Synthetic Oversampling of Instances Using Clustering

Article *in* International Journal of Artificial Intelligence Tools · January 2013

DOI: 10.1142/S0218213013500085

3 authors, including:

Eduardo F Morales
Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE)
219 PUBLICATIONS **1,338** CITATIONS
SEE PROFILE

Jesus A. Gonzalez
Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE)
95 PUBLICATIONS **722** CITATIONS
SEE PROFILE

International Journal on Artificial Intelligence Tools
Vol. 22, No. 2 (2013) 1350008 (21 pages)

DOI: 10.1142/S0218213013500085

World Scientific
www.worldscientific.com

SYNTHETIC OVERSAMPLING OF INSTANCES USING CLUSTERING

ATLÁNTIDA I. SÁNCHEZ, EDUARDO F. MORALES and JESUS A. GONZALEZ
Instituto Nacional de Astrofísica, Óptica y Electrónica,
Computer Science Department, Luis Enrique Erro 1, 72840 Tonantzintla, México
{*atlantida,emorales,jagonzalez*}*@inaoep.mx*

Received 15 August 2011
Accepted 3 January 2013
Published

Imbalanced data sets in the class distribution is common to many real world applications. As many classifiers tend to degrade their performance over the minority class, several approaches have been proposed to deal with this problem. In this paper, we propose two new cluster-based oversampling methods, SOI-C and SOI-CJ. The proposed methods create clusters from the minority class instances and generate synthetic instances inside those clusters. In contrast with other oversampling methods, the proposed approaches avoid creating new instances in majority class regions. They are more robust to noisy examples (the number of new instances generated per cluster is proportional to the cluster's size). The clusters are automatically generated. Our new methods do not need tuning parameters, and they can deal both with numerical and nominal attributes. The two methods were tested with twenty artificial datasets and twenty three datasets from the UCI Machine Learning repository. For our experiments, we used six classifiers and results were evaluated with recall, precision, F-measure, and AUC measures, which are more suitable for class imbalanced datasets. We performed ANOVA and paired t-tests to show that the proposed methods are competitive and in many cases significantly better than the rest of the oversampling methods used during the comparison.

Keywords: Imbalanced datasets; oversampling; cluster-based oversampling; jittering.

1. Introduction

In many real-world classification problems, there is a strong imbalance in the class distribution. This means that most of the instances belong to some classes and only a few of them belong to other classes. In such cases, users are usually more interested in the accuracy results from the minority classes. However, some classifiers perform poorly in them. Trying to improve the classification performance of the minority class is important. This is relevant to many real-world applications such as fraud detection,[10] medical diagnosis of rare diseases,[3] unreliable communications for clients,[9] images classification,[18] detection of intestinal contractions from endoscopy video,[22] author's identification,[19] and many more.

This problem has become relevant and different ideas to deal with it have been developed, as we can see in Refs. 1, 2, 7 and Ref. 14. These approaches were proposed to improve the classification performance on the minority class, and they can be broadly divided into two types: techniques that sample the data and techniques that work at the algorithmic level. In the first case, the idea consists of sampling the data to obtain a modified, more balanced, class distribution in order to favor the minority class. This is the subject of this paper. The second type of technique usually involves extensions to existing algorithms or the development of new ones to bias them to decrease the number of classification errors on the minority class. Some of them include cost-sensitive classifiers,[20] DataBoost,[11] and MetaCost.[8]

Sampling techniques apply oversampling algorithms on the minority class and/or subsampling techniques on the majority class. In both cases, it is possible to apply random or "intelligent" sampling techniques. This paper focuses on intelligent oversampling techniques. The simplest oversampling method is known as random oversampling (ROS), which randomly duplicates instances of the minority class. However, this method might strengthen decision regions of the learner (making them smaller), and overfit the minority class.

In this paper we compare our methods based on Synthetic Oversampling of Instances (SOI) against ROS, SMOTE, and Borderline-Smote. We also show that our two proposed algorithms are significantly better than the others under different testing conditions.

The rest of this paper is organized as follows: Section 2 describes representative research related to working with imbalanced data, including the algorithms that we use in the comparison with the proposed algorithms. Section 3 describes the proposed algorithms. Section 4 provides a description of the experiments performed to validate the proposed algorithms over different datasets, with different classification algorithms, and several performance measures used for imbalanced datasets. In Section 5 the main results of our experiments are presented. In Section 6 we present a discussion of the findings of our results. Finally, Section 7 describes the main conclusions and suggests future research directions.

2. Related Work

Working with a small number of instances of one class does not allow a learning algorithm to generalize for such a class, consequently causing overfitting of the model. A widely used solution to avoid this problem is based on sampling techniques. As it was mentioned in the introduction section, in this research we focus on oversampling techniques.

A popular method that avoids replicating instances (among the intelligent oversampling methods) is called SMOTE (Synthetic Minority Oversampling Technique).[6] The main idea of SMOTE is to create new minority (positive) class instances by interpolating pairs of close minority class instances (see Figure 1 left). The pairs of close instances used to create each synthetic example are taken by

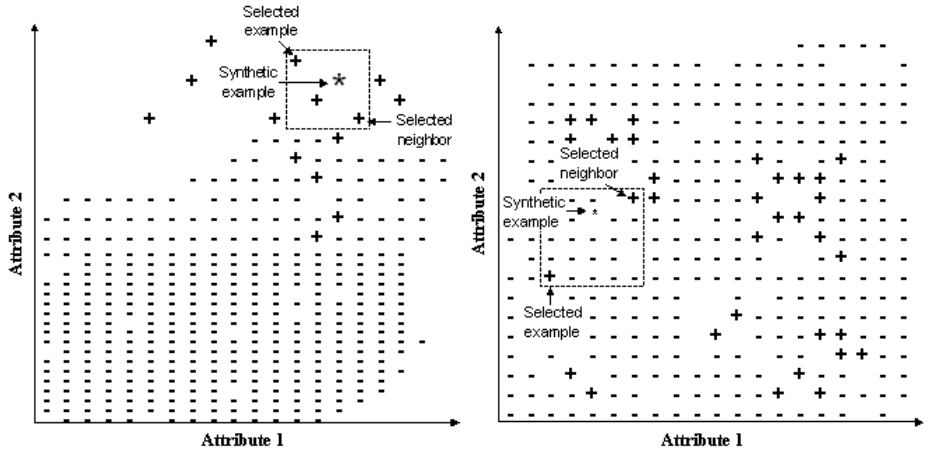

Fig. 1. The synthetic generation of minority class instances is normally performed by interpolating between a pair of minority class instances (left). However, sometimes new instances can be created inside a majority class region (right) when the chosen pair is separated by majority class instances.

randomly selecting a minority class instance and one of its k Nearest Neighbors (k normally takes a value of five). This technique is described in detail in Ref. 6. SMOTE, however, may have problems with (i) isolated instances as it may generate instances in the majority class region by interpolating between two distant minority instances (see Figure 1 right), (ii) when there is overlapping between classes, possibly due to noise, and (iii) the value of k needs to be determined for each domain, which may require a computationally expensive trial-and-error process.

Recently, Han, Wang, and Mao proposed a modification of SMOTE, called borderline-SMOTE, where new instances are created only at the border of the minority class decision region.[12] This method improves SMOTE by generating new instances only from the instances located at the borderline of the minority class. This method has two variants: Borderline-Smote1 (borderSM1), in which the instances are generated only with instances of the positive class, and Borderline-Smote2 (borderSM2), which interpolates between one positive and one negative class instance to generate a new one. Borderline-Smote, however, may have problems when there is a non-well defined border between instances of both classes, as it does not generate instances in such cases.

