Noname manuscript No.
(will be inserted by the editor)

Surrounding neighborhood-based SMOTE for learning from imbalanced data sets

V. García · J. S. Sánchez · R. Martín-Félez · R. A. Mollineda

the date of receipt and acceptance should be inserted later

Abstract Many traditional approaches to pattern classification assume that the problem classes share similar prior probabilities. However, in many real-life applications, this assumption is grossly violated. Often, the ratios of prior probabilities between classes are extremely skewed. This situation is known as the class imbalance problem. One of the strategies to tackle this problem consists of balancing the classes by resampling the original data set. The SMOTE algorithm is probably the most popular technique to increase the size of the minority class by generating synthetic instances. From the idea of the original SMOTE, we here propose the use of three approaches to surrounding neighborhood with the aim of generating artificial minority instances, but taking into account both the proximity and the spatial distribution of the examples. Experiments over a large collection of databases and using three different classifiers demonstrate that the new surrounding neighborhood-based SMOTE procedures significantly outperform other existing over-sampling algorithms.

Keywords Imbalance · Over-sampling · Surrounding neighborhood · Nearest centroid neighborhood · Gabriel graph · Relative neighborhood graph · SMOTE

1 Introduction

Class imbalance constitutes a challenging problem that has recently received much attention in a wide variety of research fields such as Data Mining, Machine Learning and Pattern Recognition [16,31,43,47]. The class imbalance problem occurs when one or several classes (the majority classes) vastly outnumber the other classes (the minority classes), which are usually the most important classes and often with the highest misclassification costs. It has been observed that class imbalance may affect the performance of most standard classification systems, which assume a relatively well-balanced class distribution and equal misclassification costs [28]. This issue results particularly relevant in a variety of real-life applications, such as the diagnosis of infrequent diseases [10,35], credit risk assessment [4,27], detection of software defects [30], fraud detection in telecommunications [15,24], prediction of customer insolvency [11], text categorization [9,44], and detection of oil spills in radar images [32]. In two-class imbalanced problems, the examples of the minority class are typically referred to as positive, whereas the instances of the majority class are referred to as negative.

The issue of class imbalance has been addressed by numerous approaches at both data and algorithmic levels [23]. The methods at the algorithmic level modify the existing learning algorithms for biasing the discrimination process towards the minority class [2,38,39], whereas the data level solutions consist of artificially resampling the original data set, either by over-sampling the minority class [2,33,49] and/or under-sampling the majority class [3,7,20,21], until the classes are approximately equally represented.

In general, the resampling strategies have been the most investigated because they are independent of the underlying classifier and can be easily implemented for any problem [18]. However, the methods at the data level also present some drawbacks because they artificially alter the original class distribution. For example, under-sampling may throw out potentially useful data,

Institute of New Imaging Technologies, Department of Computer Languages and Systems, Universitat Jaume I, Av. Vicent Sos Baynat s/n, 12071 Castellón de la Plana, Spain
E-mail: sanchez@uji.es (J. S. Sánchez)

while over-sampling artificially increases the size of the data set and consequently, it worsens the computational burden of the learning algorithm. Despite conclusions about what is the most suitable resampling strategy for the class imbalance problem are divergent, several studies have reported that over-sampling usually performs better than under-sampling [3,21,46].

The present paper concentrates on the over-sampling strategy and more specifically, extends the well-known SMOTE (Synthetic Minority Over-sampling TEchnique) algorithm [7] by exploiting an alternative neighborhood formulation, namely *surrounding neighborhood* [40]. A key feature of this type of neighborhood is that the neighbors of a sample are considered in terms of both proximity and spatial distribution with respect to the sample, showing some practical advantages over the conventional neighborhood that is only based on the minimum distance. The use of the surrounding neighborhood to over-sample the minority class generates new synthetic examples that will be homogeneously distributed around the original positive instances, contributing to spread the influence region of the minority class. The thorough experimental study carried out proves the significant performance gains of our approach when compared to other state-of-the-art algorithms.

The rest of this paper is organized as follows. Section 2 reviews the SMOTE algorithm and some of its most relevant extensions. In Section 3, the general concept of surrounding neighborhood and two of its implementations are presented. The modification of SMOTE based on the different surrounding neighborhood formulations is introduced in Section 4. Section 5 describes the experimental framework, including the data sets, the over-sampling algorithms, the classifiers, the performance evaluation metrics and the statistical tests used in the present analysis. In Section 6, the experiments are carried out and their results are discussed. Finally, Section 7 remarks the main conclusions and outlines possible directions for future research.

2 The SMOTE algorithm and some variants

The simplest strategy to expand the minority class is random over-sampling (ROS), which corresponds to a non-heuristic method that balances the class distribution through a random replication of positive examples [3,36]. Although effective, this method may increase the likelihood of overfitting since it makes exact copies of the minority class instances [7].

In order to avoid overfitting, Chawla et al. [7] proposed the SMOTE algorithm to up-size the minority class. Instead of merely replicating positive instances, this method generates artificial examples of the minority class by interpolating existing instances that lie close together. It first finds the k positive nearest neighbors for each minority class example and then, the synthetic examples are generated in the direction of some or all of those nearest neighbors.

SMOTE allows the classifier to build larger decision regions that contain nearby instances of the minority class. Depending upon the amount of over-sampling required, a certain number of instances from the k nearest neighbors are randomly chosen. In the experiments reported in the original paper, k is set to five. The generation procedure for each minority class example can be explained as follows: (i) take the difference between the feature vector (instance) under consideration and one of its k minority class nearest neighbors; (ii) multiply this difference by a random number between 0 and 1; and (iii) add it to the feature vector that corresponds to the new synthetic example of the minority class.

Although SMOTE has proved to be an effective tool for handling the class imbalance problem, it may overgeneralize the minority class as it does not take care of the distribution of majority class neighbors, especially when the minority class is very sparse with respect to the majority class. As a result, the generation of synthetic examples may increase the overlapping between classes [37]. From the original SMOTE algorithm, several modifications have further been proposed in the literature, most of them pursuing to determine the region in which the positive examples should be generated. Among them, one of the most widely-known generalizations corresponds to the Borderline SMOTE (B-SMOTE) algorithm [20], which consists of using only positive examples close to the decision boundary since these are more likely to be misclassified.

The Safe-Level SMOTE (SL-SMOTE) algorithm [5] calculates a "safe level" coefficient (sl) for each minority class example, which is defined as the number of other minority class instances among its k neighbors. If the coefficient sl is equal or close to 0, such an example is considered as noise; if sl is close to k, then this example may be located in a safe region of the minority class. The idea is to direct the generation of new synthetic examples close to safe regions.

Finally, other less known extensions of SMOTE are the FSMOTE algorithm proposed by Zhang et al. [50], which utilizes fractal interpolation theory to generate the synthetic positive examples, and the LLE-based SMOTE [48] that implements the locally linear embedding algorithm to map the high-dimensional data onto a low-dimensional space where the synthetic instances of the minority class are then generated and mapped back to the original input space. On the other hand,

the MSMOTE algorithm [25] divides the instances of the minority class into three groups: safe, border and latent noise instances. When MSMOTE generates new examples, the strategy to select the nearest neighbors depends on the group to which the instance belongs. For safe instances, the algorithm randomly selects a data point from the k neighbors; for border instances, it only selects the nearest neighbor; for latent noise instances, it does nothing. Maciejewski and Stefanowski introduced the LN-SMOTE [37], which exploits more precisely information about the local neighborhood of the considered examples. The SMOTEBoost [8] combines SMOTE with the standard boosting procedure: it utilizes SMOTE for improving the prediction of the minority class and boosting for not affecting the accuracy over the entire data set.

3 Surrounding neighborhood

Intuitively, the concept of neighborhood should be such that the neighbors are as close to an instance as possible but also, the neighbors should lie as homogeneously around it as possible. The second condition is a consequence of the first in the asymptotic case but in some practical situations, the geometrical location may become much more important than the actual distances to appropriately characterize an instance by means of its neighborhood [40]. As the traditional neighborhood takes into account the first property only, the nearest neighbors may not be placed symmetrically around the instance if the neighborhood in the data set is not spatially homogeneous. In fact, it has been shown that the use of local distance measures can significantly improve the classifier behavior in the finite sample size case [41].

Alternative neighborhood definitions have been proposed as a way to overcome the problem just pointed out. These consider both proximity and symmetry so as to define the general concept of *surrounding neighborhood* [40]: they try to search for neighbors of an instance close enough (in the basic distance sense), but also in terms of their spatial distribution with respect to that instance. The *nearest centroid neighborhood* and the *graph neighborhood* are two representative examples of the surrounding neighborhood, which have demonstrated to behave better than the conventional nearest neighborhood for a number of pattern classification problems [40,51].

