Cluster-Based Minority Over-Sampling for Imbalanced Datasets

Kamthorn PUNTUMAPON\(^{(a)}\), Student Member, Thanawin RAKTHAMAMON\(^{(b)}\), Nonmember, and Kitsana WAIYAMAI\(^{(c)}\), Member

SUMMARY Synthetic over-sampling is a well-known method to solve class imbalance by modifying class distribution and generating synthetic samples. A large number of synthetic over-sampling techniques have been proposed; however, most of them suffer from the over-generalization problem whereby synthetic minority class samples are generated into the majority class region. Learning from an over-generalized dataset, a classifier could misclassify a majority class member as belonging to a minority class. In this paper a method called TRIM is proposed to overcome the over-generalization problem. The idea is to identify minority class regions that compromise between generalization and overfitting. TRIM identifies all the minority class regions in the form of clusters. Then, it merges a large number of small minority class clusters into more generalized clusters. To enhance the generalization ability, a cluster connection step is proposed to avoid over-generalization toward the majority class while increasing generalization of the minority class. As a result, the classifier is able to correctly classify more minority class samples while maintaining its precision. Compared with SMOTE and extended versions such as Borderline-SMOTE, experimental results show that TRIM exhibits significant performance improvement in terms of F-measure and AUC. TRIM can be used as a pre-processing step for synthetic over-sampling methods such as SMOTE and its extended versions. key words: imbalanced data, cluster-based minority over-sampling, synthetic minority over-sampling

1. Introduction

The class imbalance problem has been identified as one of the ten most challenging problems in data mining \([1]\). A dataset is considered imbalanced when its class distribution is skewed. In our work, we focus on the classification of binary classes, consisting of a minority and majority class. The minority class has a much smaller proportion of class samples compared to the majority class. Most traditional objective functions, such as accuracy, bias the number of class samples. Therefore, several methods are always biased toward the majority class. In other words, a classification method misclassifies a minority class member as belonging to the majority class.

A large number of methods to solve class imbalance have been proposed, and they can be classified into two categories: under-sampling and over-sampling. Synthetic over-sampling methods insert synthetic minority class samples to balance the distribution of the two classes, whereas under-sampling removes redundant majority class samples. The advantage of synthetic over-sampling is its generalization ability to produce a decision region larger and less specific toward the original dataset. Several synthetic over-sampling techniques \([2]-[7]\) have been proposed. However, almost all of them suffer from the over-generalization problem where synthetic minority class samples are generated in the majority class region. In the over-generalized situation, a classifier always misclassifies a large number of majority class samples without covering any additional minority class samples. The situation is more severe in extreme imbalanced datasets. For example, an imbalanced dataset is collected from a manufacturing line where the minority class represents defective products and the majority class represents good products. Only 1% misclassification of the majority class members (false positive rate) could cover several million good product records. Even if a classifier is able to reach a single-digit false positive rate, it may not be applied in real life.

SMOTE \([2]\) is the state of the art synthetic over-sampling algorithm. It generates synthetic data along the line segment between minority data members and their selected nearest neighbors using kNN. kNN could be considered as a simple criterion to avoid over-generalization. However, SMOTE still suffers from the over-generalization problem because its neighbor selection mechanism is performed without considering information from the majority class. This paper proposes a solution to the over-generalization problem in SMOTE and its extensions.

In \([8]\), we proposed a pruning based procedure for searching a precise and generalized region for synthetic minority over-sampling. This paper adds a second step called the Cluster Connection Step in order to enhance the generalization ability in oversampling minority class sample. The idea is to avoid over-generalization towards the majority class while maintaining the generalization of minority class concepts. To increase generalization, a method for merging minority class clusters to obtain more generalized clusters is proposed. All pairs of the minority class samples that can be connected between two minority class clusters are considered. Then, the best pairs are selected based on their proximity to the majority class samples, and are used to generate high-quality synthetic minority class samples. By connecting minority class clusters, the decision region of the minority class is enlarged and their generalization is increased. Thus, TRIM can be used as a pre-processing tech-
niques for over sampling methods [2]–[7] that are developed based on SMOTE. To evaluate performance of TRIM as a pre-processing method, we combine it with SMOTE and BSMOTE (an extended version of SMOTE). Experimental results show that with TRIM, we obtained comparable performance on 113 experiments, significant improvement on 76 experiments, and lower performance on 9 experiments in terms of F-measure.

2. Related Work

2.1 Over-Sampling Methods

Most of the synthetic minority over-sampling techniques [2]–[5], [9] are designed based on the Synthetic Minority Over-sampling TÉchnique (SMOTE) [2]. SMOTE employs k-nearest neighbors (k-NN) to select two minority class samples (source and its nearest neighbor). Then, a synthetic data value is randomly generated along the line segmented between the two minority class samples. That is, SMOTE uses only minority class samples to generate new synthetic data without considering the majority class samples. How SMOTE generates synthetic data can be interpreted as merging the two minority class samples into the same pattern. With two different pattern nearest neighbors, the synthetic data could be generated into the majority class region. The situation is more severe on sparse minority class datasets with a small number of samples. As a result, the decision regions of the output dataset are larger and less specific than the original dataset. That is, the minority class is generalized and even over-generalized. Several extensions of SMOTE have been developed to overcome this problem.

