Neurocomputing 216 (2016) 596–610

Contents lists available at ScienceDirect

Neurocomputing

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

# EPRENNID: An evolutionary prototype reduction based ensemble for nearest neighbor classification of imbalanced data

Sarah Vluymans [a,b,c,*], Isaac Triguero [b,d,e], Chris Cornelis [a,c], Yvan Saeys [b,d]

[a] Department of Applied Mathematics, Computer Science and Statistics, Ghent University, Belgium
[b] Data Mining and Modeling for Biomedicine, VIB Inflammation Research Center, Ghent, Belgium
[c] Department of Computer Science and Artificial Intelligence, University of Granada, Spain
[d] Department of Internal Medicine, Ghent University, Belgium
[e] School of Computer Science, University of Nottingham, Jubilee Campus, Wollaton Road, Nottingham NG8 1BB, United Kingdom

ARTICLE INFO

*Article history:*
Received 27 November 2015
Received in revised form
7 July 2016
Accepted 4 August 2016
Communicated by Swagatam Das
Available online 11 August 2016

*Keywords:*
Imbalanced data
Prototype selection
Prototype generation
Differential evolution
Nearest neighbor

ABSTRACT

Classification problems with an imbalanced class distribution have received an increased amount of attention within the machine learning community over the last decade. They are encountered in a growing number of real-world situations and pose a challenge to standard machine learning techniques. We propose a new hybrid method specifically tailored to handle class imbalance, called EPRENNID. It performs an evolutionary prototype reduction focused on providing diverse solutions to prevent the method from overfitting the training set. It also allows us to explicitly reduce the underrepresented class, which the most common preprocessing solutions handling class imbalance usually protect. As part of the experimental study, we show that the proposed prototype reduction method outperforms state-of-the-art preprocessing techniques. The preprocessing step yields multiple prototype sets that are later used in an ensemble, performing a weighted voting scheme with the nearest neighbor classifier. EPRENNID is experimentally shown to significantly outperform previous proposals.



## 1. Introduction

Class imbalance is present in a dataset when its instances are unevenly distributed among the classes. It is encountered in many real-world situations such as medical diagnosis [1], microarray data analysis [2] or software quality evaluation [3]. Many applications are inherently prone to class imbalance, motivating the increased amount of attention to this issue within the machine learning community [4].

The class imbalance problem [5] refers to the fact that the performance of learning algorithms can be severely hampered by data imbalance. In this work, we focus on two-class imbalanced classification, where the elements of the *majority* class outnumber those of the *minority* class. Traditionally, the majority elements are denoted as *negative*, whereas the minority elements are referred to as *positive*. Standard classification techniques may not perform well in this context, as they internally assume equal class distributions. Consequently, over the last decade, a considerable amount of work has been proposed in the specialized literature to alleviate the imbalance problem [6–8]. Some approaches work at the data level, while others develop custom classification processes. At the data level, the so-called data sampling methods modify the training dataset to produce a better balance between classes [9,10]. Solutions at the algorithm level are modifications of existing methods and internally deal with the intrinsic challenges of imbalanced classification [11,12].

Prototype reduction techniques [13] were originally developed to simplify large training datasets in order to improve the noise tolerance, the speed and the storage requirements of learning models [14,15]. They can be applied to imbalanced datasets [16–18] as a data level approach, balancing majority and minority classes. Two main families of prototype reduction techniques exist in the literature: prototype selection (PS) [19] and prototype generation (PG) [20]. The former is limited to selecting a subset of instances from the original training data, while the latter can create new artificial instances to better adjust the decision boundaries of the classes. However, PG methods are known to be susceptible to overfitting [20,21]. The best performing models are evolutionary-based techniques, such as differential evolution [22]. In [23], the authors showed that a hybrid setting of PS and PG can significantly improve the classification process in a balanced class setting. To the best of our knowledge, no hybrid PS-PG techniques

* Corresponding author at: Department of Applied Mathematics, Computer Science and Statistics, Ghent University, Belgium.
*E-mail address:* Sarah.Vluymans@UGent.be (S. Vluymans).

http://dx.doi.org/10.1016/j.neucom.2016.08.026
0925-2312/

have been developed to deal with imbalanced classification problems so far.

In this paper, we propose a combined model for the classification of two-class imbalanced data, integrating both a hybrid preprocessing and a classification step. We extend the framework of [23] for use in the presence of class imbalance, considerably modifying both PS and PG stages. We also aim at introducing diversity in the process. The multiple prototype sets resulting from the preprocessing step are further combined in a custom ensemble for classification. The classification step is an extension of the $k$ nearest neighbor classifier ($k$NN [24]). We call our method EPRENNID, an Evolutionary Prototype Reduction based Ensemble for Nearest Neighbor classification of Imbalanced Data.

The main contributions of this work are as follows:

- We first introduce a new evolutionary PS method specifically tuned to handle class imbalance. Although it is related to undersampling methods, it takes a step away from them by allowing the removal of minority elements from the dataset, as in [25]. Most existing methods do not allow such kind of reduction of non-representative or noisy elements from the positive class.
- To alleviate the overfitting issues of prototype reduction models, we take advantage of the evolutionary nature of the proposed method. Instead of yielding a single reduced set, EPRENNID provides several well-performing and diverse ones.
- The evolutionary PG method used in this work [23] has been modified to handle the class imbalance problem.
- Finally, the optimized prototype sets are used in a classifier ensemble, using an adaptive scheme selecting the most suitable prototype sets to classify each single target instance with $k$NN.

To analyze the performance of our proposal, we carry out an extensive experimental study on 35 two-class imbalanced datasets, categorized into different groups corresponding to the difficulty of identifying minority elements. We compare our model with state-of-the-art models and apply non-parametric statistical tests to check whether there are significant differences among them.

The remainder of this paper is structured as follows. In Section 2, we review the PS and PG schemes and provide more details on related work in imbalanced classification. Section 3 introduces the proposed model, with a detailed explanation of the separate preprocessing and classification phases. We have conducted a comprehensive experimental study. Its setup is described in Section 4, while Section 5 lists and discusses our results. Finally, Section 6 formulates the conclusions of this work and outlines future research directions.

## 2. Preliminaries and related work

This section provides the necessary background for the remainder of the paper. Section 2.1 presents prototype selection and generation techniques, focusing on the methods on which our model is based. Section 2.2 introduces the problem of classification with imbalanced datasets and its evaluation is recalled in Section 2.3.

### 2.1. Prototype reduction

Prototype reduction techniques aim to reduce the available training set $T = \{x_1, x_2, \ldots, x_n\}$ of labeled instances to a smaller set of prototypes $S = \{y_1, y_2, \ldots, y_r\}$, with $r < n$ and each $y_i$ either drawn from $T$ or artificially constructed. The set $S$, rather than the entire set $T$, is used afterwards to train the classifier.

These methods are commonly combined and designed to be used with the $k$NN classifier. This lazy learning algorithm [26] assigns new input instances to the class to which the majority of their $k$ nearest neighbors in the training set belongs. Despite its performance, it suffers from several drawbacks such as low efficiency, high storage requirements and sensitivity to noise. PS and PG techniques can be beneficial to alleviate these issues. To that end, the instances contained in $S$ should form a good representation of the original class distributions. Furthermore, their size relative to that of $T$ should be small enough in order to considerably reduce the storage and execution time requirements of $k$NN.

A PS method reduces $T$ to $S$ by selecting a subset of its instances. This implies that for every instance $y_i \in S$ there exists an element $x_j \in T$ such that $y_i = x_j$. In [19], a taxonomy for PS methods was proposed and an extensive experimental study was conducted. The main difference between PG and PS is that the former can either select elements from $T$ or construct artificial ones, while the latter is restricted to selecting elements from $T$. Therefore, a set $S$ constructed by a PG method is not necessarily a subset of $T$, allowing for a larger flexibility in the construction of $S$. For PG methods, a related taxonomy has been proposed in [20]. In what follows, we describe the PS and PG methods on which we base our proposal.

#### 2.1.1. Steady state memetic algorithm for instance selection

The Steady State Memetic Algorithm (SSMA) is a genetic algorithm for PS. In several experimental studies (e.g. [19,23]), it has been shown to be one of the best-performing PS methods, which is due to its optimization procedure performed in each iteration. As a genetic algorithm, it evolves a population of $I$ individuals, the chromosomes, over a number of generations $G$. Each individual corresponds to a candidate subset and is encoded as a bitstring, where a 0 in the $i$th position means that the $i$th element of $T$ is not included in the subset, while a 1 means that it is. The quality of an individual, that is, how good a solution it is, is evaluated by a so-called fitness function. To calculate the fitness of a candidate subset $S$, SSMA uses a combined criterion, namely the accuracy of the $k$NN classifier on the entire training set $T$ using $S$ as prototype set and the reduction in size of $S$ relative to $T$.

The population is optimized over the subsequent generations, such that the final fittest individual corresponds to an optimal solution. To guide the evolution, it uses two genetic operators: crossover and mutation. In each generation, two parents are selected to produce two new individuals by means of the Half Uniform Crossover (HUX) procedure: positions in which the parents take on the same value are simply copied to the children, while for the remaining ones, each child randomly copies half of each parent.

Afterwards, random mutation is applied to the children. This procedure changes the value of a randomly selected position with probability $p$. The most defining aspect of the SSMA method is its use of an optimization procedure, the so-called meme. This is an iterative optimization process that pursues a double objective to improve individuals of the population: the reduction of the number of selected prototypes and the enhancement of the classification accuracy. The meme is applied on a generated child when its fitness value is higher than the current lowest fitness in the population. When its fitness is lower, the optimization is only executed with a small probability. We refer to the original proposal [27] for a detailed description.

#### 2.1.2. Scale factor local search in differential evolution

Scale Factor Local Search in Differential Evolution (SFLSDE) [28] was shown to be one of the top performing PG methods in the experimental study of [23]. It is a positioning adjustment algorithm, optimizing the positions of the instances in the dataset. The method uses differential evolution (DE [29,22]), which follows the

evolutionary framework, evolving a population of candidate solutions over a number of generations. The evolution is guided by custom mutation and crossover operators. In general, for each individual $x_i$, mutation is achieved by randomly selecting two other chromosomes $x_1$ and $x_2$ from the current population. A new individual is created by increasing $x_i$ by the difference of $x_1$ and $x_2$, weighted by a scale factor $F > 0$. A number of different mutation operators exist, but we have chosen to use the DE/RandToBest/1 strategy, which makes use of the current fittest $x_{best}$ individual in the population. It increases $x_i$ by both the difference of the two randomly selected individuals as well as the difference of $x_i$ and $x_{best}$, weighting both terms by $F$. After mutation, crossover is performed, randomly modifying the mutated individual in certain positions. The crossover is guided by another user-specified parameter $Cr$.

SFLSDE is a memetic DE algorithm and modifies the general mutation and crossover schemes, integrating two local searches. The method uses adaptive values for the $F$ and $Cr$ parameters. Specifically, each instance $x_i$ has its custom values $F_i$ and $Cr_i$ values assigned to it, which are updated in each iteration. When updating the scale factors $F_i$, two local searches are used: the golden section search and hill-climbing. We refer to [28] for further details.

### 2.2. Imbalanced classification

In a wide range of classification problems, the number of instances that belong to each class can be radically skewed. Standard classifiers tend to be biased towards the majority class, although the minority class is normally the most interesting class.

Several approaches have been developed to alleviate the imbalance problem either at the data level or by designing especially designed classification methods that address the particular challenges associated to these types of problems. At the data level, *data sampling methods* modify the dataset to find a better balance between the classes, such that class imbalance should not hinder a posterior classification process. A first group consists of *undersampling* methods, which remove a part of the majority class. This can be done in a random way [9] or more complex heuristics for selecting majority class candidates for removal can be put in place [30,16]. By reducing the size of the dataset, undersampling methods are actually performing prototype selection. However, they are constrained in their application, as they are usually only allowed to reduce the majority class, leaving the minority elements untouched.

The strategy adopted by a second kind of methods consists of finding a more favorable balance between classes by means of *oversampling* the minority class. The size of this class is increased by adding duplicates of existing minority instances or by constructing artificial elements based on the ones at hand. A straightforward approach is presented in [9]. It involves the duplication of randomly selected minority elements. The SMOTE technique [10] laid the foundation of more complex oversampling methods. Instead of duplicating existing minority elements and thereby increasing their weight in the dataset, it generates a number of synthetic instances assigning them to the minority class. Several later proposals (e.g. [31–33]) are modifications of SMOTE, replacing some of its random components by more complex procedures.

Finally, several *hybrid* data sampling methods, both undersampling the majority and oversampling the minority class, have been designed as well. They often combine an initial oversampling step by posterior data cleaning [9]. The first phase usually results in a perfectly balanced dataset, on which the data cleaning is executed. The latter can be performed on either the entire intermediate set or be restricted to the newly generated instances. Alternatively, a complete intertwining of the oversampling and undersampling approaches can be set up, generating minority elements and removing majority instances at the same time [34,35].

Apart from the data level approaches discussed above, some specific classification algorithms tolerating class imbalance have been proposed as well. These include the *cost-sensitive learners*, like cost-sensitive *k*NN [12], cost-sensitive C4.5 [11], cost-sensitive SVM [36,37] and cost-sensitive neural networks [38], which modify traditional classifiers by assigning different costs to the misclassification of minority and majority instances. These costs are used in the construction of the classification model and reduce the dominance of majority over minority elements.