3.1 Nearest centroid neighborhood

The first definition of surrounding neighborhood comes from the nearest centroid neighborhood (NCN) concept [6]. Let p be a sample whose k neighbors should be found from a set of n points $X = \{x_1, \ldots, x_n\}$. These k neighbors are such that (a) they are as near p as possible and, (b) their centroid is also as close to p as possible. Both conditions can be satisfied through the following iterative procedure:

Algorithm 1 Computation of NCN

Input:
$X = \{x_1, \ldots, x_n\}$ Input data set
k Number of neighbors to search
p Query point
Output:
$Q = \{q_1, \ldots, q_k\}$ Set of k nearest centroid neighbors

The first NCN of p is its nearest neighbor, say q_1
for $i = 2$ to k **do**
  The i'th neighbor q_i is such that the centroid of this and all previously selected neighbors, $q_1, \ldots, q_{i-1}$ is the closest to p
end for

This definition leads to a type of neighborhood in which both closeness and spatial distribution of neighbors are taken into account because of the centroid criterion. Besides, the proximity of the nearest centroid neighbors to the sample is guaranteed because of the incremental nature of the way in which those are obtained from the first nearest neighbor. However, note that the iterative procedure outlined in Algorithm 1 clearly does not minimize the distance to the centroid because it gives precedence to the individual distances instead. On the other hand, the region of influence of the NCN results bigger than that of the traditional nearest neighborhood (NN); as can be seen in Fig. 1, the 4-NCN (a, b, c, d) of a given point p enclose a region quite bigger than the region determined by the 4-NN (a, e, f, g).

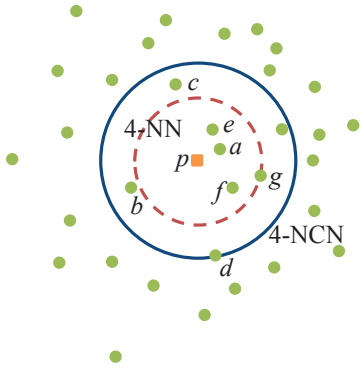

Fig. 1 An example of NCN compared to the traditional NN

3.2 Graph neighborhood

A proximity graph defined on a set $X = \{x_1, \ldots, x_n\}$ is an undirected graph $G = (V, E)$, which comprises a set of nodes $V = X$ and a set of edges $E : V \times V$, such that $(x_i, x_j) \in E$ if and only if the points x_i and x_j fulfill some mutual neighborhood criterion; then x_i is said to be neighbor of x_j and vice versa. The set of graph neighbors of a given point constitutes its *graph neighborhood* [40]. The graph neighborhood of a subset $S \subseteq V$ consists of the union of all the graph neighbors of every node in S.

Two well-known examples of proximity graphs are the Gabriel graph (GG) and the relative neighborhood graph (RNG) [29], which are subgraphs of the Delaunay triangulation (DT): RNG $\subseteq$ GG $\subseteq$ DT. As definitions for and properties of the GG and RNG are widely available in the literature, only essential concepts needed in this paper are reproduced here.

3.2.1 Gabriel graph

Let $d(\cdot, \cdot)$ be the Euclidean distance between two points in $\mathbb{R}^d$. The set of edges in a GG consists of the pairs of points that satisfy the following relation:

$$(x_i, x_j) \in E \Longleftrightarrow d^2(x_i, x_j) \leq d^2(x_i, x_k) + d^2(x_j, x_k) \quad \forall x_k \in X,\ k \neq i, j$$

Geometrically, two points x_i and x_j are said to be Gabriel neighbors if and only if there is no other point of X lying in the *hypersphere of influence* $\Gamma(x_i, x_j)$ centered at their middle point and whose diameter is the distance between x_i and x_j. In Fig. 2, for example, both p and q are Gabriel neighbors of the point a, but r is not because q lies inside the sphere of influence determined by the points a and r.

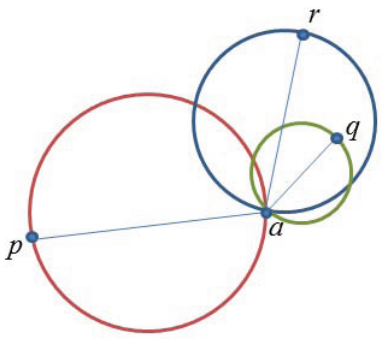

Fig. 2 An example of Gabriel neighborhood

3.2.2 Relative neighborhood graph

In a similar fashion, the set of edges that belong to an RNG comprises the pairs of points that fulfill the following neighborhood property:

$$(x_i, x_j) \in E \Longleftrightarrow d(x_i, x_j) \leq max[d(x_i, x_k), d(x_j, x_k)] \quad \forall x_k \in X,\ k \neq i, j$$

In this case, its corresponding geometric interpretation is based on the concept of *lune* Λ_{x_i, x_j}, which is defined as the disjoint intersection between two hyperspheres centered at x_i and x_j and whose radii are equal to the distance between them. Two points x_i and x_j are said to be *relative neighbors* if and only if their lune does not contain other points of the set X. In Fig. 3, the points q and a are not relative neighbors because p lies inside their lune; conversely, p and a are relative neighbors because their lune is empty.

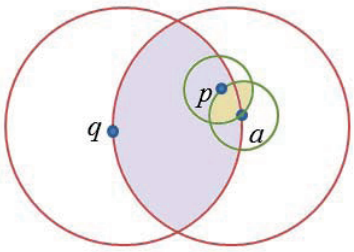

Fig. 3 An example of relative neighborhood

4 Surrounding SMOTE

As already mentioned, the surrounding neighborhood methods have successfully been applied to a number of pattern classification and data mining problems. These approaches can effectively help in several situations (finite sample size case), in which training instances do not fully represent the underlying statistics and/or the distance used (irrelevant in the asymptotic case) exhibits some undesirable properties. In fact, the ultimate goal of the surrounding neighborhood is to overcome some shortcomings of the conventional NN-based techniques.

Hence, based upon the analysis just stated, we here propose to employ the three surrounding neighborhood realizations (NCN, GG, and RNG) for over-sampling the minority class by means of a modification of the standard SMOTE algorithm.

SMOTE finds the k positive nearest neighbors for each minority class example in the training set and then, it generates artificial samples in the direction of some (or all) of those nearest neighbors. Instead of nearest neighbors, now we propose to select surrounding positive neighbors for each instance of the minority class. The rationale behind this modification of the original SMOTE algorithm is that these surrounding neighbors will extend the region of new synthetic samples and therefore, it seems that the resulting over-sampled set can describe better the decision boundaries.

The Surrounding SMOTE algorithm (using the NCN concept) can be written as follows:

Algorithm 2 Surrounding SMOTE

Input:
$P = \{p_1, \ldots, p_{min}\}$ Training positive examples
min Number of minority examples
N Number of synthetic instances to generate for each positive example
k Number of neighbors
Output:
$Synthetic$ Set of artificial instances

for $i = 1$ to min **do**
  Find k nearest centroid neighbors of p_i
  for $j = N$ to 1 **do**
    Choose randomly one of the k nearest centroid neighbors of p_i, say $q(p_i)$
    $diff = p_i - q(p_i)$
    $gap =$ random number between 0 and 1
    $newSample = p_i + gap * diff$
    $Synthetic \longleftarrow newSample$
  end for
end for

The size of the set of synthetic instances will be $N \times min$. Note that in the case of GG and RNG, we do not have to provide the number of neighbors (k), since each training positive instance may have a different number of graph neighbors. Apart from this difference, the rest of the procedure for Surrounding SMOTE using proximity graphs will be exactly the same as the one reported in Algorithm 2.

From a practical point of view, one disadvantage of the graph neighborhood compared to the nearest centroid neighborhood is its higher computational cost. The graph neighbors of a set of points can be computed exhaustively by using the brute-force method (i.e. by testing all pairs of samples in the set X), with a complexity of $O(n^3)$. Nevertheless, in the case of the GG and RNG, there exist heuristic methods that allow to considerably reduce the number of pairs to be tested for graph neighbors and whose computational cost is close to $O(n^2)$ [29]. In addition, it is worth noting that the Surrounding SMOTE only computes the graph neighbors that belong to the minority class (usually consisting of a very small number of instances), what results in a low complexity algorithm.

5 Experimental set-up

An empirical comparison between the three Surrounding SMOTE methods here proposed and other oversampling algorithms has been performed over a total of 39 data sets taken from the KEEL Data Set Repository (`http://www.keel.es/dataset.php`). Note that all the original multi-class databases have firstly been transformed into two-class problems. Table 1 summarizes the main characteristics of the data sets, including the imbalance ratio (IR), that is, the number of negative examples divided by the number of positive examples. The fifth and sixth columns in Table 1 indicate the original classes that have been employed to shape the positive and negative classes, respectively. For example, in Glass567 database the classes 5, 6 and 7 have been combined to form a unique minority class, whereas the original classes 1, 2 and 3 have been joined to represent the majority class.