Borderline-SMOTE (BSMOTE) [3] uses the basic assumption that data nearby the decision boundary has greater chance to be misclassified than those far from the decision boundary. The authors’ further proposed a criterion to identify borderline samples based on k-NN. A minority class sample is identified as a borderline sample if it has sufficient number of majority class samples among its k-nearest neighbors. Only borderline samples and their neighbors are used to generate synthetic minority class samples. As a result, synthetic data is generated within the overlapped region between the two classes. Like SMOTE, Borderline-SMOTE suffers from over-generalization. The algorithm may cause over-generalization due to focused sampling within the overlapped area.

Margin-guided Synthetic Over-sampling (MSYN) [9] uses 1-NN margin to estimate generalization of the synthetic data. The algorithm sorts synthetic data having a large margin on both minority and majority classes. SMOTE is used to generate 10-fold of the synthetic minority class samples. The proportion of synthetic data is selected based on its margin. MSYN tends to generate synthetic data on a well separated region while avoiding the boundary region. Although MSYN can be used to avoid over-generalization, it uses all the features to select a synthetic sample. Many existing classification methods use several good feature sets to classify unseen data. In this paper, MSYN is used as a comparative method to pre-process data before using SMOTE to generate synthetic minority class samples.

Ordinal graph-based over-sampling (OGO-SP) [7] uses the k-NN algorithm to generate a graph between three ordinal classes in which edges and vertices respectively represent nearest neighbors and data samples. The k-NN graph is generated within the same class and inter-classes. The hypothesis is that the distance between two adjacent classes is lower than the distance between non-adjacent classes. Therefore, the shortest path algorithm Dijkstra is used to prune out some edges which are not shortest paths. Finally, the well-constructed graph is used to generate the new synthetic samples via using SMOTE. In this paper, we focus on the binary class classification instead of the ordinal classification.

Majority Weighted Minority Oversampling (MW-MOTE) [6] uses k-NN algorithm to identify both borderline majority class and hard-to-learn minority class samples. The borderline majority samples are located nearby a large number of minority class samples while the hard-to-learn minority class samples are located nearby the borderline majority samples. Agglomerative clustering technique is used for partitioning the minority class samples into multiple clusters. Both borderline majority class and hard-to-learn minority class samples are used to calculate weight of each minority class sample. Finally, based on weight and the minority class clusters, SMOTE is used to generate the synthetic minority class samples. In this work, the minority class samples are clustered based on the objective function that maximizes precision while maintaining adequate generalization. The samples located between two clusters can be used to generate synthetic minority class samples as well, if they are potential of being good minority class samples.

2.2 Under-Sampling Methods

Under-sampling methods balance the class distribution by removing some samples from the majority class. The problem with these methods is information lose. A large number of under-sampling methods [10]–[13] have been proposed to cope with the problem of information lose.

Cluster-based under-sampling [10] (CUS) method uses the K-mean to cluster data. For each cluster, a ratio of majority class sample is calculated. The higher the ratio, the greater will be the number of majority class samples to be contained in the majority class region. The ratio is used to determine the number of majority class samples which will be removed from the original dataset. The authors proposed both random and selective under-sampling strategies. The selective under-sampling strategy uses distance as a ranking criterion. However, CUS relies on the clustering algorithm which is not guaranteed to capture the majority class region.

One-sided sampling [11] (OSS) has been proposed to remove redundant and overlapped samples from the majority class. It starts by constructing a dataset that includes every minority class sample and a small subset of the major-
ity class samples. The initialized dataset is used to identify redundant majority class samples based on the 1-NN classification model. Each correctly classified majority class sample will be labeled as a redundant sample. The overlapped minority and majority class sample is identified using Tomek link [12]. Since Tomek link is used on every majority class sample, the algorithm is susceptible to noise in the minority class. Even with a single noise minority class sample, a majority class region can be removed.

3. Methodology

TRIM is a preprocessing method for synthetic oversampling algorithms. The design of TRIM is based on SMOTE. Our main objective is to redesign the selection process to generate synthetic minority class samples with higher quality than SMOTE. The basic idea is to search for clusters of minority class samples and compromise between generalization and precision.

Our proposed method is composed of two main steps: the minority class clustering and cluster connection.

Figure 1 shows the result of the minority class clustering step. Each cluster is represented with a dashed rectangle. In those clusters, most of the majority class samples are pruned out, while all the minority class samples are contained in some particular clusters.

Figure 2 shows the output of the cluster connection step. The shape of each minority class cluster is illustrated by a dashed line. Each connection between two cluster members is represented by a solid line. Synthetic data are then generated from each connected cluster and their solid connection lines. The main objective here is to avoid overgeneralization of the minority class concepts while maintaining their generalization.

3.1 Minority Class Clustering Step

In this step, the algorithm recursively splits the entire dataset into smaller clusters. At the beginning, the entire dataset is considered as one big cluster. Then, the cluster is split into a number of smaller clusters with respect to the proposed criterion, TRIM. This criterion is based on a cluster’s precision and generalization. Although both majority and minority class data can be contained in a cluster, most of the majority class samples are pruned. More specifically, majority class samples located outside a cluster are removed from the computation. Hence, in the further steps, only the minority class data inside each cluster are considered; we call this step minority class clustering.

