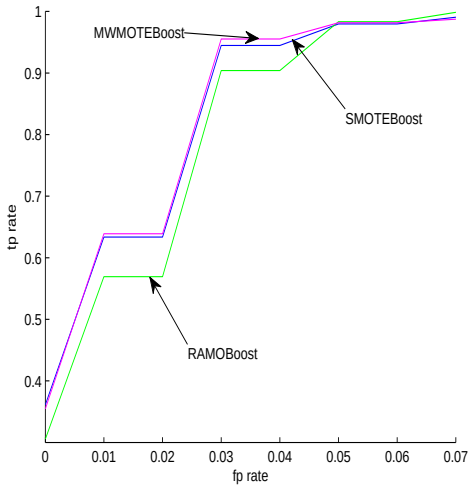
Dataset	MWMOTEBoost	SMOTEBoost	Difference	Rank
Abalone	0.85124	0.87004	-0.0188	-15
Breast-cancer-diagnostic	0.98192	0.98019	0.00173	+7
Breast-cancer-original	0.97742	0.97706	0.00036	+1
Breast-tissue	0.87313	0.82539	0.04774	+17
Ecoli	0.92313	0.92228	0.00085	+5
Glass	0.96636	0.95498	0.01138	+13
Ionosphere	0.9333	0.93401	-0.00071	-4
Libra	0.97421	0.96471	0.0095	+11
Phoneme	0.82678	0.81615	0.01063	+12
Pima	0.77364	0.76728	0.00636	+9
Robot	0.88622	0.87886	0.00736	+10
Satimage	0.9061	0.87751	0.02859	+16
Vehicle	0.98314	0.98169	0.00145	+6
Wine	0.98863	0.98819	0.00044	+2
Yeast	0.88123	0.86797	0.01326	+14
Spambase	0.94709	0.94425	0.00284	+8
texture	0.98535	0.98585	-0.0005	-3
$T = min\{R^+, R^-\} = min\{133, 22\} = 22$				

Table 5.4: Simulation on real world datasets: Significance test of averaged AUC between MWMOTEBoost, and RAMOBoost [18].

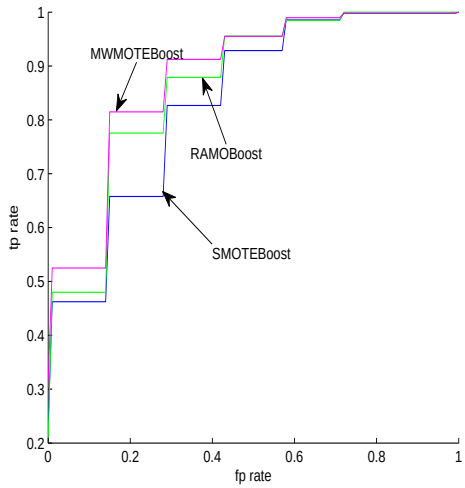
Dataset	MWMOTEBoost	RAMOBoost	Difference	+Rank
Abalone	0.85124	0.86828	-0.01704	-15
Breast-cancer-diagnostic	0.98192	0.97957	0.00235	+6
Breast-cancer-original	0.97742	0.97562	0.0018	+3
Breast-tissue	0.87313	0.85603	0.0171	+16
Ecoli	0.92313	0.91258	0.01055	+11
Glass	0.96636	0.95226	0.0141	+14
Ionosphere	0.9333	0.92074	0.01256	+12
Libra	0.97421	0.97222	0.00199	+4
Phoneme	0.82678	0.80491	0.02187	+17
Pima	0.77364	0.7877	-0.01406	-13
Robot	0.88622	0.88785	-0.00163	-2
Satimage	0.9061	0.90408	0.00202	+5
Vehicle	0.98314	0.98042	0.00272	+7
Wine	0.98863	0.98514	0.00349	+8
Yeast	0.88123	0.87641	0.00482	+9
Spambase	0.94709	0.94129	0.0058	+10
texture	0.98535	0.98544	-0.00009	-1
$T = min\{R^+, R^-\} = min\{124, 31\} = 31$				



(a)



(b)



(c)

Figure 5.1: Averaged ROC curves of SMOTEBoost [31], RAMOBoost [18], and MWMOTE-Boost on real world datasets: (a) Breast Cancer Diagnostic. (b) Breast Cancer Original. (c) Breast Tissue.