In Ref. 5, the authors introduced the Cluster-SMOTE algorithm. Their idea is that identifying even an approximation of the minority class regions would enhance the performance on global classification. Cluster-SMOTE simply creates clusters from the minority class examples with the k-means algorithm and then applies SMOTE to each cluster. In Cluster-SMOTE the parameter k, used as seed to find the clusters, is defined by the user. Results presented in Ref. 5 showed that, although there was not a clear winner, Cluster-SMOTE was preferred over SMOTE when evaluating with convex hull analysis over the ROC graph. If the selected k is smaller

than the actual number of clusters of the minority class, Cluster-SMOTE can still produce instances within majority class regions. The problem with Cluster-SMOTE is that the user has to know (or find) the right number of clusters (parameter k), and this is a really difficult task.[16] In fact, this problem is known as "the fundamental problem of cluster validity".[25] In our algorithms, the goal is to release the user from this task.

A recent method presented in Ref. 15 shows how, by adding noise to the new instances, the class imbalance problem can be reduced. They created an over and undersampling technique with noise to work with boosting algorithms. Their algorithm, called JOUS-Boost, combines over/undersampling with jittering. The way in which new examples are created (to avoid only replicating instances) consists on the addition of a small amount of random noise or jittering to the original values of the attributes of the instances. Uniform noise $(-v\sigma_j, v\sigma_j)$ is added according to the standard deviation of the attributes (denoted as σ), and a parameter (v) that is tuned to optimize performance. The value of v depends on the data set characteristics. This algorithm is designed to work with AdaBoost. It considers only numerical attributes, and the amount of jittering depends on the standard deviation of the attributes in the whole dataset. Our algorithms work with any classical classifier, with both numerical and nominal attributes. The amount of jittering, in the case of SOI-CJ, depends on the standard deviation of each attribute in the whole dataset and in each cluster, producing a more conservative strategy. Finally, they do not depend on any tuning parameter.

In this paper, we propose two new oversampling methods called SOI-C and SOI-CJ. In contrast with Cluster-SMOTE, our methods automatically generate clusters, so they do not depend on the definition of a k parameter. They also avoid the generation of instances in a majority class region. The number of new generated instances is proportional to the size of the clusters, so they are more robust to noise as very little oversampling will be produced around isolated (possibly noisy) examples. Finally, our algorithms deal with both numerical and nominal attributes.

It has recently been shown that ROS may achieve better performance than SMOTE and Borderline-Smote.[21]

3. Synthetic Oversampling of Instances

Our general approach, called SOI (Synthetic Oversampling of Instances), starts by generating clusters with the minority class instances. It then creates new instances within those clusters (see Figure 2). An advantage of the cluster-based strategy is that it can be used to identify possibly noisy examples in singleton clusters. The total number of instances added to each cluster is proportional to the cluster size. This strategy is less susceptible to noisy examples as it generates very few new instances from isolated (possibly noisy) examples. Our cluster-based strategy also avoids creating instances in a region of the majority class, as many Smote-based algorithms do, and it is able to create instances even when majority class

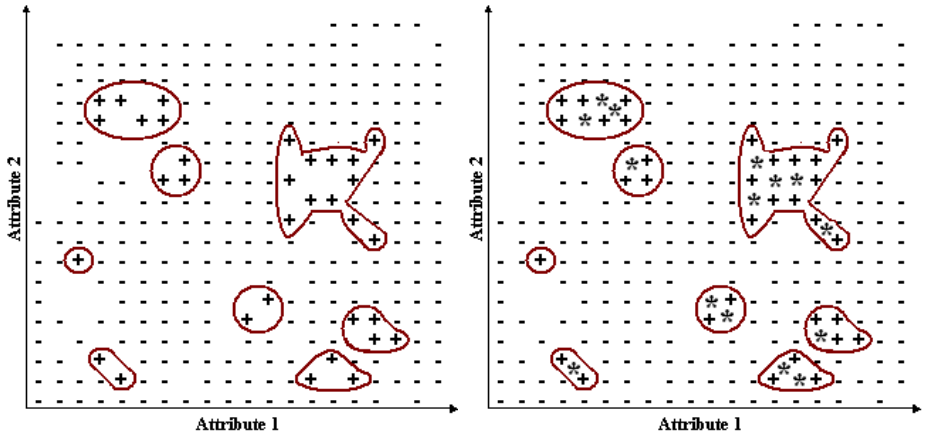

Fig. 2. The general idea of the proposed algorithms is to construct minority class clusters (left) and to generate instances within clusters (right).

instances are not close to the minority class instances, as opposed to Borderline-Smote, stressing the decision regions.

3.1. *Creation of clusters*

Clustering commonly refers to the classification of objects into groups (called clusters) where instances that belong to the same cluster are more similar to each other than instances from different clusters. In this approach, the term cluster refers to a group of instances belonging to the same class (the minority class). When we generate minority class clusters, each instance is considered as a seed to create a new cluster, while the rest of the instances are assigned to the cluster according to their distance to the seed (as we will describe through the algorithms). In this work, we use the Heterogeneous Value Difference Metric (HVDM) defined between two instances x and y as follows:[23]

$$HVDM(x, y) = \sqrt{\sum_{a=1}^{m} d_a^2(x_a, y_a)}$$

where m is the number of attributes and d_a returns the difference between the values of attribute a, depending on its type (numerical or nominal) so that each attribute contributes to the total HVDM value according to:

$$d_a(x_a, y_a) = \begin{cases} vdm(x_a, y_a) & \text{if } a \text{ is nominal} \\ Euclidean(x_a, y_a) & \text{if } a \text{ is continuous} \end{cases}$$

where vdm is evaluated as:

$$vdm_a(x_a, y_a) = \sum_{i=1}^{n} \sqrt{p(c_i|x_a) - p(c_i|y_a)}$$

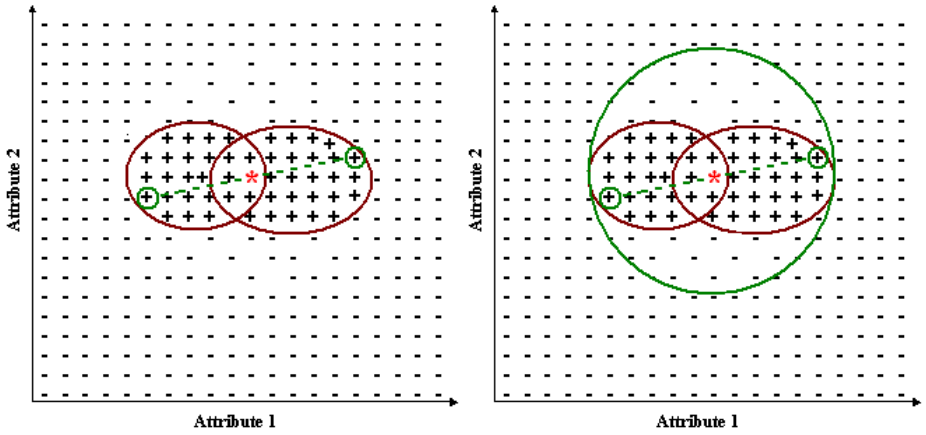

Fig. 3. Extreme instances of two clusters and their mean point (left). Potential influence area of the merged clusters for generating minority class instances (right).

where n is the number of classes, $p(c_i|x_a)$ is the probability of a particular class value (c_i) given an attribute value (x_a), and $Euclidean(x, y)$ is the Euclidean distance between values x and y. Using the *vdm* distance measure, two values are considered to be closer if they have high correlation with the output class.

