Synthetic minority oversampling technique for multiclass imbalance problems

Tuanfei Zhu[a], Yaping Lin[a,*], Yonghe Liu[b]

[a] College of Information Science and Engineering, Hunan University, 116 Lu Shan South Road, Changsha, China
[b] Department of Computer Science & Engineering, University of Texas at Arlington, 701 S Nedderman Dr, Arlington, U.S.

ARTICLE INFO

ABSTRACT

Multiclass imbalance data learning has attracted increasing interests from the research community. Unfortunately, existing oversampling solutions, when facing this more challenging problem as compared to two-class imbalance case, have shown their respective deficiencies such as causing serious over generalization or not actively improving the class imbalance in data space. We propose a k-nearest neighbors (k-NN)-based synthetic minority oversampling algorithm, termed SMOM, to handle multiclass imbalance problems. Different from previous k-NN-based oversampling algorithms, where for any original minority instance the synthetic instances are randomly generated in the directions of its k-nearest neighbors, SMOM assigns a selection weight to each neighbor direction. The neighbor directions that can produce serious over generalization will be given small selection weights. This way, SMOM forms a mechanism of avoiding over generalization as the safer neighbor directions are more likely to be selected to yield the synthetic instances. Owing to this, SMOM can aggressively explore the regions of minority classes by configuring a high value for parameter k, but do not result in severe over generalization. Extensive experiments using 27 real-world data sets demonstrate the effectiveness of our algorithm.

© 2017 Published by Elsevier Ltd.

1. Introduction

1.1. Motivation

The imbalanced data learning is an important type of classification problems in machine learning and data mining, where certain classes (minority classes) possess much fewer instances than other classes (majority classes) while the minority classes often remain the primary interest. As most standard learning algorithms seek the minimization of overall training errors, skewed class distribution could result in that the resulting classifier provides unfavorable prediction performance for the minority classes. Over the past decade, a number of solutions have been proposed to deal with the imbalanced learning problem [1–6], most of which are inherently designed for two-class imbalance. However, two class cannot cover all scenarios in real world applications, where multiclass data with imbalance tends to occur more often. Compared with the two-class scenario, the problem of learning from multiclass imbalanced data is more difficult. This is not only because the higher number of classes increases the concept complexity of data, but more because

the possible existence of multiple minority classes (termed multiminority) and multiple majority classes (termed multimajority) complicates the situation of imbalance [7].

In this paper, we focus our attention on data oversampling method which is devoted to rebalance original skewed data by introducing new minority class instances to combat multiclass imbalance problems. Data oversampling method addresses the imbalance problem at the most fundamental level, and is very popular in the existing literature [1,8]. Its main advantage is being versatile as it is independent of any specific classifiers. However, when handling multiclass imbalanced data, oversampling techniques face new challenges not present in two-class scenario. First, multiclass imbalanced data might have multimajority. The minority class instances are easier to be ignored by the learning algorithm as a result of the existence of multiple strong concepts from the majority classes. To better recognize the minority class, oversampling techniques should possess the capability that can greatly broaden and strengthen the region of minority class. Second, multiclass imbalanced data may have multiminority, oversampling techniques should produce as little negative impacts as possible in the process of oversampling because the negative impacts can be amplified due to the existence of multiple minority classes.

Therefore, to better handle multiclass imbalance problems with data oversampling, we want to design an oversampling algorithm

* Corresponding author.
E-mail addresses: zhutuanfei@hnu.edu.cn (T. Zhu), yplin@hnu.edu.cn, yplinhn@163.com (Y. Lin), yonghe@cse.uta.edu (Y. Liu).

that can aggressively broaden the regions of minority classes to better protect the minority class instances while decreasing significantly the negative effects in the oversampling process of minority classes. Below, we first detail the insufficient and inadaptability of previous oversampling algorithms in terms of handling multiclass imbalanced data, followed by explanation of our method.

1.2. Limits of previous oversampling techniques

The most simple oversampling method is Random OverSampling (ROS) which yields new instances by randomly replicating original instances. In multiclass imbalanced data, the data space of one minority class may be invaded by multiple majority classes. It is more likely that minority class instances sparsely intersperse in the instances of other classes. In order for the learning algorithms to not create very specific and error-prone decision regions for minority classes (i.e., overfitting), newly introduced instances may need to actively expand the regions of minority classes and mitigate the class imbalance in data space [7]. However, ROS is unable to reduce this kind of imbalance as its new instances come from the replication of original instances.

Synthetic Minority Oversampling TechniquEs (SMOTE) [2] was proposed for enlarging the region of minority class by generating synthetic instances in feature space. In SMOTE, the generation of a synthetic instance can be described as:

$$x^{syn} = x^i + (x^j - x^i). * \gamma \tag{1}$$

where x^i is the minority class instance under consideration, x^j is a instance randomly selected from the k-nearest minority neighbors of x^i; γ is a vector in which each element is a random number from [0, 1]; the symbol ".*" denotes element-wise multiplication. It can be observed that generating a synthetic instance refers to two original instances. For convenience, the instances like x^i and x^j are respectively called primary reference instance (PRI) and **assistant reference instance (ARI)** in this paper. According to (1), x^{syn} is created along the line segment joining x^i and x^j, and the direction of line segment relative to x^i is determined by x^j. Since x^j belongs to the k-nearest minority neighbors of x^i, x^{syn} will be only created in the directions of k-nearest minority neighbors of x^i. Hence, the generation method above is called k-nearest neighbor-based (k-NN-based) method [8].

Although SMOTE decreases the risk of overfitting [2], it can raise the problem of over generalization which the considered minority class area is generalized to the region of majority class due to the generation of wrong synthetic instances [9]. When SMOTE is applied to multiclass imbalanced data, over generalization may become a more serious problem as wrongful generalization of one minority class can also happen in the regions of other minority classes to harm the learnability of minority class instances.

Following the footsteps of SMOTE, many oversampling algorithms were developed such as ADASYN [10], RAMO [4], Safe-level-SMOTE (SL-SMOTE) [11] and MWMOTE [8]. Most of these approaches are designed to handle two-class imbalance. If they are used to handle multiclass imbalanced data, either certain class decomposition technique or class transformation is needed for converting multiclass problem into several two-class tasks. One of the most popular decomposition schemes is One-against-all (OAA) [12]. In OAA, a k-class multiclass problem is converted into k two-class subproblems. Each subproblem is constructed by using the instances of class c_i $(i = 1 \ldots k)$ as the positive instances and the instances of other classes as the negative instances. For class transformation, all of the classes except the considered class are incorporated into the negative class for forming a two-class task [3,13]. When tackling multiclass imbalanced data, the difference of using this way and OAA decomposition is that the former merges all new instances generated for each class together and constructs a

classifier, the latter will oversample and train on each two-class subproblem separately. However, regardless of class decomposition and transformation, the resulting two-class tasks have to lose some information of original multiclass data (e.g., for both OAA decomposition and the class transformation, the label information of $k - 2$ classes will be lost in each two-class task) [7,12]. Hence, the generated instances based on the two-class tasks cannot adequately embody or fit the distribution of the whole multiclass data.

Recently, a Mahalanobis Distance-based Over-sampling technique (MDO) [3] was specially proposed for multiclass imbalance problems. The main idea of MDO is to preserve the covariance structure of the minority class instances and reduce the risk of class overlapping. In order to reduce the overlapping between different class regions, MDO only generates the synthetic instances in dense areas of minority class regions. This can actually result in that the borderline minority class instances will not be protected.

Based on the above analyses, for handling multiclass imbalanced data, existing oversampling algorithms either may produce more severe negative impacts such as SMOTE, or have limited ability to protect minority class instances well (e.g., ROS and MDO). In addition, a common deficiency for them is that the generation of new instances is not based on the whole multiclass imbalanced data to consideration.

1.3. Our method

Thereby motivated, we design a k-NN-based Synthetic Minority Oversampling algorithm to combat Multiclass imbalance problems (SMOM). Unlike previous k-NN-based oversampling where the synthetic instances are generated in the directions of k-nearest neighbors randomly, SMOM, for each neighbor direction, gives a corresponding selection weight which represents how much probability it is used as the direction of generating synthetic instances. The assignment of selection weight for a neighbor direction mainly considers the over-generalization factor that reflects the potential negative effects to other classes (especially to other minority classes) if the synthetic instances are generated along this neighbor direction. Those neighbor directions causing serious over generalization will be given small selection weights. By this way, SMOM can establish a mechanism of avoiding over generalization.

Owing to this, SMOM, on one hand, decreases the risk of introducing noisy synthetic instances into the regions of incorrect minority classes. The minority classes can be better modeled as the overlapping among minority classes is restricted. On the other hand, it is practicable that adopting a high value of k for SMOM to dramatically expand minority class regions. The minority class instances can be better protected. We evaluate our SMOM over 27 multiclass imbalanced data sets with four different classifiers. The experiment results show that SMOM is superior than other compared oversampling algorithms in terms of MG, MAUC, and the recall on the smallest minority class.

The remainder of this paper is organized as follows: A brief review on related works is provided in Section 2. Section 3 presents SMOM in detail. The experimental study is shown in Section 4. Section 5 concludes the paper.

2. Related works

Although a large number of research works in class imbalance problems have been done, we only provide a brief overview for those works involved into oversampling as data oversampling is our interest in this paper.

Between-class imbalance, within-class imbalance and class overlapping are three main factors for causing the difficulty of learning from imbalanced data [1,14,15]. All oversampling algorithms can address the between-class imbalance by replicating
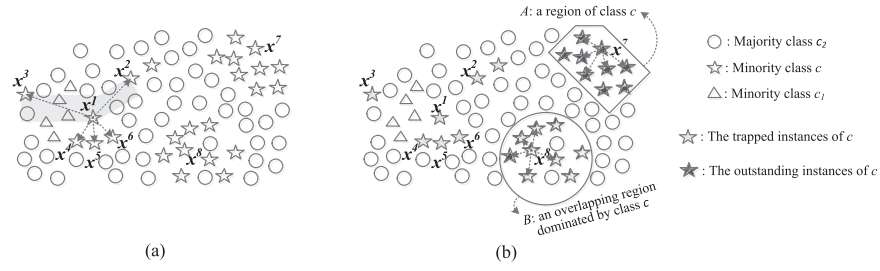
Fig. 1. An imbalanced data with three classes. (a) x^1's k-nearest neighbors from class c are shown by arrow ($k = 5$). (b) Two local regions, A and B, dominated by class c.

original minority class instances or adding new synthetic minority instances. Some of them are also capable of dealing with the within-class imbalance. These algorithms either use clustering techniques to find the sub-clusters of minority class so that greater oversampling levels can be imposed on the instances from those smaller sub-clusters [16], or measure the difficulty of learning for each minority instance by considering its local characteristics so as to assign higher oversampling weights into the instances with higher difficulty levels [4,8,13]. The class overlapping is a special factor with regard to using oversampling techniques to combat imbalance problems. On one hand, oversampling can enhance the information expression of minority class, but on the other hand the new overlapping may be manufactured in the non-overlapping regions of original data as a result of the generation of noisy synthetic samples [3,8,17], which is harmful to constructing the prediction model with good generalization ability. There are two ways to alleviate the class overlapping brought from oversampling. The first way is to apply data editing technologies into the oversampled data for removing the possible noisy and overlapped instances [17,18]. The second way is to avoid the generation of noisy and wrong synthetic instances in advance. e.g., in [8,13], the minority class instances are divided into several small clusters, the synthetic instances are only allowed to be created within such clusters; in paper [3], the synthetic instances are only generated for the instances which settle in the dense minority class areas.

