Knowledge-Based Systems 000 (2018) 1–13

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Imbalance: Oversampling algorithms for imbalanced classification in R

Ignacio Cordón, Salvador García*, Alberto Fernández, Francisco Herrera

DaSCI Andalusian Institute of Data Science and Computational Intelligence, University of Granada, Spain

A R T I C L E I N F O

Article history:
Received 1 March 2018
Revised 14 June 2018
Accepted 25 July 2018
Available online xxx

Keywords:
Oversampling
Imbalanced classification
Machine learning
Preprocessing
SMOTE

A B S T R A C T

Addressing imbalanced datasets in classification tasks is a relevant topic in research studies. The main reason is that for standard classification algorithms, the success rate when identifying minority class instances may be adversely affected. Among different solutions to cope with this problem, data level techniques have shown a robust behavior. In this paper, the novel `imbalance` package is introduced. Written in R and C++, and available at *CRAN* repository, this library includes recent relevant oversampling algorithms to improve the quality of data in imbalanced datasets, prior to performing a learning task. The main features of the package, as well as some illustrative examples of its use are detailed throughout this manuscript.

1. Introduction

The imbalance classification problem is probably one of the most researched problems in the machine learning framework [1–5]. Its classical definition is a binary classification problem where we are given a set of *training instances*, labeled with two possible classes and a set of unlabeled instances, namely *test set*, to classify them using the information provided by the former one. When the size of a class, which usually represents the most important concept to predict, is much lower than the other one, we have an imbalance classification problem.

In these cases, standard classification learning algorithms may be biased toward the majority class examples. In order to address this issue, a common approach is to apply a preprocessing stage for rebalancing the training data. This can be carried out either by undersampling, i.e. removing majority class instances, or oversampling, i.e., introducing new minority class instances. Since undersampling may remove some relevant instances, oversampling is usually preferred. Additionally, this procedure is intended to reinforce the concept represented by the minority class, so that the learning algorithm will be guided to avoid misclassifying these examples.

Plenty of excellent oversampling algorithms arise every day in the scientific literature, but the software is rarely released. To the best of our knowledge, there are just few open source libraries and packages that include methods and techniques related to imbalanced classification.

First, regarding Java we may find a complete module of imbalanced classification in the `KEEL` software suite [6]. It comprises a very complete collection of external and internal approaches, as well as a large number of ensemble methods that work at both levels.

For Python, there exists a very recent tool-box named as `imbalanced-learn` [7]. Similar to `KEEL`, it includes solutions based on preprocessing and ensemble learning.

Finally, for R we may find several packages at *CRAN* which include oversampling and undersampling methods. Specifically, we must refer to `unbalanced` [8], `smotefamily` [9], and `rose` [10,11].

However, there are two main issues associated with the aforementioned software solutions. On the one hand, only `imbalanced-learn` allows a straightforward representation of the preprocessed datasets. This fact is very important in order to acknowledge the actual areas that are reinforced for the minority class examples. On the other hand, and possibly the most significant point, we have observed that none of them contain the latest approaches proposed in the specialized literature.

In this paper, we present a novel, robust and up-to-date R package including preprocessing techniques for the imbalance classification. Named as `imbalance`, it aims to provide both the state-of-the-art methods for oversampling algorithms, as well as some of the most recent techniques which still lack an implementation in the R language. In this sense, we intend to provide a significant contribution to the already available tools that address the same problem.

To present this novel package, the rest of the manuscript is arranged as follows. First, Section 2 presents the soft-

* Corresponding author.
E-mail address: salvagl@decsai.ugr.es (S. García).

https://doi.org/10.1016/j.knosys.2018.07.035
0950-7051/© 2018 Elsevier B.V. All rights reserved.

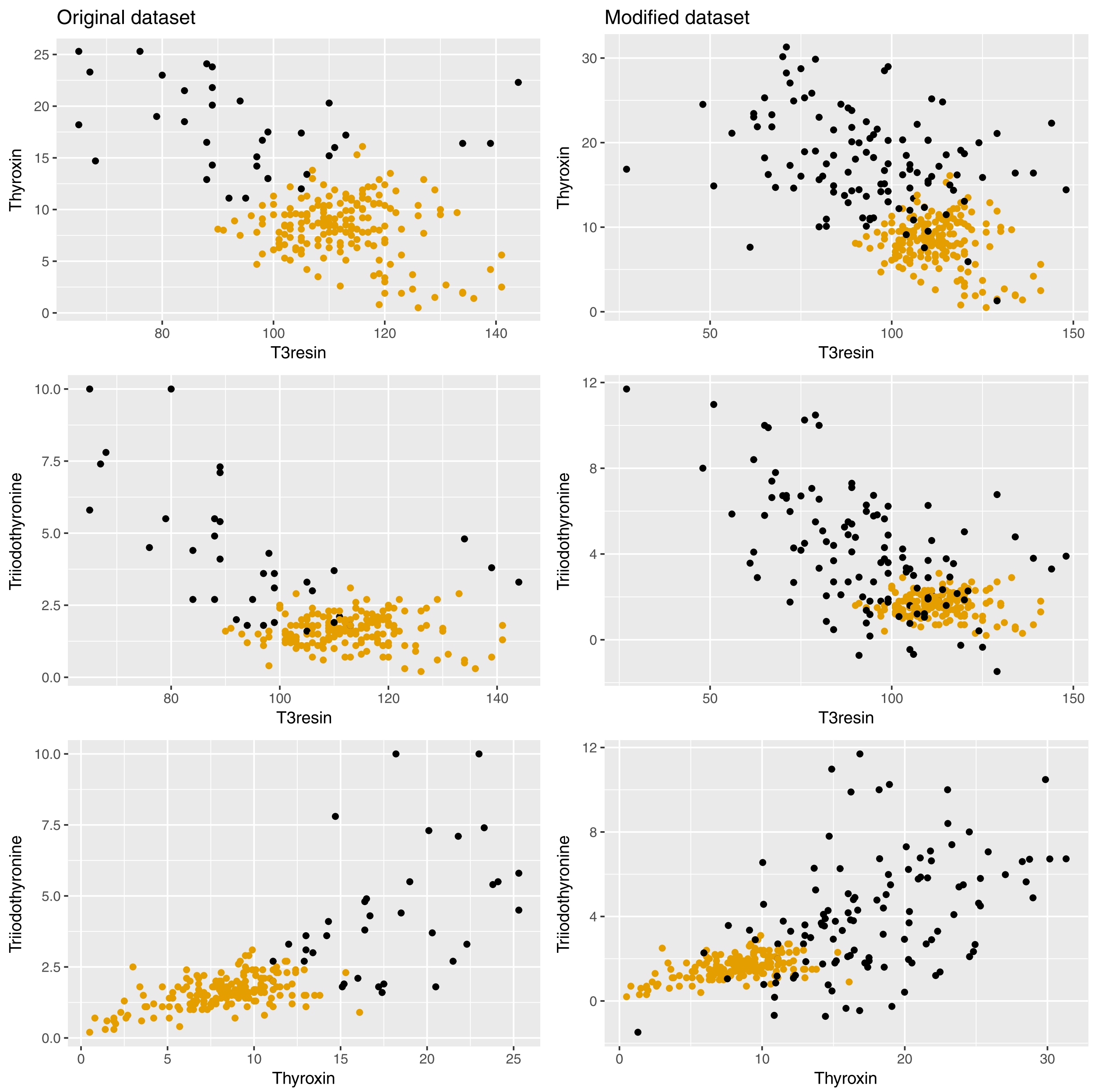

Fig. 1. PDFOS applied to `newthyroid1` dataset.

Table 1
Comparison of the proposed `imbalance` package to the available R packages for imbalanced classification.