For each cluster (initially represented by its seed), we add to it the set of minority class instances closest to it (with the restriction that there cannot be a majority class instance closer to the seed than any of the minority class instances of that set).

Once each cluster has been assigned its closest minority class instances, the algorithm merges clusters with common elements as follows. First, the common elements are deleted from the smallest cluster. Then, the farthest pair of instances between the two clusters is used to obtain the mean point between them (see Figure 3 left). This point corresponds to a new instance that takes as values the average of the two instances values for numerical attributes and both values in the case of nominal attributes (or one if it is the same for both instances). The algorithm then finds all the instances that are contained in a hypersphere with its center at the mean instance, with a radius of length equal to one of the points used to find the mean instance (see Figure 3 right). If most of the covered instances belong to the minority class, then the two clusters are merged. Otherwise, the clusters are not merged (see Figure 4).

The idea is to create clusters whose predominant elements belong to the minority class. In the experiments reported in this paper, the clusters are merged if the number of minority class instances is larger than the number of other covered instances, until it is not longer possible to merge clusters (see Algorithm 1). However, other more conservative merging criteria could be used as well.

Once the clusters have been created, new instances are generated within each cluster. We have implemented two different strategies to generate instances. In both

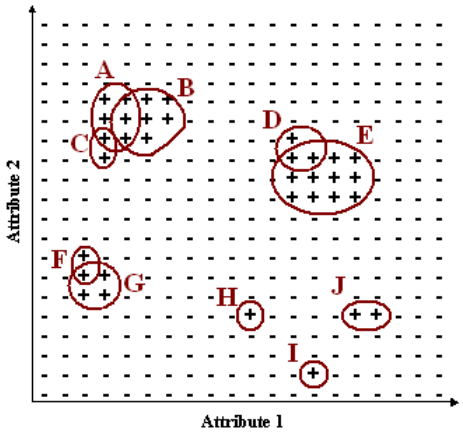

Fig. 4. Identification and merging of clusters.

Algorithm 1 Creation of minority class clusters.

Input: E: all instances and E^+: Minority class instances
Output: Clusters: C_f
{Form clusters}
for all $e_i \in E^+$ **do**
Create a new cluster $C_i = \{e_i\}$ with instance e_i as seed
Create an ascending sorted list (SL) with all the other instances ($E - \{e_i\}$) according to their distance with e_i.
end for
while the closest instance e_k is from the minority class ($e_k \in E^+$) **do**
remove the instance from SL and
add it to the cluster $C_i = C_i \cup \{e_k\}$
end while
{Merge clusters}
while there is an intersection between clusters **do**
select two clusters with an intersection
remove common elements from the smaller cluster
find the mid point (m) between the most distant elements in both clusters
obtain the distance (d) between m and the distant elements
if most of the elements covered from m within distance d are from the minority class **then**
merge clusters
else
discard merging and continue with another pair of clusters
end if
end while

cases, a jittering process is used to create new instances for single element clusters, as explained below.

3.2. *SOI-C*

Our first method creates new instances following an approach similar to SMOTE, mainly by interpolating between minority class instances. Instead of considering only the k neighbors, our approach, called SOI-C (SOI by Clustering), interpolates between a randomly selected pair of minority class instances belonging to the same cluster. In this case, we do not have to tune the number of neighbors to consider (as SMOTE does). The number of neighbors to choose from depends on the number of elements of each cluster. This reduces the possibility of creating instances in the majority class area. Finally, we do not need to tune the number of clusters, as Cluster-SMOTE does, as they are automatically constructed (see Algorithm 2).

Algorithm 2 Synthetic generation of instances within clusters using interpolation.

Input: C: all minority class clusters, E^+: minority class instances, N: number of synthetic instances to generate

```
for all c_i ∈ C do
    Prop = N / |E+| {percentage of new instances}
    N_i = Prop × |c_i|
    New_i = ∅
    for j = 1 to N_i do
        randomly select a pair of elements e_j, e_k ∈ c_i with j ≠ k
        new = interpolate(e_j, e_k) {See Algorithm 3}
        New_i = {new} ∪ New_i
    end for
    c_i = c_i ∪ New_i
end for
```

For numerical attributes, the interpolation is performed as it is done in SMOTE. Given a pair of numerical attributes from two instances, we calculate their difference. We then generate a random number between 0 and 1. After that, we create a new value by adding the obtained difference, multiplied by the random number, to the smaller value.

For nominal attributes, a new value is created according to the current distribution of values of attributes, considering only the attribute values of instances inside that cluster. This creates nominal values according to the current values in the cluster, reducing the possibility of creating instances more similar to those of other classes. This idea is better explained with an example: suppose that there are ten instances in a cluster, five of them have the value "blue", three have the value

"red", and two have the value "yellow" for a given attribute. Then, the value of that attribute for the new synthetic instance will be selected as "blue" with a probability of 50%, as "red" with 30%, and "yellow" with 20%. This process is described in Algorithm 3.

Algorithm 3 Synthetic generation of one instance using interpolation.

Input: e_1, e_2, two instances, where each instance is characterized by a set of attributes $\{a_1, a_2, \ldots, a_n\}$.
Output: Synthetic instance ($Synt$)
  for $i = 1$ to n (number of attributes) **do**
    if a_i is numerical **then**
      diff $= |a_{i,1} - a_{i,2}|$ {Difference between the values of the i-th. attribute in both instances}
      gap = random number between 0 and 1
      if $a_{i,1} \geq a_{i,2}$ **then**
        $Synt_i = a_{i,1} + gap * \text{diff}$
      else
        $Synt_i = a_{i,2} + gap * \text{diff}$
      end if
    else if a_i is nominal **then**
      $Synt_i$ = generate a new nominal value using the value distribution of a_i in the current cluster
    end if
  end for

3.3. *SOI-CJ*

The second approach, called SOI-CJ (SOI by Clustering and Jittering), uses a jittering process to generate synthetic instances "around" instances inside the cluster. In the case of numerical attributes, it first evaluates the standard deviation of the minority class elements inside the cluster and then the standard deviation of all the elements belonging to the minority class. It selects the minimum standard deviation between them and creates a new value for that attribute as follows: $new = current + random * std\_dev$, where $random$ is a random number between $\{-1, 1\}$. A more conservative jittering process is performed as we also consider the standard deviation of an attribute within a cluster. For the case of nominal attributes, we use the same approach that we used in SOI-C. We do not need to tune the amount of jittering, as it depends on the available data and on the data of each cluster (see Algorithm 4).

Our algorithms are simple, have no tuning parameters, are less sensitive to noisy examples, and generate instances within minority class clusters. In the following sections, we present the experimental setups and main results.

Algorithm 4 Synthetic generation of instances within clusters using jittering.

Input: C: all minority class clusters, E^+: minority class instances, N: number of synthetic instances to generate

for all $c_i \in C$ **do**
  $Prop = \frac{N}{|E^+|}$ {percentage of new instances}
  $N_i = Prop \times |c_i|$
  obtain standard deviation for all numerical attributes within cluster ($\vec{std}_{c_i}$) and within E^+ ($\vec{std}_{E_i^+}$).
  Let $\vec{std} = min(std_{c_i}, std_{E_i^+}) \forall i$
  $New_i = \emptyset$
  for $j = 1$ until N_i **do**
    randomly select an instance $e_i \in c_i$
    $new = jitter(e_i, \vec{std})$ {see Algorithm 5}
    $New_i = \{new\} \cup New_i$
  end for
  $c_i = c_i \cup New$
end for

Algorithm 5 Synthetic generation of one instance using jittering.