A number of *ensemble techniques* have also been specifically designed to handle the classification of imbalanced data with multiple applications [39]. Most commonly, they use a standard ensemble learning technique, such as the boosting [40] or bagging [41] schemes, and incorporate some heuristics to deal with class imbalance or cost-sensitive models [42]. Other ensemble-based approaches analyze the influence of noisy data in imbalanced classification [43]. Prominent and recent examples include the SMOTEBoost [44], SMOTEBagging [45], RB-Bagging [46], NBBag [47], and EUSBoost [48] methods. Very recent proposals also deal with multi-class imbalanced data [49].

### 2.3. Evaluation of imbalanced classifiers

In this section, we review the important issue of the evaluation of the classification performance on imbalanced data. Table 1 presents a generic confusion matrix for binary classification problems, displaying the number of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN) obtained in a classification experiment. As this paper focuses on binary problems, we restrict this matrix to the binary case as well, but its generalization to more than two classes is straightforward.

In traditional classification applications, the performance of a classifier is commonly assessed by the classification accuracy (percentage of correctly classified examples, that is, $acc = \frac{TP+TN}{n}$, where $n$ is the size of the dataset). In the presence of class imbalance this measure usually provides misleading results, because it does not distinguish between the number of correct labels of different classes, making it sensitive to skewness in class distributions [50].

As an alternative to the overall accuracy, the geometric mean g_mean is often used [16,51,52]. This measure is defined as

$$g\_mean = \sqrt{\frac{TP}{TP+FN} \cdot \frac{TN}{TN+FP}}.$$

Another widely used evaluation measure in this domain is the Area under the ROC-curve (AUC) [7,9,16]. A ROC-curve is defined for probabilistic classifiers on binary problems and reflects the trade-off between its true positive $TP$ and false positive $FP$ rates. The area under it expresses how well the classifier achieves this, in a single measure.

For a discrete classifier, outputting actual class labels rather than class probability estimates, a ROC-curve can be constructed by converting its crisp output to the required class probabilities. As noted in [53], one needs to consider the inner workings of the method to extract these values. For example, when applying the

**Table 1**
Confusion matrix obtained after classification of a two-class dataset.

| Actual/predicted | Positive | Negative |
|---|---|---|
| Positive | TP | FN |
| Negative | FP | TN |

$k$NN classifier on a binary problem, an instance is assigned to the class to which the majority of its $k$ nearest neighbors belong. The probabilities of belonging to the positive and negative classes can be set to $\frac{k_+}{k}$ and $\frac{k_-}{k}$ respectively, where $k_+$ represents the number of positive elements among the $k$ neighbors and $k_-$ the number of negative ones. As shown in [54], when $k=1$, the AUC is computed as

$$AUC = \frac{1 + TP - FN}{2}.$$

The difference between AUC and g_mean is that the AUC presents a global picture of the strength of the classifier, varying the threshold of how likely instances should belong to the positive class to be assigned to it (except in the case of $k$NN and $k=1$), while g_mean solely considers the standard decision criterion, assigning instances to the class to which they most likely belong.

As discussed in [53], ROC-curves are insensitive to changes in class distribution, rendering the AUC a proper measure to use in the classification of imbalanced data. This results from the fact that the points of the curve are determined using the row-wise ratios of the confusion matrix. By using the rows separately, the ROC-curve and the AUC do not depend on the actual class distribution.

## 3. Proposed model: EPRENNID

In this section, we introduce our new model for the classification of imbalanced data, incorporating both a preprocessing and a classification step. The former involves the combination of PS and PG and the latter uses an ensemble of well-performing prototype sets, provided by the preprocessing step, in a weighted voting scheme. Its different stages are depicted in Fig. 1.

The description of EPRENNID is divided into two main parts. In Section 3.1 we discuss its preprocessing phase and in Section 3.2, we proceed with the classification part of our model, which is an ensemble approach using $k$NN.

### 3.1. Preprocessing: a hybrid prototype reduction model for imbalanced data

In this section, we propose a hybrid prototype reduction model to preprocess imbalanced data. Based on the model presented in [23], in which the authors combined PS and PG models for standard classification, our proposal will hybridize these two processes to alleviate the weaknesses of the isolated models in the imbalanced context.

By means of PS, a number of well-performing prototype sets are generated, which are further optimized using a PG method. The underlying motivation for applying such hybridization is that PG models are more flexible than PS techniques, allowing us to obtain more accurate reduced sets that are not limited to selecting a subset of instances from the original training set. However, PG models also suffer from several drawbacks such as initialization issues (appropriate choice of the number of prototypes per class) and more complex search spaces, in which PS models can be exploited to ease the posterior PG process. More details about the benefits of hybridizing PS and PG can be found in [23]. We use the most successful combination suggested by their experiments: the PS method SSMA (Section 2.1.1) and the PG method SFLSDE (Section 2.1.2).

The hybrid prototype reduction step incorporated in EPRENNID is considerably different from the proposal of [23] in order to alleviate the overfitting problems of these models and handle imbalance problems.

- Firstly, we provide a wide variety of reduced sets with a newly proposed PS method, SSMA$_{Imb}$. This method entirely replaces the SSMA step in [23] to take into account the class imbalance, allowing elimination of both positive and negative examples (Section 3.1.1).
- Secondly, the optimization performed by SFLSDE has been modified as well (named SFLSDE$_{Imb}$), by means of a new objective function, more appropriately evaluating the performance of a prototype set in a classification process (Section 3.1.2).
- Finally, we combine the above two methods in a hybrid setting, optimizing multiple SSMA$_{Imb}$ generated prototype sets with SFLSDE$_{Imb}$. We use a diversity mechanism to select a diverse set of well-performing prototype sets to deal with the overfitting problem often encountered by PG. They are later optimized in separate populations, out of which a final diverse set is again selected to be used in the classification step. In this way, during classification, EPRENNID has the flexibility to select prototype sets that have proven to perform well in the neighborhood of a specific target, instead of relying on one prototype set to perform well in the entire feature space (Section 3.1.3).

#### 3.1.1. SSMA$_{Imb}$

Even though SSMA performs very well on balanced data, it fails when faced with class imbalance. A preliminary experimental study [54] showed that the direct application of this method significantly worsens the classification performance and tends to remove all the examples from the minority class. Nevertheless, its good performance on balanced data, its use of the optimization step and its flexibility motivated us to adapt it to tackle imbalanced problems. We have kept the defining aspects of SSMA in place, i.e. it remains a steady state memetic algorithm, but we have integrated some imbalance-resistant heuristics at three crucial

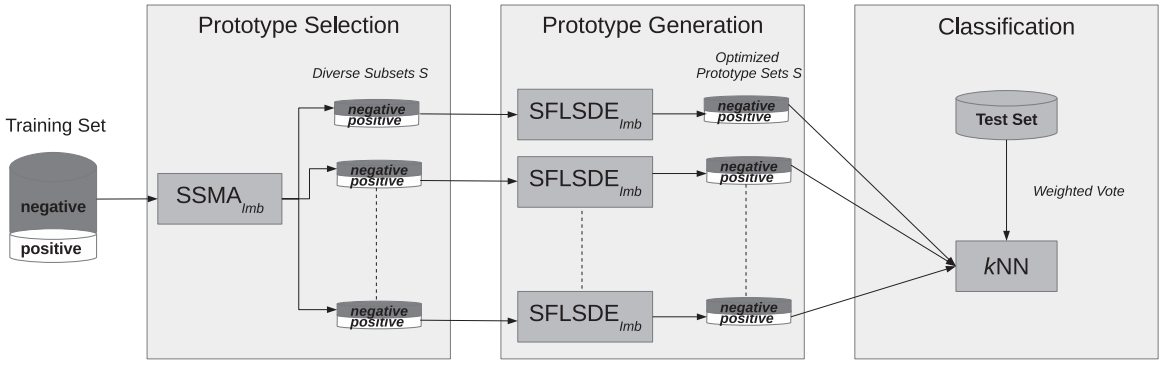


**Fig. 1.** Schematic workflow of EPRENNID. In a first step, diverse prototype sets (*numPS*) are selected from the training data using the proposed SSMA$_{Imb}$ algorithm. These prototype sets are refined by using SFLSDE$_{Imb}$ to optimize the positioning of the prototypes of every subset. Finally, the resulting pre-processed datasets are used in a weighted vote to classify test instances.

points: the fitness function, the parent selection mechanism and the meme optimization. We call the modified method $SSMA_{Imb}$.

*Fitness function modifications*: The first change that needs to be made lies with the fitness function, as it has some clear shortcomings in the context of class imbalance. By evaluating the classification performance by the accuracy and explicitly using the reduction, small subsets consisting of mostly negative elements can easily attain high fitness values and give the impression of representing high quality subsets. As an example, consider a training set $T$ consisting of 10 positive and 90 negative elements and a singleton candidate subset $S$ of one negative instance. Using the 1NN rule as the classifier, all instances are classified as negative, yielding an accuracy of 90%. The reduction rate would be 99%. The combination results in a high fitness value, even though this set will never be able to classify any positive element correctly. To remedy this situation, we propose a new fitness function, similar to the one used in [16]:

$$fitness(S) = g\_mean - \left|1 - \frac{1}{IR_S}\right| \cdot P, \tag{1}$$

where g_mean replaces the accuracy to evaluate the classification performance of $S$. This is determined by using $k$NN and leave-one-out cross-validation. The value $IR_S$ corresponds to the *imbalance ratio* of the set $S$. This measure evaluates how imbalanced a set is and is defined as $IR_S = \frac{Maj_S}{Min_S}$, where $Maj_S$ and $Min_S$ correspond to the cardinality of the majority and minority classes in $S$ respectively. It is important to note that the majority and minority classes in $S$ do not necessarily correspond to those in $T$. Finally, the parameter $P$ determines the weight of the second term and therefore how much class imbalance is penalized in $S$. The authors of [16] proposed to use $P=0.2$ and we have adopted this value as well. We evaluated other values for $P$ in a preliminary experimental study, but no significant differences in performance were observed, so we decided to use its default value. The new fitness function favors subsets $S$ with a good classification performance and that are not too imbalanced. The fitness function is used to decide whether the meme optimization is applied or not. In a later stage of our proposal (see Section 3.1.3), it is also applied in the selection of the fittest individuals.

*Parent selection mechanism*: When selecting parents to create offspring, the original SSMA method assigns a higher probability of being selected to individuals with a higher fitness. However, $SSMA_{Imb}$ does not use the fitness measure for such purpose, because it is more focused on providing good classification performance ($P=0.2$). The parent selection procedure of $SSMA_{Imb}$ aims to explicitly favor more balanced sets in the population as well, reducing the imbalanced ratio while keeping the g-mean high, independently of any parameter $P$. Thus, we propose a new measure, defined as

$$Sel(S) = 2 \cdot \frac{g\_mean \cdot \frac{1}{IR_S}}{g\_mean + \frac{1}{IR_S}}, \tag{2}$$

which corresponds to the harmonic mean of g_mean and $\frac{1}{IR_S}$. The harmonic mean of two values tends more strongly to the smaller one, meaning that both inputs should attain high values for it to be large. In this case, this corresponds to a high value for g_mean and low IR. Chromosomes attaining higher $Sel(\cdot)$ values have a higher probability of being selected for reproduction.

The HUX operator must deal with a constraint posed by imbalanced problems: avoid creating children that do not contain any positive instances. For example, consider two parents that differ in all genes in a dataset with $|T| = 11$, with 3 positive and 8 negative instances:

$$\underbrace{1\,0\,1}_{Minority}\ \underbrace{0\,1\,0\,1\,0\,1\,0\,1}_{Majority} \quad \text{and} \quad \underbrace{0\,1\,0}_{Minority}\ \underbrace{1\,0\,1\,0\,1\,0\,1\,0}_{Majority}.$$

In the construction of their children, randomly half of the positions of each parent are used. This could yield a child without minority class elements:

$$\underbrace{0\,0\,0}_{Minority}\ \underbrace{1\,0\,1\,1\,0\,1\,0\,1}_{Majority}.$$

To tackle this issue, the HUX operator will be initially limited to the positions of the majority elements. This creates two partial children with only majority elements. Next, the minority positions are filled up in each child. When both parents take on the same value, this value is copied to the child. Otherwise, we set the position to 1 while there are fewer minority than majority elements in the chromosome. When a perfect balance has been achieved, we go back to selecting a random value. This procedure depends on the order in which the minority genes are considered. We first use the genes set to 1 in the fittest parent, randomly ordered.

*Meme optimization*: Finally, the meme optimization procedure has been modified as well. First, its evaluation of the classification performance by the accuracy has been replaced by the AUC. Note that our new fitness function uses the g_mean measure, while we now evaluate classification performance by the AUC. By incorporating the two measures in the PS algorithm, we avoid overfitting one of them and instead aim to optimize both. Furthermore, instead of trying to increase the reduction, we are allowing only majority positions to be set to 0 and minority positions to be set to 1. The majority and minority classes are again determined within the chromosome at hand. When the chromosome is perfectly balanced, that is, the same number of elements for both classes are selected, no optimization is performed. Note that an empty set $S$ is also perfectly balanced, but in such a situation $S$ is optimized by adding an arbitrary element of each class. By setting majority genes to 0 and minority genes to 1, $IR_S$ can only decrease, possibly up to a point where a perfect balance is achieved and an imbalance in the other direction would be created. To prevent this, the optimization halts prematurely when this occurs. We present the modified optimization procedure in Algorithm 1.