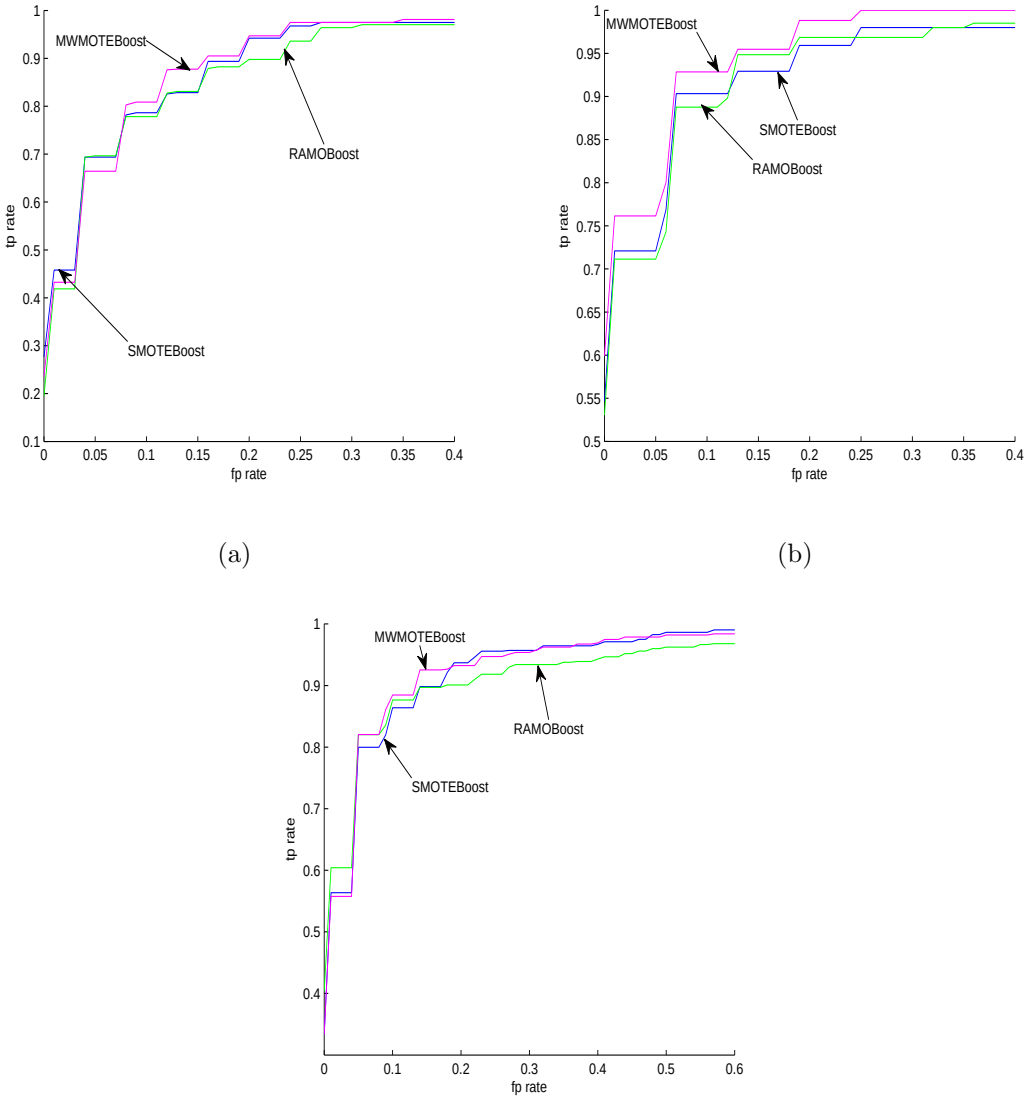
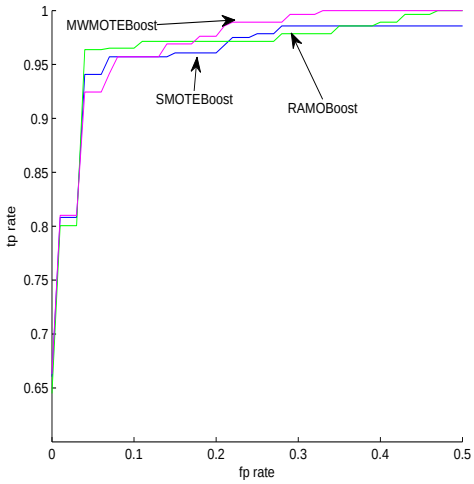
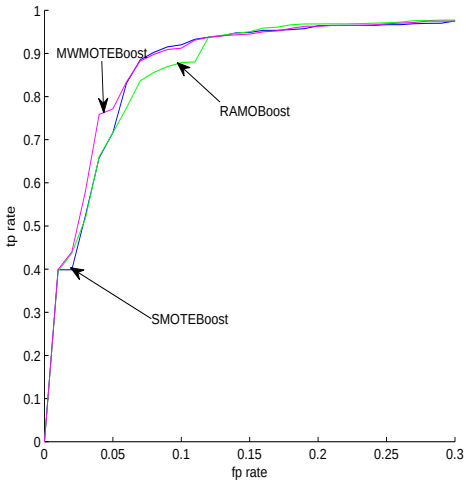