Input: e_1, instance characterized by a set of attributes $\{a_1, a_2, \ldots, a_n\}$ and $\vec{std}$ a vector with the standard deviation of all the attributes

Output: Synthetic instance ($Synt$)

for $i = 1$ to n (number of attributes) **do**
  if a_i is numerical **then**
    $random$ = random value in $\{-1, 1\}$
    $std\_dev_i \in \vec{std}$ standard deviation of attribute a_i
    $Synt_i = current + random * std\_dev_i$
  else if a_i is nominal **then**
    $Synt_i$ = generate a new nominal value using the value distribution of a_i in the current cluster
  end if
end for

4. Experiments

In our experimental setting, we worked with problems of two classes and generated as many synthetic instances as needed to balance the classes (for all the oversampling methods). In the case of datasets with more than two classes, we selected the minority class and merged the rest of the classes into a single majority class.

We tested our methods with forty three datasets. Twenty three were obtained from the UCI Machine Learning Repository[3] and twenty were artificially constructed. The datasets from UCI show different imbalance ratios, number of at-

tributes, and have continuous and categorical attributes. The artificial datasets were generated to test the performance of the algorithms with a variety of controlled conditions.

For the artificial datasets, we considered different conditions as explained by Japkowicz *et al.*[13] We generated ten datasets with two continuous attributes (continuous); five datasets with two nominal attributes (nominal); and five datasets with one continuous and one nominal attribute (mixed). The ten continuous datasets were generated as follows: each instance belongs to a bi-dimensional domain in which each attribute is in the [0, 1] range, and each instance is associated with one of two classes. The input range for each dimension is divided in $2n$ regular intervals, where n is a complexity parameter that takes values from 1 to 5. With this configuration, there are $2(2n)$ areas in the bi-dimensional domain to create instances where adjacent areas have opposite output classes. The instances in the $2(2n)$ areas are randomly generated. We consider an imbalance ratio of 1:10 for five datasets and 1:20 for the other five datasets. In order to create the five nominal datasets, we considered two nominal attributes with four possible values each. The values for each attribute of an instance were randomly selected. We considered an imbalance ratio of 1:10 for the five nominal datasets. For the creation of the five combined datasets, we used the same technique used in the continuous datasets for the first attribute, and the same technique used in the nominal datasets for the second attribute. The characteristics of all datasets are shown in Table 1.

Although accuracy is typically used to evaluate the performance of machine learning algorithms, this is not appropriate when the data classes are imbalanced. This is because accuracy averages the performance of all the classes, hiding the accuracy on the minority class. The evaluation metrics used for imbalanced domains are typically obtained from the confusion matrix. The results of our experiments were obtained using recall, precision, F-measure, and AUC (Area under the ROC curve) of the minority class. These measures have shown to give a better estimate of the result for the minority class classification.

We decided to use well-known and commonly-used learning algorithms to evaluate our oversampling methods that have also been tested on class imbalanced problems. The following algorithms from WEKA[24] were used in the tests: J48 (the Weka implementation of C4.5), Naive Bayes, IBK with k=3, PART, Multilayer perceptron, and AdaBoostM1 with J48.

We performed an ANOVA test to verify if the use of our oversampling technique produced a significant improvement in the performance of the algorithms. Additionally, we performed paired t-tests between our proposed algorithms and the rest of the oversampling algorithms with 95% of confidence.

We performed a ten-fold cross validation experiment ten times in which we created synthetic instances only for the training set, and averaged the results. This process is repeated for all the oversampling methods (ROS, SMOTE, borderSM1, borderSM2, SOI-C, and SOI-CJ) and all the performance metrics (Recall, Precision, F-measure, and AUC) over all the datasets.

Table 1. Description of the datasets used in the experiments, with its name, its type indicated in parenthesis for the non-artificial datasets, *M* for Mixed, *C* for continuous and *N* for nominal, the size (number of instances), the percentage of minority class instances (M.C.) and the number of attributes.

Name	Size	M.C.	Attr.	Name	Size	M.C.	Attr.
Mixed01	110	9.09%	2	Continuos01	110	9.09%	2
Mixed02	110	9.09%	2	Continuos02	110	9.09%	2
Mixed03	110	9.09%	2	Continuos03	264	9.09%	2
Mixed04	110	9.09%	2	Continuos04	352	9.09%	2
Mixed05	221	9.05%	2	Continuos05	671	8.94%	2
Nominal01	110	9.09%	2	Continuos06	315	4.76%	2
Nominal02	110	9.09%	2	Continuos07	420	4.76%	2
Nominal03	120	16.67%	2	Continuos08	420	4.76%	2
Nominal04	120	16.67%	2	Continuos09	840	4.76%	2
Nominal05	300	16.67%	2	Continuos10	660	9.09%	2
Abalone1 (M)	600	16.66%	9	Escalon (C)	100	19%	3
Balloons (N)	76	46.05%	5	Glass (C)	214	7.94%	10
BreastCancer (C)	198	23.74%	33	Haberman (C)	306	26.47%	4
Bupa (C)	345	42.03%	7	Hayes-Roth (N)	132	22.72%	4
Car1 (N)	700	10%	7	Heart (N)	267	20.60%	23
Car2 (N)	700	10%	7	Hepatitis (M)	128	20.31%	20
CardioVasc (C)	312	17.63%	6	Imayuscula (C)	100	12%	3
Cinco (C)	106	18.87%	3	PostOpPatient (M)	88	27.27%	9
Coil (M)	184	19.57%	18	Raro (C)	82	29.26%	3
Cpu (C)	162	4.32%	9	Servo (N)	161	31%	5
Diabetes (C)	768	35%	9	Wine (C)	178	27%	14
Ecoli (M)	332	10.54%	8				

5. Results

We present the results of our experiments for the J48 algorithm in Tables 2 to 9. It is important to mention that although in this paper we only show results for the J48 algorithm, similar tables were produced for all the datasets, for each classifier, and for each performance metric. All these tables are reported in Ref. 17, in which we show one table for each combination of performance metric, kind of data, and classifier.

In these tables, the first column corresponds to the dataset name. The second, to the percentage of oversampling performed over the minority class. The third column shows the original performance of the algorithm (without oversampling). The rest of the columns show the name of the oversampling technique used (ROS, SMOTE, borderSM1, borderSM2, SOI-C, and SOI-CJ) for each of the datasets of the corresponding row. A "**" in the first column means that the ANOVA test found that the results are in general, statistically significant and they are not a consequence of randomness introduced by the oversampling methods. In the rest of the columns, a "*" means a statistically significant difference with a confidence level of 95% with SOI-C and a "+" means a statistically significant difference with SOI-CJ. The best results are shown in bold face.

Table 2. Recall results with artificial datasets for the J48 classification algorithm.