Recently, multiclass imbalance problems have received increasing attention. A few of oversampling techniques that can directly handle multiclass imbalanced data were proposed. Lin et al., developed a dynamic SMOTE radial basis function algorithm, DSRBF. DSRBF uses a memetic algorithm for optimizing radial basis neural networks. The class which has the worse accuracy in the different generations of memtic algorithm is oversampled by SMOTE to improve the sensitivity of minority classes. Fernández-Navarro et al. [6] designed a dynamic sampling algorithm based on multilayer perceptrons (MLP), DyS. In DyS, the minority instances will have higher probabilities to be selected to train MLP. In this way, the resulting MLP classifier is biased towards the minority classes. However, both DSRBF and DyS depend on specific learning algorithms.

3. SMOM: a *k*-NN-based synthetic minority oversampling algorithm for multiclass imbalance problems

Previous k-NN-based oversampling algorithms randomly select ARI from the k-nearest minority neighbors of PRI when yielding the synthetic instances for PRI. Fig. 1a shows an imbalanced distribution with three classes. Assuring the minority class c is being considered to oversample, and x^1 is PRI. x^2, x^3, x^4, x^5 and x^6 are x^1's k-nearest neighbors from class c ($k = 5$). According to (1), the two shaded rectangular regions in the neighbor directions $\vec{x^1 x^2}$ and $\vec{x^1 x^3}$ are respectively these two directions' generalization regions in which the synthetic instances can appear. Obviously, se-

lecting which one of neighbor directions is equivalent to filling the synthetic instances in which one of corresponding generalization regions. If the synthetic instances are created along the neighbor directions $\vec{x^1 x^2}$ and $\vec{x^1 x^3}$, they will erroneously fall in the regions not belonging to class c, resulting in over generalization.

To address this problem, our SMOM for the neighbor direction computes a corresponding selection weight to be the generation direction of synthetic instances. The computation of selection weight for a neighbor direction mainly considers the over generalization factor that measures the possible negative impacts for all other classes when the synthetic instances are created along this neighbor direction. The neighbor directions that may cause serious over generalization will obtain low selection weights.

Consider the Fig. 1a again, as the neighbor directions $\vec{x^1 x^2}$ and $\vec{x^1 x^3}$ can respectively introduce the synthetic instances towards the regions of class c_2 and c_1, causing the negative effects for the learning of these two classes, SMOM will assign lower selection weights to $\vec{x^1 x^2}$ and $\vec{x^1 x^3}$ than the other neighbor directions of x^1. Hence, SMOM has the ability to avoid over generalization. This advantage can also make our SMOM to refer to more neighbors for the selection of ARI without having to concern the problem of over generalization, which helps to yield more informative synthetic instances.

For some minority class instances, however, computing the selection weights for their directions of k-nearest neighbors can be omitted. This can be understood from Fig. 1b in which the surroundings of x^7 and x^8 primarily distribute the instances from class c. It is reliable that whichever their neighbor directions is used to generate synthetic instances. In fact, most regions may be mainly occupied by the instances from a single class even though there are multiple concepts of classes in the whole feature space. Consider Fig. 1b where the regions A and B are dominated by class c. The c class instances located in A and B are just like the instances x^7 and x^8, which a considerable number of neighbors from class c are distributed nearby them without crossing the regions of other classes. We, in this paper, term those c class instances located in the region dominated by class c as the outstanding instances of class c since they stand out in comparison with its neighbouring other classes instances. The other instances of class c are called the trapped instances of class c as they seem to be trapped in their neighbouring instances from other classes.

Based on the above reason, we designed a neighborhood-based clustering algorithm (i.e., NBDOS) to find the outstanding instances of a class, the remaining instances naturally constitute the trapped instances set of this class.

In summary, when oversampling class c, the main process of SMOM can be described as follows (the complete process is presented in Algorithm 1):

1) Dividing the instances set of class c, S^c, into the outstanding instances set Oi^c and the trapped instances set Ti^c by using NBDOS .

Algorithm 1 SMOM(D, c, ζ, $k1$, $k2$, rTh, nTh, $w1$, $w2$, $r1$, $r2$).

Input: D: The whole multiclass imbalanced data; c: The minority class under consideration; ζ: Number of synthetic instances to be generated; $k1$: Number of nearest neighbors used to generate the synthetic instances; $k2$, rTh, and nTh: The parameters used in the clustering algorithm NBDOS; $w1$, $w2$, $r1$, and $r2$: The parameters used for calculating the selection weights;

Output: SI: The generated synthetic instances

1: Obtain the instances of class c, S^c, and the instances of other classes, $S^{\tilde{c}}$, from D respectively.

2: For each instance $x^i \in S^c$:

   1) Find the nearest $k3$ instances from x^i, $x^i.N^c_{k3}$, in S^c, where $k3 = max\{k1, k2\}$; record the distance between x^i and its $k1$th nearest instance as $x^i.r_{k1}$, and obtain the nearest $k1$ instances from x^i, $x^i.N^c_{k1}$, in $x^i.N^c_{k3}$.

   2) Find the nearest $k3$ instances from x^i, $x^i.N^{\tilde{c}}_{k3}$, in $S^{\tilde{c}}$; during this process, choose out those instances, T, from $S^{\tilde{c}}$ which the distances of they to x^i are no larger than $x^i.r_{k1}$, then construct $x^i.Fs \leftarrow T \cup x^i.N^c_{k1}$. For all instances of $x^i.Fs$, save their indexes and corresponding distances from x^i to $x^i.Fd$ and $x^i.Fi$, respectively.

   3) Find the $k2$-nearest neighbors set of x^i, $x^i.N_{k2}$, in the union of $x^i.N^c_{k3}$ and $x^i.N^{\tilde{c}}_{k3}$.

3: Obtain the cluster labels of instances of S^c, $S^c.cl$, by using the clustering algorithm NBDOS, i.e., $S^c.cl \leftarrow NBDOS(S^c, k2, S^c.N_{k2}, rTh, nTh)$.

4: Divide S^c into the outstanding instances Oi^c and the trapped instances Ti^c: For each instance $x^i \in S^c$, If $x^i.cl \neq 0$, $Oi^c \leftarrow Oi^c \cup \{x^i\}$; otherwise, $Ti^c \leftarrow Ti^c \cup \{x^i\}$.

5: For each instance $x^i \in Ti^c$, computing the selection weights for its $k1$ neighbor directions, $x^i.sw$: For each $x^j \in x^i.N^c_{k1}$

   1) If $x^j \in Oi^c$ and $x^i \in x^j.N^c_{k1}$, then $x^i.sw(x^j) \leftarrow 1 + w1/e$

   2) If $x^i \in x^j.N^c_{k1}$, and $x^j.sw(x^i)$ has been computed, then $x^i.sw(x^j) \leftarrow x^j.sw(x^i)$

   3) Otherwise, according to $x^i.Fi$ and $x^i.Fd$, choose outthe instances, $x^i.Ss(x^j)$, from $x^i.Fs$ which the distances to x^i are no larger than the distance between x^i and x^j, then call dPN to obtain all of the instances located in $x^i.PN(x^j)$, i.e., $N_{PN} \leftarrow dPN (x^i, x^j, x^i.Ss(x^j))$. The selection weight $x^i.sw(x^j)$ is obtained based on N_{PN}, $w1$, $w2$, $r1$ and $r2$ by using (2) in Section 3.2.1.

6: For each instance $x^i \in Ti^c$, obtain the probability distribution in terms of selecting its neighbor directions, $x^i.P$:

   1) For all $k1$ neighbor directions of x^i, if there is no PN neighborhood only containing the instances of class c, add x^i itself to $x^i.N^c_{k1}$ and assign $x^i.sw(x^i) \leftarrow 1 + w1/e$.

   2) For each instance $x^j \in x^i.N^c_{k1}$, $x^i.P(x^j) \leftarrow x^i.sw(x^j)/\sum_{x^l \in x^i.N^c_{k1}} x^i.sw(x^l)$.

7: For any $x^i \in S^c$, obtain the number of synthetic instances to be generated for it (denoted by $x^i.g$) by allocating ζ into the members of S^c as evenly as possible.

8: Generate the synthetic instances set SI. For each $x^i \in S^c$:

   1) If $x^i \in Ti^c$, a neighbor instance x^j is selected from $x^i.N^c_{k1}$ in the weighted sampling way according to the probability distribution $x^i.P$. Otherwise, x^j is randomly selected from $x^i.N^c_{k1}$.

   2) Using formula (1) to create a synthetic instance si, $si \leftarrow x^i + (x^j - x^i).*\gamma$, then $SI \leftarrow SI \cup \{si\}$.

   3) repeat 1) and 2) $x^i.g$ times.

2) Acquiring the selection weights for the neighbor directions. For each instance of Ti^c, computing the selection weight for each of its k neighbor directions. For each instance of Oi^c, the selection weights in all of its k neighbor directions are simply assumed to be equal.

3) Generating the synthetic instances for class c. For any instance of class c, its k neighbor directions are selected as the direction of generating synthetic instances according to their respective selection weights.

3.1. Performing the clustering algorithm NBDOS to divide the instances

3.1.1. Motivation

The motivation of NBDOS is to find those c class instances located in the regions dominated by class c, i.e., Oi^c. It can make all instances of Oi^c to avoid the computation of selection weights for their neighbor directions. The dominated regions of class c may be separated from each other, each such region corresponds to a cluster of outstanding instances containing all c class instances falling into this region. All instances outside any cluster of outstanding instances naturally constitute the clustering of the trapped instances of class c, i.e., Ti^c.

It is worth pointing out that many existing oversampling approaches also embedded clustering techniques in their algorithmic processes. For these approaches, there are generally two motivations behind the application of clustering. One is to identify meaningful sub-clusters (/sub-concepts) for a single class [5,16,19]. This kind of oversampling algorithms is usually dependent on an accurate result of clustering as they aim to solve the within-class imbalance (/disjuncts) by imposing different operations for the different sub-concepts (/clusters) [5,16]. In SMOM, NBDOS is just used to find the outstanding instances of the considered class. It needs not to confront the challenge of accurately discovering sub-concepts. Another one is to partition the data space of minority class into small pieces for reducing the generation of wrong synthetic instances [8,13]. The oversampling methods motivated by this often adopt the agglomerative clustering algorithms with high time complexity. The resulting clusters are usually the hyperspherical shape having size limitation. Our NBDOS which is actually derived from the well-known density-based clustering DBSCAN [20] can discover the arbitrary shaped clusters with good efficiency.

3.1.2. NBDOS: neighborhood-based clustering for discovering the clusters of outstanding instances

NBDOS is similar with DBSCAN in many regards. The biggest difference between them is that DBSCAN is conducted on the whole data to detect the data objects above a certain density level, and NBDOS is performed on one-class data to discover the instances that lie inside the data regions dominated by this class. We first provide some basic definitions and lemmas in NBDOS, then present the concrete implementation of NBDOS.

Definition 1. (soft core instance) For any $x^i \in S^c$, if the proportion of c class instances in its k-nearest neighbors set is not less than rTh, then x^i is a soft core instance of class c.

According to Definition 1, the soft core condition of NBDOS depends on the parameters k and rTh. In DBSCAN, a point p is core point, if its Esp-neighborhood contains at least $MinPts$ points, where Esp-neighborhood of p is a points set $N_{Esp}(p) = \{q \mid dist(p, q) \leq Esp\}$ and $MinPts$ is threshold to determine whether Esp-neighborhood of p is dense. Similar to the Esp-neighborhood of DBSCAN, we correspondingly give a definition of $\mathcal{G}^c_k$-neighborhood.