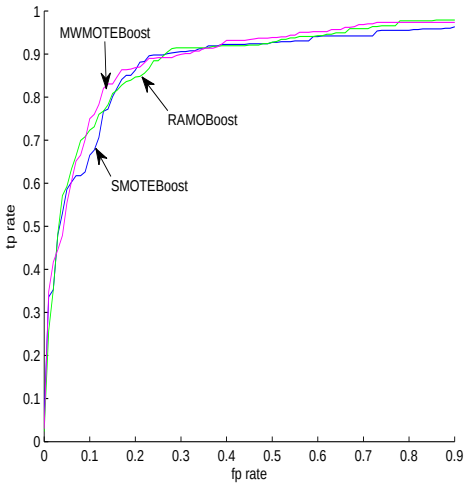
Figure 5.2: Averaged ROC curves of SMOTEBoost [31], RAMOBoost [18], and MWMOTE-Boost on real world datasets: (a) Ecoli. (b) Glass. (c) Ionosphere.



(a)



(b)



(c)

Figure 5.3: Averaged ROC curves of SMOTEBoost [31], RAMOBoost [18], and MWMOTE-Boost on real world datasets: (a) Libra. (b) Spambase. (c) Yeast.

both MWMOTE-Boost and RAMOBoost [18]. The number of boosting iteration, T is set to 20 for all methods. The other simulation parameters are similar to those used in experiments for MWMOTE (4). Tables 5.1 and 5.2 show the results obtained from the simulation experiments. The table shows the values of accuracy, precision, recall, F-measure, G-mean, and AUC performance measures. All these measures were obtained after 10-fold cross validation of the dataset. Figures 5.1-5.3 show the ROC graphs of MWMOTE-Boost, SMOTEBoost, and RAMOBoost algorithms over some representative datasets.

From the performance measures of Tables 5.1 and 5.2, we see that, MWMOTE-Boost outperforms each of SMOTEBoost and RAMOBoost in most of the datasets. In many of these datasets, the performance of MWMOTE-Boost is well over the performance of the other two methods. The reason is clearly due to the improved oversampling technique MWMOTE. The ROC graphs (Figs. 5.1-5.3) also show the better behavior of MWMOTE-Boost algorithm where ROC graphs of MWMOTE-Boost are well above the ROC graphs of other two methods. To compare the significance of performance difference, we evaluate Wilcoxon signed-rank significance test on AUC and the results are shown in Tables 5.3 and 5.4. The values of T are found to be 22 and 31 respectively, which are below the critical value (35). Therefore, the significance test also justifies MWMOTE-Boost's dominance over SMOTEBoost and RAMOBoost.