Property	`Imbalance`	`Unbalanced`	`Smotefamily`	`Rose`
Version	1.0.0	2.0	1.2	0.0-3
Date	2018-02-18	2015-06-26	2018-01-30	2014-07-15
#Techniques	12	9	6	1
Undersampling	✗	√	✗	✗
Oversampling	√	√	√	√
SMOTE (& var.)	√	√	√	✗
Advanced OverS.	√	✗	✗	✗
Filtering	√	√	✗	✗
Wrapper	√	√	✗	✗
Visualization	√	✗	✗	✗

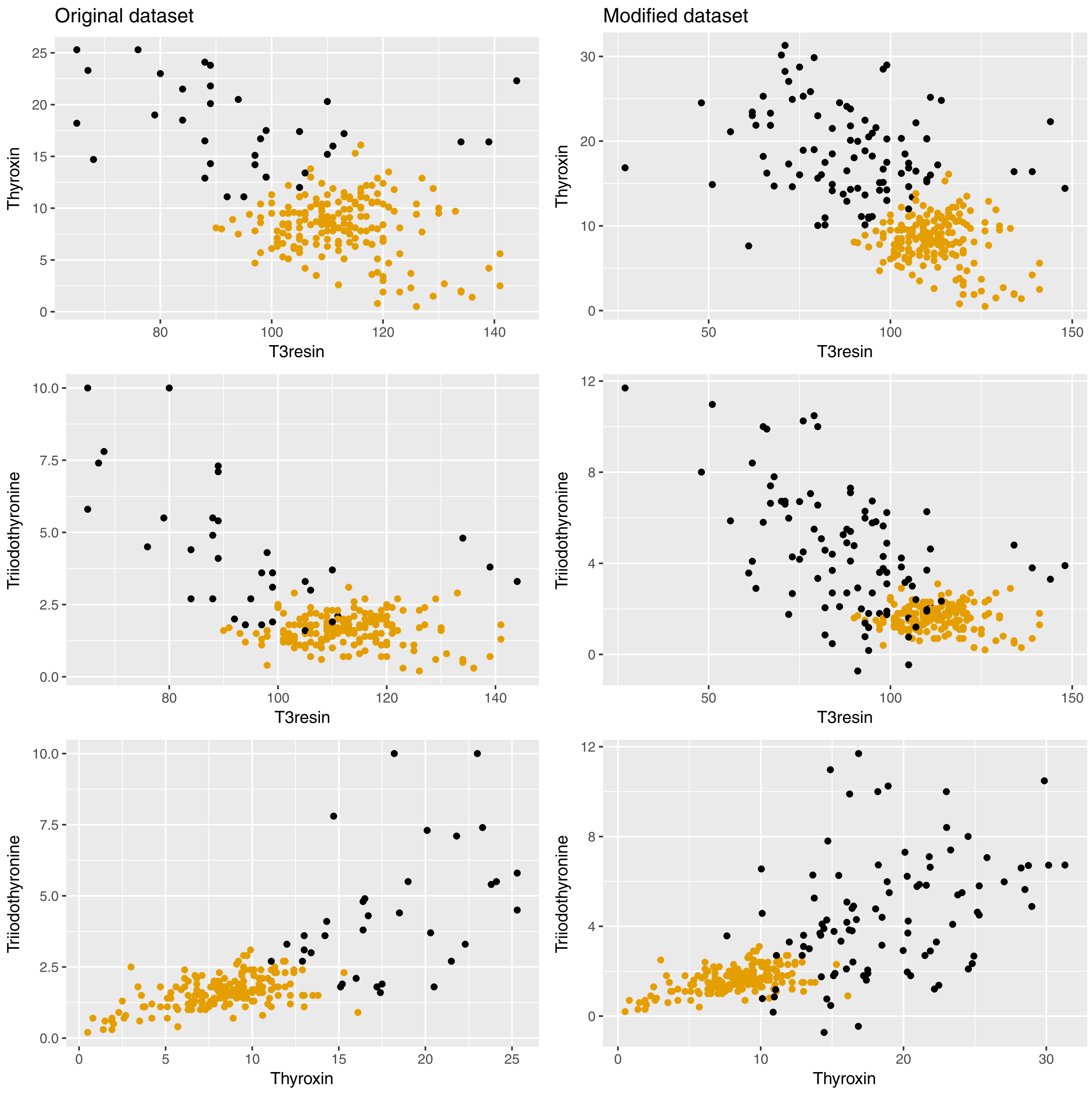

Fig. 2. PDFOS + NEATER applied to `newthyroid1`.

Table 2
Code metadata (mandatory).

Nr.	Code metadata description	Please fill in this column
C1	Current code version	1.0.0
C2	Permanent link to code/repository used of this code version	*github.com/ncordon/imbalance*
C3	Legal Code License	GPL (≥ 2)
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	R ($\geq 3.3.0$), C++
C6	Compilation requirements, operating environments and dependencies	Rcpp, bnlearn, KernelKnn, ggplot2, mvtnorm
C7	If available Link to developer documentation/manual	*ncordon.github.io/imbalance*
C8	Support email for questions	*nacho.cordon.castillo@gmail.com*

ware framework and enumerates the implemented algorithms. Then, Section 3 shows some illustrative examples. Finally, Section 4 presents the conclusions.

2. Software

The significance of oversampling techniques in imbalanced classification is beyond all doubt. The main reason is related to a smart generation of new artificial minority samples in those areas that need reinforcement for the learning of class-fair classifiers.

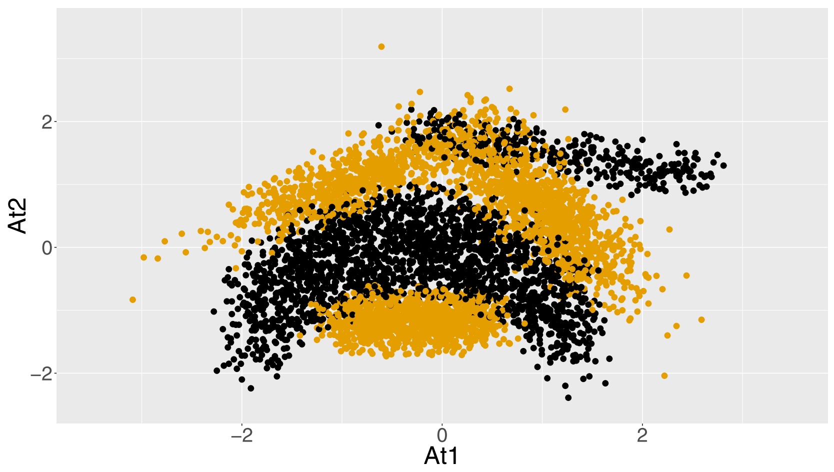

Fig. 3. Original `banana` dataset.

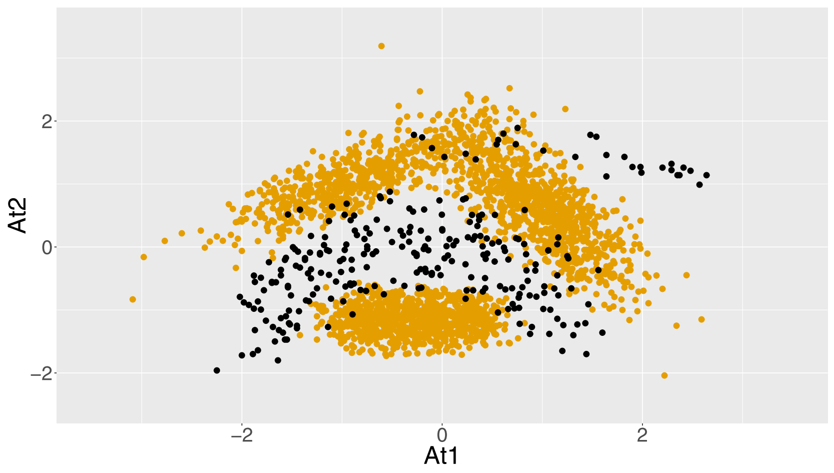

Fig. 4. Imbalanced `banana` dataset.