Definition 2. ($\mathcal{G}^c_k$-neighborhood) For any soft core instance of class c, x^i, its $\mathcal{G}^c_k$-neighborhood is defined as a instances set $H^c_k(x^i) =$

$\{x^i\} \cup \{x^j \in x^i.N_k | x^j$ is a instance of class $c\} \cup \{x^j \in x^i.RN_k | x^j$ is a soft core instance of class $c\}$, where N_k and NR_k denote k-nearest and reverse k-nearest neighbors set, respectively.

To detect the arbitrarily shaped regions dominated by class c, NBDOS must satisfy a key condition that $\mathcal{G}_k^c$-neighborhood is based on a binary predicate which is symmetric and reflexive [21]. From Definition 2, one can see that, for a soft core instance, its $\mathcal{G}_k^c$-neighborhood consists of three parts: this instance itself, all instances of class c in its N_k and all of the soft core instances of class c in its NR_k. Such a constitution assures that those neighboring instances of class c can be included in its $\mathcal{G}_k^c$-neighborhood, and $\mathcal{G}_k^c$-neighborhood can satisfy reflexivity and symmetry (proved in Lemma 1). Note that only the soft core instances in NR_k are included into $\mathcal{G}_k^c$-neighborhood, which makes each instance of $\mathcal{G}_k^c$-neighborhood coming from the dominated regions of class c.

Lemma 1. *(the reflexivity and symmetry of $\mathcal{G}_k^c$-neighborhood) For any soft core instance $x^i \in S^c$, $x^i \in H_k^c(x^i)$; and for any two soft core instances x^i, $x^j \in S^c$, if $x^j \in H_k^c(x^i)$ then $x^i \in H_k^c(x^j)$.*

Proof. The reflexivity: It is obvious for $x^i \in H_k^c(x^i)$ according to the definition of $H_k^c(x^i)$. The symmetry: If x^i is identical to x^j, the symmetry of $\mathcal{G}_k^c$-neighborhood apparently hold. If x^i is different to x^j, $x^j \in H_k^c(x^i)$ means that either $x^j \in x^i.N_k$, or $x^j \in x^i.RN_k$ and x^j is a soft core instance of class c. For the former, $x^i \in x^j.RN_k$ and x^i is a soft core instance, so $x^i \in H_k^c(x^j)$. For the latter, $x^i \in x^j.N_k$, it also obtains $x^i \in H_k^c(x^j)$. $\square$

The implementation of NBDOS is described in Algorithm 2. In line 2, the cluster labels of all c class instances are initialized to 0, which indicates all instances default to the trapped instances. In lines $3 - 13$, it finds all soft core instances of class c and their respective H_k^c. In lines $15 - 20$, it detects the clusters of outstanding instances of class c literally. Each time the function of *expandCluster* is called to discover a cluster, and the label of cluster is assigned the current cluster number *curId*.

expandCluster contains four steps to discover a cluster of outstanding instances. 1) An unmarked soft core instance x^i is used as the initialized soft core instances set for the current cluster to be discovered, *sfC* (line 27). 2) Draw out a instance x^j from *sfC*. For each instance in $H_k^c(x^j)$, if it has not been discovered in any previous cluster, *curId* is assigned to its cluster label (i.e., add this instance into the current cluster, line 33). Furthermore, if it is a soft core instance, add it into *sfC* (line 34). 3) Remove x^j from *sfC* after the examinations for all instances of $H_k^c(x^j)$ (line 38). 4) Repeat 2) and 3) until *sfC* is empty. The discovering process of current cluster starts with x^i and gradually expands by introducing the unmarked instances of class c that are near to those instances existing in the current cluster. The reflexivity and symmetry of $\mathcal{G}_k^c$-neighborhood ensures that, beginning with an arbitrary soft core instance falling into a region dominated by class c, *expandCluster* can exactly obtain the cluster which corresponds to this region [21].

When all soft core instances of class c have been marked (i.e., their cluster labels are not equal to 0), the detecting process for the clusters of outstanding instances is finished. Next, a postprocessing is applied to all discovered clusters to drop those clusters that the number of members is smaller than *nTh* by reassigning 0 to the cluster labels of their members (lines $21 - 25$). The outputted $S^c.cl$ provides two information: 1) The instances with non-0 cluster labels belong to the clustering of outstanding instances, and the instances whose cluster labels are 0 form the clustering of trapped instances (Step 4 of Algorithm 1). 2) For the instances with non-0 cluster label, the different cluster labels correspond to the different clusters of outstanding instances.

Algorithm 2 NBDOS(S^c, k, $S^c.N_k$, rTh, nTh).

Input: S^c: The instances of class c; k: The number of nearest neighbors; $S^c.N_k$: The k-nearest neighbors lists of S^c; rTh: The minimal proportion of c class instances which should be achieved for the soft core instances in their k-nearest neighbors; nTh: The minimal number of members required for the discovered clusters.

Output: $S^c.cl$: The cluster labels of all instances of class c

1: Initialize the soft core instances set to be empty, $sfS^c \leftarrow \varnothing$
2: Initialize the cluster labels of S^c to 0, $S^c.cl \leftarrow 0$
3: **for all** $x^i \in S^c$ **do**
4:     $Tem \leftarrow$ all c class instances in $x^i.N_k$
5:     **if** $round(\frac{|Tem|}{k}) \geq rTh$ **then**
6:         $sfS^c \leftarrow sfS^c \cup \{x^i\}$
7:         $H_k^c(x^i) \leftarrow Tem$
8:     **end if**
9: **end for**
10: **for all** $x^i \in sfS^c$ **do**
11:     $Tem \leftarrow$ find those instances whose k-nearest neighbors include x^i
12:     $H_k^c(x^i) \leftarrow H_k^c(x^i) \cup (Tem \cap sfS^c)$
13: **end for**
14: Initialize the current cluster number to 0, $curId \leftarrow 0$
15: **for all** $x^i \in sfS^c$ **do**
16:     **if** $x^i.cl = 0$ **then**
17:         $curId \leftarrow curId + 1$
18:         $S^c.cl \leftarrow expandCluster(sfS^c, x^i, curId, S^c.cl)$
19:     **end if**
20: **end for**
21: **for** $i \leftarrow 1, curId$ **do**
22:     $C^i \leftarrow$ find those instances whose cluster labels are i
23:     **if** $|C^i| < nTh$ **then** $C^i.cl \leftarrow 0$
24:     **end if**
25: **end for**

26: **function** $expandCluster(sfS^c, x^i, curId, S^c.cl)$
27:     $sfC \leftarrow \{x^i\}$
28:     $x^i.cl \leftarrow curId$
29:     **while** $sfC \neq \varnothing$ **do**
30:         $x^j \leftarrow$ draw out a instance from sfC
31:         **for all** $x^l \in H_k^c(x^j)$ **do**
32:             **if** $x^l.cl = 0$ **then**
33:                 $x^l.cl \leftarrow curId$
34:                 **if** $x^l \in sfS^c$ **then** $sfC \leftarrow sfC \cup \{x^l\}$
35:                 **end if**
36:             **end if**
37:         **end for**
38:         $sfC \leftarrow sfC \setminus \{x^j\}$
39:     **end while**
40:     **return** $S^c.cl$
41: **end function**

Some previous works also involved into defining different kinds for the minority class instances. In literature [22,23], classifying a minority class instance depends upon the local characteristics of this instance through analyzing its k-nearest neighbors. A instance which most of its k-nearest neighbors have the same (/different) class with it belongs to the safe (/noise) instances, otherwise it belongs to the borderline instances. It should be pointed out that the outstanding and trapped instances are different from the safe and borderline instances, respectively. The former emphasizes whether a minority class instance lie in the regions dominated by the class which it belongs to, while the latter focuses on examining the distribution of instances in the k-nearest neighborhood of this single
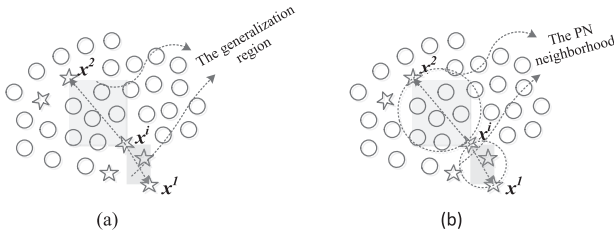
Fig. 2. x^i is primary reference instance. x^1 and x^2 are its two neighbors. $\vec{x^i x^1}$ and $\vec{x^i x^2}$ are two neighbor directions of x^i. (a) The generalization regions of $\vec{x^i x^1}$ and $\vec{x^i x^2}$. (b) The PN neighborhoods of $\vec{x^i x^1}$ and $\vec{x^i x^2}$.

minority instance. We present the detailed differences in the supplementary material due to space consideration.

3.2. Acquiring the selection probabilities for the neighbor directions

In SMOM, the selection weight of a neighbor direction reflects how much probability that it is selected to generate synthetic instances. For the instances of Oi^c, as mentioned earlier, it is reasonable that the synthetic instances are randomly created along their directions of $k1$-nearest neighbors from class c. SMOM actually only computes the selection probabilities for the neighbor directions of Ti^c. There are two steps to obtain the selection probability distribution on the neighbor directions of a trapped instance: Firstly, compute the selection weight for each of the $k1$ neighbor directions of this instance. Secondly, convert all of the computed selection weights into a probability distribution.

3.2.1. The computation of selection weights

Consider Fig. 2a, $\vec{x^i x^1}$ and $\vec{x^i x^2}$ are two neighbor directions of x^i. The two shaded rectangular regions are their corresponding generalization regions in which the synthetic instances can appear. As selecting which one of neighbor directions is equivalent to filling the synthetic instances in which one of corresponding generalization regions. The computation of selection weight for a neighbor direction can be transformed to measure how aggressively its generalization region should place the synthetic instances.

Two factors need to be considered when the synthetic instances of class c are filled into a region. The first factor is the risk of over generalization. e.g., as shown in Fig. 2a, the generalization region of $\vec{x^i x^1}$ is apparently more reliable than the generalization region of $\vec{x^i x^2}$. The second factor is the level of difficulty with respect to learning the instances of class c. e.g., compared to the generalization region of $\vec{x^i x^1}$, the generalization region of $\vec{x^i x^2}$ obviously needs more synthetic instances of class c so that the c class instances within it is able to be learned. To quantify these two factors, the instances distributed in the generalization region must be acquired before. However, the generalization region is hyperrectangle. It is unfavorable to locate instances. We use a kind of hypersphere region to approximatively represent it. As shown in Fig. 2b, for the generalization region of $\vec{x^i x^1}$ ($/\vec{x^i x^2}$), a sphere centered at the midpoint of x^i and x^1 ($/x^2$), with radius being the half distance between them compactly covers it. We call such sphere the PN neighborhood due to the characteristic through PRI and its Neighbor. Correspondingly, for a neighbor direction $\vec{x^i x^j}$, its PN neighborhood is denoted as $x^i.PN(x^j)$. The PN neighborhood is employed to substitute the generalization region below.