All the experiments have been carried out using the Weka learning environment [19] with the 1-NN decision rule, the C4.5 decision tree and the multi-layer perceptron (MLP) neural network, whose parameter values used in the experiments are given in Table 2. We have adopted a 5-fold cross-validation method to estimate the AUC measure: each data set has been divided into five stratified blocks of size $n/5$ (where n denotes the total number of examples in the data set), using four folds for training the classifiers and the remaining block as an independent test set. Therefore, the results correspond to the average over the five runs.

Each classifier has been applied to the original (imbalanced) training sets and also to sets that have been preprocessed by the three implementations of the Surrounding SMOTE (NCN-SMOTE, GG-SMOTE, RNG-SMOTE) and seven state-of-the-art over-sampling approaches taken from the KEEL data mining software tool [1]. Apart from the original SMOTE and two of its variants (B-SMOTE and SL-SMOTE), other four oversampling algorithms have been included in this study: ROS, agglomerative hierarchical clustering (AHC), adjusting the direction of the synthetic minority class examples (ADOMS), and adaptive synthetic (ADASYN). The Euclidean distance has been used with all the algorithms tested. The number of neighbors has been set to 5 for NCN-SMOTE, SMOTE, B-SMOTE, SL-SMOTE, ADOMS and ADASYN, and these neighbors have been searched among the minority class instances. The data sets have been balanced to the 50% distribution.

Table 1 Data sets used in the experimental analysis. The table is arranged in ascending order of the imbalance ratio

Data Set	#Examples	#Attributes	IR	Positive Class	Negative Class
Glass2	214	9	1.82	2	1,3,5,6,7
Wisconsin2	683	9	1.86	2	1
Pima2	768	8	1.87	2	1
Iris1	150	4	2.00	1	2,3
Glass1	214	9	2.06	1	2,3,5,6,7
Yeast2	1484	8	2.46	2	1,3,4,5,6,7,8,9,10
Haberman2	306	3	2.78	2	1
Vehicle3	846	18	2.88	3	1,2,4
Vehicle2	846	18	2.90	2	1,3,4
Vehicle4	846	18	2.99	4	1,2,3
Glass567	214	9	3.20	5,6,7	1,2,3
Vehicle1	846	18	3.25	1	2,3,4
Ecoli2	336	7	3.36	2	1,3,4,5,6,7,8
NewThyroid2	215	5	5.14	2	1,3
Ecoli3	336	7	5.46	3	1,2,4,5,6,7,8
Segment1	2308	19	6.02	1	2,3,4,5,6,7
NewThyroid3	215	5	6.17	3	1,2
Glass7	214	9	6.38	7	1,2,3,5,6
Yeast4	1484	8	8.10	4	1,2,3,5,6,7,8,9,10
Ecoli4	336	7	8.60	4	1,2,3,5,6,7,8
PageBlocks2345	5472	10	8.79	2,3,4,5	1
Yeast5vs1	514	8	9.08	5	1
Yeast5vs347810	528	8	9.35	5	3,4,7,8,10
Vowel1	988	13	9.98	1	2,3,4,5,6,7,8,9,10,11
Glass3	192	9	10.29	3	1,2,7
Glass3vs12567	214	9	11.59	3	1,2,5,6,7
Shuttle1	1829	9	13.87	1	5
Yeast8vs2	459	7	14.30	8	2
Glass5	214	9	15.46	5	1,2,3,6,7
Ecoli5	336	7	15.80	5	1,2,3,4,6,7,8
PageBlocks3	472	10	15.86	3	2,5
Glass6	184	9	19.44	6	1,2,7
Yeast8vs2459	693	8	22.10	8	2,4,5,9
Glass6vs12357	214	9	22.78	6	1,2,3,5,7
Yeast9	482	8	23.10	9	1
Yeast5	1484	8	28.10	5	1,2,3,4,6,7,8,9,10
Yeast8vs12910	947	8	30.57	8	1,2,9,10
Yeast6	1484	8	32.73	6	1,2,3,4,5,7,8,9,10
Yeast7	1484	8	39.11	7	1,2,3,4,5,6,8,9,10

Table 2 Parameters used in the classifiers

	Parameters
1-NN	Euclidean distance; Attribute values normalized
C4.5	Pruned tree; Confidence factor = 0.25; Minimum number of instances per leaf = 2
MLP	Learning rate = 0.3; Momentum = 0.2; Training time = 500; Hidden layers = (attributes + classes)/2; Attribute values normalized

The AHC over-sampling method [10] involves three major steps: (1) using single- and complete-linkage to form a dendogram, (2) gathering clusters from all levels of the dendogram and computing the cluster centroids as synthetic examples, and (3) concatenating centroids with the original minority class instances. The ADOMS algorithm [45] generates synthetic positive examples along the first principal component axis of local data distribution (made up of the positive instance being analyzed and its k neighbors). Finally, the ADASYN approach [22] is based on the idea of using a density distribution as a criterion to adaptively determine the number of synthetic examples to be generated for each minority class instance according to its level of difficulty in learning; in practice, the algorithm results in a balanced data set more focused on those positive instances that are harder to learn.

5.1 Performance evaluation metrics

Many measures have been developed for performance evaluation on imbalanced classification problems. Most

of them are based on the 2×2 confusion matrix as illustrated in Table 3.

Table 3 Confusion matrix for a two-class problem

	Predicted positive	*Predicted negative*
Positive class	True Positive (TP)	False Negative (FN)
Negative class	False Positive (FP)	True Negative (TN)

The most commonly used metric for measuring the performance of learning systems is the overall accuracy (and its counterpart, the error rate), which can be easily computed as follows:

$$Acc = \frac{TP + TN}{TP + FN + TN + FP} \tag{1}$$

Nevertheless, researchers have demonstrated that, when the prior class probabilities are very different, the overall accuracy is not appropriate because it does not consider misclassification costs, is strongly biased to favor the majority class, and is very sensitive to class skews [11,14,26]. Thus, in domains with imbalanced data, alternative metrics that measure the classification performance on positive and negative classes independently are required.

Two straightforward metrics that evaluate the classification performance on the majority and minority classes independently are the true positive rate (or sensitivity or recall) and the true negative rate (or specificity), that is, the percentage of instances (positive and negative, respectively) correctly classified. In general, in imbalanced problems more attention should be given to sensitivity than to specificity [34]:

$$sensitivity = \frac{TP}{TP + FN} \tag{2}$$

$$specificity = \frac{TN}{TN + FP} \tag{3}$$

Alternative performance evaluation criteria include the area under the ROC curve (AUC), the geometric mean of accuracies, the precision, the F-measure and the area under the precision-recall curve, among others. In general, these are good indicators of classification performance on imbalanced data because they are independent of the distribution of examples between classes.

The AUC, which constitutes one of the most commonly used metrics in the context of skewed class distributions, will be the method employed in the present paper to evaluate the performance of a variety of over-sampling techniques. For a binary problem, the AUC measure defined by a single point on the ROC curve is also referred to as balanced accuracy or macro-average, which can be computed as follows [42]:

$$AUC = \frac{sensitivity + specificity}{2} \tag{4}$$

5.2 Statistical tests

The AUC results have further been tested for statistically significant differences by means of non-parametric tests, which are generally preferred over the parametric methods because the usual assumptions of independence, normality and homogeneity of variance are often violated due to the non-parametric nature of the problems [12,17].

Both pairwise and multiple comparisons have been used in this paper. First, the Iman-Davenport's statistic has been applied to determine whether there exist significant differences among the over-sampling strategies. The process starts by computing the Friedman's ranking of the algorithms for each data set independently according to the AUC results: as there are eleven competing strategies, the ranks for each data set go from 1 (best) to 11 (worst); in case of ties, average ranks are assigned. Then the average rank of each algorithm across all data sets is computed. Under the null-hypothesis, which states that all the algorithms are equivalent, the Friedman's statistic can be computed as follows:

$$\chi_F^2 = \frac{12N}{K(K+1)} \left[\sum_j R_j^2 - \frac{K(K+1)^2}{4} \right] \tag{5}$$

where N denotes the number of data sets, K is the total number of algorithms, and R_j is the average rank of the algorithm j.