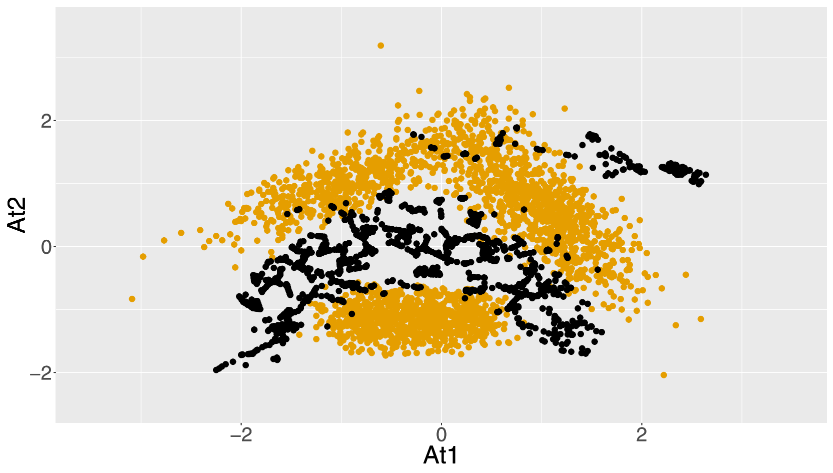

Fig. 5. SMOTE applied to imbalanced `banana`.

Since the original SMOTE algorithm in 2002 [12], many different approaches have been designed to improve the classification performance under different scenarios [13,14]. In this new developed software package, we have compiled some of the newest oversampling algorithms, which are listed below:

- `mwmote`. The *Majority Weighted Minority Oversampling Technique* (MWMOTE), first proposed in [15], is an extension of the original SMOTE algorithm [12,13]. It assigns higher weight to borderline instances, undersized minority clusters and examples near the borderline of the two classes.
- `racog`, `wracog`. *Rapidly Converging Gibbs* (RACOG) and *wrapper-based RACOG* (wRACOG), both proposed by [16], work for discrete attributes. They generate new examples with respect to an approximated distribution using a Gibbs Sampler scheme. RACOG needs the number of instances to generate be-

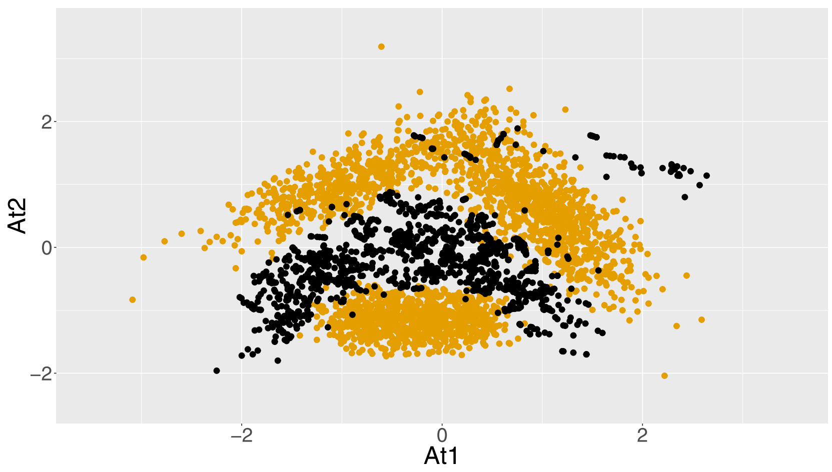

Fig. 6. MWMOTE applied to imbalanced `banana`.

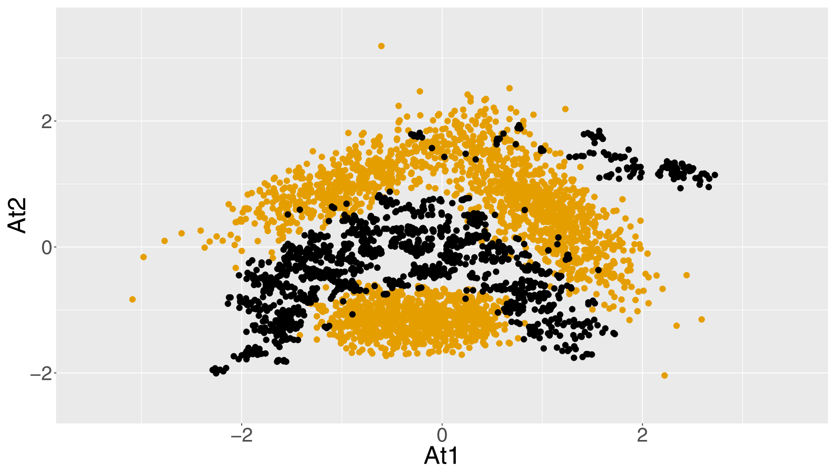

Fig. 7. RWO applied to imbalanced `banana`.

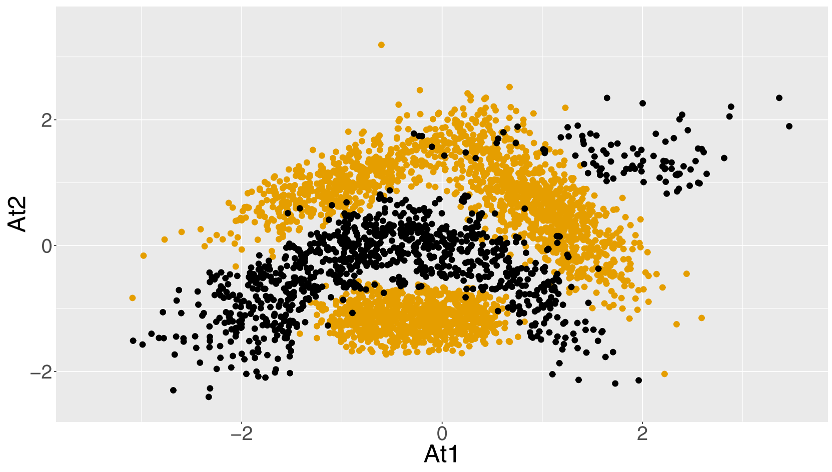

Fig. 8. PDFOS applied to imbalanced `banana`.

forehand. wRACOG requires a target classifier to show no improvement to stop generating examples.

- `rwo`. *Random Walk Oversampling* (RWO) is an algorithm introduced by [17], which generates synthetic instances so that mean and deviation of numerical attributes remain close to the original ones.
- `pdfos`. *Probability Distribution density Function estimation based Oversampling* (PDFOS) was proposed in [18]. It uses multivariate Gaussian kernel methods to locally approximate the minority class.

Apart from those oversampling methods, we provide a filtering method called `neater`. The *filteriNg of ovErsampled dAta using non cooperaTive gamE theoRy* (NEATER), introduced in [19], is highly based on game theory. It discards the instances with higher probability of belonging to the opposite class, based on each instance neighborhood.

The package also includes the method `oversample`, which is a *wrapper* that eases calls to the described and already existing methods.

To evaluate the oversampling process, we propose a visual method, called `plotComparison`. It plots a pairwise comparative grid of a selected set of attributes, both in the original dataset and the oversampled one. That way, if a proper oversampling has been performed, we expect to see larger minority clusters in the resulting dataset.

In addition to this, `imbalance` includes some datasets from the KEEL [6,20] repository (http://www.keel.es/datasets.php), which can be used to perform experiments. Additional datasets can be easily imported under a single constraint: they must contain a class column (not necessarily the last one) having two different values.

To conclude this section, we show in Table 1 a comparison of the main features of this novel `imbalance` package with respect to the previous software solutions available at CRAN: `unbalanced` [8], `smotefamily` [9], and `rose` [10,11]. Among the properties that are contrasted, we included the latest date of release, the number of preprocessing techniques that are available, and if it includes different approaches, namely undersampling, oversampling, SMOTE (and variants/extensions), advanced oversampling (techniques beyond SMOTE), and filtering methods. We also show whether an automatic wrapper procedure is available, and if it includes a visualization of the preprocessed output.