For a neighbor direction, its selection weight is computed as follows :

$$1/\underbrace{e^{r1\frac{\gamma^{mi}}{\gamma^c}+r2E_{mi}+w2(r1\frac{\gamma^{ma}}{\gamma^c}+r2E_{ma})}}_{overgeneralization\,factor}+w_1\underbrace{e^{-\frac{\gamma^c}{\gamma^{ma}+\gamma^{mi}+\gamma^c}}}_{difficulty\,factor} \tag{2}$$

e is the natural constant. γ^c, γ^{ma} and γ^{mi} denote the number of c class instances, majority class instances and minority class

instances in the PN neighborhood of this neighbor direction, respectively. Note that both the majority and minority classes mentioned here do not include class c. The class c is separately treated for convenience. E_{mi} is the entropy value reached by the minority classes in the PN neighborhood (we call E_{mi} the minority class entropy). It is equal to $(\sum_{c_i \in miC}(\frac{\gamma^{c_i}}{\gamma^{mi}}\log\frac{\gamma^{c_i}}{\gamma^{mi}}))/Z_{mi}$, where miC is the set of minority classes and Z_{mi} is the maximum entropy value that can be achieved by the minority classes (i.e., $\log|miC|$). Z_{mi} makes E_{mi} range from 0 to 1. Similarly, E_{ma} is the majority class entropy and is equal to $(\sum_{c_i \in maC}(\frac{\gamma^{c_i}}{\gamma^{ma}}\log\frac{\gamma^{c_i}}{\gamma^{ma}}))/Z_{ma}$, where maC is the set of majority classes and Z_{ma} is the maximum majority class entropy (i.e., $\log|maC|$). The values of all above variables can be obtained according to the class distribution of PN neighborhood. In addition, $w1$, $w2$, $r1$ and $r2$ are four parameters. We describe them below.

(2) consists of two parts that respectively consider the over generalization factor and the difficulty factor. The over generalization factor concerns the negative effects to other classes if the synthetic instances of class c fall into the PN neighborhood. The difficulty factor highlights the level of difficulty for learning the instances of class c. If a PN neighborhood brings lesser over generalization and has higher degree of learning difficulty, its corresponding neighbor direction will gain higher selection weight. However, there is a negative correlation between the over generalization factor and the difficulty factor. $w1$ is a weighting parameter to balance their relative importance. In light of our observation, the over generalization should be mainly considered for the selection weight. $w2$ is another weighting parameter to adjust the effect of majority classes for the over generalization factor. As the avoidance of over generalization should put more emphasis on the minority classes, the value of $w2$ should be smaller than 1. In addition, the parameters $r1$ and $r2$ are used to rescale $\frac{\gamma^{mi}}{\gamma^c}(/\frac{\gamma^{ma}}{\gamma^c})$ and $E_{mi}(/E_{ma})$ by multiplying with them, respectively.

In (2), the expression of difficulty factor is straightforward. $\frac{\gamma^c}{\gamma^{ma}+\gamma^{mi}+\gamma^c}$ is the proportion of c class instances in the PN neighborhood. The lower this proportion is, the larger the difficulty factor is. However, the expression of over generalization factor is relatively complicated. In fact, it is based on the following four important observations:

1. For arbitrary two PN neighborhoods, the negative impacts that the synthetic instances of class c may cause to the minority classes are more significant on the PN neighborhood with higher $\frac{\gamma^{mi}}{\gamma^c}$. This observation is obvious as higher $\frac{\gamma^{mi}}{\gamma^c}$ indicates the minority classes are more dominant in the PN neighborhood, the generated synthetic instances of class c are more likely to be noisy. Consider Fig. 3a, supposing x^1 is PRI. The $\frac{\gamma^{mi}}{\gamma^c}$ in $x^1.PN(x^4)$ and $x^1.PN(x^5)$ are $\frac{2}{2}$ and $\frac{1}{2}$, respectively. We can see that the neighbor direction $\vec{x^1 x^4}$ can introduce the synthetic instances of class c into a more uncertain region in terms of class c_1 than $\vec{x^1 x^5}$. Therefore, the over generalization factor in (2) is positively correlated with $\frac{\gamma^{mi}}{\gamma^c}$.

2. For arbitrary two PN neighborhoods, if their $\frac{\gamma^{mi}}{\gamma^c}$ is equal, the negative impacts that the synthetic instances of class c may cause to the minority classes are more significant on the PN neighborhood with more minority classes. It can be understood from that if a PN neighborhood includes more minority classes, the synthetic instances of class c can negatively impact the more minority classes. Observe Fig. 3b, $x^2.PN(x^1)$ and $x^2.PN(x^3)$ have equal $\frac{\gamma^{mi}}{\gamma^c}$, but $x^2.PN(x^3)$ involved the minority classes c_1 and c_2. $\vec{x^2 x^3}$ can result in that the synthetic instances of class c are created in the boundary region between c_1 and c_2. Both c_1 and c_2 suffer from negative disturbance. The over general-
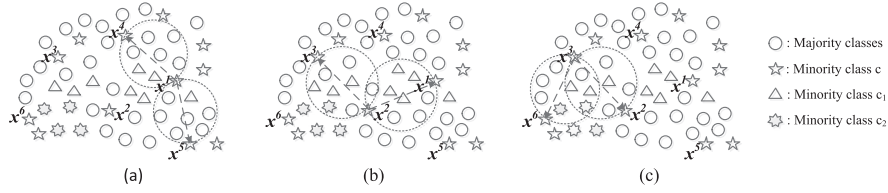
Fig. 3. The figures (a)–(c) illustrate three observations with respect to the computation of selection weight for the neighbor direction.
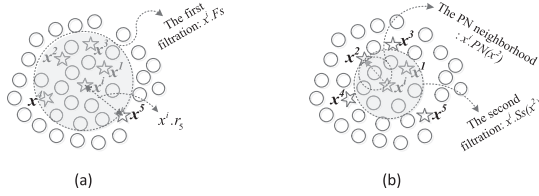

Fig. 4. x^i is primary reference instance. x^1, x^2, x^3, x^4 and x^5 are its k-nearest neighbors from the same class with it (k is equal to 5 here). Figure illustrates a two-round filtration operation. (a) The first filtration performed on all PN neighborhoods of x^i. (b) The second filtration performed on the PN neighborhood of $x^i\vec{x^2}$.

ization factor incorporates the number of minority classes into consideration by using E_{mi}. If there are more minority classes, the value of E_{mi} that can be reached will be larger. The over generalization factor is positively correlated with E_{mi}.

3. For arbitrary two PN neighborhoods, if they have the equal $\frac{\gamma^{mi}}{\gamma^c}$ and the same number of minority classes, the negative impacts that the synthetic instances of class c may cause to the minority classes are more significant on the PN neighborhood with more uniform distribution of minority classes. The reason is that if a PN neighborhood is more evenly occupied by different minority classes, the synthetic instances generated in this PN neighborhood may be less discriminative. Consider $x^3.PN(x^6)$ and $x^3.PN(x^2)$ as shown in Fig. 3c, $x^3\vec{x^6}$ makes both minority class c_2 and c_1 equally expose to $x^3.PN(x^6)$ while $x^3\vec{x^2}$ mainly crosses the region of c_1. Therefore, the synthetic instances of class c may be more conceptual ambiguity if they are generated in $x^3.PN(x^6)$. E_{mi} is impacted by the evenness extent of minority classes distribution. The more uniform the distribution of minority classes is, the larger E_{mi} may be.

4. The three observations above are also suitable for considering the negative impacts to the majority classes. The difference is that the over generalization occurring on the majority classes can be paid less attention as compared to the minority classes. Hence, (2) uses $w2$ to adjust the importance of majority classes for the over generalization factor.

The acquisition of selection weight for a neighbor direction contains two parts of works. The first part is to find all instances located inside the corresponding PN neighborhood, N_{PN}. The second part is to compute the selection weight based on N_{PN} by using (2). Obviously, the former takes most of the computational cost.

In order to find N_{PN} efficiently, we perform a two-round filtration operation to early exclude those instances which are impossible to locate in the PN neighborhood. Assuming x^i shown in Fig. 4a is PRI. x^1, x^2, x^3, x^4 and x^5 are its k nearest neighbors (k is 5 here). It is obvious that the distance between x^i and x^5, $x^i.r_5$, is the maximum value among the distances from x^i to the instances that lie inside any of the PN neighborhoods of x^i. Therefore, before computing N_{PN} for any PN neighborhood of x^i, the first filtration can be performed to find those instances, $x^i.Fs$, that the distances to x^i are no larger than $x^i.r_5$ (i.e., as shown in Fig. 4a, all instances outside the dotted circle can be ignored). The implementation of the first filtration is given in steps $2.1 - 2.2$ of Algorithm 1.

It should be noted that $x^i.N_{k1}^c$ are just all those instances of class c whose distances from x^i are no greater than $x^i.r_{k1}$. The second filtration appears before computing N_{PN} for a specific PN neighborhood. As shown in Fig. 4b, before computing the N_{PN} of $x^i.PN(x^2)$, it can choose out the instances, $x^i.Ss(x^2)$, from $x^i.Fs$ which the distances to x^i are no larger than the distance between x^i and x^2. The second filtration is presented in step 5.3. $x^i.Fs$ and $x^i.Ss$ are respectively the resulting instances set after performing the first and second round filtration for x^i in the process of finding N_{PA}.

Moreover, as the instances belonging to the Ss of a PN neighborhood are actually distributed in a hypersphere neighborhood whose radius is the double of the radius of this PN neighborhood (e.g., as $x^i.PN(x^2)$ and $x^i.Ss(x^2)$ shown in Fig. 4b), Ss may contain many instances not belonging to the corresponding PN neighborhood. Hence, further examination is needed. The function dPN is used to determine which instances from Ss are located in the corresponding PN neighborhood (Step 5.3). In dPN, the distance of each instance in Ss to the center of the considered PN neighborhood is computed, if the distance is not greater than the radius of PN neighborhood, this instance belongs to N_{PN} (the details of dPN are presented in Algorithm 1S in the supplementary material).

It should be noted that, however, not all of the neighbor directions must call dPN to compute their selection weights. Supposing $x^i \in Ts^c$, and $x^j \in x^i.N_{k1}^c$. If $x^j \in Os^c$ and $x^i \in x^j.N_{k1}^c$. It indicates that x^i is close enough to a cluster of outstanding instances. Hence, directly assigning $1 + \frac{w1}{e}$ into $x^i.sw(x^j)$ is reasonable (Step 5.1), where the "$1 + \frac{w1}{e}$" implies $x^i.PN(x^j)$ is a clean neighborhood only containing the instances of class c. If $x^i \in x^j.N_{k1}^c$, and $x^j.sw(x^i)$ has been computed, $x^j.sw(x^i)$ can be assigned to $x^i.sw(x^j)$ (Step 5.2), as $x^j.PN(x^i)$ represents the same neighborhood with $x^i.PN(x^j)$.

We noticed that the oversampling algorithms SL-SMOTE and LN-SMOTE [24] were also concerned with the problem of how to generate safe synthetic instances. They define the safe level of a sample as the number of minority class sample in its k-nearest neighbors. The synthetic samples are always created closer to the one having higher safe level in PRI and ARI. However, only considering the distribution of instances in the k-nearest neighborhoods of PRI and ARI may be insufficient for completely covering the corresponding generalization region especially when ARI is far away from PRI. Our SMOM to compute the selection weight for a neighbor direction is based on considering the whole multiclass distribution in the corresponding PN neighborhood. This PN neighborhood contains this neighbor direction's generalization region in which the synthetic instances may appear. Hence, the computation of selection weight in our SMOM used more precise and sufficient information.

3.2.2. Convert the selection weights to the selection probabilities

Some trapped instances of class c may be potential outliers and noise. They are likely far away from all other instances of class c. Hence, it is possible that the over generalization may be inevitable whichever the neighbor direction is selected. To handle this problem, SMOM for each trapped instance checks its all neighbor directions firstly. If there is no PN neighborhood that only contains the instances of class c, add it into its own neighbors set and then

Table 1
Description of characteristics of experimental data sets.