Dataset	%over	Original	ROS	SMOTE	BSM1	BSM2	SOI-C	SOI-CJ
Nominal01**	900	0.000	0.490	0.440	0.030*+	0.030*+	**0.500**	**0.500**
Nominal02**	900	0.000	0.320	0.200*+	0.000*+	0.000*+	**0.330**	**0.330**
Nominal03**	400	0.000	**0.530**	0.430*+	0.035*+	0.035*+	0.515	0.515
Nominal04**	400	0.000	**0.520**	0.440*+	0.015*+	0.040*+	**0.520**	**0.520**
Nominal05**	400	0.000	0.622*+	0.482*+	0.010*+	0.010*+	**0.624**	**0.624**
Numeric01	800	0.900	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
Numeric02**	800	0.000	0.185*+	0.695+	0.605*+	0.675+	0.690+	**0.750***
Numeric03**	800	0.008	0.015*+	0.220*+	0.322*+	0.348*+	0.575	**0.588**
Numeric04**	800	0.000	0.038*+	0.057*+	0.180*+	0.173*+	**0.248**	0.235
Numeric05**	800	0.000	0.000*+	**0.732***+	0.573	0.517	0.485	0.525
Numeric06	300	0.950	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
Numeric07**	300	0.000	0.412*+	0.838	0.810	**0.860**	0.782	0.815
Numeric08**	300	0.261	0.094*+	0.150*+	0.408*+	0.381*+	0.571	**0.580**
Numeric09**	300	0.000	0.067*+	0.192*+	0.274*+	0.248*+	0.598	**0.615**
Numeric10**	300	0.000	0.002*+	0.182*+	0.257*+	0.267*+	0.614	**0.634**
Mixed01**	900	0.900	0.900+	0.900+	0.900+	0.900+	0.900+	**1.000***
Mixed02**	900	0.000	0.640*+	0.680*+	0.250*+	0.270*+	0.730	**0.740**
Mixed03**	900	0.210	0.610+	0.670+	0.430*+	0.530+	0.660+	**0.810***
Mixed04	900	0.470	0.720	0.730	0.730	**0.750**	0.710	**0.750**
Mixed05**	900	0.290	**0.645**	0.570	0.505	0.560	0.615	0.625

Table 3. Precision results with artificial datasets for the J48 classification algorithm.

Dataset	%over	Original	ROS	SMOTE	BSM1	BSM2	SOI-C	SOI-CJ
Nominal01**	900	0.000	0.098	**0.126**	0.009	0.009	0.089	0.089
Nominal02**	900	0.000	0.053	**0.070**	0.000	0.000	0.052	0.052
Nominal03**	400	0.000	**0.217**	0.209	0.021*+	0.017*+	0.205	0.205
Nominal04**	400	0.000	0.201	**0.208**	0.009*+	0.022*+	0.190	0.190
Nominal05**	400	0.000	**0.166**	0.146	0.017*+	0.017*+	0.161	0.161
Numeric01**	800	0.900	**1.000**	**1.000**	**1.000**	0.893*+	**1.000**	0.995
Numeric02**	800	0.000	0.338*+	0.303*+	0.398*+	0.379*+	**0.492**+	0.415*
Numeric03**	800	0.020	0.043*+	0.059*+	0.131*+	**0.137***+	0.117	0.105
Numeric04**	800	0.000	**0.132**	0.080*+	0.097*+	0.114	0.106	0.128
Numeric05**	800	0.000	0.000*+	0.088	**0.100**	0.092	0.093	0.089
Numeric06**	300	1.000	**0.995**	**0.995**	**0.995**	0.920	**0.995**	**0.995**
Numeric07	300	0.000	0.623	0.596	0.539	0.530	**0.633**	0.619
Numeric08**	300	0.298	0.285	0.155*+	**0.330***+	0.317*+	0.247	0.244
Numeric09	300	0.000	0.314	0.285	0.330	**0.343**	0.280	0.280
Numeric10**	300	0.000	0.020*+	**0.176**	0.117	0.144	0.157	0.158
Mixed01**	900	0.900	0.860	0.855	0.860	0.813	0.873	**0.932**
Mixed02**	900	0.000	0.133	0.153	0.145	0.122	0.159	**0.173**
Mixed03**	900	0.210	0.560	**0.595**	0.388	0.428	0.499	0.471
Mixed04	900	0.410	0.303	0.302	**0.361**	0.321	0.348	0.297
Mixed05**	900	0.371	0.257	**0.364**	0.309	0.262	0.306	0.224

Table 4. F-measure results with artificial datasets for the J48 classification algorithm.

Dataset	%over	Original	ROS	SMOTE	BSM1	BSM2	SOI-C	SOI-CJ
Nominal01**	900	0.000	0.162*+	**0.190***+	0.014*+	0.014*+	0.150	0.150
Nominal02**	900	0.000	0.090	**0.094**	0.000*+	0.000*+	0.089	0.089
Nominal03**	400	0.000	**0.300**	0.271	0.026*+	0.023*+	0.282	0.282
Nominal04**	400	0.000	0.279	0.268	0.011*+	**0.028*+**	0.273	0.273
Nominal05**	400	0.000	**0.260**	0.222*+	0.012*+	0.012*+	0.254	0.254
Numeric01**	800	0.900	**1.000**	**1.000**	**1.000**	0.927	**1.000**	**1.000**
Numeric02**	800	0.000	0.236*+	0.399*+	0.442*	0.456*	**0.540**+	0.487*
Numeric03**	800	0.011	0.022*+	0.088*+	0.175	**0.186**	0.171	0.166
Numeric04**	800	0.000	0.057	0.050	0.101	**0.109**	0.093	0.093
Numeric05**	800	0.000	0.000*+	0.156	**0.162**	0.148	0.120	0.127
Numeric06**	300	0.967	**0.997**	**0.997**	**0.997**	0.950	**0.997**	**0.997**
Numeric07**	300	0.000	0.459*+	**0.674**	0.618	0.628	0.673	**0.674**
Numeric08**	300	0.271	0.131*+	0.141*+	**0.345**	0.314	0.304	0.313
Numeric09**	300	0.000	0.108*+	0.159*+	0.203*+	0.210*+	0.283	**0.287**
Numeric10**	300	0.000	0.003*+	0.137*+	0.145*+	0.154*+	0.228	**0.236**
Mixed01**	900	0.900	0.873+	0.870+	0.873+	0.845+	0.882+	**0.954***
Mixed02**	900	0.000	0.216	0.240	0.172*+	0.157*+	0.255	**0.271**
Mixed03**	900	0.210	0.577	**0.618***+	0.402*+	0.460*+	0.547	0.568
Mixed04	900	0.428	0.396	0.403	**0.455**	0.420	0.441	0.391
Mixed05**	900	0.312	0.340	**0.426**	0.357	0.336	0.385	0.308

Table 5. AUC results with artificial datasets for the J48 classification algorithm.

Dataset	%over	Original	ROS	SMOTE	BSM1	BSM2	SOI-C	SOI-CJ
Nominal01**	900	0.500	0.541	0.602	**0.636**	0.634	0.532	0.532
Nominal02**	900	0.507	0.405	0.420	**0.471**	**0.471**	0.404	0.404
Nominal03	400	0.499	0.577	0.565	0.563	0.556	**0.590**	**0.590**
Nominal04	400	0.496	0.499	0.522	0.518	**0.535**	0.497	0.497
Nominal05	400	0.500	0.526	0.518	0.523	0.523	**0.537**	**0.537**
Numeric01	800	0.950	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
Numeric02**	800	0.500	0.361*+	0.798	0.708*+	0.725*+	0.845	**0.874**
Numeric03**	800	0.507	0.555	0.494	0.569	**0.579**	0.508	0.520
Numeric04	800	0.500	**0.496**	0.456	0.440	0.459	0.467	0.452
Numeric05**	800	0.500	0.417	0.414	**0.486**	0.460	0.427	0.443
Numeric06	300	0.975	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
Numeric07**	300	0.500	0.569*+	**0.893**	0.832	0.859	0.878	0.870
Numeric08	300	0.640	0.567	0.482	**0.608**	0.587	0.483	0.498
Numeric09	300	0.500	0.517	0.475	0.529	0.518	0.505	**0.540**
Numeric10	300	0.491	**0.474**	0.459	0.458	0.470	0.445	0.462
Mixed01	900	0.950	0.998	0.997	**0.999**	0.998	0.997	**0.999**
Mixed02**	900	0.491	0.702	0.672	**0.754**	0.736	0.699	0.724
Mixed03**	900	0.642	**0.970**	0.897	0.692*+	0.772*+	0.953	0.930
Mixed04	900	0.716	0.741	0.744	0.757	0.753	**0.764**	0.759
Mixed05	900	0.674	**0.748**	0.684	0.686	0.700	0.716	0.722

Table 6. Recall results with real datasets for the J48 classification algorithm.