5.5 Summary

In this chapter, MWMOTE-Boost algorithm is proposed which uses MWMOTE oversampling technique inside AdaBoost.M2 ensemble algorithm in a manner similar to existing RAMOBoost algorithm [18]. Experiments are performed on real world data sets using neural network base classifier. The results show that MWMOTE-Boost outperforms existing RAMOBoost and SMOTEBoost in most of the data sets in several performance metrics. This is mainly the MWMOTE-Boost's oversampling procedure (MWMOTE) which helps it achieve better performance than RAMOBoost and SMOTEBoost algorithms.

Chapter 6

Conclusion

Many oversampling algorithms exist in literature which deal with imbalanced data sets by creating synthetic samples for the minority data set. In this paper, we identify the difficulties and insufficiencies that existing oversampling algorithms may face in many different scenarios of data samples. We propose a novel oversampling technique MWMOTE which tries to alleviate the problems of existing techniques. Our MWMOTE effecively selects a set of hard to learn minority samples from which synthetic data generation is required. MWOTE then adaptively weights these minority samples depending on its importance in data set. The weighted minority samples are then used to generate a better set of synthetic data samples. The data generation mechanism of MWMOTE is based on unsupervised clustering which ensures that the generated synthetic samples reside inside minority area, thus avoiding any wrong or noisy synthetic data generation observed by existing techniques. Finally, MWMOTE was integrated with AdaBoost.M2 algorithm in a manner similar to RAMOBoost [18] algorithm to provide a better stand-alone algorithm than existing RAMOBoost. The proposed algorithm MWMOTE-Boost tries to improve both the majority and minority class performance without sacrificing any majority class performance introduced by the oversampling technique.

Extensive experiments have been carried out to evaluate how well MWMOTE performs in different data sets in comparison with other oversampling algorithms. Some artificially

created data set and real world datasets were used for evaluating the performance. The experimental results show that, MWMOTE can outperform existing techniques in terms of a good number of overall performance measures such as accuracy, precision, F-measure, G-mean, and AUC. Finally, similar experiments were carried out to show the effectiveness of MWMOTE-Boost algorithm as a stand-alone classifier. The experiments shows that, MWMOTE-Boost algorithm can outperform the state-of-the-art RAMOBoost and SMOTEBoost algorithms in several performance measures.

6.1 Future Research

We can investigate several other research issues using MWMOTE. Firstly, the application of MWMOTE in multi-class problems. The objective will be how efficiently MWMOTE can be applied to multi class data sets. The second research issue is the consideration of nominal features with MWMOTE. In this paper, we have considered datasets with continuous features only. So, MWMOTE can be generalized to handle features of any type. Thirdly, 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. Fourthly, the MWMOTE oversampling technique can be integrated with some other undersampling methods existing in literature and we can investigate whether they together can give better results than single MWMOTE oversampling procedure. Fifthly, MWMOTE involves a number of parameters, which can be optimized for best performance depending on the specific problem at hand.

Bibliography

[1] H. He, and E. A. Garcia. Learning from imbalanced data. *IEEE Trans. Knowl. Data Eng.*, 21(10):1263-1284, 2009.

[2] P.M. Murphy, and D.W. Aha. UCI repository of Machine learning databases. University of California Irvine, Department of Information and Computer Science.

[3] D. Lewis,and J. Catlett. Heterogeneous Uncertainty Sampling for Supervised Learning. In *Proc. of the Eleventh International Conference of Machine Learning*, pages 148-156, 1994.

[4] T.E. Fawcett, and F. Provost. Adaptive Fraud Detection. *Data Mining and Knowledge Discovery*, 3(1):291-316, 1997.

[5] M. Kubat, R.C. Holte, and S. Matwin. Machine Learning for the Detection of Oil Spills in Satellite Radar Images. *Machine Learning*, 30:195-215, 1998.