**Algorithm 1.** Optimization procedure of $SSMA_{Imb}$.

**Require**: A chromosome $S = \{s_1, s_2, \ldots, s_n\}$
**Ensure**: The optimized chromosome
Determine the majority and minority class in $S$.
**while** there are untested positions **do**
  $S_* \leftarrow S$.
  Select a random untested minority position with $s_i=0$ or majority position with $s_i=1$.
  Change the value of the selected position in $S^*$.
  $AUC_{S^*} \leftarrow$ AUC of $k$NN, using $S^*$ as prototype set.
  $AUC_S \leftarrow$ AUC of $k$NN, using $S$ as prototype set.
  $gain \leftarrow AUC_{S^*} - AUC_S$.
  **if** $gain \geq \mu$ **then**
    $S \leftarrow S^*$
  **end if**
**if** $IR_S = 1$ **then**
    halt the optimization
  **end if**
**end while**

In the original algorithm, the value $\mu$ is a given threshold. If

$\mu \geq 0$, a modified chromosome is only accepted when it results in a higher AUC of the classifier. When $\mu < 0$, individuals can also be accepted when they correspond to a set $S$ with lower classification performance, preventing a premature convergence to a local optimum. The value of this parameter is adaptive. It is initialized as $\mu = 0$, specification by the user is not required. When after a given number of generations the performance of the best chromosome in the population has not improved, it is increased internally by 0.001. When the reduction corresponding to the best chromosome has not increased for a given number of populations, $\mu$ is decreased by 0.001. The value 0.001 was chosen as it is used in combination with the AUC, which is a number in the interval $[0, 1]$.

#### 3.1.2. $SFLSDE_{Imb}$

As a second step in the preprocessing phase, we apply a PG algorithm. We have opted to combine $SSMA_{Imb}$ with a modified version of SFLSDE. This combination yielded the best results in [23], making it a potential candidate for extension to imbalanced classification problems. Nevertheless, the authors used the accuracy of the 1NN classifier to evaluate the fitness of the individuals during DE. Following our discussion in Section 2.3, the accuracy is not an appropriate fit performance measure for class imbalance in the training set and we have accordingly changed it to g_mean. We do not modify the other DE operators of mutation and crossover as we did in the PS step. The PG method is applied after the dataset has been preprocessed by $SSMA_{Imb}$. The individuals coming from this PS step are already more balanced, implying that the genetic operators in SFLSDE do not have to be specifically tuned to handle class imbalance. As before, we denote the modified DE algorithm as $SFLSDE_{Imb}$.

#### 3.1.3. Hybridizing $SSMA_{Imb}$ and $SFLSDE_{Imb}$

As $SSMA_{Imb}$ is a genetic algorithm, it encounters a high number of candidate prototype sets during its run. Even though these do not correspond to the final fittest solution, they might still constitute valid alternatives performing well in the classification. To use these solutions and enhance the performance of our algorithm, we therefore do not restrict ourselves to selecting only the fittest solution found by $SSMA_{Imb}$, but rather select a diverse set of fit individuals, setting up a voting committee for the final classification step (Section 3.2).

The user specifies the desired number *numPS* of prototype sets, which are selected from among the 50% fittest chromosomes encountered during the entire execution. The selection procedure is described in Algorithm 2. The first set is chosen as the overall fittest one. The remaining sets are selected by an incremental procedure, continuously adding subsets diverse enough from the ones previously selected, until *numPS* have been chosen.

The diversity measure between two prototype sets $S_1$ and $S_2$ is based on Yule's $Q$-statistic [55]. This is a well-known measure in the ensemble community that ensures a good level of diversity in the classification behavior of two datasets [56]. It has recently been shown that promoting diversity by means of the $Q$-statistic in an ensemble for imbalanced classification has a positive effect on its performance evaluated by both AUC and g_mean [57]. The examples of the training set $T$ are classified with the $1NN$ rule, using both prototype sets $S_1$ and $S_2$ as reference sets. Their diversity is computed as follows:

$$diversity(S_1, S_2) \leftarrow 1 - \frac{n_{00}n_{11} - n_{01}n_{10}}{n_{00}n_{11} + n_{01}n_{10}}, \tag{3}$$

where $n_{00}$ represents the number of instances that none of the classifiers predicted correctly, $n_{11}$ the number of samples correctly classified by both of them, and $n_{01}$ and $n_{10}$ counts the number of samples predicted by $S_1$ and not by $S_2$ and vice versa.

**Algorithm 2.** Selection of a diverse set of fit prototype sets.

**Require**: A complete set **S** of candidate prototype sets, an integer *numPS*
**Ensure**: Set *div* of *numPS* selected prototype sets

```
div ← {S}, where S is the fittest individual in S
numPS ← numPS − 1
while numPS > 0 do
  diversity_max ← 0
  S_best ← null
  for all S ∈ S do
    diversity_curr ← 0
  for all P ∈ div do
    diversity_curr ← diversity_curr + diversity(S, P)
  end for
  if diversity_curr > diversity_max then
    diversity_max ← diversity_curr
    S_best ← S
  end if
 end for
 T ← T\S_best
 div ← div ∪ {S_best}
 numPS ← numPS − 1
end while
```

Each selected prototype set undergoes position adjustment by $SFLSDE_{Imb}$, in order to further optimize the positions of the prototypes. Initial populations on which DE is being applied should contain individuals covering the entire population space, which is why [29] suggested filling it in a random way. In our framework, each prototype selected after the PS step seeds a separate population. The population is generated randomly, but the structure of the seeding chromosome is preserved, in that all individuals have the same class sizes.

After execution of $SFLSDE_{Imb}$, instead of selecting the final fittest individual from each population, we use a procedure similar to the one described in Algorithm 2 to preserve the diversity initially injected between the populations. We have incorporated a slight modification in this stage, namely by weighting the diversity measure by the fitness of the subset. In this way, we achieve a trade-off between fitness and diversity, obtaining a set of diverse prototype sets without undoing the efforts of the optimization.

### 3.2. Classification

In [58], an ensemble approach to $k$NN classification using PS was introduced. In each iteration of a boosting algorithm, PS was applied to train a classifier to improve the classification of difficult instances. PS is used to construct subsets of the training set able to better classify these difficult instances. Although we are also using an ensemble approach for the $k$NN classifier, incorporating PS, the classification process set up in EPRENNID is very different from the one in [58]. We are not using a boosting scheme, but constructing an ensemble based on a diverse set of preprocessed prototype sets $S$. These are used in a weighted vote to perform the classification. In particular, when classifying a target instance $x$, prototype sets performing well in the neighborhood of $x$ are assigned a larger weight in the vote compared to ones not performing particularly well there.

For each target instance $x$, different weights are assigned to all prototype sets. For each set $S$, we consider the $K_s$ nearest neighbors of $x$ in the training set. Each neighbor is classified by 1NN using $S$ as prototype set. The weight of $S$ is set equal to the number of

correctly classified neighbors and is therefore an integer contained in the interval $[0, K_s]$. When all sets have been processed, the weights are normalized by dividing them all by the maximal weight that was encountered.

When the weights have been determined, EPRENNID proceeds with the final classification of $x$. The instance is classified by $k$NN *numPS* times, using each prototype set once, where $k$ is specified by the user. Each set votes for the class to which it assigns $x$, using its computed weight. Finally, $x$ is assigned to the class with the highest number of votes.

## 4. Experimental setup

This section discusses our experimental setup. We introduce the datasets on which our experiments are run (Section 4.1), the state-of-the-art methods for imbalanced classification to which EPRENNID is compared (Section 4.2), the evaluation measures (Section 4.3) and the statistical tests that we used (Section 4.4).

### 4.1. Datasets

We have selected 35 two-class imbalanced datasets on which all methods are executed. They were constructed by taking real-world datasets available from the UCI [59] or KEEL dataset [60] repositories and consequently merging or removing classes until only two remain. This procedure is common practice and has been used in other experimental studies as well (e.g. [2,48,52,61]).

Different kinds of minority class examples may have a different influence on learning classifiers [62,63]. To enrich the performed analysis, we have further divided the datasets into three subgroups, *dense*, *medium* and *sparse*, which represent different degrees of difficulty to recognize minority elements. Minority instances in dense datasets are grouped closely together, while they are more spread out in sparse datasets.

Inspired by [62], we use the local neighborhood of minority elements to consider them as *safe*, *borderline*, *rare* or *outliers*. In this work, we propose an alternative definition that is more conservative than the one used in [62]. For each minority instance, we determine its five nearest neighbors in the dataset and denote it as:

- *Safe*: The five nearest neighbors of this instance all belong to the minority class.
- *Borderline*: The instance has one or two majority class elements among its five nearest neighbors.
- *Rare*: The instance has three or four of its five nearest neighbors belonging to the majority class.
- *Outlier*: All five nearest neighbors of the instance belong to the majority class.

The three groups of datasets are constructed based on the division of their minority instances among these four types.

- In a *dense* dataset, at least half of the minority elements are *safe* or *borderline*.
- On the other hand, when more than half of the minority instances are *rare* elements or *outliers*, the dataset is considered *sparse*.
- In all remaining cases, the dataset is assigned to the *medium* group.

Table 2 provides an overview of all datasets. We specify the number of instances they contain and the degree of imbalance between the two classes, represented by the IR. The datasets are divided among the three groups, of which the sizes range between 11 and 13.

### 4.2. Methods

We have chosen a number of popular, well-performing data sampling methods to compare our model with. Preprocessing methods are used in conjunction with a later classification step. Since the base classification in EPRENNID is performed by the $k$NN classifier, we have opted to use it for the other data sampling methods as well for a fair comparison.

Below, we provide a short descriptive overview, including specifications with regard to their parameter settings. A specific choice of parameters over the different data sources may result in better performance, but our purpose here is to analyze the general performance of the techniques without a time-consuming parameter tuning step. Their operations should provide good enough results even though the parameters are not optimized for a particular problem. For this reason, we always use the default values recommended by their developers.

**Table 2**
Description of the real-world two-class datasets used in the experimental study. This table lists the number of instances (Inst) and the IR of the dataset, measuring the degree of imbalance between the majority and minority classes.

| | Dataset | Inst | IR | Dataset | Inst | IR |
|---|---|---|---|---|---|---|
| DENSE | abalone-3vs11 | 502 | 32.47 | page-blocks0 | 5472 | 8.79 |
| | ecoli4 | 336 | 15.80 | segment0 | 2308 | 6.02 |
| | glass6 | 214 | 6.38 | shuttle2vs5 | 3316 | 66.67 |
| | haberman | 306 | 2.78 | texture2redvs34 | 1042 | 23.81 |
| | iris0 | 150 | 2.00 | vehicle2 | 846 | 2.88 |
| | kddcup-bovsb | 2233 | 73.43 | wisconsin | 683 | 1.86 |
| | new-thyroid1 | 215 | 5.14 | | | |
| MEDIUM | abalone17vs78910 | 2338 | 39.31 | segment6redvs345 | 1002 | 82.50 |
| | appendicitis | 106 | 4.05 | shuttle67vs1redB | 2023 | 86.96 |
| | cleveland0vs4 | 173 | 12.31 | vehicle0 | 846 | 3.25 |
| | ecoli3 | 336 | 8.60 | wdbc-MredBvsB | 372 | 23.80 |
| | glass4 | 214 | 15.46 | yeast4 | 1484 | 28.10 |
| | movementlibras1 | 336 | 13.00 | | | |
| SPARSE | abalone20vs8910 | 1916 | 72.69 | shuttle6vs23 | 230 | 22.00 |
| | ecoli0147vs2356 | 336 | 10.59 | wdbc-MredvsB | 365 | 44.63 |
| | glass5 | 214 | 22.78 | winequality-red4 | 1599 | 29.17 |
| | ionosphere-bredvsg | 235 | 22.50 | winequality-white3vs7 | 900 | 44.00 |
| | magic-hredvsgred | 2645 | 54.10 | yeast0256vs3789 | 1004 | 9.14 |
| | phoneme-1redvs0red | 2543 | 46.98 | | | |

**Table 3**
Parameter settings for the EPRENNID method.

| Phase | Parameter values |
|---|---|
| $SSMA_{Imb}$ | *Evaluations* = 10000, *Population* = 30, $p_{mutation}$ = 0.001, $k$ = 1 |
| $SFLSDE_{Imb}$ | *Iterations* = 500, $F_l$ = 0.1, $F_u$ = 0.9, *iterSFGSS* = 8, *iterSFHC* = 20 |
| Hybridization | *numPS* = 40, *Population* = 10 |
| Classification | $K_s$ = 5, $k$ = 1, 3, 5 |

- Borderline-SMOTE2 (Border2, [31]): This oversampling procedure is a modification of SMOTE. It uses minority elements located near the decision boundaries as seeds for the construction of artificial instances. Artificial minority elements are introduced on the line segment between the seed instance and a randomly selected element from among their $k$ nearest positive neighbors. For each seed, one synthetic element is also generated on the line segment connecting it to its nearest negative neighbor.
- SMOTE-TL (SMT-TL, [9]): A Tomek Link (TL) is defined as a pair of opposite class elements which are located more closely to each other than to any other element in the dataset. The SMOTE-TL method consists of first applying SMOTE and afterward removing all pairs of elements that form a TL.
- SMOTE-$RSB_*$ (SMT-RSB [64]): Similar to SMOTE-TL, this method first applies SMOTE on the dataset and afterward removes certain instances. All original instances are automatically retained, but the synthetic elements are required to belong to the rough lower approximation [65] of the minority class. If they do not satisfy this criterion, they are removed.
- NCR [30]: This undersampling method seeks to remove harmful majority instances. It uses the Edited Nearest Neighbor method with $k$=3 [66] to remove noisy negative instances. When a negative element is misclassified by 3NN, it is removed from the dataset. When a positive instance is misclassified by the 3NN rule, all of the negative instances contributing to this misclassification are removed.
- Spider2 [35]: This is a hybrid data sampling method. It has a number of options to be set, where we have chosen the best ones put forward by [35]. In a first phase, negative instances that are misclassified by 3NN are relabeled as positive. Secondly, a number of duplicates of misclassified positive elements are added to the dataset.
- $SSMA_{Imb}$ (Section 3): We have also included our modified $SSMA_{Imb}$ algorithm described above, leaving out the PG optimization step and the ensemble classification procedure. This allows us to determine whether the added complexity of the latter two steps improves the performance of $SSMA_{Imb}$ or whether the actual strength of our model lies in the PS step alone.
- IPADE-ID (IPADE [18]): This is a previously proposed method for imbalanced classification using DE, making it an interesting competitor of EPRENNID. It is an extension of the IPADE method [67] to the imbalanced domain. In both the internal workings as well as the final classification step of IPADE-ID, we are using the $k$NN classifier. This makes for a fair comparison with EPRENNID and the data sampling methods, as they use $k$NN as well.
- Evolutionary undersampling (EUS [16]): Based on the PS method CHC [68], the authors proposed a modified version to handle class imbalance, similar to what has been done in this paper for the SSMA method. They developed two settings, that either focus on balancing the dataset (EBUS) or optimizing the classification performance (EUSCM). Furthermore, they provide an option to perform a global selection (GS), removing instances of both classes in the reduction process, or majority selection (MS), solely reducing the majority class. Their custom fitness functions use either the AUC or g_mean to evaluate the classification performance. We have selected the best performing setting, the EBUS-MS method, using g_mean in its fitness evaluations.

