Applied Soft Computing 25 (2014) 26–39

Contents lists available at ScienceDirect

Applied Soft Computing

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

Training algorithms for Radial Basis Function Networks to tackle learning processes with imbalanced data-sets

M.D. Pérez-Godoy*, Antonio J. Rivera, C.J. Carmona, M.J. del Jesus

Department of Computer Science, University of Jaén, 23071 Jaén, Spain

ARTICLE INFO

Article history:
Received 9 November 2012
Received in revised form 6 May 2014
Accepted 12 September 2014
Available online 22 September 2014

Keywords:
Imbalanced data-sets
Radial Basis Function Networks
Training algorithms
Evolutionary computation

ABSTRACT

Nowadays, many real applications comprise data-sets where the distribution of the classes is significantly different. These data-sets are commonly known as imbalanced data-sets. Traditional classifiers are not able to deal with these kinds of data-sets because they tend to classify only majority classes, obtaining poor results for minority classes. The approaches that have been proposed to address this problem can be categorized into three types: resampling methods, algorithmic adaptations and cost sensitive techniques.

Radial Basis Function Networks (RBFNs), artificial neural networks composed of local models or RBFs, have demonstrated their efficiency in different machine learning areas. Centers, widths and output weights for the RBFs must be determined when designing RBFNs.

Taking into account the locally tuned response of RBFs, the objective of this paper is to study the influence of global and local paradigms on the weights training phase, within the RBFNs design methodology, for imbalanced data-sets. Least Mean Square and the Singular Value Decomposition have been chosen as representatives of local and global weights training paradigms respectively. These learning algorithms are inserted into classical RBFN design methods that are run on imbalanced data-sets and also on these data-sets preprocessed with re-balance techniques. After applying statistical tests to the results obtained, some guidelines about the RBFN design methodology for imbalanced data-sets are provided.

1. Introduction

Many real applications are associated to data-sets with an implicit imbalance of the existing classes. This fact implies that the number of instances of certain classes is much lower than the instances of the other classes. The importance of these data-sets resides in the fact that a minority class usually represents the concept of interest, for example patients with illness in a medical diagnosis problem; whereas the other class represents the counterpart of that concept (healthy patients). For this kind of problems standard classifier algorithms have a bias toward the majority class. This is due to the fact that the mechanisms inside these classifiers are positively weighted to predict the majority class defined by the accuracy metrics. The different methods proposed for addressing this problem can be categorized into three groups [1]: resampling methods [2], algorithmic adaptations [3], and cost sensitive techniques [4].

Radial Basis Function Networks (RBFNs) [5] are one of the most important Artificial Neural Network paradigms in the field of Machine Learning. RBFNs have important features such as: a simple topological structure; the possibility of extracting rules [6,7]; a universal approximation capability [8]; and that each neuron/RBF has a characteristic locally-tuned response. RBFNs have been successfully applied in most important machine learning areas [9] such as classification [10,11], regression [12,13] or time series forecasting [14,15]. Examples of application fields where RBFNs have been demonstrated their good behavior are: engineering problems [16–18], medical diagnosis [19,20] or web mining [21,22], among others. They have also obtained outstanding results in imbalanced problems [23]. Typically, RBFNs are designed in two stages: First, RBF parameters (center and width) are calculated with different techniques [9]; and then, weights are obtained with local or more global training algorithms. A global training algorithm considers all the data-set instances at the same time in a matrix that is resolved using numerical methods, whereas a local training algorithm separately trains each instance, only modifying weighs of the nearest RBFs.

From these premises, the objective of this paper is to carry out a study that characterizes the most suitable RBFN training methodology (local and global) taking into account the known locally-tuned response of the RBFs and avoiding the above mentioned bias toward the majority class on imbalanced data-sets.

* Corresponding author: Tel.: +34 953 212892; fax: +34 953 212472.
E-mail address: lperez@ujaen.es (M.D. Pérez-Godoy).

http://dx.doi.org/10.1016/j.asoc.2014.09.011
1568-4946/

Thus, this paper analyzes the characteristics of Least Mean Square (LMS) [24] and Singular Value Decomposition (SVD) [25], as local and global weights training representative methods respectively. In this way, the behavior of both training algorithms is studied with different RBFN design methods: an Incremental algorithm, a Clustering method, a traditional Evolutionary Computation method (Pittsburgh codification) and CO^{2}RBFN [10], an evolutionary cooperative–competitive RBFN design method.

During experimentation, the RBFN design algorithms were applied to a largest group of data-sets with different imbalance ratio (IR) and to the same group of data-sets preprocessed in order to re-balance them. The analysis of the results let us to obtain a set of guidelines for the use of weights training algorithms in imbalanced frameworks.

The paper is organized as follows: First, Section 2 introduces RBFNs. Section 3 describes LMS and SVD, the two weights training algorithms for RBFNs. Section 4 details the most important paradigms for RBFN design. In Section 5, the imbalance problem is analyzed. The experimental framework of this research is specified in Section 6 and the results and their analysis are shown in Section 7. Finally, the conclusions of the paper are outlined in Section 8.