From Table 1 we may conclude that `imbalance` is a complete solution with many relevant oversampling techniques. It is by far the one with the largest number of approaches, and the only one that includes oversampling approaches beyond the traditional SMOTE scheme. Finally, we must stress the relevance of the visualization feature, which can be very useful for practitioners to check the areas where the minority class is mainly reinforced (Table 2).

3. Examples of use

The following example loads the dataset `newthyroid1`, included in our package, and applies the algorithm PDFOS to the dataset, requesting 80 new instances. `newthyroid1` is a classical dataset that has a series of 5 different medical measurements as attributes, and classifies every patient in hyperthyroidism (42 instances) or non-hyperthyroidism (173 instances). Once the algorithm has been applied, we plot a pairwise visual comparison between first three attributes of the original and modified datasets. Result can be observed in Fig. 1, and in Fig. 2, after applying filtering.

```
# Load the previously installed imbalance package
library("imbalance")
# Load the dataset newthyroid1, included in imbalance
data(newthyroid1)

# Compute the imbalance ratio of newthyroid1 (that is,
# proportion of minority examples with respect to majority
   ones)
imbalanceRatio(newthyroid1)
# 0.1944444


# Generate 80 new minority instances for newthyroid1 using
# pdfos algorithm
newSamples <- pdfos(
    newthyroid1,
    numInstances = 80
)

# Add the new samples to the original newthyroid1 dataset and
# asssign it to a newDataset variable
newDataset <- rbind(
    newthyroid1,
    newSamples
)

# Compare the three first variables of the extended dataset
# and the original one
plotComparison(
    newthyroid1,
    newDataset,
    attrs = names(newthyroid1)[1:3]
)

# Filter synthetic examples from newSamples, so that
# only relevant ones remain in the dataset
filteredSamples <- neater(
    newthyroid1,
    newSamples,
    iterations = 500
)

# Add the new filtered samples to the original dataset
filteredNewDataset <- rbind(
    newthyroid1,
    filteredSamples
)

# Compare the three first variables of the extended filtered
# dataset and the original one
plotComparison(
    newthyroid1,
    filteredNewDataset,
    attrs = names(newthyroid1)[1:3]
)
```

The banana dataset, which has been included in the package, is a binary dataset with 5300 samples and two attributes (apart from class attribute), artificially developed to represent a banana when plotted in two dimensions with each class filled with a different color. A straightforward *Random Undersampling* has yield an imbalance dataset (with 10% imbalance ratio) from the original dataset, both observable in Figs. 3 and 4, respectively. Original dataset has been included as banana-orig in the package; imbalanced one, as banana. We provide a visual comparison between results of applying an oversample to reach 50% of imbalance ratio in banana, and a later filtering using NEATER. Applied oversampling techniques are SMOTE (Fig. 5), MWMOTE (Fig. 6), RWO (Fig. 7) and PDFOS (Fig. 8).

Algorithm 1 MWMOTE oversampling.

```
Require: $S^+ = \{x_1, \ldots, x_m\}$, minority instances
Require: $S^- = \{y_1, \ldots, y_m\}$, majority instances
Require: $T$, requested number of synthetic examples
Require: $k_1$, KNN parameter to filter noisy instances of $S^+$
Require: $k_2$, KNN parameter to compute boundary $U \subseteq S^-$
Require: $K_3$, KNN parameter to compute boundary $V \subseteq S^+$
Require: $\alpha$, tolerance for the closeness level to the borderline
Require: $C$, weight of the closeness factor to the borderline
Require: $C_{clust}$
 1: Initialize $S' = \emptyset$
 2: For each $x \in S^+$, compute its $k_1$ KNN neighbourhood, $NN^{k_1}(x)$
 3: Let $S^+_f = S^+ - \{x \in S^+ : NN^{k_1}(x) \cap S^+ = \emptyset\}$
 4: Compute $U = \bigcup_{x \in S^+_f} NN^{k_2}_-(x)$
 5: Compute $V = \bigcup_{x \in U} NN^{k_3}_+(x)$
 6: For each $x \in V$, compute $P(x) = \sum_{y \in U} I_{\alpha,C}(x, y)$
 7: Normalize $P(x)$ for each $x \in V$, $P(x) = \frac{P(x)}{\sum_{z \in V} P(z)}$
 8: Compute
    $$T_{clust} = C_{clust} \cdot \frac{1}{|S^+_f|} \sum_{x \in S^+_f} \min_{y \in S^+_f, y \neq x} d(x, y)$$
 9: Let $L_1, \ldots, L_M \subseteq S^+$ be the clusters for $S^+$, with $T_{clust}$ as threshold
10:
11: for $t = 1, \ldots, T$ do
12:     Pick $x \in V$ with respect to $P(x)$
13:     Uniformly pick $y \in L_e$ inside $L_e \ni x$, where $x \in L_e$
14:     Uniformly pick $r \in [0, 1]$
15:     $S' = S' \cup \{x + r(y - x)\}$
16: end for
17:
18: return $S'$, synthetic examples
```

4. Conclusions

Class imbalance in datasets is one of the most decisive factors to take into account when performing classification tasks. To improve the behavior of classification algorithms in imbalanced domains, one of the most common and effective approaches is to apply oversampling as a preprocessing approach. It works by creating synthetic minority instances to increase the number of representatives belonging to that class.

In this paper we have presented the imbalance package for R. It was intended to alleviate some drawbacks that arise in current software solutions. Firstly, to provide useful implementations for those novel oversampling methods that were not yet available

Algorithm 2 RACOG oversampling.

```
Require: $S = \{x_1, \ldots, x_m\}$, positive examples
Require: $\beta$, burnin
Require: $\alpha$, lag
Require: $T$, requested number of synthetic examples
 1: $P =$ approximation of the $S$ distribution
 2: $S' = \emptyset$
 3: $M = \left\lceil \frac{T}{m} \right\rceil \cdot \alpha + \beta$
 4:
 5: for $t = 1, \ldots, M$ do
 6:     $S = \mathrm{GibbsSampler}(S, P)$
 7:     if $t > \beta$ and $t \mod (\alpha) = 0$ then
 8:         $S' = S' \cup S$
 9:     end if
10: end for
11:
12: $S' =$ Pick $T$ random instances from $S'$
13: return $S'$, synthetic examples
```

Algorithm 3 wRACOG oversampling.

```
Require: $S_{train} = \{z_i = (x_i, y_i)\}_{i=1}^m$, train instances
Require: $S_{val}$, validation instances
Require: wrapper
Require: $T$, requested number of synthetic examples
Require: $\alpha$, tolerance parameter
 1: $S = S^+_{train}$
 2: $P =$ approximation of the $S$ distribution
 3: Build a model with wrapper and $S_{train}$
 4: Initialize $S' = \emptyset$
 5: Initialize $\tau = (\underset{1)}{+\infty}, \ldots, \underset{T)}{+\infty})$
 6:
 7: while The standard deviation of $\tau \geq \alpha$ do
 8:     $S = \mathrm{GibbsSampler}(S, P)$
 9:     $S \supseteq S_{misc} =$ misclassified examples by model
10:     Update $S' = S' \cup S_{misc}$
11:     Update train set $S_{train} = S_{train} \cup S_{misc}$
12:     Build new model with wrapper, $S_{train}$
13:     Let $s =$ sensitivity of model over $S_{val}$
14:     Let $\tau = (\tau_2, \ldots, \tau_T, s)$
15: end while
16:
17: return $S'$, synthetic examples
```

for researchers and practitioners. Secondly, to include a visualization environment for the sake of observing those areas of the minority class that are actually reinforced by means of preprocessing. Finally, to enable a simpler integration of these methods with the existing oversampling packages at CRAN.