In addition to these data sampling models, some representative ensemble-based techniques are also considered:

- SMOTEBagging [45]: This bagging procedure constructs bootstrap samples of the training set by applying the SMOTE oversampling method. The resampled version of the majority class is always obtained via random resampling. In sampling the minority class for a bootstrap sample, a given percentage is obtained via random resampling, while the other part is constructed by creating synthetic minority instances with SMOTE. The percentage of random resampling is varied between the different bootstrap samples. A balance between the two classes is guaranteed in each sample.
- SMOTEBoost [44]: This ensemble method uses SMOTE in each iteration of the AdaBoost.M2 boosting algorithm [69]. The oversampling step creates synthetic minority elements in order to better represent previously misclassified minority instances, thereby implicitly increasing their weights in the current interation.
- EUSBoost [48]: Similar to SMOTEBoost, EUSBoost embeds the EUS method described above in AdaBoost.M2. In each boosting round, the undersampling step reduces the majority class to a subset.

The parameter settings of EPRENNID are presented in Table 3. $SSMA_{Imb}$ is a modified version of SSMA and we have used the same default parameters as the latter method, as proposed in [27]. In the PG step, we use the default parameters used in [23] for $SFLSDE_{Imb}$. For more detail on these parameter values, we refer to [23,28]. In the hybridization step, we use 40 populations of 10 individuals each. The populations are kept small on purpose, to avoid that they all converge to the same solution, which would result in a loss of diversity. In order to provide a more global picture, we have set $k$ to 1, 3 and 5 for the $k$NN classifier used in the final classification step.

### 4.3. Evaluation measures

We evaluate the classification performance by the two popular methods discussed in Section 2.2: the AUC and g_mean. All results are obtained by means of five-fold stratified cross validation. We note that both EPRENNID and all included methods, apart from NCR and Spider2, contain random components. To account for this degree of randomness, we repeat the experiments 10 times and report the averages and standard deviation over these 10 runs.

### 4.4. Statistical analysis

In order to test for significance in the observed differences in the experimental results, we apply non-parametric statistical tests, as recommended in [70,71]. We use the Friedman test [72] to verify whether any significant differences in performance are present among a group of methods. When the $p$-value of this test is lower than a specified significance level $\alpha$, the null hypothesis of equivalent performance is rejected and we conclude that significant differences exist among the methods.

To determine where these significant differences occur, we apply the Nemenyi post hoc test. In this test, the performance of two classifiers is significantly different only if their average ranks differ by a certain critical distance. The critical distance depends

on the number of algorithms, the number of datasets and the critical value for a significance level provided by a Studentized range statistic. The result of the Nemenyi post hoc test is plotted with an average ranks diagram. The ranks are depicted on the axis, so that the best algorithms are at the right side of the diagram. A line with the length of the critical distance is drawn between those algorithms that do not differ significantly (in performance) for a significance level of $\alpha = 0.05$. More information about these tests and other statistical procedures can be found at http://sci2s.ugr.es/sicidm/.

## 5. Experimental results

We have conducted a thorough experimental study comparing our method to the current state-of-the-art in imbalanced classification. This section presents and interprets our experimental results, including a statistical analysis. In Section 5.1, we consider the internal reduction associated with our model. Section 5.2 presents an initial overview of the classification results. We divide the further discussion of these results into two main parts. Section 5.3 compares EPRENNID with data sampling models, while Section 5.4 presents a comparison with ensemble-based models. Due to the extent of our experimental analysis, we are unable to list all results here. The complete results are reported on the associated web page http://www.cwi.ugent.be/sarah.php.

### 5.1. A note on reduction

Before discussing the classification performance of our proposal in detail, we briefly note that the average reduction of the prototype sets in the ensemble in EPRENNID is 0.6413 (dense), 0.8123 (medium), 0.8904 (sparse) and 0.7733 (all). The global reduction after the SSMA$_{Imb}$ step however is only 0.3820 (dense), 0.6108 (medium), 0.7393 (sparse) and 0.5662 (all). The global reduction is computed by taking the union of the prototype sets and comparing it to the full training set. Since we guarantee a level of diversity between the sets in the ensemble, the global reduction is noticeably lower than the average reduction. Since reduction is not the most relevant measure for our method, we do not further compare these values with those obtained by the data sampling methods and instead focus on the classification performance.

### 5.2. Overview of results

In Table 4, we present a compact overview of the classification results of all included methods, using both the AUC and g_mean as evaluation measures. As noted above, we consider three different values for $k$ in the classification step of EPRENNID. The data sampling methods are combined with 1NN, 3NN and 5NN. The ensemble methods are only evaluated for $k = 1$, as motivated in Section 5.4. We list the average values for each dataset group, combined with the average standard deviation over 10 runs where applicable. The reader can refer back to this table throughout our discussion in the remainder of the paper. The full results can be consulted at http://www.cwi.ugent.be/sarah.php.

### 5.3. Comparison with data sampling models

In this section, we compare EPRENNID to the selected data sampling methods. In addition to Table 4, Table 5 lists the full results of the classification by 1NN evaluated using the AUC. The

**Table 4**
Overview of all classification results.

| 1NN | AUC | | | g_mean | | |
|---|---|---|---|---|---|---|
| | Dense | Medium | Sparse | Dense | Medium | Sparse |
| EPRENNID | 0.9626 ± 0.003 | 0.9514 ± 0.004 | 0.9020 ± 0.011 | 0.9289 ± 0.006 | 0.8651 ± 0.017 | 0.7372 ± 0.018 |
| Border2 | 0.9291 ± 0.004 | 0.8135 ± 0.018 | 0.6926 ± 0.053 | 0.9294 ± 0.004 | 0.8130 ± 0.019 | 0.693 ± 0.056 |
| SMT-TL | 0.9313 ± 0.005 | 0.8575 ± 0.019 | 0.7763 ± 0.029 | 0.9311 ± 0.005 | 0.8574 ± 0.020 | 0.776 ± 0.030 |
| SMT-RSB | 0.9267 ± 0.004 | 0.8154 ± 0.026 | 0.7084 ± 0.043 | 0.9269 ± 0.004 | 0.8160 ± 0.029 | 0.708 ± 0.045 |
| NCR | 0.9263 | 0.8358 | 0.7400 | 0.9228 | 0.8067 | 0.747 |
| Spider2 | 0.9181 | 0.8027 | 0.7110 | 0.9137 | 0.6425 | 0.568 |
| SSMA$_{Imb}$ | 0.9326 ± 0.005 | 0.8722 ± 0.017 | 0.7873 ± 0.034 | 0.9318 ± 0.005 | 0.8712 ± 0.018 | 0.782 ± 0.035 |
| IPADE | 0.9190 ± 0.009 | 0.8795 ± 0.022 | 0.8111 ± 0.026 | 0.9168 ± 0.010 | 0.8687 ± 0.032 | 0.783 ± 0.044 |
| EUS | 0.9276 ± 0.007 | 0.8737 ± 0.014 | 0.7836 ± 0.028 | 0.9272 ± 0.008 | 0.8734 ± 0.015 | 0.785 ± 0.030 |
| SMOTEBagging | 0.9544 ± 0.006 | 0.9210 ± 0.010 | 0.8516 ± 0.014 | 0.9285 ± 0.007 | 0.8536 ± 0.013 | 0.717 ± 0.016 |
| SMOTEBoost | 0.9419 ± 0.014 | 0.8671 ± 0.036 | 0.7797 ± 0.036 | 0.8441 ± 0.061 | 0.3677 ± 0.148 | 0.262 ± 0.096 |
| EUSBoost | 0.9329 ± 0.015 | 0.8580 ± 0.056 | 0.7254 ± 0.073 | 0.8828 ± 0.050 | 0.6250 ± 0.227 | 0.349 ± 0.174 |
| 3NN | | | | | | |
| EPRENNID | 0.9637 ± 0.002 | 0.9522 ± 0.006 | 0.8845 ± 0.013 | 0.9217 ± 0.004 | 0.8752 ± 0.014 | 0.7412 ± 0.028 |
| Border2 | 0.9367 ± 0.010 | 0.8372 ± 0.049 | 0.7257 ± 0.069 | 0.9368 ± 0.011 | 0.8378 ± 0.052 | 0.7257 ± 0.073 |
| SMT-TL | 0.9399 ± 0.007 | 0.8844 ± 0.029 | 0.8161 ± 0.037 | 0.9397 ± 0.008 | 0.8843 ± 0.030 | 0.8166 ± 0.040 |
| SMT-RSB | 0.9409 ± 0.009 | 0.8661 ± 0.031 | 0.7431 ± 0.049 | 0.9406 ± 0.009 | 0.8663 ± 0.032 | 0.7435 ± 0.052 |
| NCR | 0.9393 | 0.8847 | 0.7825 | 0.9152 | 0.7628 | 0.5531 |
| Spider2 | 0.9487 | 0.8624 | 0.7484 | 0.9182 | 0.7844 | 0.6111 |
| SSMA$_{Imb}$ | 0.9417 ± 0.011 | 0.8985 ± 0.034 | 0.8150 ± 0.044 | 0.9350 ± 0.012 | 0.8885 ± 0.035 | 0.7972 ± 0.048 |
| IPADE | 0.8575 ± 0.035 | 0.7876 ± 0.058 | 0.7447 ± 0.052 | 0.4011 ± 0.225 | 0.5416 ± 0.210 | 0.4436 ± 0.205 |
| EUS | 0.9378 ± 0.013 | 0.8967 ± 0.028 | 0.7961 ± 0.044 | 0.9374 ± 0.014 | 0.8967 ± 0.030 | 0.7959 ± 0.047 |
| 5NN | | | | | | |
| EPRENNID | 0.9601 ± 0.002 | 0.9457 ± 0.006 | 0.8763 ± 0.012 | 0.9137 ± 0.006 | 0.8651 ± 0.019 | 0.6921 ± 0.038 |
| Border2 | 0.9396 ± 0.014 | 0.8553 ± 0.052 | 0.7488 ± 0.071 | 0.9397 ± 0.014 | 0.8553 ± 0.055 | 0.7488 ± 0.075 |
| SMT-TL | 0.9443 ± 0.010 | 0.9019 ± 0.019 | 0.8302 ± 0.039 | 0.9439 ± 0.011 | 0.9015 ± 0.020 | 0.8298 ± 0.042 |
| SMT-RSB | 0.9429 ± 0.012 | 0.8816 ± 0.033 | 0.7584 ± 0.058 | 0.9430 ± 0.013 | 0.8821 ± 0.035 | 0.7589 ± 0.062 |
| NCR | 0.9479 | 0.8998 | 0.7982 | 0.9048 | 0.6954 | 0.4496 |
| Spider2 | 0.9466 | 0.8851 | 0.7662 | 0.9189 | 0.7847 | 0.6267 |
| SSMA$_{Imb}$ | 0.9414 ± 0.017 | 0.8915 ± 0.042 | 0.7967 ± 0.057 | 0.9331 ± 0.018 | 0.8767 ± 0.047 | 0.7776 ± 0.062 |
| IPADE | 0.8760 ± 0.032 | 0.8153 ± 0.048 | 0.7582 ± 0.046 | 0.4746 ± 0.220 | 0.6184 ± 0.164 | 0.5153 ± 0.176 |
| EUS | 0.9387 ± 0.017 | 0.9001 ± 0.033 | 0.7868 ± 0.051 | 0.9378 ± 0.018 | 0.8999 ± 0.035 | 0.7873 ± 0.056 |

**Table 5**
Classification performance of 1NN evaluated by the AUC of the classifier in all the datasets. The results of all algorithms, apart from NCR and Spider2, were taken as averages over 10 runs. We report the corresponding standard deviation as well. The best results in each group of datasets is printed in bold.