The χ_F^2 is distributed according to the Chi-square distribution with $K-1$ degrees of freedom, when N and K are big enough. However, as the Friedman statistic produces an undesirably conservative effect [13], the Iman-Davenport's statistic constitutes a better alternative. This is distributed according to the F-distribution with $K-1$ and $(K-1)(N-1)$ degrees of freedom:

$$F_F = \frac{(N-1)\chi_F^2}{N(K-1) - \chi_F^2} \tag{6}$$

If the null-hypothesis of equivalence is rejected, we can then proceed with a post hoc test. In this work, the Holm's post hoc test has been employed to ascertain

whether the best (control) algorithm performs significantly better than the remaining techniques [17].

Afterwards, the Wilcoxon's paired signed-rank test has been used to find out statistically significant differences between each pair of over-sampling algorithms. This statistic ranks the differences in performances of two algorithms for each data set, ignoring the signs, and compares the ranks for the positive and the negative differences. Let d_i be the difference between the performance scores of the two algorithms on i-th out of N data sets. The differences are ranked according to their absolute values. Let R^+ be the sum of ranks for the data sets on which the first algorithm outperforms the second, and R^- the sum of ranks for the opposite. Ranks of $d_i = 0$ are split evenly among the sums; if there is an odd number of them, one is ignored:

$$R^+ = \sum_{d_i>0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i)$$
$$R^- = \sum_{d_i<0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i) \quad (7)$$

Let Z be the smaller of the sums, $Z = \min(R^+, R^-)$. If Z is less than or equal to the value of the distribution of Wilcoxon for N degrees of freedom, the null-hypothesis that both algorithms perform equally well can be rejected.

6 Experimental results and discussion

The aim of the present study is three-fold. First, we want to establish whether the surrounding neighborhood is able to properly handle the class imbalance problem and to what extent its application can be robust across different classifiers. Second, we are also interested in investigating whether or not the surrounding versions of SMOTE outperform other classical over-sampling algorithms. Finally, we try to find out which of the three surrounding implementations yields the best performance in terms of the AUC metric.

In order to make more comprehensible the experimental results, Table 4 shows the average AUC values across all databases obtained with the 1-NN, C4.5 and MLP classifiers using the different over-sampling methods. The detailed results over each problem are given in the Appendix. As expected, classification with the imbalanced data sets produces the poorest performance, irrespective of the classifier used. However, the most important observation is that the results of the Surrounding SMOTE algorithms are among those of the best performing methods, especially in the case of NCN-SMOTE and GG-SMOTE realizations. Therefore, it appears that the use of the surrounding neighborhood to over-sample the minority class leads to balanced data sets with a better representation of the underlying class distribution, which contributes to better classification results according to the AUC metric.

Table 4 Average AUC values

	1-NN	C4.5	MLP
Imbalanced	0.804	0.798	0.809
SMOTE	0.830	0.829	0.845
B-SMOTE	0.821	0.816	0.839
SL-SMOTE	0.804	0.820	0.847
ROS	0.804	0.820	0.842
AHC	0.823	0.826	0.845
ADOMS	0.826	0.822	0.853
ADASYN	0.828	0.834	0.844
NCN-SMOTE	0.847	0.835	0.851
GG-SMOTE	0.846	0.835	0.855
RNG-SMOTE	0.839	0.833	0.846

The Friedman's average ranks for the three classification models have been plotted in Fig. 4, which can be taken as a further confirmation of the findings with the AUC values. For the 1-NN classifier, GG-SMOTE and NCN-SMOTE clearly arise as the over-sampling algorithms with the lowest rankings, that is, the highest performance in average; when using the C4.5 decision tree, GG-SMOTE and ADASYN are the techniques with the best rankings, followed by NCN-SMOTE and RNG-SMOTE; for the MLP neural network, the NCN-SMOTE, ADOMS and GG-SMOTE methods yield the lowest rankings. Despite the use of imbalanced data sets produces the highest (worst) average ranks with all classifiers, it is worth noting that the rankings of SL-SMOTE and ROS are not too far in the case of the 1-NN rule.

With the aim of checking whether our first conclusions can be supported by non-parametric statistical tests, the Iman-Davenport's statistic has been com-

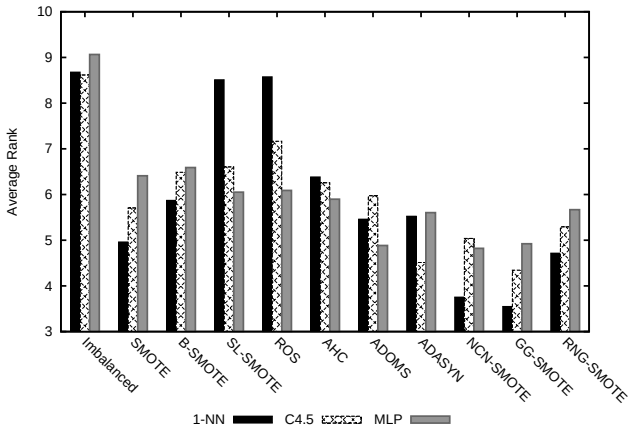

Fig. 4 Friedman's average ranks

puted using Equation 6 to discover whether or not the AUC results are significantly different. This computation yielded $F_F = 17.41$ for 1-NN, $F_F = 6.15$ for C4.5, and $F_F = 5.54$ for MLP. As the critical values for the F-distribution with $K - 1 = 11 - 1 = 10$ and $(K - 1)(N - 1) = (11 - 1)(39 - 1) = 380$ degrees of freedom at confidence levels of 90% and 95% are $F(10, 380)_{0.90} = 1.62$ and $F(10, 380)_{0.95} = 1.86$, the null-hypothesis that all strategies here explored perform equally well can be rejected. Consequently, we can now carry on with a Holm's post hoc test, using the best over-sampling method for each classifier as the respective control algorithm.

Table 5 reports the z values, the p-values and the adjusted α's calculated using the Holm's procedure, where the symbol "**" indicates that the null-hypothesis of equivalence with the control algorithm is rejected at a significance level of $\alpha = 0.05$. For each classifier, the algorithms have been ordered from the smallest to largest p-values.

Table 5 Results obtained with the Holm's test for $\alpha = 0.05$

i	Algorithm	z	p-value	α/i
1-NN (GG-SMOTE is the control method)				
10	Imbalanced**	6.827887	0	0.005000
9	ROS**	6.691330	0	0.005556
8	SL-SMOTE**	6.605981	0	0.006250
7	AHC**	3.772408	0.000162	0.007143
6	B-SMOTE**	3.089619	0.002004	0.008333
5	ADASYN**	2.628737	0.008570	0.010000
4	ADOMS**	2.543388	0.010978	0.012500
3	SMOTE	1.877669	0.060426	0.016667
2	RNG-SMOTE	1.553344	0.120341	0.025000
1	NCN-SMOTE	0.273115	0.784764	0.050000
C4.5 (GG-SMOTE is the control method)				
10	Imbalanced**	5.684216	0	0.005000
9	ROS**	3.755338	0.000173	0.005556
8	SL-SMOTE**	3.004270	0.002662	0.006250
7	B-SMOTE**	2.850643	0.004363	0.007143
6	AHC	2.543388	0.010978	0.008333
5	ADOMS	2.167854	0.030170	0.010000
4	SMOTE	1.809390	0.070390	0.012500
3	RNG-SMOTE	1.263159	0.206532	0.016667
2	NCN-SMOTE	0.921765	0.356651	0.025000
1	ADASYN	0.221906	0.824387	0.050000
MLP (NCN-SMOTE is the control method)				
10	Imbalanced**	5.650077	0	0.005000
9	B-SMOTE	2.355621	0.018492	0.005556
8	SMOTE	2.116645	0.034290	0.006250
7	ROS	1.689902	0.091047	0.007143
6	SL-SMOTE	1.638693	0.101277	0.008333
5	AHC	1.433856	0.151613	0.010000
4	RNG-SMOTE	1.126601	0.259911	0.012500
3	ADASYN	1.041253	0.297758	0.016667
2	GG-SMOTE	0.136558	0.891380	0.025000
1	ADOMS	0.085349	0.931984	0.050000

The results of the Holm's test given in Table 5 reveal the superiority of the surrounding strategies with the 1-NN classifier: the GG-SMOTE approach performs significantly better than all the other algorithms, except SMOTE, NCN-SMOTE and RNG-SMOTE. Focusing on the results of the C4.5 decision tree, one can observe that GG-SMOTE appears significantly better than ROS, SL-SMOTE and B-SMOTE, but it is statistically equivalent to AHC, ADOMS, SMOTE, RNG-SMOTE, NCN-SMOTE and ADASYN. With the MLP neural network, the control algorithm NCN-SMOTE significantly outperforms the non-preprocessed imbalanced data set, but behaves equally well as all the other over-sampling techniques.