Dataset	%over	Original	ROS	SMOTE	BSM1	BSM2	SOI-C	SOI-CJ
Abalone1	100	0.385	0.465	0.460	0.455	0.450	0.465	**0.470**
Balloons**	100	0.774	0.731*+	0.797*+	0.806	0.817	**0.823**	**0.823**
Breast_Cancer	200	0.186	0.320*+	0.388	0.390	0.386	**0.404**	**0.404**
Bupa**	100	0.512	0.539*+	0.673*+	0.670*+	0.683*+	**0.724**	0.718
Cardiovascular	400	0.052	0.193*+	0.510	0.537	**0.543**	0.470	0.458
Coil	200	0.247	0.172*+	0.244	0.236	0.264	0.242	**0.244**
Cpu	1000	0.871	0.086*+	0.843	0.586*+	0.629*+	0.771+	**0.871***
Ecoli**	600	0.547	0.724*+	0.816	0.800	0.816	0.818	**0.829**
Glass**	700	0.033	0.006*+	0.506	0.211	0.206	0.467	**0.517**
Heart**	300	0.477	0.515*+	0.615+	0.600*+	**0.665**	0.633	0.658
Hepatitis**	300	0.478	0.581	0.600	0.626	**0.670**	0.596	0.648
Postoperative	100	0.104	0.104	**0.154**	0.133	**0.154**	0.142	0.137
Car1**	700	0.916	0.920	0.946	0.947	**0.980**	0.934	0.974
Car2**	700	0.850	**0.991**	0.943	0.951	0.900*+	0.970	0.940
Cinco**	300	0.445	0.200*+	0.525*+	0.555*+	**0.635**	0.615	0.630
Diabetes**	100	0.557	0.515*+	0.638*+	0.683*+	0.732	**0.738**	0.721
Escalon**	300	0.105	0.365*+	0.805	0.785	**0.840**	0.795	0.825
Haberman**	200	0.417	0.467*+	0.736	**0.844***+	0.840*+	0.701	0.733
Hayes-roth**	200	0.473	**1.000**	**1.000**	0.987	0.980	**1.000**	0.997
Imayuscula**	600	0.042	0.058*+	**0.667**	0.392*+	0.450*+	0.567	0.633
Raro**	100	0.692	0.383*+	0.746	0.679*+	**0.762**	0.704	0.700
Servo**	100	0.772	0.788+	0.802	0.804	0.814	0.814	**0.888**
Wine	100	0.883	0.981	**0.990**	**0.990**	**0.990**	0.988	0.988

As can be seen from these tables, all the tested algorithms win in some datasets and lose in others. Similar results are obtained when we used other classifiers. In order to present our results in a clearer way we also made a summary of the results obtained according to each performance metric. For the twenty artificial datasets with six classifiers, there are one hundred and twenty results for each performance metric to show the performance of the six oversampling methods (four in the state of the art, SOI-C, and SOI-CJ). In the case of the twenty three UCI datasets, there are one hundred and thirty eight results. In Tables 10 to 13 we only show a summary of those results in which the ANOVA or pair-t tests showed statistical significance. The results of these tables are presented in pairs N_r/N_c, where N_r denotes the number of times that the oversampling method of that row is better than the oversampling method of that column, and N_c denotes the times that the oversampling method of that column is better than the oversampling method of that row. For example, in Table 10 the number of times that SOI-C is significantly better/worse than the other algorithms in the artificial datasets is: 56/1 for ROS, meaning that SOI-C is significantly better (with 95% of confidence level) than ROS fifty six times and significantly worse only once, 63/7 for SMOTE, 75/3 for borderSM1, 70/4 for borderSM2 and 2/19 for SOI-CJ. This means that although the number of overall wins and ties for SOI-C is roughly equivalent to that of the other algorithms, when

Table 7. Precision results with real datasets for the J48 classification algorithm.

Dataset	%over	Original	ROS	SMOTE	BSM1	BSM2	SOI-C	SOI-CJ
Abalone1	100	0.640	**0.743**	0.646	0.585*+	0.552*+	0.639	0.687
Balloons**	100	0.737	**0.736**	0.691	0.718	0.637	0.637	0.632
Breast_Cancer	200	0.257	0.289*+	0.302	0.315	0.289*+	**0.347**	0.342
Bupa**	100	0.609	**0.692**	0.602	0.592	0.603	0.579	0.585
Cardiovascular	400	0.097	**0.279**	0.181	0.189	0.177	0.212	0.200
Coil	200	0.222	0.192	0.176	**0.225**	0.206	0.196	0.192
Cpu	1000	0.729	0.055*+	0.369	0.285	0.252	0.345	**0.413**
Ecoli**	600	0.724	**0.709**	0.616	0.683	0.653	0.606	0.595
Glass**	700	0.043	0.004*+	0.220	**0.164**	0.135	0.145	0.159
Heart**	300	0.511	0.492	0.479	0.500	0.498	0.509	**0.511**
Hepatitis**	300	0.526	0.604	0.539	**0.647**	0.606	0.533	0.561
Postoperative	100	0.179	0.137	0.150	0.155	0.165	**0.174**	0.162
Car1**	700	0.879	**0.932**	0.918	0.887	0.812	0.882	0.772
Car2**	700	0.789	**0.989**	0.856	0.846	0.694	0.929	0.732
Cinco	300	0.580	0.312	0.245	0.259	0.284	0.341	**0.342**
Diabetes**	100	0.662	**0.641**	0.621	0.554	0.540	0.585	0.620
Escalon	300	0.123	0.467	**0.586**	0.552	0.508	0.508	0.487
Haberman**	200	0.420	**0.456***+	0.347	0.294*+	0.288	0.372	0.361
Hayes-roth**	200	0.967	**1.000**	**1.000**	0.919	0.764*+	**1.000**	0.971
Imayuscula**	600	0.037	0.097*+	0.245	**0.399**	0.330	0.225	0.193
Raro**	100	0.860	0.424	0.447	**0.606**	0.563	0.444	0.457
Servo**	100	0.814	0.820	0.810	0.818	0.807	**0.829**	0.677
Wine	100	0.925	0.966	**0.969**	0.968	0.968	0.966	**0.969**

compared with each algorithm, its superiority is clearly shown as it has more results that are significantly better than the other algorithms.

In case of SOI-CJ we have the following results: ROS 67/1, SMOTE 80/6, borderSM1 84/1, borderSM2 81/1, SOI-C 19/2. Similar results are reported for precision, F-measure and AUC for both types of datasets. It means that when our proposed methods loose, their difference in results are not statistically significant, while they are when they have better results.