For a given cluster, we define \(|\text{minority}||\) as the number of the minority class samples in the cluster, \(|\text{majority}||\) as the number of majority class samples, and \(N\) as the total number of data including both majority and minority class data so \(N = |\text{minority}|| + |\text{majority}||\). The precision of a given cluster is simply defined by Eq. (1)

\[
\text{precision} = \frac{|\text{minority}||}{N} \tag{1}
\]

To obtain precise minority class clusters while maintaining their generalization, we define an objective function, TRIM, using Eq. (2). The TRIM criterion is a product of precision (ratio of minority class data in a cluster) and generalization, \(|\text{minority}||\). The higher its value, the more precise and generalized a cluster is obtained.

\[
\text{TRIM} = \frac{|\text{minority}||^2}{N} \tag{2}
\]

In each dimension, a splitting point is identified as the midpoint between two adjacent data elements. For any specific splitting point, we define \(|\text{minority}_{\text{left}}||\) and \(|\text{minority}_{\text{right}}||\), as the number of minority data in the left and the right cluster, respectively. Similarly, \(N_{\text{left}}\) and \(N_{\text{right}}\) are the number of data in the left and the right cluster. With this notation, \(\text{TRIM}_{\text{split}}\) can be defined as

\[
\text{TRIM}_{\text{split}} = \max \left( \frac{|\text{minority}_{\text{left}}||^2}{N_{\text{left}}}, \frac{|\text{minority}_{\text{right}}||^2}{N_{\text{right}}} \right) \tag{3}
\]

In this step, the majority class data are filtered out.
While the TRIM criterion represents the performance of the cluster before splitting, TRIM\textsubscript{split} represents the performance of the split clusters. The higher TRIM split is, the better the split clusters are. This iterative splitting process continues until there is no better TRIM\textsubscript{split} than TRIM (TRIM\textsubscript{split} ≤ TRIM).

The algorithm uses a greedy approach similar to a traditional decision tree. Each attribute is independent from each other, and is considered one at a time. While the clustering algorithm focuses on splitting minority class data, it also prunes out majority class data if possible. Notice that the majority class sample is not in the calculation when it does not belong to the minority class cluster. Pseudo-code for the minority class clustering step is as follows.

**Algorithm** Minority Clustering(D)
**Input:** dataset (D), threshold (minPrecision)
**Output:** list of minority cluster (clusterList)

1. \( S = \{ \} \)
2. \( S.add(D) \)
3. while (\( S \) is not empty)
   4. \( clus = S.pop() \)
   5. \( clus = ConvexHullEstimation(clus) \)
   6. \( trim = ComputeTRIM(clus) \) // using Eq. (1)
   7. \( S = \) set of all splitting points in cluster \( clus \)
   8. \( trim\textsubscript{split} = max_{sp \in S} (TrimSplit(clus, sp)) \) // Eq. (3)
   9. if (\( trim\textsubscript{split} > trim \)) // the higher, the better
      10. split cluster \( clus \) to \( clus\textsubscript{left} \) and \( clus\textsubscript{right} \)
      11. \( S.add(clus\textsubscript{left}) \)
      12. \( S.add(clus\textsubscript{right}) \)
      13. else
      14. if precision(clus) > minPrecision
      15. clusterList.add(clus)
      16. end if
      17. end if
   18. end while
   19. return clusterList

**Function** ConvexHullEstimation (clus)

20. \( trim = ComputeTRIM(clus) \)
21. \( S = \) set of all splitting points in \( clus \) that subject to this constraint: \(|\text{majority}_{left}|=0 \) or \(|\text{majority}_{right}|=0 \)
22. \( trim\textsubscript{split} = max_{sp \in S} (ComputeTrimSplit(clus, sp)) \)
23. if (\( trim\textsubscript{split} > trim \))
24. split cluster \( clus \) to \( clus\textsubscript{left} \) and \( clus\textsubscript{right} \)
25. if \( (\text{majority}_{left} == 0) \),
26. \( clus = clus\textsubscript{right} \)
27. else
28. \( clus = clus\textsubscript{left} \)
29. end if
30. \( clus = ConvexHullEstimation(clus) \)
31. end if

The algorithm starts with a single cluster containing all the data points. It consists of two main steps which are Convex Hull Estimation and Cluster Splitting. The Convex Hull Estimation (CE) step iteratively splits data into left and right clusters. The purpose is to prune out the majority class samples located outside the minority cluster. To achieve this, a constraint is defined such that either left or right cluster must contain only the majority class samples. For every splitting point, the criterion TRIM\textsubscript{split} is calculated using the defined constraint. This is to ensure that one of the split cluster will contain all the minority class samples of its parent cluster. The majority class cluster is then removed from calculation. The other cluster proceeds to the next iteration which terminates when the stopping criterion TRIM\textsubscript{split} is not better than the previous iteration. The result is a rectangular cluster containing minority class samples. Detail of this step is shown in line 20-32.