As several algorithms exhibit similar behaviors, especially with the C4.5 and MLP classifiers, we have run a Wilcoxon's test between each pair of techniques for each classification model. The upper diagonal half of Tables 6–8 summarizes this statistic for a significance level of $\alpha = 0.10$ (10% or less chance), whereas the lower diagonal half corresponds to a significance level of $\alpha = 0.05$. The symbol "•" indicates that the method in the row significantly outperforms the method in the column, and the symbol "○" means that the method in the column performs significantly better than the method in the row.

Table 6 Summary of the Wilcoxon's statistic for the over-sampling methods with the 1-NN classifier. Upper and lower diagonal halves are for $\alpha = 0.10$ and $\alpha = 0.05$, respectively

		(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)
(1)	Imbalanced	–	○	○			○	○	○	○	○	○
(2)	SMOTE	•	–	•	•	•	•			○	○	○
(3)	B-SMOTE	•		–	•	•				○	○	○
(4)	SL-SMOTE		○	○	–		○	○	○	○	○	○
(5)	ROS		○	○		–	○	○	○	○	○	○
(6)	AHC	•	○		•	•	–		○	○	○	○
(7)	ADOMS	•			•	•		–		○	○	○
(8)	ADASYN	•			•	•			–	○	○	○
(9)	NCN-SMOTE	•	•	•	•	•	•	•	•	–		•
(10)	GG-SMOTE	•	•	•	•	•	•	•	•		–	•
(11)	RNG-SMOTE	•		•	•	•	•		•	○	○	–

With the 1-NN classifier, the original SMOTE algorithm performs significantly better than SL-SMOTE, ROS and AHC at both significance levels, whereas there are not significant differences between SMOTE and B-SMOTE at a significance level of $\alpha = 0.05$. The most remarkable observation from Table 6 is that NCN-SMOTE and GG-SMOTE are significantly better than the remaining methods at both significance levels, what demonstrates the suitability of these over-sampling algorithms to consistently produce well-balanced training sets for further classification with the 1-NN model.

When using the C4.5 decision tree, Table 7 shows that there are less statistically significant differences than in the previous case of the 1-NN rule. Nonetheless, the GG-SMOTE algorithm performs significantly better than B-SMOTE, SL-SMOTE, ROS, AHC and

Table 7 Summary of the Wilcoxon's statistic for the over-sampling methods with the C4.5 classifier. Upper and lower diagonal halves are for $\alpha = 0.10$ and $\alpha = 0.05$, respectively

		(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)
(1)	Imbalanced	–	○	○	○	○	○	○	○	○	○	○
(2)	SMOTE	●	–	●		●						
(3)	B-SMOTE	●	○	–					○	○	○	○
(4)	SL-SMOTE	●			–				○	○	○	○
(5)	ROS	●	○			–			○	○	○	○
(6)	AHC	●					–		○	○	○	
(7)	ADOMS	●						–	○		○	
(8)	ADASYN	●		●	●	●	●	●	–			
(9)	NCN-SMOTE	●		●	●	●	●			–		
(10)	GG-SMOTE	●		●	●	●	●	●			–	
(11)	RNG-SMOTE	●				●						–

ADOMS at both significance levels. On the other hand, NCN-SMOTE is also significantly better than those methods (except to ADOMS) at both significance levels. The original SMOTE algorithm and the ADASYN technique perform equally well as the methods based on the surrounding neighborhood here introduced.

In the case of the MLP neural network, ADOMS and NCN-SMOTE appear to be the algorithms with most differences, being significantly better than SMOTE, B-SMOTE, ROS and AHC for $\alpha = 0.05$; besides, at a significance level of $\alpha = 0.10$, ADOMS also performs significantly better than SL-SMOTE and ADASYN, whereas NCN-SMOTE is also significantly better than the RNG-SMOTE approach. Finally, for $\alpha = 0.10$, the GG-SMOTE method is significantly superior to SMOTE, B-SMOTE, SL-SMOTE, RNG-SMOTE, ROS, AHC and ADASYN.

Table 8 Summary of the Wilcoxon's statistic for the over-sampling methods with the MLP classifier. Upper and lower diagonal halves are for $\alpha = 0.10$ and $\alpha = 0.05$, respectively

		(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)
(1)	Imbalanced	–	○	○	○	○	○	○	○	○	○	○
(2)	SMOTE	●	–					○		○	○	
(3)	B-SMOTE	●		–				○	○	○	○	
(4)	SL-SMOTE	●			–			○			○	
(5)	ROS	●				–		○		○	○	
(6)	AHC	●					–	○		○	○	
(7)	ADOMS	●	●	●		●	●	–	●			
(8)	ADASYN	●							–		○	
(9)	NCN-SMOTE	●	●	●		●	●			–		●
(10)	GG-SMOTE	●	●	●		●					–	●
(11)	RNG-SMOTE	●								○		–

As a summary of the Wilcoxon's tests for an easier analysis, the three values in the cells of Table 9 show how many times each method has been significantly-better/same/significantly-worse than the rest of over-sampling strategies at significance levels of $\alpha = 0.10$ and $\alpha = 0.05$ for each classifier. The results here reported corroborate the discussion of previous tables, proving the practical relevance of over-sampling the minority class irrespective of the classification model (using the imbalanced set is significantly worse than employing a training set that has been preprocessed by some over-sampling algorithm). This summary also allows to clearly state the superiority of the NCN-SMOTE and GG-SMOTE algorithms over the remaining methods, especially with the 1-NN classifier.

Table 9 Summary of how many times the over-sampling techniques have been significantly-better/same/significantly-worse

	1-NN		C4.5		MLP	
	$\alpha_{0.10}$	$\alpha_{0.05}$	$\alpha_{0.10}$	$\alpha_{0.05}$	$\alpha_{0.10}$	$\alpha_{0.05}$
Imbalanced	0/2/8	0/2/8	0/0/10	0/0/10	0/0/10	0/0/10
SMOTE	5/2/3	4/4/2	3/7/0	3/7/0	1/6/3	1/6/3
B-SMOTE	3/3/4	3/4/3	1/4/5	1/5/4	1/5/4	1/6/3
SL-SMOTE	0/2/8	0/2/8	1/5/4	1/6/3	1/7/2	1/9/0
ROS	0/2/8	0/2/8	1/4/5	1/4/5	1/6/3	1/6/3
AHC	3/2/5	3/3/4	1/6/3	1/6/3	1/6/3	1/7/2
ADOMS	3/4/3	3/5/2	1/7/2	1/7/2	7/3/0	5/5/0
ADASYN	4/3/3	3/4/3	6/4/0	6/4/0	2/6/2	1/9/0
NCN-SMOTE	9/1/0	9/1/0	5/5/0	5/5/0	6/4/0	5/4/1
GG-SMOTE	9/1/0	9/1/0	6/4/0	6/4/0	8/2/0	4/6/0
RNG-SMOTE	8/2/0	6/2/2	4/6/0	2/8/0	1/7/2	1/8/1

7 Final conclusions and future work

This paper has focused on the problem of expanding the minority class so as to balance the class distribution of the training set. Three modifications of the original SMOTE algorithm have been proposed, all them based upon the concept of surrounding neighborhood. In particular, we have used the NCN, the GG and the RNG in the step of selecting neighbors for further generation of artificial positive examples. The aim of these alternatives is to take both proximity and spatial distribution of neighbors into account in order for extending the regions of the minority class.

Experimental results over 39 databases using three different classifiers (1-NN, C4.5 and MLP) have demonstrated that the Surrounding SMOTE methods achieve significant improvements in terms of the AUC measure with respect to the original SMOTE algorithm and other existing over-sampling procedures. From the three surrounding alternatives, the NCN-SMOTE and GG-SMOTE appear to be the strategies with the highest performance according to the average AUC value and the average Friedman's rank. A further analysis with the Wilcoxon's statistic has allowed to observe that both NCN-SMOTE and GG-SMOTE have performed significantly better than most of the remaining methods, especially when using the 1-NN classification rule. Despite differences are less significant in the case of the C4.5 decision tree and the MLP neural network,

these two surrounding approaches to SMOTE are still the best over-sampling algorithms.

Finally, future research will be mainly addressed to incorporate a filtering phase into the general structure of the Surrounding SMOTE algorithms in order to remove any example (either positive or negative) that could be considered noisy or atypical. Another avenue for further investigation concentrates on the study of alternative methods to be exploited in the phase of generation of synthetic positive examples. Also, we are interested in analyzing the behavior of both the surrounding-based approaches and other SMOTE-like algorithms as a function of the imbalance ratio of the data sets.