Data	M	L	F	Class distribution	Description of classes	IR
vehicle	846	3	18	199/429/218	'DH'/'SL'/'NO'	2.22
sat	4374	4	36	1533/626/707/1508	'1'/'4'/'5'/'7'	5.31
ecoli	327	4	7	143/77/52/35	'cp'/'im'/'pp'/'imU'	7.86
automobile	205	3	71	25/153/27	'1'&'2'/'3'&'4'&'5'/'6'	9.87
voice9_6	413	6	10	100/43/58/115/59/38	'1'/'2'&'9'/'5'/'6'/'7'/'8'	12.11
vowel5	990	5	10	180/90/360/270/90	'0'&'1'/'2'/'3'&'4'&'5'&'6'/'7'&'8'&'9'/'10'	13.83
vowel7	990	7	10	90/180/90/180/90/180/180	'0'/'1'&'2'/'3'/'4'&'5'/'6'/'7'&'8'/'9'&'10'	15.00
robot	5456	4	4	826/2097/2205/328	'Slight RT'/'Sharp RT'/'Move F'/'Slight LT'	15.89
pendigits	7494	5	16	2339/719/2998/719/719	'1'&'6'/'2'&'3'/'4'/'5'	16.55
page-blocks	560	4	10	329/28/88/115	'0'&'1'&'2'/'3'/'4'&'5'&'6'&'7'/'8'&'9'	20.91
housing5	506	5	13	77/239/123/36/31	'1'/'2'/'3'/'4'/'5'	24.16
SWD	1000	4	10	32/352/399/217	'2'/'3'/'4'/'5'	28.84
wine-red	1571	4	11	53/681/638/199	'4'/'5'/'6'/'7'/'8'	30.34
voice9_7	413	7	10	100/22/58/115/59/38/21	'1'/'2'/'5'/'6'/'7'/'8'/'9'	32.74
LEV	1000	5	4	93/280/403/197/27	'0'/'1'/'2'/'3'/'4'	40.41
machine	209	5	6	152/27/13/7/10	'1'&'2'/'3'&'4'/'5'&'6'/'7'&'8'/'9'&'10'	54.46
abalone8	2148	8	10	126/203/267/487/634/259/115/57	'14'/'13'/'12'/'11'/'10'/'6'/'5'/'4'	59.52
plates-faults5	1941	5	27	158/72/55/793/863	'1'/'4'/'5'/'3'&'6'/'2'&'7'	61.05
plates-faults7	1941	7	27	158/190/391/72/55/402/673	'1'/'2'/'3'/'4'/'5'/'6'/'7'	61.11
ESL	488	5	4	62/14/351/28/23	'1'&'2'/'3'/'4'&'5'&'6'/'7'/'8'&'9'	62.73
ERA	1000	9	4	92/142/181/172/158/118 /88/31/18	'1'/'2'/'3'/'4'/'5'/'6'/'7'/'8'/'9'	79.84
yeast5	982	5	8	463/429/35/30/25	'CYT'/'NUC'/'EXC'/'VAC'/'POX'&'ERL'	85.75
ERA5	1000	5	4	92/771/88/31/18	"1'/'2'/'3'/'4'/'5'	93.42
yeast7	1055	7	8	463/244/163/51/79/35/20	'CYT'/'NUC'/'ME2'/'ME1'/'EXC'/'VAC'/'POX'	95.18
abalone10	2297	10	10	57/115/259/391/634/487/126/103/67/58	'4'/'5'/'6'/'7'/'10'/'11'/'14'/'15'/'16'/'17'	123.58
yeast8	1484	8	8	463/429/244/163/81/44/35/25	'CYT'/'NUC'/'MIT'/'ME3'/'ME2'&'VAC'/'ME1'/'EXC'/'POX'&'ERL'	130.15
housing10	506	10	13	22/55/85/154/84/39/29 /7/10/21	'1'/'2'/'3'/'4'/'5'/'6'/'7'/'8'/'10'	152.35

assign $1 + w1/e$ to the selection weight of its own direction (Step 6.1 of Algorithm 1). A probability distribution in terms of selecting its neighbor directions is obtained by normalizing the selection weights in all of its neighbor directions (Step 6.2).

3.3. Generating the synthetic instances

The specific process to create the synthetic instances is presented in step 8. Three elements are needed to generate the synthetic instances for a instance of class c: 1) Its $k1$-nearest neighbors from class c. 2) The selection weights (/probabilities) of the neighbor directions of this instance. 3) The amount of synthetic instances that need to be yielded for this instance (Step 7).

One point worth making is that our SMOM generates an equal number of synthetic samples for each minority instance. There are some previous oversampling algorithms that provide different weights to different minority class instances (a higher weight helps in creating more synthetic instances for the corresponding minority instance) [4,8,10]. These methods usually place more emphasis on the protection of those minority instances near class boundary by assigning more weights to them. One potential risk in them is that some boundary regions may be overly filled by the synthetic instances from the considered minority class, resulting in a significant deterioration for the classification performance of the other classes [4], especially when the total amount of synthetic instances required to be generated is large.

4. Experimental study

4.1. Setup

Data Sets. 27 multiclass imbalanced data sets are used in our experiments. Their detailed characteristics are summarized in Table 1. *M, L,* and *F* respectively denote the number of instances, classes and features in the data set. The column of "Description of classes" shows which original classes of prediction domain constituted the class distribution of experimental data. *IR* is the overall imbalanced degree measured for multiclass imbalanced data. It is

defined as:

$$IR = \sum_{i=1}^{C-1} \sum_{j>i} \left(\frac{N_i}{N_j} - 1 \right) \tag{3}$$

where N_i and N_j correspond respectively to the instance size of class c_i and class c_j (the classes have been sorted in descending order ,i.e., $N_i \geq N_j$ if $j \geq i$). From (3), IR will be equal to 0 if all of the classes have the same size.

Note that these experimental data sets vary in size, class distribution, feature number and IR to ensure a reliable assessment of performance. A few of them (housing5, SWD, LEV, machine, ESL, ERA, ERA5 and housing10) are the ordinal regression benchmark data, which have been widely used in the ordinal regression literature [13,25]. The others are available from UCI [26]. As there is no criterion to distinguish minority classes and majority classes in multiclass imbalance data, we simply classify the classes whose instances size greater (/smaller) than $\frac{M}{L}$ as the majority (/minority) classes.

Assessment metrics. Recall, precision, and F-measure [21] are widely discussed single-class metrics in two-class imbalance problems. Recall and precision respectively assess the completeness and exactness of prediction. F-measure incorporates both of them to express their tradeoff. Parameter β is used to adjust the relative importance of recall and precision, and is usually set to one.

$$Recall = \frac{TP}{TP + FN} \quad Precision = \frac{TP}{TP + FP}$$

$$F - measure = \frac{(1 + \beta^2)RecallPrecision}{\beta^2 Recall + Precision}$$

In multiclass imbalance problems, MG [27] and MAUC [28] are the most commonly used measures. They are respectively defined as:

$$MG = \left(\prod_{i=1}^{|C|} Recall_{c_i} \right)^{\frac{1}{|C|}} \quad MAUC = \frac{2}{|C|(|C|-1)} \sum_{i<j} (\widehat{A}(c_i, c_j))$$

where C is the set of classes; $Recall_{c_i}$ is the recall on class c_i; $\widehat{A}(c_i, c_j) = [\widehat{A}(i|j) + \widehat{A}(j|i)]/2$, $\widehat{A}(i|j)$ is the probability that a ran-

Table 2

Average performance results ($mean_{SD}$), and mean ranks of NONE, ROS, SMOTE, ADASYN, SL-SMOTE, MWMOTE, MDO and SMOM over all data sets (the results of ADASYN, SL-SMOTE and MWMOTE are displayed in gray as they are originally prepared to handle two-class imbalanced data).

Method	C4.5		AdaBoost (C4.5)		MLP		NB		The overall
	Avg-MG	Mean rank	Avg-MG	Mean rank	Avg-MG	Mean rank	Avg-MG	Mean rank	mean rank
NONE	$0.417_{\pm0.386}$	6.963	$0.446_{\pm0.418}$	7.111	$0.405_{\pm0.409}$	7.389	$0.372_{\pm0.370}$	5.796	6.815
ROS	$0.505_{\pm0.337}$	4.685	$0.511_{\pm0.380}$	5.278	$0.534_{\pm0.337}$	3.519	**$0.453_{\pm0.326}$**	3.685	4.292
SMOTE	$0.512_{\pm0.334}$	2.852	$0.526_{\pm0.362}$	3.556	$0.537_{\pm0.334}$	3.407	$0.437_{\pm0.332}$	3.537	3.338
ADASYN	$0.493_{\pm0.343}$	4.167	$0.519_{\pm0.366}$	3.889	$0.529_{\pm0.340}$	4.074	$0.41_{\pm0.338}$	4.222	4.088
SL-SMOTE	$0.482_{\pm0.357}$	4.963	$0.509_{\pm0.376}$	4.704	$0.508_{\pm0.347}$	5.056	$0.447_{\pm0.333}$	3.815	4.635
MWMOTE	$0.469_{\pm0.367}$	4.389	$0.495_{\pm0.386}$	3.982	$0.513_{\pm0.344}$	4.482	$0.407_{\pm0.346}$	4.907	4.440
MDO	$0.44_{\pm0.377}$	6.037	$0.481_{\pm0.394}$	5.222	$0.480_{\pm0.354}$	6.037	$0.28_{\pm0.343}$	7.019	6.079
SMOM	**$0.525_{\pm0.332}$**	**1.944**	**$0.542_{\pm0.354}$**	**2.259**	**$0.550_{\pm0.330}$**	**2.037**	$0.435_{\pm0.333}$	**3.019**	**2.315**

Method	C4.5		AdaBoost (C4.5)		MLP		NB		The overall
	Avg-MAUC	Mean rank	Avg-MAUC	Mean rank	Avg-MAUC	Mean rank	Avg-MAUC	Mean rank	mean rank
NONE	$0.835_{\pm0.107}$	5.519	$0.864_{\pm0.118}$	4.648	$0.877_{\pm0.096}$	4.63	$0.86_{\pm0.102}$	4.667	4.866
ROS	$0.842_{\pm0.115}$	5.259	$0.869_{\pm0.109}$	6.13	$0.877_{\pm0.098}$	4.278	$0.863_{\pm0.096}$	4.278	4.986
SMOTE	$0.848_{\pm0.111}$	3.037	$0.873_{\pm0.107}$	3.907	$0.88_{\pm0.098}$	3.556	$0.869_{\pm0.094}$	3.222	3.431
ADASYN	$0.84_{\pm0.112}$	4.963	$0.874_{\pm0.108}$	3.722	$0.878_{\pm0.097}$	4.352	$0.864_{\pm0.097}$	4.796	4.458
SL-SMOTE	$0.841_{\pm0.117}$	5.222	$0.87_{\pm0.110}$	5.5	$0.872_{\pm0.104}$	5.907	$0.865_{\pm0.098}$	4.5	5.282
MWMOTE	$0.834_{\pm0.118}$	4.963	$0.862_{\pm0.121}$	4.907	$0.866_{\pm0.107}$	5.889	$0.854_{\pm0.109}$	5.63	5.347
MDO	$0.84_{\pm0.110}$	4.741	$0.869_{\pm0.114}$	5.259	$0.873_{\pm0.099}$	5.148	$0.846_{\pm0.097}$	6.389	5.384
SMOM	**$0.853_{\pm0.108}$**	**2.296**	**$0.878_{\pm0.104}$**	**1.926**	**$0.884_{\pm0.096}$**	**2.241**	**$0.871_{\pm0.093}$**	**2.519**	**2.246**