| | EPRENNID | Border2 | SMT-TL | SMT-RSB | IPADE | $SSMA_{Imb}$ | EUS | NCR | Spider2 |
|---|---|---|---|---|---|---|---|---|---|
| abalone-3vs11 | 1.0000 ± 0.000 | 1.0000 ± 0.000 | 1.0000 ± 0.000 | 1.0000 ± 0.000 | 0.9993 ± 0.001 | 0.9988 ± 0.000 | 0.9990 ± 0.001 | 1.0000 | 1.0000 |
| ecoli4 | 0.9743 ± 0.015 | 0.9059 ± 0.013 | 0.9035 ± 0.013 | 0.9110 ± 0.011 | 0.9231 ± 0.026 | 0.9078 ± 0.007 | 0.8913 ± 0.019 | 0.8623 | 0.8702 |
| glass6 | 0.9472 ± 0.013 | 0.8654 ± 0.006 | 0.8813 ± 0.008 | 0.8670 ± 0.009 | 0.9096 ± 0.022 | 0.9164 ± 0.009 | 0.8817 ± 0.027 | 0.8852 | 0.8659 |
| haberman | 0.6422 ± 0.009 | 0.5866 ± 0.020 | 0.5852 ± 0.018 | 0.5555 ± 0.015 | 0.6060 ± 0.015 | 0.5657 ± 0.015 | 0.5558 ± 0.008 | 0.5727 | 0.5734 |
| iris0 | 1.0000 ± 0.000 | 1.0000 ± 0.000 | 1.0000 ± 0.000 | 1.0000 ± 0.000 | 0.9995 ± 0.002 | 1.0000 ± 0.000 | 1.0000 ± 0.000 | 1.0000 | 1.0000 |
| kddcup-bovsb | 1.0000 ± 0.000 | 1.0000 ± 0.000 | 1.0000 ± 0.000 | 1.0000 ± 0.000 | 0.9999 ± 0.000 | 1.0000 ± 0.000 | 1.0000 ± 0.000 | 1.0000 | 1.0000 |
| new-thyroid1 | 0.9985 ± 0.000 | 0.9887 ± 0.005 | 0.9813 ± 0.008 | 0.9800 ± 0.006 | 0.9785 ± 0.008 | 0.9767 ± 0.008 | 0.9871 ± 0.003 | 0.9917 | 0.9663 |
| page-blocks0 | 0.9766 ± 0.002 | 0.9074 ± 0.002 | 0.9239 ± 0.003 | 0.9133 ± 0.002 | 0.8646 ± 0.005 | 0.9142 ± 0.014 | 0.9137 ± 0.012 | 0.9096 | 0.8923 |
| segment0 | 0.9996 ± 0.001 | 0.9947 ± 0.000 | 0.9946 ± 0.000 | 0.9947 ± 0.000 | 0.9626 ± 0.005 | 0.9905 ± 0.001 | 0.9882 ± 0.001 | 0.9939 | 0.9856 |
| shuttle2vs5 | 1.0000 ± 0.000 | 1.0000 ± 0.000 | 1.0000 ± 0.000 | 1.0000 ± 0.000 | 0.9995 ± 0.000 | 1.0000 ± 0.000 | 1.0000 ± 0.000 | 1.0000 | 1.0000 |
| texture2redvs34 | 0.9884 ± 0.003 | 0.9026 ± 0.003 | 0.9104 ± 0.007 | 0.9162 ± 0.003 | 0.9034 ± 0.015 | 0.9416 ± 0.005 | 0.9432 ± 0.014 | 0.9068 | 0.8953 |
| vehicle2 | 0.9960 ± 0.001 | 0.9607 ± 0.002 | 0.9513 ± 0.003 | 0.9538 ± 0.003 | 0.8279 ± 0.012 | 0.9366 ± 0.008 | 0.9313 ± 0.004 | 0.9430 | 0.9108 |
| wisconsin | 0.9914 ± 0.001 | 0.9659 ± 0.002 | 0.9756 ± 0.002 | 0.9556 ± 0.002 | 0.9728 ± 0.003 | 0.9751 ± 0.003 | 0.9676 ± 0.004 | 0.9769 | 0.9749 |
| Mean **dense** | 0.9626 ± 0.003 | 0.9291 ± 0.004 | 0.9313 ± 0.005 | 0.9267 ± 0.004 | 0.9190 ± 0.009 | 0.9326 ± 0.005 | 0.9276 ± 0.007 | 0.9263 | 0.9181 |
| abalone17vs78910 | 0.9353 ± 0.004 | 0.5912 ± 0.052 | 0.7492 ± 0.018 | 0.7110 ± 0.023 | 0.8334 ± 0.009 | 0.7999 ± 0.025 | 0.7808 ± 0.018 | 0.6932 | 0.6298 |
| appendicitis | 0.7722 ± 0.010 | 0.7540 ± 0.005 | 0.7292 ± 0.013 | 0.7344 ± 0.013 | 0.7581 ± 0.021 | 0.7268 ± 0.010 | 0.7069 ± 0.007 | 0.7115 | 0.7218 |
| cleveland0vs4 | 0.9690 ± 0.005 | 0.7777 ± 0.020 | 0.8428 ± 0.024 | 0.6271 ± 0.099 | 0.8636 ± 0.049 | 0.8485 ± 0.044 | 0.9171 ± 0.022 | 0.8325 | 0.7116 |
| ecoli3 | 0.9208 ± 0.009 | 0.8137 ± 0.009 | 0.8502 ± 0.009 | 0.8137 ± 0.018 | 0.8775 ± 0.020 | 0.8418 ± 0.013 | 0.8391 ± 0.010 | 0.8150 | 0.8398 |
| glass4 | 0.9728 ± 0.004 | 0.8617 ± 0.007 | 0.8883 ± 0.017 | 0.8292 ± 0.030 | 0.8597 ± 0.031 | 0.8776 ± 0.008 | 0.8883 ± 0.018 | 0.8892 | 0.8158 |
| movementlibras1 | 0.9911 ± 0.006 | 0.9358 ± 0.003 | 0.9024 ± 0.004 | 0.9474 ± 0.012 | 0.9121 ± 0.024 | 0.9340 ± 0.003 | 0.9518 ± 0.010 | 0.9376 | 0.9118 |
| segment6redvs345 | 0.9997 ± 0.000 | 0.7953 ± 0.054 | 0.8846 ± 0.064 | 0.7948 ± 0.054 | 0.9719 ± 0.026 | 0.9553 ± 0.028 | 0.9308 ± 0.005 | 0.8641 | 0.8162 |
| shuttle67vs1redB | 1.0000 ± 0.000 | 1.0000 ± 0.000 | 1.0000 ± 0.000 | 1.0000 ± 0.000 | 0.9750 ± 0.015 | 0.9817 ± 0.011 | 0.9837 ± 0.012 | 0.9750 | 0.9750 |
| vehicle0 | 0.9852 ± 0.001 | 0.9332 ± 0.002 | 0.9381 ± 0.003 | 0.9282 ± 0.003 | 0.8884 ± 0.014 | 0.9212 ± 0.001 | 0.9131 ± 0.007 | 0.9216 | 0.9132 |
| wdbc-MredBvsB | 0.9960 ± 0.001 | 0.8194 ± 0.014 | 0.8772 ± 0.040 | 0.8941 ± 0.005 | 0.9269 ± 0.033 | 0.8859 ± 0.031 | 0.8769 ± 0.039 | 0.7986 | 0.8000 |
| yeast4 | 0.9232 ± 0.004 | 0.6666 ± 0.033 | 0.7710 ± 0.018 | 0.6900 ± 0.029 | 0.8082 ± 0.005 | 0.8220 ± 0.015 | 0.8217 ± 0.012 | 0.7552 | 0.6944 |
| Mean **medium** | 0.9514 ± 0.004 | 0.8135 ± 0.018 | 0.8575 ± 0.019 | 0.8154 ± 0.026 | 0.8795 ± 0.022 | 0.8722 ± 0.017 | 0.8737 ± 0.014 | 0.8358 | 0.8027 |
| abalone20vs8910 | 0.9175 ± 0.012 | 0.6475 ± 0.043 | 0.7714 ± 0.018 | 0.7251 ± 0.026 | 0.8712 ± 0.018 | 0.7748 ± 0.019 | 0.8020 ± 0.030 | 0.6862 | 0.6348 |
| ecoli0147vs2356 | 0.9439 ± 0.005 | 0.8258 ± 0.006 | 0.8799 ± 0.010 | 0.8244 ± 0.007 | 0.8853 ± 0.016 | 0.8956 ± 0.013 | 0.8754 ± 0.017 | 0.8707 | 0.8320 |
| glass5 | 0.9902 ± 0.003 | 0.8377 ± 0.050 | 0.8298 ± 0.050 | 0.8377 ± 0.050 | 0.8633 ± 0.069 | 0.8920 ± 0.075 | 0.8537 ± 0.066 | 0.9280 | 0.8854 |
| ionosphere-bredvsg | 0.8556 ± 0.033 | 0.6852 ± 0.062 | 0.7606 ± 0.058 | 0.5352 ± 0.112 | 0.7048 ± 0.061 | 0.7431 ± 0.068 | 0.7117 ± 0.061 | 0.6478 | 0.6478 |
| magic-hredvsgred | 0.8723 ± 0.010 | 0.5327 ± 0.074 | 0.7446 ± 0.018 | 0.6127 ± 0.044 | 0.7763 ± 0.025 | 0.7271 ± 0.026 | 0.7158 ± 0.006 | 0.6563 | 0.5691 |
| phoneme-1redvs0red | 0.8817 ± 0.007 | 0.5491 ± 0.068 | 0.7106 ± 0.024 | 0.6154 ± 0.047 | 0.7598 ± 0.012 | 0.7605 ± 0.019 | 0.7625 ± 0.014 | 0.6079 | 0.5998 |
| shuttle6vs23 | 0.9786 ± 0.023 | 0.9349 ± 0.020 | 0.9566 ± 0.023 | 0.9349 ± 0.023 | 0.9600 ± 0.021 | 0.9271 ± 0.023 | 0.8808 ± 0.026 | 0.9000 | 0.9000 |
| wdbc-MredvsB | 0.9969 ± 0.001 | 0.9457 ± 0.004 | 0.9446 ± 0.005 | 0.8914 ± 0.009 | 0.9469 ± 0.002 | 0.8784 ± 0.057 | 0.9466 ± 0.038 | 0.9000 | 0.9000 |
| winequality-red4 | 0.7190 ± 0.009 | 0.3641 ± 0.157 | 0.5939 ± 0.028 | 0.5144 ± 0.060 | 0.6528 ± 0.016 | 0.6091 ± 0.026 | 0.6265 ± 0.009 | 0.5658 | 0.5243 |
| winequality-white-3vs7 | 0.9250 ± 0.007 | 0.5290 ± 0.089 | 0.5874 ± 0.076 | 0.5475 ± 0.084 | 0.6980 ± 0.039 | 0.6745 ± 0.042 | 0.6821 ± 0.031 | 0.5710 | 0.5409 |
| yeast0256s3789 | 0.8413 ± 0.008 | 0.7665 ± 0.010 | 0.7597 ± 0.005 | 0.7533 ± 0.007 | 0.8039 ± 0.008 | 0.7784 ± 0.006 | 0.7620 ± 0.016 | 0.8059 | 0.7870 |
| Mean **sparse** | 0.9020 ± 0.011 | 0.6926 ± 0.053 | 0.7763 ± 0.029 | 0.7084 ± 0.043 | 0.8111 ± 0.026 | 0.7873 ± 0.034 | 0.7836 ± 0.028 | 0.7400 | 0.7110 |

data sampling methods were combined with 1NN and the value of $k$ in the classification step of EPRENNID was set to 1 as well. For each group of datasets, the result of the best-performing method is printed in bold. In order not to clutter this discussion, we do not list the complete results of the g_mean measure nor for 3NN and 5NN. The results of the statistical analyses relevant to this section can be found in Fig. 2, plotting the average ranks diagrams. We note that we have taken the entire group of 35 datasets to perform the statistical analysis, rather than doing this group-wise, as the sizes of the groups are rather small. We discuss the results of the two evaluation measures, AUC and g_mean, separately in Sections 5.3.1 and 5.3.2 respectively. In Section 5.3.3, we compare EPRENNID to the data sampling methods in terms of their runtime.

### 5.3.1. *Analysis of the AUC results*

The conducted experimental study represented in Fig. 2 suggests that EPRENNID dominates all previous proposals with respect to the AUC for all values of $k$. For 1NN, the IPADE-ID method outperforms most data sampling methods, showing the strength of DE in the imbalanced domain, but it is further improved upon by EPRENNID. For higher values of $k$, IPADE-ID does not perform well, being dominated by all data sampling methods, while EPRENNID still comes out on top.

An interesting point to observe is that the absolute differences in the obtained values increase with the difficulty of the datasets, going from the dense to the sparse groups. This conclusion is similar to the one drawn in the experimental study of [35], which also investigated the effect of the distribution of the minority instances on the performance of several methods. In particular, taking the results of 1NN in Table 5 as an example, we observe that the difference between the best and worst performing methods increases from 0.0445 for the dense group, over 0.1487 for the medium group to 0.2094 for the sparse datasets. Furthermore, even though the AUC of EPRENNID does decrease with the difficulty of the dataset, this decrease is less prominent than for other methods. For instance, while EPRENNID only loses 0.0606, its closest competitors IPADE-ID and EUS face a decrease in AUC of about 0.1079 and 0.1440. Although their dataset-wise results are not printed here, similar conclusions can be drawn for 3NN and 5NN. This observation shows the good performance of EPRENNID for all types of class imbalance.