The Clustering Splitting (CS) step is similar to the CE step without constraint. Therefore, minority class samples are split into left and right clusters. In this step, a minority class cluster (parent) is split into more precise clusters based on the TRIM criterion. Every splitting point is calculated and evaluated to ensure that each new cluster will have the TRIM\textsubscript{split} greater than its parent cluster (TRIM). At each splitting point, the algorithm will split a specific cluster having the highest TRIM\textsubscript{split}. This step takes as input a rectangular minority class cluster, and it generates two rectangular clusters, as output. The next iteration, the two split clusters will be returned to the CE step. When there is no better resulting clusters, the CS step returns the current cluster as output. The detail of CS step is shown in line 7-17.

In SMOTE, the synthetic samples are generated within the convex hull of minority class samples. With TRIM, a cluster shape can be a rectangle (in 2D) or a hyper-rectangle (in higher dimensions). Each cluster always covers the convex hull of its minority samples. Thus, TRIM can be used as a criterion to estimate the performance of the SMOTE algorithm via its rectangular shape clusters.

Although each minority class cluster is well-approximated, some clusters can be considered as noise. To filter out noise, a user-defined threshold on minimum precision (minPrecision) is used to prune noise clusters. The precision of each cluster is calculated using Eq. (1). Intuitively, if minPrecision is set too high, even good clusters may be considered as noise. However, if minPrecision is set too low, noise clusters with few minority class samples will be used for generating synthetic data in the next step. Note that minPrecision is fixed at 0.3 in all of our experiments.

In summary, the algorithm initially considers the entire dataset as one cluster. Then, it determines the best rectangular-shaped clusters by splitting the large clusters in two smaller ones if the TRIM criterion is improved. Improvement of this criterion is to ensure a better trade-off between precision and generalization.

Here, complexity analysis of the proposed algorithm is given. Recall that, the algorithm calculates TRIM on every splitting point. Take \( M \) as the number of dimensions or attributes, and \( N \) as the number of data points. For each dimension, the maximum number of splitting points is \( N - 1 \). In the extreme case where one data point is always split in every iteration, the maximum number of iterations for split-
ting is $N - 1$. Hence, the time complexity of the algorithm is $O(MN^2)$.

3.2 Cluster Connection Step

In the previous step, a minority class clustering algorithm was described. The algorithm clusters the minority class samples based on their precision and generalization. As a result of the minority class clustering, a cluster’s shape can be a rectangle (in 2D) or a hyper-rectangle (in higher dimension). However, when shape of a minority class pattern is complex, this step generates multiple small rectangular-shape clusters. Thus, loss of cluster generalization is unavoidable. It follows that the quality of the generated synthetic data is limited because they are generated from only minority class samples within the same cluster.

In this section, we propose an algorithm to connect two minority class samples from different clusters based on a proximity criterion. The proposed criterion is a proximity of a line segment connecting two minority class samples and a majority class sample. In SMOTE, a line segment connecting two minority class samples is used to generate synthetic data. Therefore, the proposed criterion represents the proximity of synthetic data and a majority class sample.

Given two base points, $A$ and $B$, a vertex point $C$, the angle $\theta$ between $\overrightarrow{CA}$ and $\overrightarrow{CB}$ can be expressed mathematically using the following Eq. (4).

$$\theta_c = \arccos \left( \frac{\overrightarrow{CA} \cdot \overrightarrow{CB}}{||\overrightarrow{CA}|| ||\overrightarrow{CB}||} \right)$$

(4)

Note that our algorithm handles both minority and majority class samples differently; the base points must be of the minority class, and the vertex point must be of the majority class. Figure 3 shows a toy example containing three data points: $A$, $B$, and $C$. The minority class samples ($A$ and $B$) and majority class sample ($C$) are shown as circles and rectangle, respectively.

The $\theta_c$ is an angle between vector $\overrightarrow{CA}$ and $\overrightarrow{CB}$. On the other hand, $\theta_c$ could be considered as the proximity of the majority class sample $C$ over the line segment connecting $A$ and $B$. In the example, the cosine of $\theta_A$ and $\theta_B$ could be considered as a proximity measure between the vector $\overrightarrow{AC}$ on $\overrightarrow{AB}$ and the vector $\overrightarrow{BC}$ on $\overrightarrow{BA}$, respectively. With $\theta_C = 2\pi - \theta_A - \theta_B$. Therefore, $\theta_C$ could be interpreted as the proximity of the vertex point over the line segment. The larger is $\theta_C$, the higher the proximity is between data point $C$ and the line segment connecting $A$ and $B$. $\theta_C$ is in the range $(0, \pi]$. The data point $C$ is on the line segment if and only if the $\theta_C$ is equal to $\pi$.

From Eq. (4), the criterion can be used to evaluate only a single majority class sample at a time. To evaluate a line segment and its majority class samples, the average $\theta$ of majority class samples located nearby the line segment is calculated. $k$-nearest of majority class neighbors are determined based on two minority class samples and a virtual sample. A virtual sample $V$ is defined as a point located in the middle of the line segment connecting two minority class samples. The set of majority nearest neighbors ($S$) represents the majority class samples located nearby the line segment. Two base minority class samples and one virtual sample are used to represent this line segment. The definition of the proximity, called $\text{AngleSim}$, of the line segment connecting two minority class sample ($A$ and $B$) and its nearby majority class sample is shown as

$$\text{AngleSim}_{A,B} = \frac{1}{|S|} \sum_{x \in S} \arccos \left( \frac{x \cdot \overrightarrow{AB}}{||x|| ||\overrightarrow{AB}||} \right)$$

(5)

Define $S$, a union of three sets: $k$-nearest of majority class sample of $A$, $B$, and $V$ (a virtual sample).