Method	C4.5		AdaBoost (C4.5)		MLP		NB		The overall
	Avg-Recall	Mean rank	Avg-Recall	Mean rank	Avg-Recall	Mean rank	Avg-Recall	Mean rank	mean rank
NONE	$0.542_{\pm0.345}$	6.537	$0.577_{\pm0.370}$	6.519	$0.525_{\pm0.367}$	7.04	$0.526_{\pm0.365}$	5.333	6.357
ROS	$0.639_{\pm0.262}$	4.833	$0.634_{\pm0.307}$	5.926	$0.707_{\pm0.210}$	3.833	$0.587_{\pm0.266}$	4.556	4.787
SMOTE	$0.651_{\pm0.243}$	3.259	$0.657_{\pm0.274}$	3.296	$0.701_{\pm0.220}$	3.482	$0.6_{\pm0.253}$	3.982	3.505
ADASYN	$0.621_{\pm0.257}$	4.407	$0.64_{\pm0.289}$	4.482	$0.704_{\pm0.232}$	3.593	$0.58_{\pm0.290}$	4.148	4.158
SL-SMOTE	$0.603_{\pm0.291}$	4.852	$0.632_{\pm0.301}$	4.611	$0.651_{\pm0.258}$	5.13	$0.602_{\pm0.272}$	4.093	4.672
MWMOTE	$0.607_{\pm0.287}$	4.537	$0.63_{\pm0.305}$	3.796	$0.668_{\pm0.242}$	4.648	$0.556_{\pm0.282}$	4.926	4.477
MDO	$0.57_{\pm0.308}$	5.204	$0.615_{\pm0.317}$	5.185	$0.615_{\pm0.254}$	5.685	$0.457_{\pm0.321}$	6.074	5.537
SMOM	**$0.669_{\pm0.237}$**	**2.37**	**$0.676_{\pm0.261}$**	**2.185**	**$0.717_{\pm0.205}$**	**2.593**	**$0.62_{\pm0.255}$**	**2.889**	**2.509**

Method	C4.5		AdaBoost (C4.5)		MLP		NB		The overall
	Avg-Precision	Mean rank	Avg-Precision	Mean rank	Avg-Precision	Mean rank	Avg-Precision	Mean rank	mean rank
NONE	**$0.601_{\pm0.316}$**	3.889	$0.626_{\pm0.335}$	3.852	**$0.629_{\pm0.318}$**	**2.444**	$0.372_{\pm0.306}$	5.796	3.995
ROS	$0.555_{\pm0.292}$	4.37	$0.616_{\pm0.321}$	3.982	$0.53_{\pm0.302}$	4.926	$0.507_{\pm0.315}$	5.259	4.634
SMOTE	$0.561_{\pm0.296}$	4.222	$0.612_{\pm0.310}$	4.87	$0.547_{\pm0.305}$	4.852	$0.557_{\pm0.245}$	4.574	4.630
ADASYN	$0.549_{\pm0.304}$	4.889	$0.61_{\pm0.314}$	4.519	$0.524_{\pm0.316}$	5.778	$0.535_{\pm0.269}$	4.426	4.903
SL-SMOTE	**$0.601_{\pm0.291}$**	**3.482**	**$0.653_{\pm0.298}$**	**3.685**	$0.59_{\pm0.277}$	3.259	$0.545_{\pm0.277}$	4.667	**3.773**
MWMOTE	$0.508_{\pm0.331}$	5.444	$0.565_{\pm0.341}$	5.37	$0.503_{\pm0.329}$	5.074	$0.493_{\pm0.295}$	4.519	5.102
MDO	$0.597_{\pm0.280}$	5.13	$0.652_{\pm0.296}$	4.852	$0.603_{\pm0.285}$	4.667	**$0.571_{\pm0.318}$**	4.093	4.686
SMOM	$0.573_{\pm0.282}$	4.574	$0.631_{\pm0.292}$	4.87	$0.569_{\pm0.279}$	5	$0.57_{\pm0.242}$	**3.907**	4.588

domly drawn member of class c_j will have a lower estimated probability of belonging to class c_i than a randomly drawn member of class c_i. In our experiments, MG, MAUC, recall and precision are together used as assessment metrics (we only compute recall and precision for the smallest minority class following the suggestion of [3,7]). The performance results on each data set are obtained from the average of 10 independent runs with stratified 10-fold cross validation.

Base classifier. Four frequently-used base classifiers are selected for our experiments, including C4.5 decision tree (C4.5), AdaBoost.M1 ensemble of C4.5 (AdaBoost(C4.5)), Multilayer perceptrons (MLP) and Naive Bayes (NB). All classifiers are built by weka [29]. The changes for their default weka parameters are described as follows: C4.5 uses Laplace smoothing; Adaboost.M1 uses 51 [7] boosting iterations; the network of MLP is trained to 500 epochs with a learning rate of 0.1, and the number of hidden layer neurons is set to be 10; NB employs supervised discretization.

Compared algorithms. We compare SMOM with six existing algorithms, including ROS, SMOTE, ADASYN, SL-SMOTE, MW-MOTE and MDO. The standardized Euclidean distance is applied to measure the distance between instances. All classes are oversampled until they contain as many instances as the largest one.

For ADASYN, SL-SMOTE and MWMOTE, the class transformation is used to handle multiclass, i.e., when oversampling one of the minority classes, the other classes are taken as the corresponding majority class. With respect to the parameter settings, there are a series of parameters in SMOM. Among of them, $k1$, $w1$ and $w2$ are three key parameters, nTh should be set to a value near $k1$, and the impacts of rTh and $k2$ are more reflected on the efficiency of SMOM rather than the effectiveness. Given the space limitation, a detailed explanation on how to set the appropriate values for the parameters of SMOM is provided in the supplementary material. We set their values as follows: $k1 = 12$, $k2 = 8$, $rTh = 5/8$, $nTh = 10$, $w1 = 0.2$, $w2 = 1/2$, $r1 = 1/3$ and $r2 = 0.2$. For all of the other compared algorithms, the default parameter values are used if not otherwise specified.

4.2. Experimental results and analyses

Experiment 1. Table 2 summarizes the average performance results ($mean_{SD}$) and mean ranks of NONE, six previous oversampling algorithms and our SMOM over all data sets (the complete results are summarized in Table S3, S4, S5, S6, S7, S8, S9 and S10), where NONE denotes the experimental results with no oversampling. Best

Table 3

Significance tests of MG, MAUC, recall and precision between SMOM and each of NONE, ROS, SMOTE, ADASYN, SL–SMOTE, MWMOTE, and MDO (the results associated with ADASYN, SL–SMOTE and MWMOTE are displayed in gray as they are originally prepared to handle two-class imbalanced data).

	Method	C4.5			AdaBoost(C4.5)			MLP			NB		
	SMOM vs	R^+	R^-	T	R^+	R^-	T	R^+	R^-	T	R^+	R^-	T
MG	NONE	374	4	**4(++)**	378	0	**0(++)**	375	3	**3(++)**	338	40	**40(++)**
	ROS	290	88	**88(++)**	307	71	**71(++)**	288	90	**90(++)**	186.5	191.5	186.5(-)
	SMOTE	316	62	**62(++)**	334.5	43.5	**43.5(++)**	349	29	**29(++)**	233	145	145(+)
	ADASYN	325	53	**53(++)**	303	75	**75(++)**	295	83	**83(++)**	274	104	**104(++)**
	SL-SMOTE	347	31	**31(++)**	326	52	**52(++)**	354	24	**24(++)**	182	196	182(-)
	MWMOTE	359	19	**19(++)**	310.5	67.5	**67.5(++)**	321	57	**57(++)**	316	62	**62(++)**
	MDO	378	0	**0(++)**	337	41	**41(++)**	377	1	**1(++)**	365	13	**13(++)**
	Method	C4.5			AdaBoost(C4.5)			MLP			NB		
	SMOM vs	R^+	R^-	T	R^+	R^-	T	R^+	R^-	T	R^+	R^-	T
MAUC	NONE	303	75	**75(++)**	329	49	**49(++)**	291.5	86.5	**86.5(++)**	328	50	**50(++)**
	ROS	339.5	38.5	**38.5(++)**	369.5	8.5	**8.5(++)**	361.5	16.5	**16.5(++)**	285	93	**93(++)**
	SMOTE	316.5	61.5	**61.5(++)**	342	36	**36(++)**	337	41	**41(++)**	243	135	135(+)
	ADASYN	346.5	31.5	**31.5(++)**	306	72	**72(++)**	360	18	**18(++)**	326.5	51.5	**51.5(++)**
	SL-SMOTE	355	23	**23(++)**	376	2	**2(++)**	372.5	5.5	**5.5(++)**	318	60	**60(++)**
	MWMOTE	359.5	18.5	**18.5(++)**	357.5	20.5	**20.5(++)**	371	7	**7(++)**	367	11	**11(++)**
	MDO	325.5	52.5	**52.5(++)**	365.5	12.5	**12.5(++)**	343.5	34.5	**34.5(++)**	368	10	**10(++)**
	Method	C4.5			AdaBoost(C4.5)			MLP			NB		
	SMOM vs	R^+	R^-	T	R^+	R^-	T	R^+	R^-	T	R^+	R^-	T
Recall	NONE	349	29	**29(++)**	342	36	**36(++)**	357	21	**21(++)**	300	78	**78(++)**
	ROS	313	65	**65(++)**	319	59	**59(++)**	235	143	143(+)	270	108	**108(+*)**
	SMOTE	284.5	93.5	**93.5(++)**	310.5	67.5	**67.5(++)**	290	88	**88(++)**	265	113	**113(+*)**
	ADASYN	313.5	64.5	**64.5(++)**	329.5	48.5	**48.5(++)**	247	131	131(+)	259.5	118.5	**118.5(+*)**
	SL-SMOTE	336.5	41.5	**41.5(++)**	344	34	**34(++)**	343.5	34.5	**34.5(++)**	235.5	142.5	142.5(+)
	MWMOTE	321.5	56.5	**56.5(++)**	301.5	76.5	**76.5(++)**	311.5	66.5	**66.5(++)**	317.5	60.5	**60.5(++)**
	MDO	360.5	17.5	**17.5(++)**	355	23	**23(++)**	368	10	**10(++)**	347	31	**31(++)**
	Method	C4.5			AdaBoost(C4.5)			MLP			NB		
	SMOM vs	R^+	R^-	T	R^+	R^-	T	R^+	R^-	T	R^+	R^-	T
Precision	NONE	139.5	238.5	139.5(-)	147.5	230.5	147.5(-)	78	273	**78(−−)**	244	107	**107(+*)**
	ROS	183.5	194.5	183.5(-)	174	204	174(-)	217	161	161(+)	262	116	**116(+*)**
	SMOTE	207	171	171(+)	202.5	175.5	175.5(+)	163	215	163(-)	242	136	136(+)
	ADASYN	233.5	144.5	144.5(+)	193	185	185(+)	244	134	134(+)	236.5	141.5	141.5(+)
	SL-SMOTE	99	279	**99(−−)**	101	277	**101(−−)**	90	288	**90(−−)**	235	143	143(+)
	MWMOTE	279	99	**99(++)**	279.5	98.5	**98.5(++)**	256	122	122(+)	256	122	122(+)
	MDO	187	191	187(-)	149	229	149(-)	138	240	138(-)	167	184	167(-)

and worst results are highlighted in bold and italics, respectively. From Table 2, we can see that SMOM acquires the lowest mean ranks over various base classifiers for MG, MAUC and recall.

In terms of MG, the mean ranks of all oversampling algorithms are obviously lower than NONE. It indicates that oversampling techniques can significantly improve the recognizing ability for the minority classes. The reasons are that MG is the geometric mean of recall over all classes and the minority classes usually have lower prediction accuracies. Hence, the boost of MG will largely depend on the performance improvement of minority classes. This can also be partly confirmed by observing the experimental results in MG and recall. The ranking results of the compared oversampling algorithms, in these two metrics, are of a high similarity, for which SMOM obtains the lowest Overall Mean Rank (OMR), followed by SMOTE, ADASYN. Different from MG and recall, many compared oversampling algorithms show close or even worse mean ranks in MAUC as compared to NONE. This result shows that not all oversampling techniques are effective in MAUC when dealing with multiclass imbalanced data.