6. Discussion

Although we have only presented complete tables for one classifier, in this case for J48 from Weka, similar results are obtained for the rest of the classifiers. As expected, some classifiers tend to have better performance than others but they are not consistently better in all measures or for all datasets. The complete tables for all the algorithms can be consulted in Ref. 17.

As it can be seen from Tables 10 to 13, our cluster-based algorithms perform better than the other oversampling methods in most datasets and for most of the performance metrics, except for AUC for which they have very similar results. Our algorithms are not always better, but in general when they win, their difference

Table 8. F-measure results with real datasets for the J48 classification algorithm.

Dataset	%over	Original	ROS	SMOTE	BSM1	BSM2	SOI-C	SOI-CJ
Abalone1	100	0.468	**0.558**	0.516	0.490*+	0.484*+	0.528	0.541
Balloons**	100	0.742	0.710	0.727	**0.746**	0.706	0.707	0.703
Breast_Cancer	200	0.199	0.295*+	0.329	0.335	0.321	**0.357**	0.354
Bupa**	100	0.547	0.589*+	0.627	0.618	0.629	0.636	**0.637**
Cardiovascular	400	0.062	0.212	0.261	0.275	0.263	**0.283**	0.271
Coil	200	0.221	0.171	0.200	**0.222**	0.221	0.207	0.209
Cpu	1000	0.776	0.064*+	0.480	0.355*+	0.336*+	0.438	**0.522**
Ecoli**	600	0.589	0.694	0.686	**0.716**	0.708	0.681	0.679
Glass**	700	0.034	0.005*+	**0.297**	0.173	0.154*+	0.217	0.239
Heart**	300	0.474	0.493	0.526	0.534	0.560	0.552	**0.561**
Hepatitis**	300	0.467	0.564	0.541	0.607	**0.614**	0.537	0.574
Postoperative	100	0.176	0.109	0.143	0.128	**0.145**	0.143	0.135
Car1**	700	0.886	0.916	**0.923**	0.907	0.879	0.895	0.851
Car2**	700	0.804	**0.990**	0.891*	0.890*	0.775*+	0.945	0.814
Cinco**	300	0.480	0.235*+	0.294*+	0.327	0.366	**0.397**	0.395
Diabetes**	100	0.595	0.567*+	0.628	0.606	0.619	0.650	**0.665**
Escalon**	300	0.103	0.373*+	**0.651**	0.618	0.612	0.584	0.587
Haberman**	200	0.407	0.446	0.466	0.434	0.427	**0.473**	0.470
Hayes-roth**	200	0.621	**1.000**	**1.000**	0.946	0.844*+	**1.000**	0.982
Imayuscula**	600	0.036	0.071*+	0.306	**0.354**	0.326	0.287	0.274
Raro**	100	0.741	0.363	0.524	0.590	**0.597**	0.519	0.504
Servo**	100	0.774	0.787	0.790	0.796	0.793	**0.806**	0.756
Wine	100	0.897	0.971	**0.978**	0.977	0.977	0.975	0.976

with respect to the other algorithms is statistically significant, while when they lose, their difference is not significant. On average, half of the time our algorithms produce significantly better results than the rest of the oversampling methods.

It is interesting to note that the proposed cluster-based method that generates new instances using a jittering process is a very clear winner over the rest of the algorithms, SOI-CJ even outperforms SOI-C. One possible explanation for this behavior could be that the merging process for clusters allows for the introduction of a large number of instances of the majority class. Consequently, the method that interpolates between elements in the clusters for oversampling (SOI-C) may still be generating instances in a majority class region, while the jittering approach generates instances between jittered instances. SOI-C, however, still outperforms ROS, SMOTE, and Borderline-SMOTE.

The proposed algorithms scale poorly with the number of attributes and size of the data since they have to evaluate the distances between all the data and all the minority class examples. In the performed tests, their running times, however, were equivalent to those of Borderline-SMOTE, although they were larger than ROS and SMOTE. Faster running times can be achieved, for instance, with better data structures to find similar examples faster (e.g., kd-trees) or with parallel

Table 9. AUC results with real datasets for the J48 classification algorithm.

Dataset	%over	Original	ROS	SMOTE	BSM1	BSM2	SOI-C	SOI-CJ
Abalone1**	100	0.690	0.816	**0.838**	0.809	0.799	0.804	0.818
Balloons**	100	0.747	0.779	0.797	0.781	0.782	**0.804**	0.802
Breast_Cancer	200	0.521	0.596	0.604	0.606	0.585	**0.636**	0.632
Bupa**	100	0.649	0.735	0.728	0.722	**0.732**	0.726	0.728
Cardiovascular	400	0.527	**0.582**	0.532	0.524	0.514	0.549	0.558
Coil	200	0.554	0.576	0.552	**0.582**	0.575	0.570	0.566
Cpu	1000	0.932	0.945	0.947	0.938	0.943	0.945	**0.955**
Ecoli**	600	0.785	**0.945**	0.941	0.940	0.943	0.938	0.943
Glass	700	0.589	0.660	**0.788**	0.735	0.733	0.730	0.739
Heart**	300	0.759	0.797	0.808	0.802	0.794	0.814	**0.816**
Hepatitis**	300	0.659	0.810	0.762	**0.824**	0.817	0.769	0.783
Postoperative	100	0.412	0.362	0.356	0.358	0.368	0.363	**0.375**
Car1	700	0.966	0.995	**0.998**	0.993	0.989	0.990	0.992
Car2**	700	0.948	**0.999**	0.993	0.994	0.983	0.998	0.987
Cinco	300	0.706	0.506	0.497	0.559	0.569	**0.584**	0.571
Diabetes	100	0.756	0.782	0.795	0.774	0.774	**0.804**	0.802
Escalon**	300	0.540	0.648	**0.912**	0.904	0.911	0.848	0.844
Haberman**	200	0.614	0.656	0.677	0.662	0.669	0.691	**0.696**
Hayes-roth**	200	0.660	**1.000**	**1.000**	0.997	0.994	**1.000**	0.999
Imayuscula**	600	0.476	0.363	0.577	0.612	**0.633**	0.621	0.610
Raro**	100	0.853	0.614	0.650	0.718	**0.728**	0.657	0.649
Servo**	100	0.885	**0.940**	0.938	0.935	0.932	**0.940**	0.930
Wine	100	0.930	**0.999**	**0.999**	**0.999**	**0.999**	**0.999**	**0.999**

Table 10. Recall: Comparison among the oversampling methods with artificial datasets (top table) and with real datasets (bottom table) considering statistical significance.

	ROS	**SMOTE**	**borderSM1**	**borderSM2**	**SOI-C**	**SOI-CJ**
			Artificial Datasets			
SOI-C	56/1	63/7	75/3	70/4	–	2/19
SOI-CJ	67/1	80/6	84/1	81/1	19/2	–
			Real Datasets			
SOI-C	92/0	25/3	37/5	27/3	–	0/6
SOI-CJ	95/1	30/0	40/4	27/4	6/0	–

Table 11. Precision: Comparison among the oversampling methods with artificial datasets (top table) and real datasets (bottom table) considering statistical significance.

	ROS	**SMOTE**	**borderSM1**	**borderSM2**	**SOI-C**	**SOI-CJ**
			Artificial Datasets			
SOI-C	24/2	22/3	30/10	33/9	–	6/5
SOI-CJ	26/3	25/3	35/10	35/9	5/6	–
			Real Datasets			
SOI-C	26/14	13/0	29/12	38/1	–	5/4
SOI-CJ	24/21	10/8	24/19	37/5	4/5	–

Table 12. F-measure: Comparison among the oversampling methods with artificial datasets (top table) and real datasets (bottom table) considering statistical significance.