As future work, we propose to keep maintaining and adding functionality to our new imbalance package. Specifically, we plan to include those new oversampling techniques that are regularly proposed in the specialized literature. In this sense, we consider that there are good prospects to improve the software in the near future.

Acknowledgments

This work is supported by the Project BigDaP-TOOLS - Ayudas Fundación BBVA a Equipos de Investigación Científica 2016.

Algorithm 4 RWO oversampling.

```
Require: $S = \{x_i = (w_1^{(i)}, \dots w_d^{(i)})\}_{i=1}^m$, positive instances
Require: $T$, required number of instances
 1: Initialize $S' = \emptyset$
 2:
 3: for For each $j = 1, \dots, d$ do
 4:     if $j$–ith attribute is numerical then
 5:         $\sigma'_j = \sqrt{\frac{1}{m}\sum_{i=1}^m \left(w_j^{(i)} - \frac{\sum_{i=1}^m w_j^{(i)}}{m}\right)^2}$
 6:     end if
 7: end for
 8:
 9: Assign $M = \lceil T/m \rceil$
10: for $t = 1, \dots, M$ do
11:     for $i = 1, \dots, m$ do
12:         for $j = 1, \dots, d$ do
13:             if $j$–th attribute is numerical then
14:                 Choose $r \sim N(0, 1)$
15:                 $w_j = w_j^{(i)} - \frac{\sigma'_j}{\sqrt{m}} \cdot r$
16:             else
17:                 Choose $w_j$ uniformly over $\{w_j^{(1)}, \dots w_j^{(m)}\}$
18:             end if
19:         end for
20:         $S' = S' \cup \{(w_1, \dots, w_d)\}$
21:     end for
22: end for
23:
24: $S'$ = Choose $T$ random instances from $S'$
25: return $S'$, synthetic positive instances
```

Algorithm 5 PDFOS oversampling.

```
Require: $S = \{x_i = (w_1^{(i)}, \dots w_d^{(i)})\}_{i=1}^m$, positive instances
Require: $T$, required number of instances
 1: Initialize $S' = \emptyset$
 2: Search for $h =$ which minimizes $M(h)$
 3: Find $U$ unbiased covariance matrix of $S$
 4: Compute $U = R \cdot R^T$ with Choleski decomposition
 5:
 6: for $i = 1, \dots, T$ do
 7:     Choose $x \in S$
 8:     Pick $r$ with respect to a normal distribution, i.e. $r \sim N^d(0, 1)$
 9:     $S' = S' \cup \{x + hrR\}$
10: end for
11:
12: return $S'$, synthetic positive instances
```

Appendix A. Installation

To install our package, R language is needed (see https://www.r-project.org/ for further indications on how to install it). Once R language is properly installed, it suffices to do, in an R interpreter:

```
install.packages("imbalance")
```

This command will install the latest version of the package directly from CRAN, which is the official repository for R packages.

Appendix B. Description of the algorithms in the package

Hereafter, the new preprocessing techniques included in the `imbalance` package will be further described. To do so, Section B.1 first provides a short introduction on classification task.

Algorithm 6 NEATER filtering.

```
Require: $S = \{z_1 = (x_1, y_1), \dots z_n = (x_n, y_n)\}$, original dataset
Require: $S' = \{\bar{z}_1 = (\bar{x}_1, \bar{y}_1), \dots \bar{z}_m = (\bar{x}_m, \bar{y}_m)\}$, positive instances
Require: $k$, number of KNN neighbours.
Require: $T$, required number of iterations.
Require: $\alpha$, smooth factor.
 1: Initialize $E = \emptyset$
 2: For each $x_i \in S'$, compute its neighbourhood $NN^k(x_i) \subseteq S \cup S'$
 3: For $i = 1, \dots, n$ initialize $\delta_i = (1, 0)$ if $y_i = 1$ and $\delta_i = (0, 1)$ if $y_i = -1$
 4: For $i = n + 1, \dots, n + m$ do $\delta_i = (0.5, 0.5)$
 5:
 6: for $t = 1, \dots, T$ do
 7:     for $i = 1, \dots, m$ do
 8:         Compute total payoff:
                $u_i = \sum_{x_j \in NN^k(x_i)} g(d(\bar{x}_i, x_j)) \cdot \delta_i \cdot \delta_j^T$
 9:         Compute positive payoff:
                $u = \sum_{x_j \in NN^k(x_i)} g(d(\bar{x}_i, x_j)) \cdot (1, 0) \cdot \delta_j^T$
10:         Assign $\alpha = (\alpha + u)/(\alpha + u_{n+1})$
11:         Update $\delta_{n+1} = (\alpha, 1 - \alpha)$
12:     end for
13: end for
14:
15: for $i = 1, \dots, m$ do
16:     if $\delta_{i1} > 0.5$ then
17:         $E = E \cup \{(\bar{x}_i, 1)\}$
18:     end if
19: end for
20:
21: return $E \subseteq S'$, filtered synthetic positive instances
```

Then, Sections B.2–B.6 include the main characteristics of the algorithms together with an explanation of the upsides and downsides of every oversampling approach.

B1. Classification task

Classification problem is one of the best known problems in the machine learning framework. We are given a set of *training instances*, namely, $S = \{(x_1, y_1), \dots, (x_m, y_m)\}$ where $x_i \in X \subset \mathbb{R}^n$ and $y_i \in Y$, with $Y = \{0, 1\}$. X and Y will be called *domain* and *label set*, respectively. The *training set* will be considered as independent and identically distributed (i.i.d.) samples taken with respect to an unknown probability $\mathcal{P}$ over the set $X \times Y$, denoting that as $S \sim \mathcal{P}^m$.

A *test set* will be a set of i.i.d. instances $T = \{(\bar{x}_1, \bar{y}_1), \dots, (\bar{x}_k, \bar{y}_k)\}$, $(\bar{x}_i, \bar{y}_i) \in X \times Y$, taken with respect to $\mathcal{P}$. We denote $T_x := \{\bar{x}_1, \dots, \bar{x}_k\}$. A classifier h_S (depending on the training set S) is a function th at takes an arbitrary *test set*, lacking the labels $\bar{y}_i$, and outputs labels for each instance. That is, $h_S(T_x) = \{(\bar{x}_1, \bar{\bar{y}}_1), \dots, (\bar{x}_k, \bar{\bar{y}}_k)\}$. The aim of the classification problem is to find the classifier that minimizes the labeling error $L_{\mathcal{P}}$ with respect to the true distribution of the data, defined as follows:

$$L_{\mathcal{P}}(h_S) := \mathop{\mathbb{E}}_{\{(x,y)\}\sim\mathcal{P}} \mathbb{1}[h_S(\{x\}) \neq \{(x,y)\}]$$

Where $\mathbb{1}[\texttt{Condition}]$ returns 1 if Condition holds, and 0 otherwise. Hence, $L_{\mathcal{P}}$ represents an average of the error over the domain instances, weighted by the probability of extracting those instances.

Since we do not know the true distribution of the data, we will usually approximate $L_{\mathcal{P}}(h_S)$ using the average error over a *test set* $T = \{(\bar{x}_1, \bar{y}_1), \ldots, (\bar{x}_k, \bar{y}_k)\}$:

$$L(h_S) = \sum_{i=1}^{k} \mathbb{1}[h_S(\{\bar{x}_i,\}) \neq \{(\bar{x}_i, \bar{y}_i)\}$$

Specifically, `imbalance` package provides oversampling algorithms. Those family of procedures aim to generate a set E of synthetic positive instances based on the training ones, so that we have a new classification problem with $\bar{S}^+ = S^+ \cup E$, $\bar{S}^- = S^-$ and $\bar{S} = \bar{S}^+ \cup \bar{S}^-$ our new training set.