In terms of precision, the OMR of SL-SMOTE is best, followed by NONE, SMOM. Contrasting the recall and precision performances of SL-SMOTE, SL-SMOTE has better precision but worse recall than the other oversampling algorithms. It indicates that SL-SMOTE makes the learning models ignore the learning for the smallest minority class on some level, and fewer instances are classified to the smallest minority class as compared to the other algorithms, leading to a higher precision but lower recall.

Furthermore, we conduct Wilcoxon signed-ranks test [30] to evaluate whether SMOM can statistically outperform the other compared algorithms. The results are presented in Table 3. The symbols "++" (/"−−") and "+*" (/"−*") denote that SMOM is statistically better (/worse) than the considered algorithm at the significance level of 0.05 and 0.1, respectively; "+" signifies that SMOM is only quantitatively better, whereas "−" implies the contrary. As shown in Table 3, SMOM often significantly outperforms the other algorithms in terms of MG, MAUC and recall. For precision, SMOM has no distinct advantage, and even it is significantly worse than SL-SMOTE over C4.5, AdaBoost and MLP. We further compare

Table 4

Average performance results ($mean_{SD}$), and mean ranks of SMOTE (k=12), ADASYN ($k = 12$), SL-SMOTE ($k = 12$) and SMOM over all data sets (the results of ADASYN and SL-SMOTE are displayed in gray as they are originally prepared to handle two-class imbalanced data).

Method	C4.5		AdaBoost (C4.5)		MLP		NB		The overall
	Avg-MG	Mean rank	Avg-MG	Mean rank	Avg-MG	Mean rank	Avg-MG	Mean rank	mean rank
SMOTE (k=12)	$0.450_{\pm0.337}$	3.04	$0.527_{\pm0.362}$	2.5	$0.543_{\pm0.332}$	2.278	$0.404_{\pm0.352}$	3.074	2.723
ADASYN (k=12)	$0.506_{\pm0.341}$	2.482	$0.523_{\pm0.362}$	2.741	$0.533_{\pm0.336}$	2.889	$0.396_{\pm0.355}$	2.722	2.709
SL-SMOTE (k=12)	$0.502_{\pm0.344}$	2.963	$0.522_{\pm0.367}$	3.019	$0.523_{\pm0.338}$	3.241	$0.436_{\pm0.334}$	2.463	2.922
SMOM	$\mathbf{0.525_{\pm0.332}}$	**1.519**	$\mathbf{0.542_{\pm0.354}}$	**1.741**	$\mathbf{0.550_{\pm0.330}}$	**1.593**	$0.435_{\pm0.333}$	**1.741**	**1.649**

Method	C4.5		AdaBoost (C4.5)		MLP		NB		The overall
	Avg-MAUC	Mean rank	Avg-MAUC	Mean rank	Avg-MAUC	Mean rank	Avg-MAUC	Mean rank	mean rank
SMOTE (k=12)	$0.847_{\pm0.109}$	3.037	$0.875_{\pm0.106}$	2.426	$0.882_{\pm0.097}$	2.111	$0.868_{\pm0.094}$	2.667	2.560
ADASYN (k=12)	$0.851_{\pm0.108}$	2.278	$0.874_{\pm0.107}$	2.741	$0.88_{\pm0.096}$	2.648	$0.867_{\pm0.095}$	2.63	2.574
SL-SMOTE (k=12)	$0.846_{\pm0.113}$	3.074	$0.873_{\pm0.108}$	3.148	$0.875_{\pm0.101}$	3.574	$0.865_{\pm0.096}$	3.185	3.245
SMOM	$\mathbf{0.853_{\pm0.108}}$	**1.611**	$\mathbf{0.878_{\pm0.104}}$	**1.685**	$\mathbf{0.884_{\pm0.096}}$	**1.667**	$\mathbf{0.871_{\pm0.093}}$	**1.519**	**1.621**

Method	C4.5		AdaBoost (C4.5)		MLP		NB		The overall
	Avg-Recall	Mean rank	Avg-Recall	Mean rank	Avg-Recall	Mean rank	Avg-Recall	Mean rank	mean rank
SMOTE (k=12)	$0.631_{\pm0.253}$	3.13	$0.658_{\pm0.273}$	2.278	$0.706_{\pm0.215}$	2.259	$0.588_{\pm0.286}$	2.667	2.584
ADASYN (k=12)	$0.656_{\pm0.243}$	2.037	$0.648_{\pm0.283}$	2.87	$0.714_{\pm0.212}$	2.278	$0.587_{\pm0.307}$	2.852	2.509
SL-SMOTE (k=12)	$0.634_{\pm0.265}$	2.944	$0.642_{\pm0.298}$	3.019	$0.664_{\pm0.247}$	3.426	$0.59_{\pm0.28}$	2.889	3.070
SMOM	$\mathbf{0.669_{\pm0.237}}$	**1.889**	$\mathbf{0.676_{\pm0.261}}$	**1.833**	$\mathbf{0.717_{\pm0.205}}$	**2.037**	$\mathbf{0.62_{\pm0.255}}$	**1.593**	**1.838**

Method	C4.5		AdaBoost (C4.5)		MLP		NB		The overall
	Avg-Precision	Mean rank	Avg-Precision	Mean rank	Avg-Precision	Mean rank	Avg-Precision	Mean rank	mean rank
SMOTE (k=12)	$0.55_{\pm0.293}$	3.093	$0.614_{\pm0.301}$	2.778	$0.566_{\pm0.287}$	2.426	$0.565_{\pm0.247}$	2.593	2.723
ADASYN (k=12)	$0.563_{\pm0.285}$	2.611	$0.612_{\pm0.304}$	3.019	$0.532_{\pm0.300}$	3.093	$0.544_{\pm0.265}$	2.704	2.857
SL-SMOTE (k=12)	$0.608_{\pm0.277}$	**1.907**	$\mathbf{0.655_{\pm0.294}}$	**1.87**	$\mathbf{0.587_{\pm0.283}}$	2	$0.53_{\pm0.288}$	2.444	**2.055**
SMOM	$0.573_{\pm0.282}$	2.389	$0.631_{\pm0.292}$	2.333	$0.569_{\pm0.279}$	2.482	$\mathbf{0.57_{\pm0.242}}$	**2.259**	2.366

SMOM with SL-SMOTE in terms of F-measure as it is a good trade-off between recall and precision. The results are provided in Table S11 and S12, which shows SMOM is superior to SL-SMOTE over all base classifiers.

Experiment 2. As SMOM uses a higher value for the parameter k (i.e., $k1 = 12$) in comparison with previous k-NN-based oversampling approaches such as SMOTE, ADASYN and SL-SMOTE (their default setting about k is 5), it cannot rule out the possibility that the superiority of SMOM, in experiment 1, is due to simply employing a different value for parameter k. This experiment investigates whether the $k1$ configured to a higher value is the reason that SMOM shows the excellence above.

Like SMOM, we set the k of SMOTE, ADASYN, and SL-SMOTE to be 12. If the considered class has less than 12 instances, k is reset to the number of instances of this class minus one for all compared algorithms. Table 4 exhibits the results of mean rank (the detailed results are provided in Table S13, S14, S15 and S16). In terms of MG, MAUC and recall, SMOM still obtains the best mean ranks over all classifiers. In terms of precision, the OMR of SL-SMOTE is the lowest, followed by SMOM. It is worth noting that although SL-SMOTE is also devoted to reduce over generalization, it is the worst approach in MG, MAUC and recall. The reason may be that PRI and ARI tend to be far away from each other when k is set to a high value, utilizing their safe levels which only consider the instance distribution in the k-nearest neighborhoods is not enough for safely generating the synthetic instances. Conversely, it may restrain synthetic instances to open minority class regions.

Table 5 presents the corresponding Wilcoxon signed-ranks test results. We can find that SMOM maintains almost all statistical differences with the compared algorithms in MG, MAUC and recall. It means that the performance advantages of SMOM come from its own oversampling mechanism rather than the difference of k.

Experiment 3. One interesting question is where is the superiority of SMOM over previous k-NN-based oversampling approaches? To this end, we compare SMOM with SMOTE ($k = 12$) further. Five performance metrics were refined. The first two are the average recall on the majority classes and minority classes, respectively. They are denoted by $recall_{mins}$ and $recall_{majs}$ in turn. The

last three are the average AUCs that are respectively formed by means of averaging pairwise comparisons between any two majority classes, between one of the majority classes and one of the minority classes, and between any two minority classes. We call them $MAUC_{majs}$, $MAUC_{maj-min}$ and $MAUC_{mins}$ in sequence. Table 6 presents the results of significance test between SMOM and SMOTE ($k = 12$) in these five metrics with C4.5 classifier. One can see that SMOM statistically outperforms SMOTE in $recall_{mins}$, $MAUC_{maj-min}$ and $MAUC_{mins}$. In fact, the similar observation also exist in the experimental results of other three classifiers, which can be seen from Table S18, S19 and S20.

Further analysis. As previous k-NN-based oversampling algorithms create the synthetic instances in the directions of k-nearest neighbors randomly, it can cause that synthetic instances may be wrongly generated in the data regions of other classes, which is harmful to constructing the prediction model with good generalization ability [3]. Our SMOM assigns the selection weights to the neighbor directions by considering the over generalization factor. The neighbor directions which can result in serious over generalization will be given small weights. Hence, SMOM has ability to avoid over generalization. By means of this property, SMOM is appropriate to configure a high value of k for better protecting minority class instances, and does not have to concern the resulting negative effects to some extent. It can help to explain why SMOM, as compared to other oversampling algorithms, has a larger improvement in the prediction accuracies of minority classes (i.e., higher recall and $recall_{mins}$) and is more able to balance the recognition among different classes (i.e., higher MG).

In addition, the mechanism of avoiding over generalization in SMOM suppresses the invasion of synthetic instances for the regions of incorrect classes (especially incorrect minority classes) when oversampling one class. It can effectively reduce the overlapping occurred in minority classes, and finally promotes the learnability of all minority classes. Hence, the resulting classifier will be better at distinguishing the minority class from the majority classes and from the other minority classes (i.e., higher $MAUC_{min-maj}$ and $MAUC_{mins}$).

Table 5
Significance tests of MG, MAUC, recall and precision between SMOM and each of SMOTE (k=12), ADASYN ($k = 12$), and SL-SMOTE ($k = 12$) (the results associated with ADASYN and SL-SMOTE are displayed in gray as they are originally prepared to handle two-class imbalanced data).