Acknowledgements This work has partially been supported by Spanish Ministry of Education and Science (CSD2007–00018 and TIN2009–14205), Fundació Bancaixa (P1–1B2009–04), Generalitat Valenciana (PROMETEO/2010/028), and Universitat Jaume I (PREDOC/2008/04).

Appendix

This appendix provides three tables with the detailed results for the experimental analysis carried out in the present work. Table 10 contains the AUC values for all the databases and algorithms achieved when using the 1-NN classification rule, Table 11 shows the results with the C4.5 decision tree, and Table 12 is for the MLP neural network. The best results are highlighted in bold face.

References

1. J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera. Software tool: Data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic and Soft Computing*, 17(2–3):255–287, 2011.
2. R. Barandela, J. S. Sánchez, V. García, and E. Rangel. Strategies for learning in class imbalance problems. *Pattern Recognition*, 36(3):849–851, 2003.
3. G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard. A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explorations Newsletter*, 6(1):20–29, 2004.
4. I. Brown and C. Mues. An experimental comparison of classification algorithms for imbalanced credit scoring data sets. *Expert Systems with Applications*, 39(3):3446–3453, 2012.
5. C. Bunkhumpornpat, K. Sinapiromsaran, and C. Lursinsap. Safe-Level-SMOTE: Safe-Level-Synthetic Minority Over-Sampling TEchnique for handling the class imbalanced problem. In *Proc. of the 13th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 475–482, Bangkok, Thailand, 2009.
6. B. B. Chaudhuri. A new definition of neighborhood of a point in multi-dimensional space. *Pattern Recognition Letters*, 17(1):11–17, 1996.
7. N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.
8. N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer. SMOTEBoost: Improving prediction of the minority class in boosting. In *Proc. of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 107–119, Dubrovnik, Croatia, 2003.
9. E. Chen, Y. Lin, H. Xiong, Q. Luo, and H. Ma. Exploiting probabilistic topic models to improve text categorization under class imbalance. *Information Processing & Management*, 47(2):202–214, 2011.
10. G. Cohen, M. Hilario, H. Sax, S. Hugonnet, and A. Geissbuhler. Learning from imbalanced data in surveillance of nosocomial infection. *Artificial Intelligence in Medicine*, 37(1):7–18, 2006.
11. S. Daskalaki, I. Kopanas, and N. Avouris. Evaluation of classifiers for an uneven class distribution problem. *Applied Artificial Intelligence*, 20(5):381–417, 2006.
12. J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7(1):1–30, 2006.
13. J. Derrac, S. García, D. Molina, and F. Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18, 2011.
14. T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.
15. T. Fawcett and F. Provost. Adaptive fraud detection. *Data Mining and Knowledge Discovery*, 1(3):291–316, 1997.
16. V. Ganganwar. An overview of classification algorithms for imbalanced datasets. *International Journal of Emerging Technology and Advanced Engineering*, 2(4):42–47, 2012.
17. S. García, A. Fernández, J. Luengo, and F. Herrera. Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, 180(10):2044–2064, 2010.
18. V. García, J. S. Sánchez, and R. A. Mollineda. On the effectiveness of preprocessing methods when dealing with different levels of class imbalance. *Knowledge-Based Systems*, 25(1):13–21, 2012.
19. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.
20. H. Han, W.-Y. Wang, and B.-H. Mao. Borderline-SMOTE: A new over-sampling method in imbalanced data sets learning. In *Proc. of the International Conference on Intelligent Computing*, pages 878–887, Hefei, China, 2005.
21. G. He, H. Han, and W. Wang. An over-sampling expert system for learning from imbalanced data sets. In *Proc. of the 2nd International Conference on Neural Networks and Brain*, pages 537–541, Beijing, China, 2005.
22. H. He, Y. Bai, E. A. Garcia, and S. Li. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In *Proc. of the International Joint Conference on Neural Networks*, pages 1322–1328, Hong Kong, 2008.

Table 10 AUC results for the 1-NN classifier

Data Set	Imbalanced	SMOTE	B-SMOTE	SL-SMOTE	ROS	AHC	ADOMS	ADASYN	NCN-SMOTE	GG-SMOTE	RNG-SMOTE
Glass2	0.779	0.768	0.771	0.779	0.779	0.785	0.7879	0.774	0.777	**0.791**	0.774
Wisconsin2	0.953	0.961	0.964	0.957	0.953	0.956	0.959	0.967	0.963	**0.966**	**0.968**
Pima2	0.671	0.673	0.676	0.671	0.671	0.669	0.674	0.673	0.687	**0.688**	0.675
Iris1	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
Glass1	0.913	0.923	**0.949**	0.913	0.913	0.929	0.942	0.936	0.929	0.942	0.933
Yeast2	0.648	0.652	0.649	0.648	0.648	0.651	**0.668**	0.650	0.657	0.663	0.663
Haberman2	0.535	0.536	0.534	0.526	0.526	0.538	0.554	0.523	**0.572**	0.550	0.563
Vehicle3	0.941	0.950	0.955	0.941	0.941	0.952	0.948	0.955	**0.956**	0.950	**0.964**
Vehicle2	0.623	0.638	0.646	0.627	0.627	0.642	0.660	0.634	0.667	**0.678**	0.658
Vehicle4	0.673	0.688	0.674	0.678	0.678	0.687	0.684	0.680	0.693	**0.695**	0.677
Glass567	0.838	**0.845**	0.842	0.838	0.838	0.838	0.835	0.828	0.838	0.842	0.831
Vehicle1	0.911	0.923	0.926	0.911	0.911	0.928	0.921	0.926	**0.940**	0.939	0.937
Ecoli2	0.797	0.836	0.843	0.797	0.797	0.823	0.843	0.833	**0.845**	**0.861**	0.842
NewThyroid2	0.977	**0.992**	**0.992**	0.977	0.977	0.977	0.989	**0.992**	0.986	0.989	0.989
Ecoli3	0.906	0.907	**0.914**	0.906	0.906	0.899	0.915	0.898	0.893	0.896	0.902
Segment1	**0.995**	**0.995**	**0.995**	**0.995**	**0.995**	**0.995**	**0.995**	**0.995**	**0.995**	0.994	**0.995**
NewThyroid3	0.980	**0.994**	**0.994**	0.980	0.980	0.980	**0.994**	0.992	0.989	0.980	0.992
Glass7	0.871	0.888	0.888	0.871	0.871	0.871	0.888	0.899	**0.899**	0.877	0.869
Yeast4	0.814	0.857	0.854	0.814	0.814	0.842	0.853	0.858	0.866	**0.867**	0.866
Ecoli4	0.745	0.821	0.768	0.745	0.745	0.783	**0.841**	0.808	0.821	0.834	0.792
PageBlocks2345	0.875	0.897	0.897	0.876	0.876	0.886	0.911	0.901	**0.922**	0.913	0.913
Yeast5vs1	0.851	0.883	0.862	0.851	0.851	0.868	0.882	0.879	**0.905**	0.875	0.896
Yeast5	0.680	0.753	0.734	0.680	0.680	0.710	0.706	0.720	**0.790**	0.770	0.752
Vowel1	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	0.999	**1.000**	**1.000**
Glass3	0.580	**0.682**	0.601	0.580	0.580	0.626	0.545	**0.682**	0.667	**0.682**	0.643
Glass3vs12567	0.601	0.728	0.619	0.601	0.601	0.685	0.613	**0.731**	0.721	0.690	0.669
Shuttle1	**0.996**	**0.996**	**0.996**	**0.996**	**0.996**	**0.996**	**0.996**	**0.996**	**0.996**	**0.996**	**0.996**
Yeast8vs2	0.642	0.699	0.681	0.642	0.642	0.689	0.706	0.680	**0.752**	0.698	0.702
Glass5	0.821	0.866	0.868	0.821	0.821	0.863	0.809	0.866	0.923	0.925	**0.928**
Ecoli5	0.870	0.894	0.870	0.870	0.870	0.915	0.911	**0.917**	0.892	**0.917**	0.894
Page Blocks3	0.980	**1.000**	0.980	0.980	0.980	**1.000**	0.980	**1.000**	**1.000**	**1.000**	**1.000**
Glass6	0.836	0.883	0.883	0.836	0.836	0.877	0.836	0.883	**0.933**	0.883	0.883
Yeast8vs2459	0.573	0.615	0.597	0.573	0.573	0.598	0.625	0.585	0.653	**0.673**	0.665
Glass6vs12357	**0.893**	0.888	0.888	**0.893**	**0.893**	0.890	0.883	0.888	0.888	0.888	0.885
Yeast9	0.768	0.764	0.761	0.768	0.768	0.794	0.727	0.763	**0.807**	0.795	0.804
Yeast5	0.667	0.706	0.693	0.667	0.667	0.686	0.720	0.714	0.763	**0.773**	0.743
Yeast8vs12910	0.554	0.585	0.593	0.554	0.554	0.581	0.651	0.585	0.660	**0.698**	0.691
Yeast6	0.846	0.924	0.902	0.846	0.846	0.922	0.931	0.913	0.941	**0.952**	0.922
Yeast7	0.749	0.778	0.760	0.749	0.749	0.771	0.834	0.776	**0.847**	0.846	0.845
Average	0.804	0.830	0.821	0.804	0.804	0.823	0.826	0.828	**0.847**	0.846	0.839