We note that our new PS method $SSMA_{Imb}$ also performs tolerably well, most prominently so for 1NN. For this classifier, it yields better results than all included data sampling methods, putting it at the same level as IPADE-ID. It is interesting to observe that $SSMA_{Imb}$, a true undersampling method removing both positive and negative elements, is able to outperform undersampling (NCR, EUS), oversampling (Borderline-SMOTE2) and hybrid

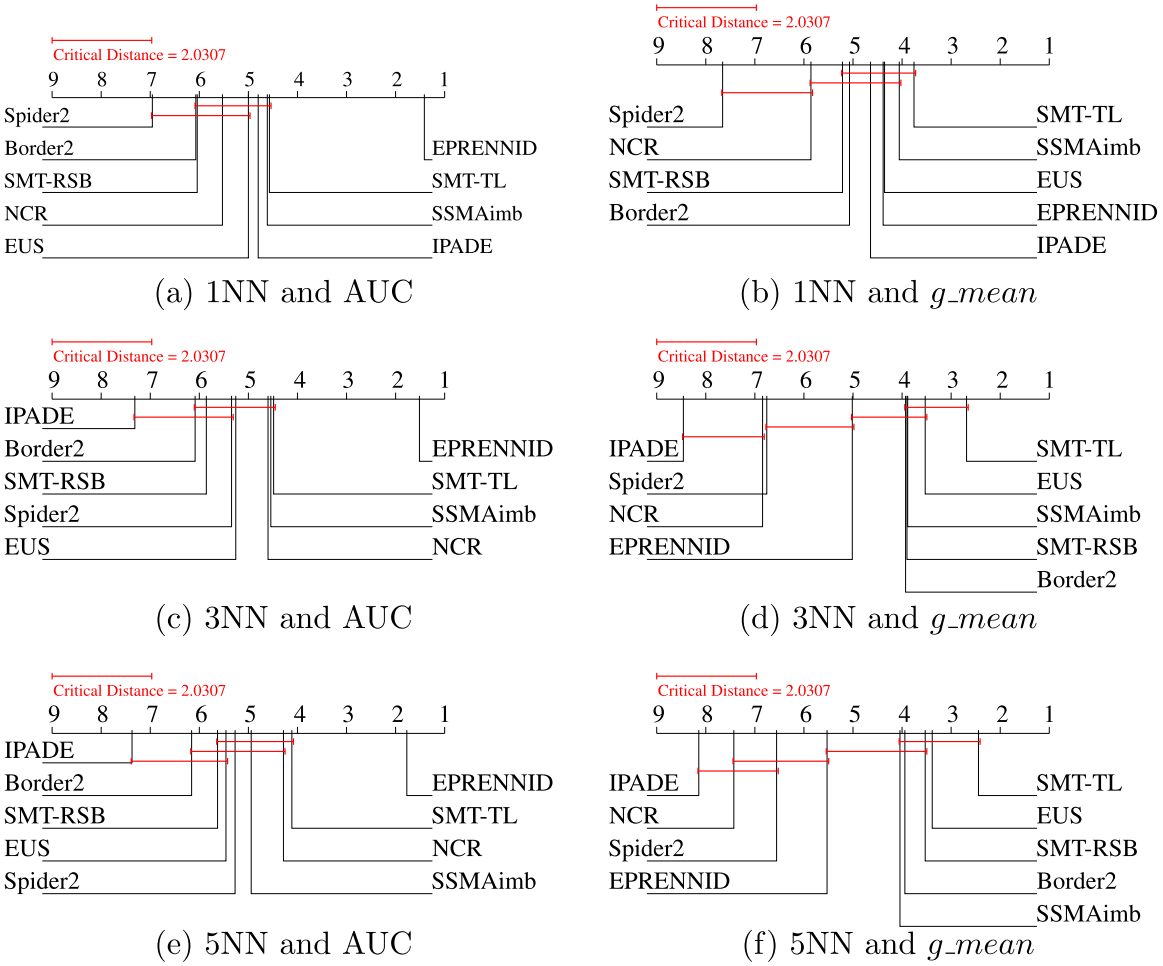


**Fig. 2.** Average ranks diagrams for AUC and g_mean using 1NN, 3NN and 5NN classifiers. Better algorithms are located on the right side of the plot (rank closer to 1). Those that differ by less than the critical distance computed for a $p$-value = 0.05 are linked by a red line.

(SMOTE-TL, SMOTE-RSB, Spider2) data sampling methods. This constitutes clear evidence that the complete protection of the minority class, incorporated by all these methods, is not necessarily justified. Allowing the removal of minority elements, which can also be noisy or redundant, provides us with an added flexibility, which makes it possible to handling class imbalance more appropriately. For higher values of $k$, SSMA$_{Imb}$ remains steadily at the top and, apart from by EPRENNID, is only improved by NCR and SMOTE-TL for 5NN. We conclude that we have proposed a strong PS method able to handle class imbalance, but it can nevertheless be further improved by hybridizing it with PG and including the ensemble classification. From Fig. 2, we observe that, for all classifiers, our method has the best rank with respect to the AUC values. Comparing EPRENNID to the others with the Nemenyi post hoc test, we conclude that it yields significantly better results than all other methods under consideration, since none of the others is located within the critical distance of our proposal. This statement holds for all three classifiers and confirms the clear dominance of our new method over the state-of-the-art in data sampling.

Finally, in Fig. 3, we visually compare the full EPRENNID model to three partially constructed ones, in order to determine whether the added complexity of the full model increases its performance. The figure is based on the performance of 1NN evaluated by the AUC. We already observed that solely applying SSMA$_{Imb}$, generating a single prototype set, yields good classification results, although they are improved upon by EPRENNID. Furthermore, optimizing this single subset by SFLSDE$_{Imb}$ does not on its own

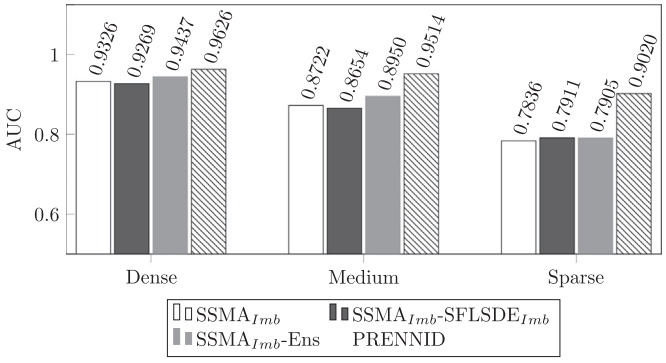


**Fig. 3.** Comparison of EPRENNID with partially constructed models. In SSMA$_{Imb}$, only the PS step is performed. SSMA$_{Imb}$+SFLSDE$_{Imb}$ is the same as EPRENNID, apart from the important fact that only one prototype set is constructed. This comparison was done for 1NN, of which the performance was evaluated by the AUC.

increase the performance, as presented by SSMA$_{Imb}$-SFLSDE$_{Imb}$ in the figure. This setup results in a slight decrease in performance, which we suspect to be due to the overfitting problem to which PG methods are prone [20]. We also consider the extension of SSMA$_{Imb}$ with the ensemble approach in EPRENNID, without optimizing the prototype sets by SFLSDE$_{Imb}$. This setting is represented by SSMA$_{Imb}$-Ens. Although giving an improvement over SSMA$_{Imb}$, it is itself clearly improved upon by introducing the optimization step by PG. The optimization of multiple diverse

prototype sets and their aggregation into a classification ensemble is shown to be truly worth the effort.

*5.3.2. Analysis of the g_mean results*

With respect to the evaluation by g_mean, the results are less favorable for EPRENNID, as shown in Table 4. For 1NN, we observe that EPRENNID still outperforms several state-of-the-art data sampling methods, but it is itself outperformed by the IPADE-ID method for medium and sparse datasets. The undersampling method EUS combined with 1NN yields better average results than EPRENNID as well. Our PS method $SSMA_{Imb}$ still exhibits very good overall behavior, again proving its obvious strength for imbalanced classification.

In combination with 3NN, we observe that the hybrid data sampling methods SMOTE-TL and $SMOTE\text{-}RSB_*$ perform better than before, placing them at the same level as EPRENNID, IPADE-ID, $SSMA_{Imb}$ and EUS. For 5NN, the hybrid data sampling methods, especially SMOTE-TL, dominate. EPRENNID yields decent results, although its overall average result for g_mean is lower than that of $SSMA_{Imb}$. Considering this phenomenon more closely, we observed that this is due to a decrease in performance of EPRENNID on sparse datasets, for which its average g_mean value is considerably lower than that of $SSMA_{Imb}$, as can be seen in Table 4. In a sparse dataset, the minority class is severely spread out over the feature space, making it more difficult for *k*NN to classify them correctly, especially for higher values of *k*. By using the ensemble approach in EPRENNID, misclassifications can build up, resulting in a decrease in performance. We conclude that in such a setting, where higher values of *k* are used to classify a sparse dataset with the *k*NN rule, it might be more appropriate to stick to the preprocessing method $SSMA_{Imb}$.

In the statistical analysis of the g_mean values (Fig. 2), SMOTE-TL is assigned the best rank for all values of *k*. For 1NN, 3NN and 5NN, SMOTE-TL is shown to significantly outperform NCR and Spider2. For 3NN and 5NN, it also performs significantly better than IPADE and EPRENNID. As noted above, upon closer examination it is revealed that the poor average result of EPRENNID in this case is due to an inferior performance on the sparse datasets.

Nevertheless, by taking the results of both evaluation measures into account, we can conclude that our new proposal of a hybrid model, integrating PS and PG in its preprocessing step and using a weighted voting procedure in its classification, is competitive with the state-of-the-art data sampling methods as well as IPADE-ID.

*5.3.3. Runtime analysis*

In this section, we present the runtimes of the different data sampling methods and our proposal, for both the preprocessing and classification steps. The results for 1NN can be found in Table 6, presenting the average group-wise runtime for EPRENNID and the data sampling methods. The values for 3NN and 5NN, for which the conclusions are similar, are available on http://www.cwi.ugent.be/sarah.php. We distinguish between the preprocessing and classification times. The former, given in seconds, refers to the necessary time to balance the dataset (data sampling methods) or the full construction of the prototype set ensemble (EPRENNID). The latter, which is given in milliseconds, is the average time spent to label one test instance based on the preprocessed dataset (data sampling methods) or using the prepared ensemble in a weighted vote (EPRENNID). We note that this comparison is somewhat unfair towards EPRENNID, as our proposal constructs an entire classification ensemble, while the data sampling methods merely resample the dataset. As such, it can be expected that the preprocessing time for our proposal is longer, as

**Table 6**
Runtime results for the data sampling methods using the 1NN as classifier.

| Preprocessing (s) | Dense | Medium | Sparse |
|---|---|---|---|
| EPRENNID | 6285.093 ± 1089.312 | 702.119 ± 50.100 | 669.144 ± 97.315 |
| Border2 | 0.251 ± 0.012 | 0.091 ± 0.011 | 0.075 ± 0.010 |
| SMT-TL | 3.283 ± 0.013 | 0.837 ± 0.008 | 1.065 ± 0.007 |
| SMT-RSB | 3.176 ± 0.137 | 1.732 ± 0.072 | 2.084 ± 0.061 |
| IPADE | 137.123 ± 51.000 | 5.891 ± 2.852 | 5.697 ± 2.798 |
| $SSMA_{Imb}$ | 113.327 ± 10.907 | 25.045 ± 2.116 | 39.547 ± 3.501 |
| EUS | 1152.029 ± 14.996 | 170.549 ± 2.473 | 129.586 ± 2.667 |
| NCR | 0.995 ± 0.039 | 0.279 ± 0.025 | 0.339 ± 0.031 |
| Spider2 | 2.212 ± 0.103 | 0.554 ± 0.043 | 0.681 ± 0.044 |
| **Classification (ms)** | **Dense** | **Medium** | **Sparse** |
| EPRENNID | 137.997 ± 28.917 | 70.786 ± 8.813 | 18.261 ± 4.164 |
| Border2 | 488.872 ± 47.487 | 143.804 ± 14.858 | 145.282 ± 15.684 |
| SMT-TL | 473.463 ± 37.858 | 109.449 ± 9.733 | 142.340 ± 14.580 |
| SMT-RSB | 485.849 ± 42.917 | 142.452 ± 13.077 | 144.506 ± 17.950 |
| IPADE | 5.075 ± 1.043 | 3.735 ± 0.721 | 3.031 ± 0.558 |
| $SSMA_{Imb}$ | 36.198 ± 2.991 | 7.298 ± 1.504 | 5.876 ± 1.137 |
| EUS | 36.215 ± 2.541 | 7.582 ± 1.036 | 6.304 ± 0.889 |
| NCR | 251.820 ± 25.394 | 57.258 ± 5.336 | 72.309 ± 7.640 |
| Spider2 | 278.728 ± 24.861 | 83.234 ± 9.049 | 84.696 ± 8.225 |

**Table 7**
AUC results for the ensemble-based models using 1NN as base classifier, with standard deviations over 10 runs.