	Method	C4.5			AdaBoost(C4.5)			MLP			NB		
	SMOM vs	R^+	R^-	T	R^+	R^-	T	R^+	R^-	T	R^+	R^-	T
MG	SMOTE (k=12)	333.5	44.5	**44.5(++)**	316.5	61.5	**61.5(++)**	275	103	**103(++)**	335	43	**43(++)**
	ADASYN (k=12)	330	48	48(++)	300	78	78(++)	317	61	61(++)	304.5	73.5	73.5(++)
	SL-SMOTE(k=12)	307.5	70.5	70.5(++)	305.5	72.5	72.5(++)	348	30	30(++)	232	146	146(+)
	Method	C4.5			AdaBoost(C4.5)			MLP			NB		
	SMOM vs	R^+	R^-	T	R^+	R^-	T	R^+	R^-	T	R^+	R^-	T
MAUC	SMOTE (k=12)	303	75	**75(++)**	329.5	48.5	**48.5(++)**	273	105	**105(++)**	348.5	29.5	**29.5(++)**
	ADASYN (k=12)	341	37	37(++)	333	45	45(++)	316.5	61.5	61.5(++)	313.5	64.5	64.5(++)
	SL-SMOTE(k=12)	346.5	31.5	31.5(++)	338	40	40(++)	374.5	3.5	3.5(++)	343.5	34.5	34.5(++)
	Method	C4.5			AdaBoost(C4.5)			MLP			NB		
	SMOM vs	R^+	R^-	T	R^+	R^-	T	R^+	R^-	T	R^+	R^-	T
Recall	SMOTE (k=12)	234.5	143.5	143.5(+)	283	95	**95(++)**	231.5	146.5	146.5(+)	332.5	45.5	**45.5(++)**
	ADASYN (k=12)	345	33	33(++)	320.5	57.5	57.5(++)	220.5	157.5	157.5(+)	276.5	101.5	101.5(++)
	SL-SMOTE(k=12)	303.5	74.5	74.5(++)	309.5	68.5	68.5(++)	347.5	30.5	30.5(++)	301	77	77(++)
	Method	C4.5			AdaBoost(C4.5)			MLP			NB		
	SMOM vs	R^+	R^-	T	R^+	R^-	T	R^+	R^-	T	R^+	R^-	T
Precision	SMOTE (k=12)	254	124	124(+)	264	114	**114(+*)**	201	177	177(+)	232	146	146(+)
	ADASYN (k=12)	288.5	89.5	89.5(++)	272.5	105.5	105.5(++)	271	107	107(++)	246	132	132(+)
	SL-SMOTE(k=12)	114	264	**114(-*)**	104.5	273.5	104.5(−−)	118	260	118(-*)	222.5	155.5	155.5(+)

Table 6
Significance test results between SMOM and SMOTE ($k = 12$) in terms of $recall_{majs}$, $recall_{mins}$, $MAUC_{majs}$, $MAUC_{maj-min}$, and $MAUC_{mins}$ with C4.5 as the base classifier.

Method	$Recall_{majs}$		$Recall_{mins}$		$MAUC_{majs}$		$MAUC_{maj-min}$		$MAUC_{mins}$	
	Avg-Result	Mean rank	Avg-Result	Mean rank	Avg-Result	Mean rank	Avg-Result	Mean rank	Avg-Result	Mean rank
SMOTE ($k = 12$)	**0.695**$_{\pm 0.226}$	**1.426**	0.627$_{\pm 0.238}$	1.833	**0.803**$_{\pm 0.143}$	**1.463**	0.86$_{\pm 0.1}$	1.685	0.857$_{\pm 0.128}$	1.704
SMOM $T=min\{R^+, R^-\}$	0.691$_{\pm 0.224}$ $min\{183, 195\} = 183(-)$	1.574	**0.639**$_{\pm 0.234}$ $min\{331.5, 46.5\} =$**46.5(++)**	**1.167**	**0.803**$_{\pm 0.143}$ $min\{108.5, 122.5\} = 108.5(-)$	1.537	**0.862**$_{\pm 0.1}$ $min\{269, 109\} =$**109(+*)**	**1.315**	**0.859**$_{\pm 0.129}$ $min\{289.5, 88.5\} =$**88.5(++)**	**1.296**

5. Conclusion

In this paper, we proposed a *k*-NN-based synthetic oversampling algorithm, SMOM, to deal with multiclass imbalanced problems. In SMOM, each of the neighbor direction is assigned a selection weight of being the direction of generating synthetic instances. The neighbor directions which may cause serious over generalization will be given small selection weights. In this way, SMOM builds a mechanism of avoiding over generalization. This mechanism provides two main advantages: 1) SMOM can safely and effectively broaden the regions of minority classes by simply referencing more neighbor directions. 2) It emphasizes on preventing the synthetic instances of one minority class from falling in the regions of other minority classes, so the class overlapping among minority classes can be restricted. Our experiment results demonstrated that SMOM can obtain the higher prediction accuracy and the better distinguishing ability for the minority classes than the compared oversampling algorithms.

We do not shun the point that the implementation of SMOM seems to be complex. This is mainly because it involves the clustering algorithm NBDOS and the two-round filtration. Both NBDOS and two-round filtration are used to reduce the time cost of SMOM. The computation of selection weights for a part of minority class instances can be avoided by applying NBDOS, and utilizing the two-round filtration can save a lot of distance calculations between instances. Experiments show that the time consuming of SMOM is near that of existing simple algorithms such as ADASYN in practice (the complexity analysis is provided in the supplementary material).

The effectiveness of SMOM is evaluated on 27 multiclass imbalanced data sets with C4.5, AdaBoost.M1 ensemble of C4.5, MLP and NB as the base classifiers. Six typical existing oversampling approaches are selected to compare our SMOM. The experimental results show that SMOM often statistically outperformed other compared methods in terms of MG, MAUC and recall.

As our current SMOM is only suitable to handle continuous attributes, we intend to generalize SMOM to deal with the imbalanced data with nominal and ordinal attributes in future work.

Conflict of interest

None declared.

Acknowledgments

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.patcog.2017.07.024

References

[1] H. He, E.A. Garcia, Learning from imbalanced data, IEEE Trans. Knowl. Data Eng. 21 (9) (2009) 1263–1284.
[2] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, Smote: synthetic minority over-sampling technique, J. Mach. Learn. Res. (2002) 321–357.

[3] L. Abdi, S. Hashemi, To combat multi-class imbalanced problems by means of over-sampling techniques, IEEE Trans. Knowl. Data Eng. 28 (1) (2016) 238–251.

[4] S. Chen, H. He, E.A. Garcia, Ramoboost: ranked minority oversampling in boosting, IEEE Trans. Neural Networks 21 (10) (2010) 1624–1642.

[5] P. Lim, C.K. Goh, K.C. Tan, Evolutionary cluster-based synthetic oversampling ensemble (eco-ensemble) for imbalance learning, IEEE Trans. Cybern. (2016).

[6] M. Lin, K. Tang, X. Yao, Dynamic sampling approach to training neural networks for multiclass imbalance classification, IEEE Trans. Neural Networks Learn. Syst. 24 (4) (2013) 647–660.

[7] S. Wang, X. Yao, Multiclass imbalance problems: analysis and potential solutions, IEEE Trans. Syst., Man Cybern. B, Cybern. 42 (4) (2012) 1119–1130.

[8] S. Barua, M.M. Islam, X. Yao, K. Murase, Mwmote–majority weighted minority oversampling technique for imbalanced data set learning, IEEE Trans. Knowl. Data Eng. 26 (2) (2014) 405–425.

[9] B. Wang, N. Japkowicz, Imbalanced data set learning with synthetic samples, in: Proc. IRIS Machine Learning Workshop, 2004, p. 19.

[10] H. He, Y. Bai, E.A. Garcia, S. Li, Adasyn: Adaptive synthetic sampling approach for imbalanced learning, in: Proc.IntÍ J. Conf. Neural Networks, IEEE, 2008, pp. 1322–1328.

[11] K.S. Bunkhumpornpat Chumphol, C. Lursinsap, Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem, in: In Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer Berlin Heidelberg, 2009, pp. 475–482.

[12] R. Anand, K. Mehrotra, C.K. Mohan, S. Ranka, Efficient classification for multiclass problems using modular neural networks, IEEE Trans. Neural Networks 6 (1) (1995) 117–124.

[13] I. Nekooeimehr, S.K. Lai-Yuen, Cluster-based weighted oversampling for ordinal regression (cwos-ord), Neurocomputing 218 (2016) 51–60.

[14] N. Japkowicz, S. Stephen, The class imbalance problem: a systematic study, Intell. Data Anal. 6 (5) (2002) 429–449.

[15] G.M. Weiss, Mining with rarity: a unifying framework, ACM SIGKDD Explorations Newsletter 6 (1) (2004) 7–19.

[16] T. Jo, N. Japkowicz, Class imbalances versus small disjuncts, ACM Sigkdd Explorations Newsletter 6 (1) (2004) 40–49.

[17] G.E. Batista, R.C. Prati, M.C. Monard, A study of the behavior of several methods for balancing machine learning training data, ACM Sigkdd Explor. Newslett. 6 (1) (2004) 20–29.

[18] H. Cao, X.-L. Li, D.Y.-K. Woon, S.-K. Ng, Integrated oversampling for imbalanced time series classification, IEEE Trans. Knowl. Data Eng. 25 (12) (2013) 2809–2822.

[19] J. Wu, H. Xiong, J. Chen, Cog: local decomposition for rare class analysis, Data Min. Knowl. Disc. 20 (2) (2010) 191–220.

[20] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in: Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 96, 1996, pp. 226–231.

[21] J. Sander, M. Ester, H.-P. Kriegel, X. Xu, Density-based clustering in spatial databases: the algorithm gdbscan and its applications, Data Mining Knowl. Discovery 2 (2) (1998) 169–194.

[22] K. Napierala, J. Stefanowski, Identification of different types of minority class examples in imbalanced data, in: Hybrid artificial intelligent systems, Springer, 2012, pp. 139–150.

[23] J.A. Sáez, B. Krawczyk, M. Woźniak, Analyzing the oversampling of different classes and types of examples in multi-class imbalanced datasets, Pattern Recognit. 57 (2016) 164–178.

[24] T. Maciejewski, J. Stefanowski, Local neighbourhood extension of smote for mining imbalanced data, in: Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on, IEEE, 2011, pp. 104–111.

[25] W. Chu, Z. Ghahramani, Gaussian processes for ordinal regression, J. Mach. Learn. Res. 6 (2005) 1019–1041.

[26] A. Frank, A. Asuncion, et al., Uci machine learning repository(2010).

[27] Y. Sun, M.S. Kamel, Y. Wang, Boosting for learning multiple classes with imbalanced class distribution, in: Proc. IntÍ Conf. Data Mining, IEEE, 2006, pp. 592–602.

[28] D.J. Hand, R.J. Till, A simple generalisation of the area under the roc curve for multiple class classification problems, Mach. Learn. 45 (2) (2001) 171–186.

[29] I.H. Witten, E. Frank, Data mining: practical machine learning tools and techniques, Morgan Kaufmann, 2005.

[30] G.W. Corder, D.I. Foreman, Nonparametric statistics: a step-by-step approach, John Wiley & Sons, 2014.

Tuanfei Zhu received the BS degree in information and computing science from Zhengzhou University of Light Industry, China, in 2005, and the MS degree in computer science from Sichuan University, China, in 2012. Since 2014, he has been working toward the PhD degree in the College of Information Science and Engineering, Hunan University. His research interests include class imbalance learning, ensemble learning, online learning and multilabel learning.

Yaping Lin received the BS degree in computer application from Hunan University, China, in 1982, and the MS degree in computer application from the National University of Defense Technology, China, in 1985. He received the PhD degree in control theory and application from Hunan University in 2000. He has been a professor and a PhD supervisor in Hunan University since 1996. From 2004–2005, he was a visiting researcher at the University of Texas at Arlington. His research interests include machine learning, network security, and wireless sensor networks. He is a member of the IEEE.

Yonghe Liu received his BS and MS degrees in control theory and engineering from Tsinghua University, Beijing, China, in 1998 and 1999, respectively. He received the PhD degree in computer engineering from Rice University, USA, in 2003. He is an associate professor at the Department of Computer Science and Engineering in University of Texas at Arlington, USA. His experience includes working for the DSP Solutions R&D Center at Texas Instruments. His current research interests are in wireless networking, media access control and system integration.