In summary, $\text{AngleSim}_{A,B}$ is the average angle between the selected majority class samples and the base minority class samples, $A$ and $B$. To allow two minority class samples from different clusters to be connected, the $\text{AngleSim}$ of the line segment connecting $A$ and $B$ must be lower than the average $\text{AngleSim}$ of either $A$ or $B$ to their nearest minority class neighbors within the same cluster. For example, the minority class sample $A$ is determined using $k$-nearest minority class neighbors in its cluster. The averaged $\text{AngleSim}$ is calculated on the line segments connecting $A$ and its minority class neighbors. Only line segments that can achieve better performance than the averaged $\text{AngleSim}$ will be used to generate new synthetic data.

The output from the previous step is the set of minority class clusters, $\text{clusterList}$. Define $\text{Cluster}(X)$ as a function that returns the cluster of a given minority class $X$.

**Algorithm** ClusterConnection($D$, $\text{clusterList}$)

**Input:** dataset ($D$), list of minority class cluster ($\text{clusterList}$)

**Output:** list of the allow-to-connect pairs of minority class sample ($\text{pairList}$)

1. for each $\text{minorityData}_i$ in $D$
2. count = 0; DiffClus = \{ \}
3. for each $nn$ of $k$-Nearest Neighbor of $\text{minorityData}_i$
4. if $\text{Cluster}(nn) == \text{Cluster}(\text{minorityData}_i)$
5. counter = counter + 1
6. sumAngle += $\text{SimAngle}($$\text{minorityData}_i$, $nn$)
7. $\text{pairList}.add(\text{pair}(\text{minorityData}_i, nn))$
8. else
9. DiffClus.add(nn)
10. end if
11. end for
12. avgAngle = sumAngle / counter;
13. while (DiffClus is not empty)
14. out_m = DiffClus.pop();
15. out_angle = SimAngle(minorityData_i, out_m)
16. if (out_angle < avgAngle)
17. pairList.add( pair(minorityData_i, out_m) )
18. end while
19. end for

Each pair of the minority class samples can be classified into two types: 1) same-cluster pair or 2) between-cluster pair (DiffClus). The same-cluster pair is a pair of minority class samples belonging to the same cluster. The between-cluster pair is a pair of minority class samples belonging to different clusters. The two types of minority class samples pairs will be used to generate the synthetic minority class samples. All the same-cluster pairs are used to generate the synthetic minority class samples. In case of a between-cluster pair, synthetic minority class samples are generated when its AngleSim is lower than the average AngleSim of either of the connecting minority class clusters. In a connecting minority class cluster, the average AngleSim of a cluster, AngleSim_{A_{kNN}} is calculated from its connecting minority class sample to its nearest minority class neighbors within the same cluster.

Given a sample A, define kNN_{A} as a set of minority class nearest neighbors that belong to the same cluster of A. Based on the Eq. (5), AngleSim_{A_{kNN}} can be defined as below:

\[ \text{AngleSim}_{A_{kNN}} = \frac{1}{|kNN_{A}|} \sum_{x \in kNN_{A}} \text{AngleSim}_{A,x} \quad (6) \]

More detailed explanation of the Eq. (6) is shown in the ClusterConnection pseudo code.

According to the Eq. (6), a between-cluster pair, A and B, can be used to generate minority class synthetic data, if AngleSim_{A,B} is lower than either AngleSim_{A_{kNN,A}} or AngleSim_{B_{kNN,B}}. This criterion ensures that the between-cluster pair has better performance than one of the connecting clusters in terms of AngleSim. To generate synthetic data, the SMOTE algorithm selects only pairs of minority class samples that satisfy the TRIM criteria. From these pairs, the synthetic data are randomly generated until the user-specified percentage of sampling is satisfied.

To summarize, the output of the algorithm is the set of the minority class pairs which are used to create new synthetic data.

Here, the complexity analysis of the cluster connection step is given. kNN is calculated on every minority class sample. Therefore, its time complexity is \(O(k \times |\text{minority}|)\). Therefore, the time complexity is \(O(k \times |\text{minority}| \times |\text{majority}|)\).

### 4. Experimental Results

In the following, the results of SMOTE, BSMOTE, MSYN, TRIM-SMOTE, TRIM-BSMOTE, and C4.5 (original dataset without sampling) are compared using 11 continuous datasets. All experiments were conducted using 10 random seeds on 10-fold cross validation with three levels of sampling, i.e., 100%, 300%, and 500%. AUC [14] and F-measure [15], are used to evaluate the performance. Each algorithm was used in at least 3,300 experiments. The experiments use WEKA C4.5 and SVM [16] as classification models with default configurations. For TRIM, minPrecision was set to 0.3; parameters for the other algorithms were set exactly as in their papers.