| Dataset | SMOTEBagging | EUSBoost | SMOTEBoost |
|---|---|---|---|
| abalone-3vs11 | 0.9995 ± 0.0010 | 0.9982 ± 0.0014 | 1.0000 ± 0.0000 |
| ecoli4 | 0.9564 ± 0.0104 | 0.9029 ± 0.0208 | 0.9563 ± 0.0250 |
| glass6 | 0.9186 ± 0.0250 | 0.8976 ± 0.0488 | 0.9105 ± 0.0471 |
| haberman | 0.6136 ± 0.0127 | 0.5597 ± 0.0470 | 0.5761 ± 0.0518 |
| iris0 | 1.0000 ± 0.0000 | 1.0000 ± 0.0000 | 1.0000 ± 0.0000 |
| kddcup-bovsb | 1.0000 ± 0.0000 | 0.9990 ± 0.0009 | 1.0000 ± 0.0000 |
| new-thyroid1 | 0.9921 ± 0.0022 | 0.9881 ± 0.0038 | 0.9942 ± 0.0060 |
| page-blocks0 | 0.9659 ± 0.0044 | 0.9025 ± 0.0139 | 0.9248 ± 0.0082 |
| segment0 | 0.9967 ± 0.0059 | 0.9954 ± 0.0036 | 0.9969 ± 0.0012 |
| shuttle2vs5 | 1.0000 ± 0.0000 | 1.0000 ± 0.0000 | 1.0000 ± 0.0000 |
| texture2redvs34 | 0.9816 ± 0.0063 | 0.9526 ± 0.0288 | 0.9540 ± 0.0232 |
| vehicle2 | 0.9917 ± 0.0020 | 0.9551 ± 0.0163 | 0.9686 ± 0.0120 |
| wisconsin | 0.9905 ± 0.0029 | 0.9764 ± 0.0085 | 0.9637 ± 0.0105 |
| Mean **dense** | 0.9544 ± 0.0056 | 0.9329 ± 0.0149 | 0.9419 ± 0.0142 |
| abalone17vs78910 | 0.8753 ± 0.0108 | 0.7085 ± 0.0791 | 0.7531 ± 0.0230 |
| appendicitis | 0.7185 ± 0.0183 | 0.6673 ± 0.0886 | 0.7795 ± 0.0296 |
| cleveland0vs4 | 0.9491 ± 0.0128 | 0.9242 ± 0.0410 | 0.8402 ± 0.0775 |
| ecoli3 | 0.9058 ± 0.0135 | 0.8256 ± 0.0445 | 0.8252 ± 0.0378 |
| glass4 | 0.9707 ± 0.0046 | 0.8686 ± 0.0682 | 0.8570 ± 0.0841 |
| movementlibras1 | 0.9770 ± 0.0014 | 0.9303 ± 0.0628 | 0.9599 ± 0.0325 |
| segment6redvs345 | 0.9356 ± 0.0362 | 0.9437 ± 0.0421 | 0.9002 ± 0.0333 |
| shuttle67vs1redB | 0.9789 ± 0.0005 | 0.9463 ± 0.0534 | 0.9733 ± 0.0129 |
| vehicle0 | 0.9783 ± 0.0030 | 0.9309 ± 0.0194 | 0.9331 ± 0.0114 |
| wdbc-MredBvsB | 0.9914 ± 0.0023 | 0.9272 ± 0.0445 | 0.9517 ± 0.0288 |
| yeast4 | 0.8507 ± 0.0100 | 0.7648 ± 0.0696 | 0.7647 ± 0.0230 |
| Mean **medium** | 0.9210 ± 0.0103 | 0.8580 ± 0.0557 | 0.8671 ± 0.0358 |
| abalone20vs8910 | 0.8359 ± 0.0184 | 0.6100 ± 0.0965 | 0.8024 ± 0.0286 |
| ecoli0147vs2356 | 0.9185 ± 0.0068 | 0.8801 ± 0.0521 | 0.8651 ± 0.0243 |
| glass5 | 0.9891 ± 0.0019 | 0.9127 ± 0.0374 | 0.9041 ± 0.0395 |
| ionosphere-bredvsg | 0.8771 ± 0.0167 | 0.7681 ± 0.0839 | 0.8650 ± 0.0357 |
| magic-hredvsgred | 0.7693 ± 0.0208 | 0.5455 ± 0.0819 | 0.6607 ± 0.0235 |
| phoneme-1redvs0red | 0.8150 ± 0.0115 | 0.6231 ± 0.0753 | 0.6730 ± 0.0196 |
| shuttle6vs23 | 0.9980 ± 0.0007 | 0.9618 ± 0.0285 | 0.9225 ± 0.0802 |
| wdbc-MredvsB | 0.9474 ± 0.0242 | 0.9659 ± 0.0460 | 0.8399 ± 0.0218 |
| winequality-red4 | 0.6414 ± 0.0187 | 0.5629 ± 0.0905 | 0.5674 ± 0.0297 |
| winequality-white3vs7 | 0.7410 ± 0.0198 | 0.5105 ± 0.1375 | 0.6713 ± 0.0743 |
| yeast0256vs3789 | 0.8350 ± 0.0113 | 0.6389 ± 0.0699 | 0.8057 ± 0.0164 |
| Mean **sparse** | 0.8516 ± 0.0137 | 0.7254 ± 0.0727 | 0.7797 ± 0.0358 |

is also clear in Table 6. However, we note that the preprocessing step only has to be executed once for each dataset. The resampled dataset or constructed ensemble can be reused in every subsequent classification. The classification time for EPRENNID is not high and indeed noticeably lower than that of SMOTE-TL, one of its closest competitors in terms of classification performance. Nevertheless, if the required runtime of EPRENNID cannot be afforded by the user, he can resort to our $SSMA_{Imb}$ method instead, which has a good classification performance (Table 4) and a reasonable runtime. In Section 5.4, we compare the runtime of EPRENNID to other ensemble methods, which makes for a fairer comparison.

### 5.4. Comparison with ensemble-based models

In this section, we compare EPRENNID to other ensemble-based models for imbalanced classification described in Section 4.2. We note that to establish a fair comparison between the different ensemble methods, we use the 1NN technique as base classifier in all cases. All ensembles use the same number of classifiers internally, set to the same value as the number of prototype sets used by EPRENNID.

Table 7 collects the results on all datasets, evaluating the classification performance of the 1NN classifier in terms of the AUC measure. As we saw in the comparison of EPRENNID with data sampling methods, our proposal dominates the other methods for all datasets groups. Its dominance becomes more apparent for increasing difficulty of the dataset, going from the dense, to the medium, to the sparse group. Fig. 4 depicts the statistical evaluation by means of average ranks diagrams. It shows that EPRENNID significantly outperforms SMOTEBagging, SMOTEBoost and EUSBoost with respect to the AUC obtained over all datasets. Considering the evaluation by g_mean (Table 4 and Fig. 4), our method also obtains the best average results and best rank in the statistical test. It is shown to be significantly better than the two boosting methods EUSBoost and SMOTEBoost.

We also comment on the computational complexity of EPRENNID compared to the other ensembles. In order to do so, Table 8 presents the average group-wise runtime required by each method, differentiating between building and classification times. The building time refers to the necessary time to create the ensemble, while the classification time involves the average time spent to label a test instance. The former is given in seconds, the latter in milliseconds. We observe that both are comparable between the four models. The classification time of EPRENNID is slightly higher than that of the other models, which is due to the target-specific weight construction for the prototype sets in the ensemble. This component is not present in the other ensemble-based methods. Nevertheless, taking the prediction results into account, EPRENNID may be preferred over the other three models, especially for imbalanced datasets with higher difficulty. Indeed, we observe that for the medium and sparse groups, the average building time of EPRENNID is the lowest among the four models and its classification performance is highly superior as well, as indicated in Table 4 and Fig. 4.

## 6. Conclusion and future work

In this work, we have introduced a new combined preprocessing and classification model able to cope with imbalanced data. Inspired by previous work in a balanced class setting, we proposed a new genetic PS model taking class imbalance into account. While most of the data sampling methods completely protect the minority class, the proposed model can reduce both majority and minority class examples, when necessary.

In our experiments, we were able to show that our new PS method outperforms several popular data sampling methods used in imbalanced classification. This shows that strenuously protecting minority elements is not necessarily the best option and including more flexible heuristics can prove to be more useful in dealing with class imbalance.

Secondly, we proposed to select not one, but a diverse set of

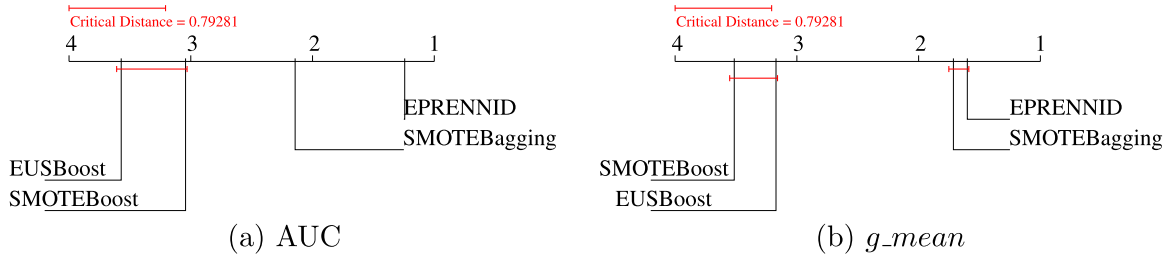


**Fig. 4.** Average ranks diagrams for AUC and g_mean using the 1NN classifier for the ensemble-based models. Better algorithms are located on the right side of the plot (rank closer to 1). Those that differ by less than the critical distance computed for a $p$-value = 0.05 are linked by a red line. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

**Table 8**
Runtime results for the ensemble-based methods.

| Building (s) | Dense | Medium | Sparse |
|---|---|---|---|
| EPRENNID | 6285.093 ± 1089.312 | 702.119 ± 50.100 | 669.144 ± 97.315 |
| SMOTEBagging | 2687.683 ± 438.284 | 1258.393 ± 86.260 | 1619.857 ± 111.510 |
| SMOTEBoost | 6895.590 ± 1112.069 | 2850.102 ± 252.999 | 3871.108 ± 386.373 |
| EUSBoost | 5901.096 ± 418.371 | 1053.477 ± 143.183 | 1158.963 ± 58.233 |
| **Classification (ms)** | **Dense** | **Medium** | **Sparse** |
| EPRENNID | 137.997 ± 28.917 | 70.786 ± 8.813 | 18.261 ± 4.164 |
| SMOTEBagging | 25.924 ± 6.102 | 28.398 ± 5.536 | 25.089 ± 4.736 |
| SMOTEBoost | 50.297 ± 10.121 | 50.409 ± 8.781 | 44.000 ± 8.653 |
| EUSBoost | 7.392 ± 0.990 | 2.968 ± 0.746 | 1.798 ± 0.224 |

well-performing prototype sets generated by the PS method. These sets were further optimized by a differential evolution scheme. As a final step, we set up an ensemble with the optimized prototype sets. The implemented voting strategy allows to assign prototype sets performing well in the neighborhood of a target instance a larger weight in its classification.

Our model does not aim to perform a significant data reduction of the dataset, but to increase the overall performance. Our experiments showed that it significantly outperforms state-of-the-art data sampling methods and ensemble-based methods, as well as a previous proposal using differential evolution in imbalanced classification, for the AUC measure. However, for the g_mean measure, it provides a similar performance in comparison to state-of-the-art models. In terms of computational cost, it is fairly similar to other ensemble-based methods.

As future work, we consider to study how an artificial injection of noisy examples may affect the behavior of our proposal. Moreover, we also intend to extend the presented approach to use other classifiers, like decision trees and support vector machines. This will require the development of custom prototype reduction methods for these classifiers in the class imbalance domain.

## Acknowledgments

The research of Sarah Vluymans is funded by the Special Research Fund (BOF) of Ghent University. Isaac Triguero held a BOF postdoctoral fellowship from Ghent University during part of the development of this work. This work was partially supported by the Spanish Ministry of Science and Technology under the Project TIN2014-57251-P and the Andalusian Research Plans P11-TIC-7765, P10-TIC-6858 and P12-TIC-2958, and by Project PYR-2014-8 of the Genil Program of CEI BioTic GRANADA.