	ROS	**SMOTE**	**borderSM1**	**borderSM2**	**SOI-C**	**SOI-CJ**
			Artificial Datasets			
SOI-C	48/2	35/10	58/2	59/1	–	9/7
SOI-CJ	48/4	40/10	57/4	56/3	7/9	–
			Real Datasets			
SOI-C	47/0	8/0	17/3	21/3	–	1/0
SOI-CJ	47/2	4/1	14/5	22/3	0/1	–

Table 13. AUC: Comparison among the oversampling methods with artificial datasets (top table) and real datasets (bottom table) considering statistical significance.

	ROS	**SMOTE**	**borderSM1**	**borderSM2**	**SOI-C**	**SOI-CJ**
			Artificial Datasets			
SOI-C	12/0	0/0	9/0	9/0	–	0/0
SOI-CJ	12/0	0/0	9/0	9/0	0/0	–
			Real Datasets			
SOI-C	16/0	4/0	2/0	3/0	–	0/0
SOI-CJ	16/0	4/0	2/0	3/0	0/0	–

computing architectures, such as CUDA. We leave the implementation of faster versions of our algorithms as future work.

7. Conclusions

Imbalanced datasets are common to many real-world applications. However, many classifiers tend to perform poorly on the minority class, which is often the class of interest. Several approaches have been previously proposed to improve the performance of algorithms in the minority class. These approaches, however, tend to perform similarly or worse than random oversampling, generate examples in majority class regions, and have problems with noisy examples. This paper presents two new cluster-based oversampling methods. The idea is to form clusters with the minority class instances and oversample within each cluster avoiding some of the problems of previous approaches. In the experiments performed, the results from the proposed algorithms are, on average, half of the time significantly better than the rest of the oversampling methods.

There are several research directions that are worth exploring. In particular, we would like to distinguish with some confidence noisy instances from truly minority class instances. The identification of singleton clusters can be a first step in this direction. We would like to test similar ideas for undersampling algorithms based

on clustering. Finally, we would like to test other merging criteria for clusters and assess their effect in the results.

Acknowledgments

The first author would like to thank CONACyT for the student grant number 3732. The authors would also like to thank the anonymous reviewers for their helpful comments in a previous version of this paper.

References

1. Rehan Akbani, Stephen Kwek, and Nathalie Japkowicz, Applying support vector machines to imbalanced datasets, *European Conference of Machine Learning*, LNCS, Vol. 3201, 2004, pp. 39–50.
2. Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria Carolina Monard, A study of the behavior of several methods for balancing machine learning training data, *ACM SIGKDD Explorations Newsletter – Special Issue on Learning from Imbalanced Datasets* **6**(1) (June 2004).
3. C. Blake and C. Merz, UCI Repository of machine learning databases, Department of Information and Computer Sciences, University of California, Irvine, 1998. http://www.ics.uci.edu/mlearn/mlrepository.html.
4. A. Bradley, The use of the area under the ROC curve in the evaluation of machine learning algorithms, *Pattern Recognition* **30**(7) (1997) 1145–1159.
5. D. A. Cieslak, N. V. Chawla and A. Striegel, Combating imbalance in network intrusion datasets, *IEEE International Conference on Granular Computing* (2006) 732–737.
6. N. V. Chawla, K. W. Bowyer, L. O. Hall and W. P. Kegelmeyer, SMOTE: Synthetic minority oversampling technique, *JAIR 16*, 2002, pp. 321–357.
7. Nitesh V. Chawla, Nathalie Japkowicz, and Aleksander Kotcz, Editorial: Special issue on learning from imbalanced data sets, *ACM SIGKDD Explorations Newsletter – Special Issue on Learning from Imbalanced Datasets* **6**(1) (June 2004).
8. P. Domingos, Metacost: A general method for making classifiers cost-sensitive, *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1999, pp. 155–164.
9. K. J. Ezzawa, M. Singh, and S. W. Norton, Learning goal oriented bayesian networks for telecommunications management, *Proceedings of the International Conference on Machine Learning, ICML'96*, Bari, Italy, Morgan Kafmann, 1996, pp. 139-147.
10. T. Fawcett and F. Provost, Combining data mining and machine learning for effective user v profile, *Proceeding of the 2nd International Conference on Knowledge Discovery and Data Mining*, Portland OR, AAAI Press, 1996, pp. 8-13.
11. H. Guo and H. L. Viktor, Learning from imbalanced data sets with boosting and data generation: The DataBoost-IM approach, *SIGKDD Explorations* **6**(1) (2004) 30–39.
12. H. Han, W. Y. Wang and B. H. Mao, Borderline-smote: A new oversampling method in imbalanced data sets learning, *In International Conference on Intelligent Computing* (*ICIC'05*), Lecture in notes in Computer Science, Vol. 3644 (2005), pp. 878–887.
13. N. Japkowicz and S. Stephen, The class imbalance problem: A systematic study, *Intelligent Data Analysis* **6**(5) (2002) 429–450.
14. Sotiris Kotsiantis, Dimitris Kanellopoulos, and Panayiotis Pintelas, Handling imbalanced datasets: A review, *GESTS International Transactions on Computer Science and Engineering* **30** (2006) 25–36.

15. D. Mease, A. J. Wyner and A. Buja, Boosted classification trees and class probability/ quantile estimation, *Journal of Machine Learning Research* **8** (2007) 409–439.
16. N. Mishra, R. Schreiber, I. Stanton, and R. E. Tarjan, Finding strongly-knit clusters in social networks, *Internet Mathematics* **5**(12) (2008) 155–174.
17. A. I. Sánchez, Synthetic instances generation for imbalanced classes, Master of Science Thesis, *National Institute of Astrophysics, Optics, and Electronics* (2008).
18. A. V. Sousa, A. M. Mendonsa, and A. Campilho, The class imbalance problem in TLC image classification, *Image Analysis and Recognition*, LNCS 4142, 2006, pp. 513–523.
19. E. Stamatatos, Text sampling and re-sampling for imbalanced author identification cases, *Proceedings of the 17th European Conference on Artificial Intelligence* (*ECAI'06*), 2006.
20. P. Turney, Types of cost in inductive concept learning, *Proceedings of the ICML'2000 Workshop on Cost Sensitive Learning*, 2000, pp. 15–21.
21. J. Van Hulse, M. T. Khoshgoftaar, and A. Napolitano, Experimental perspectives on learning from imbalanced data, *ICML 2007*, 2007, pp. 935–942.
22. F. Vilariño, P. Spyridonos, J. Vitriá, and P. Radeva, Experiments with SVM and stratified sampling with an imbalanced problem: Detection of intestinal contractions; S. Singh *et al.* (Eds.), *ICAPR*, LNCS 3687, Springer-Verlag, (ISI 0,402), 2005, pp. 783–791.
23. D. R. Wilson and T. R. Martínez, Improved heterogeneous distance functions, *Journal of Artificial Intelligence Research* **6** (1997) 1–34.
24. I. H. Witten and E. Frank, *Data Mining*: *Practical Machine Learning Tools and Techniques*, 2nd edn. (Morgan Kaufmann, 2005).
25. Jun Wang, Xi-Yuan Peng, and Yu Peng, Validity index for clustering with penalizing method, *Third International Symposium on Systems and Control in Aeronautics and Astronautics*, (ISSCAA), 8–10 June, 2010, pp. 706–710.