[6] C. X. Ling, and C. Li. Data Mining for Direct Marketing: Problems and Solutions. In *International Conference on Knowledge Discovery & Data Mining*, 1998.

[7] N. Japkowicz, C. Myers, and M. Gluck. A Novelty Detection Approach to Classification. In *Proc. of the Fourteenth Joint Conference on Artificial Intelligence*, pages 518-523, 1995.

[8] S. Clearwater, and E. Stern. A rule-learning program in high energy physics event classification. *Comput. Phys. Commun.*, 67(2):159-182, 1991.

[9] G.M. Weiss. Mining with Rarity: A Unifying Framework. *ACM SIGKDD Explorations Newsletter*, 6(1):7-19, 2004.

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

[11] J.R. Quinlan. Induction of Decision Trees. *Machine Learning*, 1(1):81-106, 1986.

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

[13] N. Japkowicz and S. Stephen. The Class Imbalance Problem: A Systematic Study. *Intelligent Data Analysis*, 6(5):429-449, 2002.

[14] T. Jo and N. Japkowicz. Class Imbalances versus Small Disjuncts. *ACM SIGKDD Explorations Newsletter*, 6(1):40-49, 2004.

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

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

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

[18] S. Chen,H. He, E. A. Garcia. RAMOBoost: Ranked Minority Oversampling in Boosting. *IEEE Trans. Neural Networks*, 21(20):1624-1642, 2010.

[19] 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.

[20] J. Laurikkala. Improving Identification of Difficult Small Classes by Balancing Class Distribution. In *Proc. Conf. AI in Medicine in Europe: Artificial Intelligence Medicine*, pages 63-66, 2001.

[21] A. Estabrooks, T. Jo, and N. Japkowicz. A Multiple Resampling Method for Learning from Imbalanced Data Sets. *Computational Intelligence*, 20:18-36, 2004.

[22] X.Y. Liu, J. Wu, and Z.H. Zhou. Exploratory Under Sampling for Class Imbalance Learning. In *Proc. Int'l Conf. Data Mining*, pages 965-969, 2006.

[23] J. Zhang and I. Mani. KNN Approach to Unbalanced Data Distributions: A Case Study Involving Information Extraction. In *Proc. Int'l Conf. Machine Learning, Workshop Learning from Imbalanced Data Sets*, 2003.

[24] M. Kubat and S. Matwin. Addressing the Curse of Imbalanced Training Sets: One-Sided Selection. In *Proc. Int'l Conf. Machine Learning*, pages 179-186, 1997.

[25] X.Y. Liu, J. Wu, and Z.H. Zhou. Exploratory Under Sampling for Class Imbalance Learning. In *Proc. Int'l Conf. Data Mining*, pages 965-969, 2006.

[26] I. Tomek. Two Modifications of CNN. *IEEE Trans. System, Man, And Cybernetics*, 6(11):769-772, 1976.

[27] D.L. Wilson. Asymptotic Properties of Nearest Neighbor Rules Using Edited Data. *IEEE Trans. Systems, Man, And Cybernetics*, 2(3), 1972.

[28] 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*, 6(1):20-29, 2004.

[29] Y. Freund and R.E. Schapire. Experiments with a New Boosting Algorithm. In *Proc. Int'l Conf. Machine Learning*, pages 148-156, 1996.

[30] Y. Freund and R.E. Schapire. A Decision-Theoretic General- ization of On-Line Learning and an Application to Boosting. *J. Computer and System Sciences*, 55(1):119-139, 1997.

[31] N.V. Chawla, A. Lazarevic, L.O. Hall, and K.W. Bowyer. SMOTEBoost: Improving Prediction of the Minority Class in Boosting. In *Proc. Seventh European Conf. Principles and Practice of Knowledge Discovery in Databases*, pages 107-119, 2003.

[32] H. Guo and H.L. Viktor. Learning from Imbalanced Data Sets with Boosting and Data Generation: The DataBoost IM Approach. *ACM SIGKDD Explorations Newsletter*, 6(1):30-39, 2004.