In the cluster connection step, k is selected based on the the sensitivity analysis given in Table 2. This table shows summation of AUC over ten datasets. Notice that when \(k \leq 5\), the overall performance is increased, and is insignificantly increased when \(k > 5\). Therefore, \(k\) is set to 5 due to the performance and time complexity.

In Table 1 the 11 UCI datasets [17] are sorted by percentage of minority class samples. The percentages vary from 3.94% to 34.8%. Each dataset has four attributes: dataset’s name, number of attributes (#Attributes), percentage of minority class sample (%Minority), and size of the dataset (Data size). All datasets are normalized using WEKA normalization [16] with the default configuration, i.e., weka.filters.unsupervised.attribute.Normalize -S 1.0 -T 0.0. Each dataset is transformed into a binary class version, i.e., minority and majority classes. The 1st-9th datasets have more than two classes, therefore, one class is considered as minority class, and the other classes grouped into the majority class. For example, yeast-ME3 contains two classes: ME3 and others.

Performance of C4.5, Support Vector Machine (SVM), MSYN, SMOTE, TRIM-SMOTE, BSMOTE, and TRIM-

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Experimental datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataset</td>
<td>#Attributes</td>
</tr>
<tr>
<td>letter-A</td>
<td>16</td>
</tr>
<tr>
<td>arrhythmia-6</td>
<td>280</td>
</tr>
<tr>
<td>ecoli-om</td>
<td>7</td>
</tr>
<tr>
<td>libras-10</td>
<td>90</td>
</tr>
<tr>
<td>glass-3</td>
<td>10</td>
</tr>
<tr>
<td>mfeat-fourier</td>
<td>16</td>
</tr>
<tr>
<td>yeast-ME3</td>
<td>9</td>
</tr>
<tr>
<td>breastTissue-fad</td>
<td>10</td>
</tr>
<tr>
<td>segment-path</td>
<td>19</td>
</tr>
<tr>
<td>haberman</td>
<td>3</td>
</tr>
<tr>
<td>pima indians diabetes</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2</th>
<th>Sensitivity analysis over ten UCI datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summation of F-measure</td>
<td>k=2</td>
</tr>
<tr>
<td>20.6</td>
<td>20.6</td>
</tr>
</tbody>
</table>
BSMOTE on the ten datasets in terms of F-measure are shown in Table 3. The original dataset without sampling (C4.5 and SVM) is used as a baseline to evaluate improved or decreased performance of the five algorithms. The bottom rows of the two tables provide a comparison of MSYN, SMOTE, and BSMOTE with the proposed algorithm TRIM-SMOTE and TRIM-BSMOTE. MSYN and SMOTE are compared with TRIM-SMOTE; whereas BMOTE is compared with TRIM-BSMOTE. Two classification techniques, i.e., C4.5 and SVM, are used to evaluate the performance of SMOTE and TRIM-SMOTE. The row labeled with (Win/Draw/Lose Significant) summarizes the number of cases where TRIM significantly outperforms, equals, or performs worse than the original sampling method. The ten-fold cross-validation with ten random seeds generate 100 value of F-Measure. To evaluate their statistical significance, F-Measure are sorted by their fold and random seeds. Wilcoxon signed rank test [18] with $p$-value less than 5% is evaluated on every cross validation result (100 value). The result is underlined when it significantly outperforms the original sampling method. A star (*) is displayed when the performance is worse than the original sampling method.

To evaluate performance of TRIM as a pre-processing method, we implement it on the two sampling methods, i.e., SMOTE and BSMOTE (extended version of SMOTE). The Wilcoxon signed-rank test is used to evaluate statistically significant performance of TRIM in terms of F-Measure. Detailed performance of TRIM and its comparative methods is given in Table 3.

Experimental results in terms of F-measure are given in Table 3. In 14 out of 66 experiments, the performance of TRIM-SMOTE with C4.5 and SVM exhibits significant improvement over SMOTE. Notice also that, in 12 out of 33 experiments, TRIM-BSMOTE is able to provide significant improvement over BSMOTE. These results show that TRIM is a pre-processing technique for over-sampling methods.