B2. MWMOTE

This algorithm, proposed by [15], is one of the many modifications of SMOTE [12], which is a classic algorithm to treat class imbalance. SMOTE generates new examples by filling empty areas among the positive instances. It updates the training set iteratively, by performing:

$$E := E \cup \{x + r \cdot (y - x)\}, \quad x, y \in S^+, r \sim N(0,1)$$

But SMOTE has a clear downside: it does not detect noisy instances. Therefore, it can generate synthetic examples out of noisy ones or even between two minority classes, which if not cleansed up, may end up becoming noise inside a majority class cluster. MWMOTE (*Majority Weighted Minority Oversampling Technique*) tries to overcome both problems. It intends to give higher weight to borderline instances, undersized minority clusters and examples near the borderline of the two clases.

Let's introduce some notations and definitions:

- $d(x, y)$ stands for the euclidean distance between x and y.
- $NN^k(x) \subseteq S$ will be the k-neighbourhood of x in S (k closest instances with euclidean distance).
- $NN_i^k(x) \subseteq S^i$, $i = +, -$ will be x's k- minority (resp. majority) neighbourhood.
- $C_f(x, y) = \frac{C}{\alpha} \cdot f\left(\frac{d}{d(x,y)}\right)$ measures the closeness of x to y, that is, it will measure the proximity of borderline instances. where $f = x\mathbb{1}_{[x \le \alpha]} + C\mathbb{1}_{[x > \alpha]}$
- $D_f(x, y) = \frac{C_f(x,y)}{\sum_{z \in V} C_f(z,y)}$ will represent a density factor, such that an instance belonging to a compact cluster will have higher $\Sigma C_f(z, y)$ than another one belonging to a more sparse cluster.
- $I_{\alpha,C}(x, y) = C_f(x, y) \cdot D_f(x, y)$, where if $x \notin NN_+^{k_3}(y)$ then $I_{\alpha,C}(x, y) = 0$.

Let $T_{clust} := C_{clust} \cdot \frac{1}{|S_f^+|} \sum_{x \in S_f^+} \min_{y \in S_f^+, y \neq x} d(x, y)$. We will also use a mean-average agglomerative hierarchical clustering of the minority instances with threshold T_{clust}, that is, we will use a mean distance:

$$dist(L_i, L_j) = \frac{1}{|L_i||L_j|} \sum_{x \in L_i} \sum_{y \in L_j} d(x, y)$$

and having started with a cluster per instance, we will proceed by joining nearest clusters until minimum of distances is lower than T_{clust}.

A few interesting considerations:

- Low k_2 is required in order to ensure we do not pick too many negative instances in U.
- For an opposite reason, a high k_3 must be selected to ensure we pick as many positive hard-to-learn borderline examples as we can.
- The higher the C_{clust} parameter, the less and more-populated clusters we will get.

B2.1. Pros and cons

The most evident gain of this algorithm is that it fixes some of the weaknesses of SMOTE. And SMOTE is still one of the main references that researches use as a benchmark to compare their algorithms. That makes a MWMOTE a state-of-the-art algorithm. Apart from that, and although the pseudocode can be quite confusing, the idea behind the algorithm is easy to understand.

On the other hand, the algorithm relies on the idea that the space between two minority instances is going to belong to a minority cluster, which seems like a reasonable hypothesis, but can lead to error in certain datasets (e.g., the minority class spread across a large number of tiny clusters).

B3. RACOG and wRACOG

These set of algorithms, proposed in [16], assume we want to approximate a discrete distribution $P(W_1, \ldots, W_d)$.

The key of the algorithm is to approximate $P(W_1, \ldots, W_d)$ as $\prod_{i=1}^{d} P(W_i \mid W_{n(i)})$ where $n(i) \in \{1, \ldots, d\}$. Chow–Liu's algorithm is used to meet that purpose. This algorithm minimizes Kullback–Leibler distance between two distributions:

$$D_{KL}(P \parallel Q) = \sum_i P(i)(\log P(i) - \log Q(i))$$

We recall the definition for the mutual information of two random discrete variables W_i, W_j:

$$I(W_i, W_j) = \sum_{w_1 \in W_1} \sum_{w_2 \in W_2} p(w_1, w_2) \log\left(\frac{p(w_1, w_2)}{p(w_1)p(w_2)}\right)$$

Let $S^+ = \{x_i = (w_1^{(i)}, \ldots, w_d^{(i)})\}_{i=1}^m$ be the unlabeled positive instances. To approximate the distribution, we do:

- Compute $G' = (E', V')$, Chow Liu's dependence tree.
- If r is the root of the tree, define $P(W_r|W_{n(r)}) := P(W_r)$.
- For each $(u, v) \in E$ arc in the tree, $n(v) := u$ and compute $P(W_v|W_{n(v)})$.

After that, a Gibbs Sampling scheme is used to extract samples with respect to the approximated probability distribution, where a badge of new instances is obtained by performing:

- Given a minority sample $x_k = (w_1^{(i)}, \ldots, w_d^{(i)})$
- Iteratively construct for each attribute

  $$\bar{w}_k^{(i)} \sim P(W_k \mid \bar{w}_1^{(i)}, \ldots, \bar{w}_{k-1}^{(i)}, w_{k+1}^{(i)}, \ldots, w_d^{(i)})$$

- Return $S = \{\bar{x}_i = (\bar{w}_1^{(i)}, \ldots, \bar{w}_d^{(i)})\}_{i=1}^m$.

B3.1. RACOG

RACOG (*Rapidly Converging Gibbs*) builds a Markov chain for each of the m minority instances, ruling out the first β generated instances and selecting a badge of synthetic examples each α iterations. That allows to lose dependence of previous values.

B3.2. wRACOG

RACOG depends on α, β and the requested number of instances. wRACOG (*wrapper-based RACOG*) tries to overcome that problem. Let *wrapper* be a binary classifier.

B3.3. Pros and cons

Clearly, RACOG and wRACOG have the advantage that they are highly based on statistical evidence/procedures, and they have guarantees to succeed in their goal. On the contrary, there exists a substantial downside of those algorithms: they only work on discrete variables, which makes them very restrictive with respect to the set of data they can be applied to.

B4. RWO

RWO (*Random Walk Oversampling*) is an algorithm introduced by [17], which generates synthetic instances so that mean and deviation of numerical attributes remain as close as possible to the original ones.

This algorithm is motivated by the central limit theorem, which states that given a collection of independent and identically distributed random variables, $W_1, \ldots, W_m$, with $\mathbb{E}(W_i) = \mu$ and $Var(W_i) = \sigma^2 < \infty$, then:

$$\lim_m P\left[\frac{\sqrt{m}}{\sigma}\left(\underbrace{\frac{1}{m}\sum_{i=1}^m W_i}_{\overline{W}} - \mu\right) \le z\right] = \phi(z)$$

where ϕ is the distribution function of $N(0, 1)$.

That is, $\frac{\overline{W}-\mu}{\sigma/\sqrt{m}} \to N(0,1)$ probability-wise.

Let $S^+ = \{x_i = (w_1^{(i)}, \ldots, w_d^{(i)})\}_{i=1}^m$ be the minority instances. Now, let's fix some $j \in \{1, \ldots, d\}$, and let's assume that j-ith column follows a numerical random variable W_j, with mean μ_j and standard deviation $\sigma_j < \infty$. Let's compute $\sigma_j' = \sqrt{\frac{1}{m}\sum_{i=1}^m \left(w_j^{(i)} - \frac{\sum_{i=1}^m w_j^{(i)}}{m}\right)^2}$ the biased estimator for the standard deviation. It can be proven that instances generated with $\bar{w}_j = w_j^{(i)} - \frac{\sigma_j'}{\sqrt{m}} \cdot r, r \sim N(0,1)$ have the same sample mean as the original ones, and their sample variance tends to the original one.