23. H. He and E. A. Garcia. Learning from imbalanced data. *IEEE Trans. on Knowledge and Data Engineering*, 21:1263–1284, 2009.
24. C. S. Hilas and P. A. Mastorocostas. An application of supervised and unsupervised learning approaches to telecommunications fraud detection. *Knowledge-Based Systems*, 21(7):721–726, 2008.
25. S. Hu, Y. Liang, L. Ma, and Y. He. MSMOTE: Improving classification performance when training data is imbalanced. In *Proc. of the 2nd International Workshop on Computer Science and Engineering*, pages 13–17, Qingdao, China, 2009.
26. J. Huang and C.-X. Ling. Using AUC and accuracy in evaluating learning algorithms. *IEEE Trans. on Knowledge and Data Engineering*, 17(3):299–310, 2005.
27. Y.-M. Huang, C.-M. Hung, and H. C. Jiau. Evaluation of neural networks and data mining methods on a credit assessment task for class imbalance problem. *Nonlinear Analysis: Real World Applications*, 7(4):720–757, 2006.
28. N. Japkowicz and S. Stephen. The class imbalance problem: A systematic study. *Intelligent Data Analysis*, 6(5):429–449, 2002.
29. J.W. Jaromczyk and G.T. Toussaint. Relative neighborhood graphs and their relatives. *Proceedings of the IEEE*, 80(9):1502–1517, 1992.
30. Y. Jiang, M. Li, and Z.-H. Zhou. Software defect detection with ROCUS. *Journal of Computer Science and Technology*, 26(2):328–342, 2011.
31. W. Klement, S. Wilk, W. Michalowski, and S. Matwin. Classifying severely imbalanced data. In *Proc. of the 24th Canadian Conference on Advances in Artificial Intelligence*, St. John's, Canada, 2011.
32. M. Kubat, R. C. Holte, and S. Matwin. Machine learning for the detection of oil spills in satellite radar images. *Machine Learning*, 30(2–3):195–215, 1998.
33. M. Kubat and S. Matwin. Addressing the curse of imbalanced training sets: One-sided selection. In *Proc. of the 14th International Conference on Machine Learning*, pages 179–186, Nashville, TN, 1997.
34. C. Lemnaru and R. Potolea. Imbalanced classification problems: Systematic study, issues and best practices. In *Enterprise Information Systems*, pages 35–50. Springer, 2012.
35. D.-C. Li, C.-W. Liu, and S. C. Hu. A learning method for the class imbalance problem with medical data sets. *Computers in Biology and Medicine*, 40(5):509–518, 2010.
36. C. X. Ling and C. Li. Data mining for direct marketing: Problems and solutions. In *Proc. of the 4th International Conference on Knowledge Discovery and Data Mining*, pages 73–79, New York, NY, 1998.
37. T. Maciejewski and J. Stefanowski. Local neighbourhood extension of SMOTE for mining imbalanced data. In *Proc. of the IEEE Symposium on Computational Intelligence and Data Mining*, pages 104–111, Paris, France, 2011.

Table 11 AUC results for the C4.5 classifier

Data Set	Imbalanced	SMOTE	B-SMOTE	SL-SMOTE	ROS	AHC	ADOMS	ADASYN	NCN-SMOTE	GG-SMOTE	RNG-SMOTE
Glass2	0.733	0.701	0.743	0.726	0.725	0.745	0.666	**0.759**	0.703	0.717	0.716
Wisconsin2	0.948	0.948	0.950	0.952	0.943	0.950	0.953	**0.961**	0.944	0.952	0.944
Pima2	0.703	0.703	0.725	0.722	0.691	0.722	0.715	0.717	0.726	**0.729**	0.716
Iris1	**0.990**	**0.990**	**0.990**	**0.990**	**0.990**	**0.990**	**0.990**	**0.990**	**0.990**	**0.990**	**0.990**
Glass1	**0.916**	0.863	0.869	0.876	0.866	0.865	0.902	0.866	0.865	0.879	0.859
Yeast2	0.668	**0.712**	0.697	0.686	0.678	0.712	0.688	0.690	0.691	0.708	0.702
Haberman2	0.576	0.608	0.593	0.619	0.622	0.581	0.617	**0.671**	0.670	0.653	0.643
Vehicle3	0.948	0.951	0.954	0.956	0.942	0.952	0.954	0.962	0.946	**0.963**	0.957
Vehicle2	0.660	0.689	0.714	0.671	0.685	0.687	**0.729**	0.713	0.707	0.699	0.722
Vehicle4	0.666	0.729	0.671	0.705	0.711	0.710	0.694	0.709	**0.739**	0.728	0.708
Glass567	0.817	**0.824**	0.806	0.799	0.767	0.789	0.782	0.799	0.821	0.786	0.792
Vehicle1	0.927	0.924	0.933	0.927	0.926	0.932	**0.935**	0.926	0.927	0.928	0.927
Ecoli2	0.861	0.890	**0.907**	0.880	0.881	0.900	0.883	0.885	0.906	0.879	0.879
NewThyroid2	0.949	0.975	0.966	0.963	0.963	0.949	0.975	**0.980**	0.963	0.954	0.952
Ecoli3	0.862	0.868	**0.900**	0.848	0.875	0.878	0.896	0.874	0.883	0.858	0.859
Segment1	0.984	0.992	0.982	0.992	0.992	**0.994**	0.989	0.984	0.992	0.992	0.993
NewThyroid3	0.949	0.946	0.952	**0.980**	0.952	0.937	0.977	0.952	0.946	0.966	0.952
Glass7	0.813	0.884	0.853	0.876	0.876	0.867	0.863	0.892	0.894	0.874	**0.898**
Yeast4	0.857	0.906	0.891	0.881	0.868	0.899	0.911	**0.938**	0.914	0.904	0.918
Ecoli4	0.728	0.837	0.815	0.725	0.783	0.801	0.828	**0.839**	0.800	0.830	0.832
PageBlocks2345	0.922	0.945	0.937	0.928	0.931	0.938	0.939	0.945	0.941	**0.950**	0.946
Yeast5vs1	0.833	0.873	0.836	0.848	0.840	0.858	0.840	0.874	**0.878**	0.877	0.869
Yeast5	0.680	**0.800**	0.777	0.680	0.748	0.759	0.752	0.781	0.741	0.772	0.751
Vowel1	0.971	0.951	0.941	0.958	0.962	0.956	**0.972**	0.967	0.968	0.968	0.938
Glass3	0.618	0.646	0.560	0.582	0.604	0.643	0.596	0.629	0.623	0.638	**0.702**
Glass3vs12567	0.669	0.667	0.653	0.628	0.655	0.711	0.512	0.728	0.723	0.582	**0.735**
Shuttle1	1.000	1.000	1.000	1.000	1.000	1.000	**1.000**	1.000	**1.000**	**1.000**	**1.000**
Yeast8vs2	0.594	0.620	0.648	0.615	0.650	0.645	0.608	**0.680**	0.644	0.670	0.626
Glass5	0.793	0.849	0.785	**0.883**	0.849	0.842	0.848	0.880	0.844	0.880	0.880
Ecoli5	0.814	0.892	0.816	0.839	0.839	**0.909**	0.904	0.858	0.864	0.866	0.839
Page Blocks3	**0.998**	**0.998**	**0.998**	**0.998**	**0.998**	**0.998**	0.997	**0.998**	**0.998**	**0.998**	**0.998**
Glass6	0.891	**0.894**	**0.894**	**0.894**	**0.894**	0.891	**0.894**	**0.894**	0.891	**0.894**	**0.894**
Yeast8vs2459	0.500	0.523	0.506	0.535	0.509	0.520	0.506	**0.613**	0.531	0.589	0.516
Glass6vs12357	0.898	0.945	**0.948**	0.943	0.943	0.890	**0.948**	0.945	0.945	0.945	0.945
Yeast9	0.500	0.747	0.664	**0.865**	0.840	0.783	0.809	0.588	0.771	0.795	0.779
Yeast5	0.595	**0.773**	0.684	0.690	0.698	0.672	0.682	0.678	0.737	0.735	0.747
Yeast8vs12910	0.616	0.565	0.614	**0.683**	0.629	0.605	0.614	0.608	0.653	0.680	0.599
Yeast6	0.883	0.913	0.902	0.857	0.879	0.924	0.923	**0.946**	0.945	0.934	**0.946**
Yeast7	0.781	0.806	0.766	0.802	0.790	0.801	0.760	0.804	**0.833**	0.818	0.815
Average	0.798	0.829	0.816	0.820	0.820	0.826	0.822	0.834	**0.835**	**0.835**	0.833