## References

[1] Y. Lee, P. Hu, T. Cheng, T. Huang, W. Chuang, A preclustering-based ensemble learning technique for acute appendicitis diagnoses, Artif. Intell. Med. 58 (2) (2013) 115–124.
[2] H. Yu, J. Ni, J. Zhao, ACOSampling: an ant colony optimization-based undersampling method for classifying imbalanced dna microarray data, Neurocomputing 101 (2013) 309–318.
[3] C. Seiffert, T. Khoshgoftaar, J. Van Hulse, A. Folleco, An empirical study of the classification performance of learners on imbalanced and noisy software quality data, Inf. Sci. 259 (2014) 571–595.
[4] E. Alpaydin, Introduction to Machine Learning, 2nd ed., MIT Press, Cambridge, MA, 2010.
[5] N. Japkowicz, The class imbalance problem: significance and strategies, in: Proceedings of the 2000 International Conference on Artificial Intelligence, vol. 1, 2000, pp. 111–117.
[6] H. He, E. Garcia, Learning from imbalanced data, IEEE Trans. Knowl. Data Eng. 21 (9) (2009) 1263–1284.
[7] V. López, A. Fernández, S. García, V. Palade, F. Herrera, An insight into classification with imbalanced data: empirical results and current trends on using data intrinsic characteristics, Inf. Sci. 250 (2013) 113–141.
[8] M. Lin, K. Tang, X. Yao, Dynamic sampling approach to training neural networks for multiclass imbalance classification, IEEE Trans. Neural Netw. Learn. Syst. 24 (4) (2013) 647–660.
[9] G. Batista, R. Prati, M. Monard, A study of the behavior of several methods for balancing machine learning training data, SIGKDD Explor. 6 (1) (2004) 20–29.
[10] N. Chawla, K. Bowyer, L. Hall, W. Kegelmeyer, SMOTE: synthetic minority oversampling technique, J. Artif. Intell. Res. 16 (2002) 321–357.
[11] K. Ting, An instance-weighting method to induce cost-sensitive trees, IEEE Trans. Knowl. Data Eng. 14 (3) (2002) 659–665.
[12] D. Hand, V. Vinciotti, Choosing k for two-class nearest neighbour classifiers with unbalanced classes, Pattern Recognit. Lett. 24 (9) (2003) 1555–1562.
[13] D.,R. Wilson, T.,R. Martinez, Reduction techniques for instance-based learning algorithms, Mach. Learn. 38 (3) (2000) 257–286.
[14] I. Kononenko, M. Kukar, Machine Learning and Data Mining: Introduction to Principles and Algorithms, Horwood Publishing Limited, Chichester, 2007.
[15] I. Triguero, D. Peralta, J. Bacardit, S. García, F. Herrera, MRPR: a mapreduce solution for prototype reduction in big data classification, Neurocomputing 150 (Part A(0)) (2015) 331–345, http://dx.doi.org/10.1016/j.neucom.2014.04.078.
[16] S. García, F. Herrera, Evolutionary undersampling for classification with imbalanced datasets: proposals and taxonomy, Evol. Comput. 17 (3) (2009) 275–306.
[17] A. de Haro-Garcia, N. Garcia-Pedrajas, A scalable method for instance selection for class-imbalance datasets, in: Proceedings of the 11th International Conference on Intelligent Systems Design and Applications (ISDA'11), 2011, pp. 1383–1390.
[18] V. López, I. Triguero, C. Carmona, S. García, F. Herrera, Addressing imbalanced classification with instance generation techniques: IPADE-ID, Neurocomputing 126 (2014) 15–28.
[19] S. García, J. Derrac, J. Cano, F. Herrera, Prototype selection for nearest neighbor classification: taxonomy and empirical study, IEEE Trans. Pattern Anal. Mach. Intell. 34 (3) (2012) 417–435.
[20] I. Triguero, J. Derrac, S. Garcia, F. Herrera, A taxonomy and experimental study on prototype generation for nearest neighbor classification, IEEE Trans. Syst. Man Cybern. Part C: Appl. Rev. 42 (1) (2012) 86–100.
[21] W. Hu, Y. Tan, Prototype generation using multiobjective particle swarm optimization for nearest neighbor classification, IEEE Trans. Cybern. (2016) 1–13, http://dx.doi.org/10.1109/TCYB.2015.2487318.
[22] S. Das, P. Suganthan, Differential evolution: a survey of the state-of-the-art, IEEE Trans. Evol. Comput. 15 (1) (2011) 4–31.
[23] I. Triguero, S. García, F. Herrera, Differential evolution for optimizing the positioning of prototypes in nearest neighbor classification, Pattern Recognit. 44 (4) (2011) 901–916.
[24] T. Cover, P. Hart, Nearest neighbor pattern classification, IEEE Trans. Inf. Theory 13 (1) (1967) 21–27.
[25] J..F. Díez-Pastor, J..J. Rodríguez, C. García-Osorio, L..I. Kuncheva, Random balance: ensembles of variable priors classifiers for imbalanced data, Knowl. Based Syst. 85 (2015) 96–111, http://dx.doi.org/10.1016/j.knosys.2015.04.022.
[26] D. Aha, Editorial, in: Lazy Learning, Springer, Dordrecht, 1997, pp. 7–10.
[27] S. García, J. Cano, F. Herrera, A memetic algorithm for evolutionary prototype selection: a scaling up approach, Pattern Recognit. 41 (8) (2008) 2693–2709.
[28] F. Neri, V. Tirronen, Scale factor local search in differential evolution, Memetic Comput. 1 (2) (2009) 153–171.
[29] R. Storn, K. Price, Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces, J. Global Optim. 11 (4) (1997) 341–359.
[30] J. Laurikkala, Improving identification of difficult small classes by balancing class distribution, in: 8th Conference on AI in Medicine in Europe, Lecture Notes on Computer Science, vol. 2001, Springer, Berlin, Heidelberg, 2001, pp. 63–66.
[31] H. Han, W. Wang, B. Mao, Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning, in: 2005 International Conference on Intelligent Computing, Lecture Notes on Computer Science, vol. 3644, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 878–887.
[32] S. Barua, M. Islam, X. Yao, K. Murase, et al., MWMOTE-majority weighted minority oversampling technique for imbalanced data set learning, IEEE Trans. Knowl. Data Eng. 26 (2) (2014) 405–425.
[33] C. Bunkhumpornpat, K. Sinapiromsaran, C. Lursinsap, Safe-level-SMOTE: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Lecture Notes on Computer Science, vol. 5476, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 475–482.
[34] J. Stefanowski, S. Wilk, Selective pre-processing of imbalanced data for improving classification performance, in: 10th International Conference in Data Warehousing and Knowledge Discovery, Lecture Notes on Computer Science, vol. 5182, Springer, Berlin, Heidelberg, 2008, pp. 283–292.
[35] K. Napierala, J. Stefanowski, S. Wilk, Learning from imbalanced data in presence of noisy and borderline examples, in: 7th International Conference on Rough Sets and Current Trends in Computing, 2010, pp. 158–167.
[36] K. Veropoulos, C. Campbell, N. Cristianini, Controlling the sensitivity of support vector machines, in: Proceedings of the International Joint Conference on Artificial Intelligence, 1999, pp. 55–60.
[37] S. Datta, S. Das, Near-bayesian support vector machines for imbalanced data classification with equal or unequal misclassification costs, Neural Netw. 70 (2015) 39–52, http://dx.doi.org/10.1016/j.neunet.2015.06.005.
[38] C. Castro, A. Braga, Novel cost-sensitive approach to improve the multilayer perceptron performance on imbalanced data, IEEE Trans. Neural Netw. Learn. Syst. 24 (6) (2013) 888–899.
[39] B. Krawczyk, M. Galar, Łukasz Jeleń, F. Herrera, Evolutionary undersampling boosting for imbalanced classification of breast cancer malignancy, Appl. Soft Comput. 38 (2016) 714–726, http://dx.doi.org/10.1016/j.asoc.2015.08.060.
[40] R. Schapire, The strength of weak learnability, Mach. Learn. 5 (2) (1990) 197–227.
[41] L. Breiman, Bagging predictors, Mach. Learn. 24 (2) (1996) 123–140.
[42] B. Krawczyk, M. Woźniak, G. Schaefer, Cost-sensitive decision tree ensembles for effective imbalanced classification, Appl. Soft Comput. 14 (Part C) (2014) 554–562, http://dx.doi.org/10.1016/j.asoc.2013.08.014.
[43] T. Khoshgoftaar, J. Van Hulse, A. Napolitano, Comparing boosting and bagging techniques with noisy and imbalanced data, IEEE Trans. Syst. Man Cybern. Part A: Syst. Humans 41 (3) (2011) 552–568.
[44] N. Chawla, A. Lazarevic, L. Hall, K. Bowyer, SMOTEBoost: Improving prediction of the minority class in boosting, in: Knowledge Discovery in Databases, Springer, Berlin, Heidelberg, 2003, pp. 107–119.
[45] S. Wang, X. Yao, Diversity analysis on imbalanced data sets by using ensemble

models, in: IEEE Symposium on Computational Intelligence and Data Mining, 2009, IEEE, Nashville, 2009, pp. 324–331.
[46] S. Hido, H. Kashima, Y. Takahashi, Roughly balanced bagging for imbalanced data, Stat. Anal. Data Min. 2 (5–6) (2009) 412–426.
[47] J. Blaszczynski, J. Stefanowski, Neighbourhood sampling in bagging for imbalanced data, Neurocomputing 150 (Part B) (2015) 529–542.
[48] M. Galar, A. Fernández, E. Barrenechea, F. Herrera, Eusboost: enhancing ensembles for highly imbalanced data-sets by evolutionary undersampling, Pattern Recognit. 46 (12) (2013) 3460–3471.
[49] L. Yijing, G. Haixiang, L. Xiao, L. Yanan, L. Jinling, Adapted ensemble classification algorithm based on multiple classifier system and feature selection for classifying multi-class imbalanced data, Knowl. Based Syst. 94 (2016) 88–104, http://dx.doi.org/10.1016/j.knosys.2015.11.013.
[50] R. Prati, G. Batista, M. Monard, Class imbalances versus class overlapping: an analysis of a learning system behavior, in: MICAI 2004: Advances in Artificial Intelligence, Springer, Berlin, Heidelberg, 2004, pp. 312–321.
[51] M. Kubat, S. Matwin, Addressing the curse of imbalanced training sets: one-sided selection, in: 14th International Conference on Machine Learning, 1997, pp. 179–186.
[52] R. Akbani, S. Kwek, N. Japkowicz, Applying support vector machines to imbalanced datasets, in: Machine Learning: ECML 2004, Springer, Berlin, Heidelberg, 2004, pp. 39–50.
[53] T. Fawcett, An introduction to ROC analysis, Pattern Recognit. Lett. 27 (8) (2006) 861–874.
[54] S. Vluymans, Instance selection for imbalanced data (Master's thesis), Ghent University, 2014, URL ⟨http://www.cwi.ugent.be/sarah/masterthesis.pdf⟩.
[55] G..U. Yule, On the association of attributes in statistics, Philos. Trans. A 194 (1900) 257–319.
[56] L. Kuncheva, C. Whitaker, Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy, Mach. Learn. 51 (2) (2003) 181–207.
[57] S. Wang, X. Yao, Relationships between diversity of classification ensembles and single-class performance measures, IEEE Trans. Knowl. Data Eng. 25 (1) (2013) 206–219.
[58] N. García-Pedrajas, Constructing ensembles of classifiers by means of weighted instance selection, IEEE Trans. Neural Netw. 20 (2) (2009) 258–277.
[59] M. Lichman, UCI Machine Learning Repository, 2013, URL ⟨http://archive.ics.uci.edu/ml⟩.
[60] J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera, KEEL data-mining software tool: data set repository, integration of algorithms and experimental analysis framework, J. Multiple-Valued Logic Soft Comput. 17 (2–3) (2011) 255–287.
[61] G. Wu, E. Chang, Class-boundary alignment for imbalanced dataset learning, in: ICML 2003 Workshop on Learning from Imbalanced Data Sets II, Washington, DC, 2003, pp. 49–56.
[62] J. Stefanowski, Overlapping, rare examples and class decomposition in learning classifiers from imbalanced data, in: Emerging Paradigms in Machine Learning, Springer, Berlin, Heidelberg, 2013, pp. 277–306.
[63] K. Napierala, J. Stefanowski, Types of minority class examples and their influence on learning classifiers from imbalanced data, Journal of Intelligent Information Systems, 2015, pp. 1–35, http://dx.doi.org/10.1007/s10844-015-0368-1.
[64] E. Ramentol, Y. Caballero, R. Bello, F. Herrera, SMOTE-RSB$_*$: a hybrid preprocessing approach based on oversampling and undersampling for high imbalanced data-sets using SMOTE and rough sets theory, Knowl. Inf. Syst. 33 (2) (2012) 245–265.
[65] Z. Pawlak, Rough sets, Int. J. Comput. Inf. Sci. 11 (5) (1982) 341–356.
[66] D. Wilson, Asymptotic properties of nearest neighbor rules using edited data, IEEE Trans. Syst. Man Cybern. 2 (3) (1972) 408–421.
[67] I. Triguero, S. García, F. Herrera, IPADE: iterative prototype adjustment for nearest neighbor classification, IEEE Trans. Neural Netw. 21 (12) (2010) 1984–1990.
[68] J. Cano, F. Herrera, M. Lozano, Using evolutionary algorithms as instance selection for data reduction in kdd: an experimental study, IEEE Trans. Evol. Comput. 7 (6) (2003) 561–575.
[69] Y. Freund, R. Schapire, A desicion-theoretic generalization of on-line learning and an application to boosting, in: Computational Learning Theory, Springer, Berlin, Heidelberg, 1995, pp. 23–37.
[70] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
[71] S. García, A. Fernández, J. Luengo, F. Herrera, Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power, Inf. Sci. 180 (10) (2010) 2044–2064.
[72] M. Friedman, The use of ranks to avoid the assumption of normality implicit in the analysis of variance, J. Am. Stat. Assoc. 32 (200) (1937) 675–701.

**Sarah Vluymans** holds an M.Sc. degree (2014) in Mathematical Informatics from Ghent University. Currently, she is a Ph.D student at Ghent University and is funded by the Special Research Fund (BOF) of that institute. She is also part of the DAMBI research group (Data mining and modeling for biomedicine) at the Inflammation Research Center, part of the Flemish Institute of Biotechnology. Her research focuses on the integration of fuzzy rough set theory in machine learning techniques.

**Isaac Triguero** received the M.Sc. and the Ph.D. degree in Computer Science from the University of Granada, Spain, in 2009 and 2014, respectively. He is currently an assistant professor in data science at the University of Nottingham, United Kingdom. He has published 25 papers in international journals. His research interests include data mining, data reduction, evolutionary algorithms, semisupervised learning, bioinformatics and big data learning.

**Chris Cornelis** holds an M.Sc. and a Ph.D. degree in Computer Science from Ghent University (Belgium). Currently, he is a postdoctoral fellow at the University of Granada supported by the Ramón y Cajal programme of the Spanish government, as well as a guest professor at Ghent University. He has cosupervised 7 Ph.D. theses and has authored over 140 papers in international journals, edited volumes and conference proceedings. He serves as an executive board member of the International Rough Set Society (IRSS). His current research interests include fuzzy sets, rough sets and machine learning.

**Yvan Saeys** is a professor at Ghent University and a group leader at the Flemish Institute for Biotechnology (VIB). He is heading the Data Mining and Modelling for Biomedicine (DaMBi) research group, an interdisciplinary team consisting of mathematicians, computer scientists, engineers and bioinformaticians. The main focus of the group is the design and application of novel data mining and machine learning techniques for applications in biomedicine and systems immunology. He has developed several state-of-the-art machine learning techniques that have been nominated as best performing techniques in the field of systems biology (winner of the DREAM5 and FlowCAP-IV challenges). He is an associate editor of the Information Fusion journal and has served on the program committee of numerous workshops focusing on the interplay between data mining and machine learning and bioinformatics.