On overall AUC, TRIM-SMOTE shows comparable results with SMOTE. For C4.5, TRIM-SMOTE provides 6 significant improvements, 23 draw results, and 4 significantly lower performance, compared with SMOTE. For SVM, TRIM-SMOTE provides 8 significant improvements, 23 draw results, and 2 significantly lower performance, compared with SMOTE. However, compared with
Table 4 Experimental result in terms of Area Under Curve (AUC).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Sampling</th>
<th>C4.5</th>
<th>SVM</th>
<th>SMOTE</th>
<th>TRIM-SMOTE</th>
<th>SMOTE SVM</th>
<th>TRIM-SMOTE SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>letter-A</td>
<td>100%</td>
<td>96.6%</td>
<td>92.3%</td>
<td>96.9%</td>
<td>97.1%</td>
<td>97.3%</td>
<td>93.5%</td>
</tr>
<tr>
<td></td>
<td>300%</td>
<td>96.6%</td>
<td>92.3%</td>
<td>97.0%</td>
<td>97.3%</td>
<td>97.1%</td>
<td>93.5%</td>
</tr>
<tr>
<td></td>
<td>500%</td>
<td>96.6%</td>
<td>92.3%</td>
<td>97.1%</td>
<td>97.2%</td>
<td>97.4%</td>
<td>93.8%</td>
</tr>
<tr>
<td>arrhythmia-6</td>
<td>100%</td>
<td>80.8%</td>
<td>49.7%</td>
<td>82.0%</td>
<td>83.8%</td>
<td>84.9%</td>
<td>56.2%</td>
</tr>
<tr>
<td></td>
<td>300%</td>
<td>80.8%</td>
<td>49.7%</td>
<td>81.2%</td>
<td>79.5%</td>
<td>86.1%</td>
<td>61.5%</td>
</tr>
<tr>
<td></td>
<td>500%</td>
<td>80.8%</td>
<td>49.7%</td>
<td>81.7%</td>
<td>79.3%</td>
<td>86.6%</td>
<td>63.1%</td>
</tr>
<tr>
<td>ecoli-om</td>
<td>100%</td>
<td>84.6%</td>
<td>70.0%</td>
<td>85.1%</td>
<td>*88.5%</td>
<td>85.7%</td>
<td>91.9%</td>
</tr>
<tr>
<td></td>
<td>300%</td>
<td>84.6%</td>
<td>70.0%</td>
<td>85.1%</td>
<td>90.3%</td>
<td>88.8%</td>
<td>91.7%</td>
</tr>
<tr>
<td></td>
<td>500%</td>
<td>84.6%</td>
<td>70.0%</td>
<td>84.8%</td>
<td>89.2%</td>
<td>90.1%</td>
<td>91.6%</td>
</tr>
<tr>
<td>libras-10</td>
<td>100%</td>
<td>77.5%</td>
<td>54.1%</td>
<td>73.3%</td>
<td>76.7%</td>
<td>77.8%</td>
<td>59.7%</td>
</tr>
<tr>
<td></td>
<td>300%</td>
<td>77.5%</td>
<td>54.1%</td>
<td>72.3%</td>
<td>79.8%</td>
<td>79.9%</td>
<td>74.6%</td>
</tr>
<tr>
<td></td>
<td>500%</td>
<td>77.5%</td>
<td>54.1%</td>
<td>72.1%</td>
<td>72.2%</td>
<td>82.4%</td>
<td>81.3%</td>
</tr>
<tr>
<td>glass-3</td>
<td>100%</td>
<td>72.0%</td>
<td>50.0%</td>
<td>71.3%</td>
<td>74.9%</td>
<td>71.8%</td>
<td>50.0%</td>
</tr>
<tr>
<td></td>
<td>300%</td>
<td>72.0%</td>
<td>50.0%</td>
<td>62.1%</td>
<td>77.1%</td>
<td>74.3%</td>
<td>50.0%</td>
</tr>
<tr>
<td></td>
<td>500%</td>
<td>72.0%</td>
<td>50.0%</td>
<td>60.6%</td>
<td>73.4%</td>
<td>71.5%</td>
<td>50.0%</td>
</tr>
<tr>
<td>mfeat-fourier</td>
<td>100%</td>
<td>98.3%</td>
<td>99.7%</td>
<td>97.9%</td>
<td>98.0%</td>
<td>98.2%</td>
<td>99.7%</td>
</tr>
<tr>
<td></td>
<td>300%</td>
<td>98.3%</td>
<td>99.7%</td>
<td>97.5%</td>
<td>98.5%</td>
<td>98.3%</td>
<td>99.7%</td>
</tr>
<tr>
<td></td>
<td>500%</td>
<td>98.3%</td>
<td>99.7%</td>
<td>97.9%</td>
<td>98.3%</td>
<td>98.3%</td>
<td>99.7%</td>
</tr>
<tr>
<td>yeast-ME3</td>
<td>100%</td>
<td>87.0%</td>
<td>74.0%</td>
<td>88.5%</td>
<td>*90.0%</td>
<td>89.4%</td>
<td>85.7%</td>
</tr>
<tr>
<td></td>
<td>300%</td>
<td>87.0%</td>
<td>74.0%</td>
<td>88.2%</td>
<td>90.6%</td>
<td>90.0%</td>
<td>88.4%</td>
</tr>
<tr>
<td></td>
<td>500%</td>
<td>87.0%</td>
<td>74.0%</td>
<td>88.9%</td>
<td>*91.7%</td>
<td>90.8%</td>
<td>88.1%</td>
</tr>
<tr>
<td>breastTissue</td>
<td>100%</td>
<td>64.4%</td>
<td>50.0%</td>
<td>63.9%</td>
<td>66.1%</td>
<td>66.5%</td>
<td>50.0%</td>
</tr>
<tr>
<td></td>
<td>300%</td>
<td>64.4%</td>
<td>50.0%</td>
<td>67.4%</td>
<td>66.4%</td>
<td>67.1%</td>
<td>*73.9%</td>
</tr>
<tr>
<td></td>
<td>500%</td>
<td>64.4%</td>
<td>50.0%</td>
<td>69.2%</td>
<td>*70.3%</td>
<td>66.8%</td>
<td>78.9%</td>
</tr>
<tr>
<td>segment-path</td>
<td>100%</td>
<td>99.6%</td>
<td>99.8%</td>
<td>99.6%</td>
<td>99.6%</td>
<td>99.6%</td>
<td>99.8%</td>
</tr>
<tr>
<td></td>
<td>300%</td>
<td>99.6%</td>
<td>99.8%</td>
<td>99.7%</td>
<td>99.7%</td>
<td>99.7%</td>
<td>99.8%</td>
</tr>
<tr>
<td></td>
<td>500%</td>
<td>99.6%</td>
<td>99.8%</td>
<td>99.7%</td>
<td>99.7%</td>
<td>*98.8%</td>
<td>99.6%</td>
</tr>
<tr>
<td>haberman</td>
<td>100%</td>
<td>61.0%</td>
<td>50.0%</td>
<td>63.7%</td>
<td>65.8%</td>
<td>66.5%</td>
<td>58.7%</td>
</tr>
<tr>
<td></td>
<td>300%</td>
<td>61.0%</td>
<td>50.0%</td>
<td>63.8%</td>
<td>64.9%</td>
<td>65.8%</td>
<td>50.9%</td>
</tr>
<tr>
<td></td>
<td>500%</td>
<td>61.0%</td>
<td>50.0%</td>
<td>64.4%</td>
<td>64.4%</td>
<td>64.2%</td>
<td>50.0%</td>
</tr>
<tr>
<td>Pima Diabetes</td>
<td>100%</td>
<td>73.1%</td>
<td>71.7%</td>
<td>71.6%</td>
<td>71.1%</td>
<td>72.2%</td>
<td>72.5%</td>
</tr>
<tr>
<td></td>
<td>300%</td>
<td>73.1%</td>
<td>71.7%</td>
<td>71.6%</td>
<td>69.9%</td>
<td>71.5%</td>
<td>67.8%</td>
</tr>
<tr>
<td></td>
<td>500%</td>
<td>73.1%</td>
<td>71.7%</td>
<td>72.1%</td>
<td>69.9%</td>
<td>71.5%</td>
<td>67.8%</td>
</tr>
</tbody>
</table>