B4.1. Pros and cons

RWO, as it was originally described in [17], uses the sample variance, instead of the unbiased sample variance. Therefore we only have guarantees that $\mathbb{E}(\sigma_j'^2) \xrightarrow[m\to\infty]{} \sigma_j^2$. If we had picked $\tau_j = \frac{1}{m-1}\sum_{i=1}^m \left(w_j^{(i)} - \frac{1}{m}\sum_{i=1}^m w_j^{(i)}\right)^2$, instead, $\mathbb{E}(\tau_j) = \sigma_j^2$ would hold. Another downside of the algorithm is its arbitrariness when it comes to non-numerical variables.

The most obvious upside of this algorithm is its simplicity and its good practical results.

B5. PDFOS

Due to the complexity of this preprocessing technique, in this case we will structure the description of its working procedure into several subsections, providing the motivation, background techniques, and finally pros and cons.

B5.1. Motivation

Given a distribution function of a random variable X, namely $F(x)$, if that function has an almost everywhere derivative, then, almost everywhere, it holds:

$$f(x) = F'(x) = \lim_{h\to 0} \frac{P(x-h < X \le x+h)}{2h}$$

Given a fixed h, that we will call *bandwidth* henceforth. Given a collection of random samples $X, X_1, \ldots, X_n$, namely $x_1, \ldots, x_n$, then an estimator for f could be the mean number of samples in $]x-h, x+h[$ (let's call this number $I_h(x_1, \ldots, x_n)$) divided by the length of the interval:

$$\widehat{f}(x) = \frac{I_h(x_1, \ldots, x_n)}{2hn}$$

If we define $\omega(x) = \begin{cases} \frac{1}{2} & |x| < 1 \\ 0 & \text{otherwise} \end{cases}$ and $w_h(x) = w\left(\left|\frac{x}{h}\right|\right)$, then we could write $\widehat{f}$ as:

$$\widehat{f}(x) = \frac{1}{nh}\sum_{i=1}^n \omega_h(x - x_i)$$

In d dimensional case, we define:

$$\widehat{f}(x) = \frac{1}{nh^d}\sum_{i=1}^n \omega_h(x - x_i)$$

B5.2. Kernel methods

If we took $w = \frac{1}{2}1_{]-1,1[}$, then $\widehat{f}$ would have jump discontinuities and we would have jump derivatives. On the other hand, we could took ω, where $w \ge 0$, $\int_\Omega \omega(x)dx = 1$, $\Omega \subseteq X$ a domain, and w even, and that way we could have estimators with more desirable properties with respect to continuity and differentiability.

$\widehat{f}$ can be evaluated through its MISE (*Mean Integral Squared Error*):

$$MISE(h) = \mathop{\mathbb{E}}_{x_1,\ldots,x_d} \int (\widehat{f}(x) - f(x))^2 dx$$

B5.3. The algorithm

PDFOS (*Probability Distribution density Function estimation based Oversampling*) was proposed in [18]. It uses multivariate Gaussian kernel methods. The probability density function of a d-Gaussian distribution with mean 0 and Ψ as its covariance matrix is:

$$\phi^\Psi(x) = \frac{1}{\sqrt{(2\pi \cdot det(\Psi))^d}} exp\left(-\frac{1}{2}x\Psi^{-1}x^T\right)$$

Let $S^+ = \{x_i = (w_1^{(i)}, \ldots, w_d^{(i)})\}_{i=1}^m$ be the minority instances. The unbiased covariance estimator is:

$$U = \frac{1}{m-1}\sum_{i=1}^m (x_i - \overline{x})(x_i - \overline{x})^T, \text{where } \overline{x} = \frac{1}{m}\sum_{i=1}^m x_i$$

We will use kernel functions $\phi_h(x) = \phi^U\left(\frac{x}{h}\right)$, where h needs to be optimized to minimize the MISE. We will pursue to minimize the following cross-validation function:

$$M(h) = \frac{1}{m^2 h^d}\sum_{i=1}^m \sum_{j=1}^m \phi_h^*(x_i - x_j) + \frac{2}{mh^d}\phi_h(0)$$

where $\phi_h^* \approx \phi_{h\sqrt{2}} - 2\phi_h$.

Once a proper h has been found, a suitable generating scheme could be to take $x_i + hRr$, where $x_i \in S^+$, $r \sim N^d(0, 1)$ and $U = R \cdot R^T$, R being upper-triangular. In case we have enough guarantees to decompose $U = R \cdot R^T$ (U must be a positive-definite matrix), we could use Choleski decomposition. In fact, we provide a sketch of proof showing that all covariance matrices are positive-semidefinite:

$$y^T\left(\sum_{i=1}^m (x_i - \overline{x})(x_i - \overline{x})^T\right)y$$

$$= \sum_{i=1}^m (\underbrace{(x_i - \overline{x})^T y)^T}_{z_i^T} \underbrace{(x_i - \overline{x})^T y)}_{z_i} = \sum_{i=1}^m ||z_i||^2 \ge 0$$

for arbitrary $y \in \mathbb{R}^d$. Hence, we need a strict positive definite matrix, otherwise PDFOS would not provide a result and will stop its execution.

B5.4. Search of optimal bandwidth

We take a first approximation to h as the value:

$$h_{Silverman} = \left(\frac{4}{m(d+2)}\right)^{\frac{1}{d+4}}$$

where d is number of attributes and m the size of the minority class.

Reshaping the equation of the cross validation function and differentiating:

$$M(h) = \frac{2}{m^2 h^d}\sum_{j>i}^{m}\phi_h^*(x_i - x_j) + \frac{1}{mh^d}\phi_{h\sqrt{2}}(0)$$

$$\begin{aligned} M'(h) = &-\frac{dh^{-1}}{mh^d}\phi_{h\sqrt{2}}(0) \\ &- \frac{2}{m^2 h^d}\Bigg[\sum_{j>i}^{m}\phi_h^*(x_i - x_j)dh^{-1} \\ &+ \sum_{j>i}^{m}\phi_h^*(x_i - x_j)h^{-3}(x_i - x_j)^T U(x_i - x_j)\Bigg] \end{aligned}$$

And we use a straightforward gradient descent algorithm to find a good h estimation.

B5.5. Pros and cons

On the one hand, PDFOS makes the assumption that the data can be locally approximated by a normal distribution. What is more, it makes the assumption that the same bandwidth gives good local results in every single point. Another disadvantage of the algorithm is that not every single covariance matrix has a Choleski decomposition (a covariance matrix can be shown to be positive-semidefinite, whereas for the Choleski decomposition to exist it needs to be a positive-definite matrix).

On the other hand, although it makes some hypothesis, they are mild assumptions compared to the results it yields. It also has an enormous theoretical component, which ensures quality results.

B6. NEATER

Once we have created synthetic examples, we should ask ourselves how many of those instances are in fact relevant to our problem. Filtering algorithms can be applied to oversampled datasets, to erase the least relevant instances.

B6.1. Game theory

Let (P, T, f) be our game space. We would have a set of players, $P = \{1, \ldots, n\}$, and $T_i = \{1, \ldots, k_i\}$, set of feasible strategies for the ith player, resulting in $T = T_1 \times \cdots \times T_n$. We can easily assign a payoff to each player taking into account his/her own strategy as well as other players' strategies. So f is given by the following equation:

$$\begin{aligned} f : T &\longrightarrow \mathbb{R}^n \\ t &\longmapsto (f_1(t), \ldots, f_n(t)) \end{aligned}$$

t_{-i} will denote $(t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n)$ and similarly we can denote $f_i(t_i, t_{-i}) = f_i(t)$.

An *strategic Nash equilibrium* is a tuple $(t_1, \ldots, t_n)$ where $f_i(t_i, t_{-i}) \geq f_i(t'_i, t_{-i})$ for every other $t' \in T$, and all $i = 1, \ldots, n$. That is, an strategic Nash equilibrium maximizes the payoff for all the players.