[33] Y. Sun, M.S. Kamel, A.K.C. Wong, and Y. Wang. Cost-Sensitive Boosting for Classification of Imbalanced Data. *Pattern Recognition*, 40(12):3358-3378, 2007.

[34] W. Fan, S.J. Stolfo, J. Zhang, and P.K. Chan. AdaCost: Misclassification Cost-Sensitive Boosting. In *Proc. Int'l Conf. Machine Learning*, pages 97-105, 1999.

[35] T. Fawcett. ROC Graphs: Notes and Practical Considerations for Data Mining Researchers. Technical Report HPL-2003-4, HP Labs, 2003.

[36] T. Fawcett. An Introduction to ROC Analysis. *Pattern Recognition Letters*, 27(8):861-874, 2006.

[37] J. Demar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7(7):1-30, 2006.

[38] G. W. Corder, and D. I. Foreman, Nonparametric Statistics for Non-Statisticians: A step-by-Step Approach. New York: Wiley, 2009.

[39] *Critical Value Table of Wilcoxon Signed-Ranks Test [Online].* http://www.euronet.n1/users/warnar/demostatistiek/tables/WILCOXON-TABEL.htm.

[40] E. M. Voorhees. Implementing Agglomerative Hiararchic Clustering Algorithms for use in Document Retrieval. *Information Processing and Management*, 22(6):465-476, 1986.

[41] H. Schutze, C. Silverstein. Projections for Efficient Document Clustering. In *SIGIR' 97: Proc. of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 74-81, 1997.

[42] N. Japkowicz. Learning from imbalanced data sets: A comparison of variuos strategies. In *Proc. Learn. Imbalanced Data Sets*, Papers AAAI Workshop, pages 10-15, 2000,

[43] D. Opitz, and R. Maclin. Popular ensemble methods: An emperical study. *J. Artificial Intell. Res.*, 11:169-198, 1999.

[44] J. R. Quinlan, C4.5: programs for machine learning, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1993.

[45] *UC Irvine Machine Learning Repository.* http://archive.ics.uci. edu/ml/.

[46] J. Demsar. Statistical comparisons of classifiers over multiple data sets. *J. Mech. Learn. Res.*, 7(7):1-30, 2006.

[47] K. Woods, C. Doss, K. Bowyer, J. Solka, C. Priebe, and W. Kegelmeyer. Comparative Evaluation of Pattern Recognition Techniques for Detection of Microcalcifications in Mammogra- phy. *Int'l J. Pattern Recognition and Artificial Intelligence*, 7(6):1417-1436, 1993.

[48] N. Japkowicz. Class imbalance: Are we focusing on the right issue?. In *Proc. 12th Int. Conf. Mach. Learn., Workshop Learn. Imbalanced Data Sets II*, 2003.

[49] D. Mease, A.J. Wyner, and A. Buja. Boosted Classification Trees and Class Probability/Quantile Estimation. *J. Machine Learning Research*, 8:409-439, 2007.

[50] N.V. Chawla, N. Japkowicz, and A. Kolcz. Editorial: Special Issue on Learning from Imbalanced Data Sets. *ACM SIGKDD Explorations Newsletter*, 6(1):1-6, 2004.

[51] G.M. Weiss. Mining with Rarity: A Unifying Framework. *ACM SIGKDD Explorations Newsletter*, 6(1):7-19, 2004.

[52] M.A. Maloof. Learning When Data Sets Are Imbalanced and When Costs Are Unequal and Unknown. In *Proc. Int'l Conf. Machine Learning, Workshop Learning from Imbalanced Data Sets II*, 2003.

[53] Y. Sun, M.S. Kamel, A.K.C. Wong, and Y. Wang. Cost-Sensitive Boosting for Classification of Imbalanced Data. *Pattern Recognition*, 40(12):3358-3378, 2007.

[54] G.M. Weiss. Mining with Rarity: A Unifying Framework. *ACM SIGKDD Explorations Newsletter*, 6(1):7-19, 2004