38. S.-H. Oh. Error back-propagation algorithm for classification of imbalanced data. *Neurocomputing*, 74(6):1058–1061, 2011.
39. A. Orriols-Puig and E. Bernadó-Mansilla. Evolutionary rule-based systems for imbalanced data sets. *Soft Computing*, 13(3):213–225, 2008.
40. J. S. Sánchez and A. I. Marqués. Enhanced neighbourhood specifications for pattern classification. In *Pattern Recognition and String Matching*, pages 673–702. Kluwer Academic Publishers, 2002.
41. R. D. Short and K. Fukunaga. A new nearest neighbour distance measure. In *Proc. of the 5th International Conference on Pattern Recognition*, pages 81–86, Miami, FL, 1980.
42. M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, 2009.
43. Y. Sun, A. K. C. Wong, and M. S. Kamel. Classification of imbalanced data: A review. *International Journal of Pattern Recognition and Artificial Intelligence*, 23(4):687–719, 2009.
44. S. Tan. Neighbor-weighted K-nearest neighbor for unbalanced text corpus. *Expert Systems with Applications*, 28(4):667–671, 2005.
45. S. Tang and S.-P. Chen. The generation mechanism of synthetic minority class examples. In *Proc. of the 5th International Conference on Information Technology and Application in Biomedicine*, pages 444–447, Shenzhen, China, 2008.
46. J. van Hulse, T. M. Khoshgoftaar, and A. Napolitano. Experimental perspectives on learning from imbalanced data. In *Proc. of the 24th International Conference on Machine Learning*, pages 935–942, Corvallis, OR, 2007.
47. J. van Hulse, T. M. Khoshgoftaar, and A. Napolitano. An exploration of learning when data is noisy and imbalanced. *Intelligent Data Analysis*, 15(2):215–236, 2011.
48. J. Wang, M. Xu, H. Wang, and J. Zhang. Classification of imbalanced data by using the SMOTE algorithm and locally linear embedding. In *Proc. of the 8th International Conference on Signal Processing*, pages 16–20, Beijing, China, 2006.
49. S.-J. Yen, Y.-S. Lee, C.-H. Lin, and J.-C. Ying. Investigating the effect of sampling methods for imbalanced data distributions. In *Proc. of the IEEE International Conference on Systems, Man, and Cybernetics*, pages 4163–4168, Taipei, Taiwan, 2006.
50. D. Zhang, W. Liu, X. Gong, and H. Jin. A novel improved SMOTE resampling algorithm based on fractal. *Journal of Computational Information Systems*, 7(6):2204–2211, 2011.
51. J. Zhang, Y.-S. Yim, and J. Yang. Intelligent selection of instances for prediction functions in lazy learning algorithms. *Artificial Intelligence Review*, 11(1):175–191, 1997.

Table 12 AUC results for the MLP classifier

Data Set	Imbalanced	SMOTE	B-SMOTE	SL-SMOTE	ROS	AHC	ADOMS	ADASYN	NCN-SMOTE	GG-SMOTE	RNG-SMOTE
Glass2	0.647	0.725	0.732	0.739	0.715	0.740	0.711	**0.751**	0.698	0.706	0.727
Wisconsin2	0.960	0.963	0.964	**0.967**	0.963	0.964	0.961	0.964	0.956	0.961	0.957
Pima2	**0.739**	0.709	0.729	0.721	0.713	0.724	0.721	0.731	0.719	0.719	0.702
Iris1	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
Glass1	0.903	0.913	0.917	0.904	0.904	0.915	**0.924**	0.919	0.906	0.900	0.909
Yeast2	0.690	0.708	0.716	0.711	0.716	0.718	**0.716**	0.714	0.719	**0.725**	0.717
Haberman2	0.580	0.620	0.615	0.604	0.597	0.588	0.616	0.628	**0.637**	0.627	0.631
Vehicle3	0.971	0.978	0.974	0.978	0.972	0.980	0.984	0.978	**0.988**	0.985	0.986
Vehicle2	0.761	0.794	0.796	0.781	0.797	0.810	**0.812**	0.811	0.798	0.802	0.788
Vehicle4	0.795	0.778	0.779	**0.817**	0.789	0.790	0.788	0.787	0.780	0.790	0.795
Glass567	0.781	0.774	0.778	0.778	0.767	0.810	0.764	0.800	0.753	0.778	**0.828**
Vehicle1	0.950	0.960	0.971	0.968	0.973	0.966	0.972	**0.975**	0.971	0.965	0.973
Ecoli2	0.867	0.887	0.880	0.861	0.867	0.867	0.870	0.862	**0.892**	0.883	0.871
NewThyroid2	0.954	0.966	0.966	0.966	0.983	0.969	**0.980**	0.966	**0.980**	**0.980**	**0.980**
Ecoli3	0.894	0.900	0.904	0.888	0.895	0.892	0.917	0.857	**0.920**	0.912	0.909
Segment1	0.993	0.998	0.998	**0.999**	**0.999**	0.997	0.994	0.995	0.997	0.996	0.996
NewThyroid3	0.954	0.952	0.952	0.952	0.952	0.952	0.975	0.952	**0.977**	**0.977**	**0.983**
Glass7	0.849	0.877	0.849	0.877	0.880	0.880	0.869	0.880	**0.888**	0.883	0.877
Yeast4	0.856	0.899	0.887	0.909	0.906	0.905	0.916	0.906	0.913	0.891	0.917
Ecoli4	0.807	0.878	0.859	0.881	0.865	0.865	0.887	0.878	0.878	**0.890**	0.854
PageBlocks2345	0.847	0.931	0.926	0.939	0.942	0.941	0.938	0.944	**0.944**	0.935	0.938
Yeast5vs1	0.855	0.884	0.869	0.872	**0.924**	0.912	0.907	0.875	0.915	0.893	0.885
Yeast5	0.717	0.746	0.749	0.744	0.759	0.748	0.788	0.762	0.758	**0.795**	0.736
Vowel1	0.994	0.989	**1.000**	0.994	**1.000**	0.999	**1.000**	**1.000**	**1.000**	0.999	0.994
Glass3	0.557	0.674	0.660	0.619	**0.696**	0.641	0.666	0.594	0.665	0.696	0.648
Glass3vs12567	0.554	**0.790**	0.667	0.765	0.754	0.759	0.772	0.721	0.792	0.775	0.759
Shuttle1	**0.996**	**0.996**	**0.996**	**0.996**	**0.996**	0.996	**0.996**	**0.996**	**0.996**	**0.996**	**0.996**
Yeast8vs2	0.626	0.679	0.637	0.699	0.662	0.684	0.712	0.722	0.678	**0.741**	0.638
Glass5	0.871	0.897	**0.930**	0.894	0.894	0.889	0.923	0.894	0.892	0.928	0.897
Ecoli5	0.891	0.912	0.889	0.914	0.914	0.909	0.900	0.914	0.861	0.911	**0.915**
Page Blocks3	0.979	0.979	0.978	0.997	0.978	0.998	**0.999**	0.977	0.998	0.976	0.957
Glass6	0.791	0.836	0.836	0.836	0.836	0.839	0.836	0.786	**0.883**	0.836	0.836
Yeast8vs2459	0.515	0.585	0.579	0.585	0.563	0.534	0.573	0.573	0.547	0.567	**0.598**
Glass6vs12357	**0.895**	0.893	0.893	0.895	0.895	0.893	0.885	0.893	0.890	0.893	0.893
Yeast9	0.773	0.724	0.787	**0.833**	0.714	0.745	0.796	0.738	0.768	0.785	0.712
Yeast5	0.663	0.780	0.786	0.752	0.758	0.767	0.777	**0.800**	0.790	0.769	0.763
Yeast8vs12910	0.531	0.622	0.611	**0.695**	0.595	0.580	0.625	0.618	0.669	0.691	0.646
Yeast6	0.844	0.941	0.862	0.918	0.919	0.930	0.927	0.930	0.945	0.933	**0.947**
Yeast7	0.681	0.816	0.789	0.795	0.806	0.871	**0.883**	0.836	0.848	0.868	0.822
Average	0.809	0.845	0.839	0.847	0.842	0.845	0.853	0.844	0.851	**0.855**	0.846