The strategy for each player will be picked with respect to a given probability:

$$\delta_i \in \Delta_i = \{(\delta_i^{(1)}, \ldots, \delta_i^{(k_i)}) \in (R_0^+)^{k_i} : \sum_{j=1}^{k_i}\delta_i^{(j)} = 1\}$$

We define $\Delta_1 \times \cdots \times \Delta_n := \Delta$ and we call an element $\delta = (\delta_1, \ldots, \delta_n) \in \Delta$ an strategy profile. Having a fixed strategy profile δ, the overall payoff for the ith player is defined as:

$$u_i(\delta) = \sum_{(t_1,\ldots,t_n)\in T}\delta_i^{(t_i)}f_i(t)$$

Given u_i the payoff for a δ strategy profile in the i-th player and $\delta \in \Delta$, we will denote

$$\delta_{-i} := (\delta_1, \ldots, \delta_{i-1}, \delta_{i+1}, \ldots, \delta_n) \tag{B.1}$$

$$u_i(\delta_i, \delta_{-i}) := u_i(\delta) \tag{B.2}$$

A *probabilistic Nash equilibrium* is a strategy profile $x = (\delta_1, \ldots, \delta_n)$ verifying $u_i(\delta_i, \delta_{-i}) \geq u_i(\delta'_i, \delta_{-i})$ for every other $\delta' \in \Delta$, and all $i = 1, \ldots, n$.

A theorem ensures that every game space (P, T, f) with finite players and strategies has a *probabistic Nash equilibrium*.

B6.2. The algorithm

NEATER (*filteriNg of ovErsampled dAta using non cooperaTive gamE theoRy*), introduced in [19], is a filtering algorithm based on game theory. Let S be the original training set, E the synthetic generated instances. Our players are $S \cup E$. Every player is able to pick between two different strategies: being a negative instance (0) or being a positive instance (1). Players of S have a fixed strategy, where the ith player would have $\delta_i = (0, 1)$ (a 0 strategy) in case it is a negative instance or $\delta_i = (1, 0)$ (a 1 strategy) otherwise.

The payoff for a given instance is affected only by its own strategy and its k nearest neighbors in $S \cup E$. That is, for every $x_i \in E$, we will have $u_i(\delta) = \sum_{j\in NN^k(x)}(x_i^T w_{ij} x_j)$ where $w_{ij} = g\big(d(x_i, x_j)\big)$ and g is a decreasing function (greater distances imply a lower payoff). In our implementation, we have considered $g(z) = \frac{1}{1+z^2}$, with d the euclidean distance.

Each step should involve an update to the strategy profiles of instances of E. Namely, if $x_i \in E$, the following equation will be used:

$$\delta_i(0) = \left(\frac{1}{2}, \frac{1}{2}\right)$$

$$\delta_{i,1}(n+1) = \frac{\alpha + u_i((1,0))}{\alpha + u_i(\delta(n))}\delta_{i,1}(n)$$

$$\delta_{i,2}(n+1) = 1 - \delta_{i,1}(n+1)$$

That is, we reinforce the strategy that is producing the higher payoff, in detriment to the opposite strategy. This method has enough convergence guarantees.

References

[1] A. Fernández, S. del Río, N.V. Chawla, F. Herrera, An insight into imbalanced big data classification: outcomes and challenges, Complex Intell. Syst. 3 (2) (2017) 105–120, doi:10.1007/s40747-017-0037-9.
[2] Q. Zou, S. Xie, Z. Lin, M. Wu, Y. Ju, Finding the best classification threshold in imbalanced classification, Big Data Res. 5 (2016) 2–8, doi:10.1016/j.bdr.2015.12.001.
[3] B. Krawczyk, M. Woźniak, G. Schaefer, Cost-sensitive decision tree ensembles for effective imbalanced classification, Appl. Soft Comput. 14 (2014) 554–562, doi:10.1016/j.asoc.2013.08.014.
[4] P. Zhou, X. Hu, P. Li, X. Wu, Online feature selection for high-dimensional class-imbalanced data, Knowl. Based Syst. 136 (2017) 187–199, doi:10.1016/j.knosys.2017.09.006.
[5] H. He, E.A. Garcia, Learning from imbalanced data, IEEE Trans. Knowl. Data Eng. 21 (9) (2009) 1263–1284, doi:10.1109/tkde.2008.239.
[6] I. Triguero, S. González, J.M. Moyano, S. García, J. Alcalá-Fdez, J. Luengo, A. Fernández, M.J. del Jesús, L. Sánchez, F. Herrera, KEEL 3.0: an open source software for multi-stage analysis in data mining, Int. J. Comput. Intell. Syst. 10 (1) (2017) 1238, doi:10.2991/ijcis.10.1.82.
[7] G. Lemaitre, F. Nogueira, C.K. Aridas, Imbalanced-learn: a python toolbox to tackle the curse of imbalanced datasets in machine learning, J. Mach. Learn. Res. 18 (2017) 1–5.

[8] A.D. Pozzolo, O. Caelen, S. Waterschoot, G. Bontempi, Racing for unbalanced methods selection., in: H. Yin, K. Tang, Y. Gao, F. Klawonn, M. Lee, T. Weise, B. Li, X. Yao (Eds.), Lecture Notes in Computer Science, 8206, Springer, 2013, pp. 24–31.

[9] W. Siriseriwan, Smotefamily: A Collection of Oversampling Techniques for Class Imbalance Problem Based on SMOTE, 2018, https://cran.r-project.org/package=smotefamily.

[10] N. Lunardon, G. Menardi, N. Torelli, ROSE: a package for binary imbalanced learning, R J. (2014) 8292.

[11] G. Menardi, N. Torelli, Training and assessing classification rules with imbalanced data, Data Min. Knowl. Discov. 28 (2014) 92122, doi:10.1007/s10618-012-0295-5.

[12] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, Smote: synthetic minority over-sampling technique, J. Arif. Intell. Res. 16 (2002) 321–357, doi:10.1613/jair.953.

[13] A. Fernandez, S. Garcia, F. Herrera, N.V. Chawla, Smote for learning from imbalanced data: progress and challenges. Marking the 15-year anniversary, J. Arif. Intell. Res. 61 (2018) 863–905.

[14] S. Das, S. Datta, B.B. Chaudhuri, Handling data irregularities in classification: foundations, trends, and future challenges, Pattern Recognit. 81 (2018) 674–693, doi:10.1016/j.patcog.2018.03.008.

[15] S. Barua, M.M. Islam, X. Yao, K. Murase, MWMOTE–majority weighted minority oversampling technique for imbalanced data set learning, IEEE Trans. Knowl. Data Eng. 26 (2) (2014) 405–425, doi:10.1109/tkde.2012.232.

[16] B. Das, N.C. Krishnan, D.J. Cook, RACOG and wRACOG: two probabilistic oversampling techniques, IEEE Trans. Knowl. Data Eng. 27 (1) (2015) 222–234, doi:10.1109/tkde.2014.2324567.

[17] H. Zhang, M. Li, RWO-sampling: a random walk over-sampling approach to imbalanced data classification, Inf. Fusion 20 (2014) 99–116, doi:10.1016/j.inffus.2013.12.003.

[18] M. Gao, X. Hong, S. Chen, C.J. Harris, E. Khalaf, PDFOS: PDF estimation based over-sampling for imbalanced two-class problems, Neurocomputing 138 (2014) 248–259, doi:10.1016/j.neucom.2014.02.006.

[19] B.A. Almogahed, I.A. Kakadiaris, NEATER: filtering of over-sampled data using non-cooperative game theory, Soft Comput. 19 (11) (2014) 3301–3322, doi:10.1109/ICPR.2014.245.

[20] J. Alcalá, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera, Keel data-mining software tool: data set repository, integration of algorithms and experimental anlysis framework, J. Mult.-Valued Log. Soft Comput. 17 (2010) 255–287.