Win/Draw/Lose Significant 18/14/1 25/7/1 16/16/1 6/23/4 - 8/232 - 11/220 -

BSMOTE, TRIM-BSMOTE provides better results. It provides 11 significant improvements and 22 draw results. Utilized with SMOTE and its extended versions, it yields most comparable performance in terms of AUC, with higher performance in terms of F-measure. For some situations, the proposed TRIM method performed worse than the existing methods. This is due to two possible reasons. The first reason is from the random step of SMOTE in generating new synthetic samples. As result, F1 and AUC are fluctuating having a big deviation between the maximum and minimum values. The second reason is that information about minority class is omitted during the minority class clustering step, using static minPrecision (0.3). These experimental results indicate that, of the two sampling algorithms, SMOTE and BMOTE, TRIM gives 26.2% significant improvements and 71.7% comparable performances in terms of F-measure. We observe 68.6% comparable performance, 25.2% significant improvement in terms of AUC. TRIM-SMOTE shows improvement over C4.5 and SVM in terms of precision (F-Measure) while maintaining comparable AUC.

5. Conclusions

Over-generalization is one of the most important problems in synthetic minority over-sampling. A majority class member can be misclassified as belonging to the minority class, when the majority class region is confused with that of minority class. To overcome over-generalization, we propose the TRIM algorithm as a pre-processing step for synthetic over-sampling methods. To evaluate its performance, TRIM is implemented as a pre-processing method for existing synthetic over-sampling techniques SMOTE, BMOTE, and MSYN. Experimental results show that performance of SMOTE, BMOTE, and MSYN are significantly increased in terms of F-measure. TRIM’s performance is stable on most of the datasets in terms of AUC. For datasets, in which classes are well separated such as the Segment-path datasets, TRIM is able to maintain classification accuracy in terms of F-measure. For complex datasets, in which over-generalization occurs such as the Arrhythmia-6 and Haberman datasets, TRIM significantly improves the performance. Thus, the experimental results indicate that TRIM
can be used for both well separated and complex datasets with over-generalization.

Acknowledgements

We would like to thank James Edward Brucker for his comments and suggestions, which helped us to considerably improve the presentation of this paper.

References


Kanthorn Puntumapon received his B.Eng. in Computer Engineer from King Mongkut’s Institute of Technology Ladkrabang, Thailand in 2005 and the M.Eng. in Kasetsart University, Thailand in 2007. He is currently a Ph.D. candidate in Computer Engineering at Kasetsart University, Thailand. His main research interests include intelligent traffic systems, data mining, and information retrieval.

Thanawin Rakthanmanon is an assistant professor at the Department of Computer Engineering, Kasetsart University, Thailand. He earned a Ph.D. in Computer Science and Engineering from the University of California, Riverside under the supervision of Prof. Eamonn Keogh. His research interests are in time series data mining and big data analysis. One of his papers won the SIGKDD2012 best research paper award.

Kitsana Waiyamai is an associate professor at the Department of Computer Engineering, Kasetsart University, Thailand. His research publications in both international journals and conference proceedings are on the design and development of efficient data mining algorithms and their applications. His current research interests include complex data mining, machine learning, and bioinformatics.