2. Radial Basis Function Networks

From a structural point of view, an RBFN is a feed-forward neural network with three layers: an input layer with n nodes, a hidden layer with m neurons or RBFs, and an output layer (Fig. 1).

The m neurons of the hidden layer are activated by a radially-symmetric basis function, $\phi_i : R^n \rightarrow R$, which can be defined in several ways [26], the Gaussian function being the most widely used (Eq. (1)):

$$\phi_i(\vec{x}) = \phi_i(e^{-(\|\vec{x}-\vec{c}_i\|/d_i)^2}) \tag{1}$$

where $\vec{c}_i \in R^n$ is the center of basis function ϕ_i, $d_i \in R$ is the width (radius), and $\|\|$ is typically the Euclidean norm on R^n. This expression is the one used in this paper as the Radial Basis Function (RBF). The output node implements the following function, where weights w_{ij} show the contribution of an RBF to the output node (Eq. (2)):

$$f_j(\vec{x}) = \sum_{i=1}^{m} w_{ij}\phi_i(\vec{x}) \tag{2}$$

The objective of any RBFN design process is to determine centers, widths and the linear output weights connecting the RBFs to the output neuron layer. The most traditional learning procedure has two stages [27–29,13,30]: first, learning of centers and widths, and then, training of output weight. In order to obtain the weights in the second stage, two paradigms are usually followed: gradient descent based methods (with a local behavior) [29,31,30] and global linear regression methods [12,32,13,31,33].

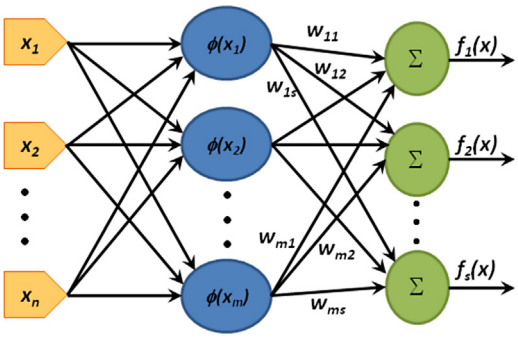

Fig. 1. RBFN topology.

One early technique to adjust the centers and widths is Clustering [34]. K-means algorithm [35] is a well-know algorithm based on clustering, which has been used for the RBFN design in [36]. For this kind of methods the center of the clusters are established as the center of the RBFs and RBF widths are typically set to the width of the previously calculated clusters or as the average distance between RBFs.

RBFNs were initially used for function approximation. For this reason, most of the RBFN design methods are based on traditional optimization techniques such as regularization [37], orthogonalization of regressors [38], gradient-based [39], or Levenberg–Marquardt [40]. These techniques can be used to decide if the RBFs aggregate or eliminate and may be considered as forward or backward selection methods [41]. The RAN algorithm [42] is one of the most well-known methods based on an incremental (aggregation) scheme.

Another important paradigm for RBFN design is Evolutionary Computation (EC) [43–45], which uses natural evolution and stochastic searching to design optimization algorithms. Reviews of EC applied to RBFN design can be found in [46,9]. In most of the proposals within this evolutionary paradigm, an individual represents a whole RBFN, and different operators are applied to the entire population to improve individual fitness [46,47]. This approach is known as the Pittsburgh representation scheme. An alternative to this evolutionary approach is the cooperative–competitive evolutionary or cooperative–coevolutionary strategy [48,49]. It provides a framework within which an individual of the population represents only a part of the solution, as it evolves in parallel and competes to survive, but at the same time cooperates to find a common solution (the complete RBFN).

In the engineering field, it can be found other RBFN design methods [17]. In [50–52] control systems are modeled using RBFNs. To developed them, first genetic algorithms are employed to determine the initial parameters of the networks. Then, the parameters of the decoupled adaptive neural network controllers are updated regarding some stability theories, such as the Lyapunov theory, and boundary-layer functions that guarantee the convergence of the state errors within a specified error bound. In the experimentation, the success of the design methods are demonstrated testing the stability and the error convergence of the modeled adaptive control laws.

Pérez-Godoy et al. developed an evolutionary cooperative–competitive RBFN design method, CO^{2}RBFN [10]. The standard version of CO^{2}RBFN, which uses the LMS algorithm for training weights, was applied to the classification problem of imbalanced data-sets [23] and achieved significant results. Other conclusions deduced in [23] were the good behavior of some RBFN design methods dealing with imbalanced data-sets.

As mentioned in this paper a more general objective is established, specifically to characterize the weights training phase inside a RBFN design process for imbalance problems. Thus, we will analyze the behavior of local and global training algorithms with different RBFN design methods such as an Incremental algorithm, a Clustering method, a traditional Evolutionary Computation method (Pittsburgh codification) and CO^{2}RBFN. Theses methods are described in Section 4.

3. Training algorithms for RBFN weights

During the training phase for RBFN weights a known set of input and output data pairs were delivered to the RBFN to compute the output layer weights. Taking into account Eq. (2), the least squares recipe is then to minimize the sum-squared-error.

$$S = \sum_{i=1}^{p} (y_i - f(x_i))^2, \quad \forall i = 1, \ldots, n \tag{3}$$

where y_i is the desired output and $f(x_i)$ is any output of the network.

In order to calculate the weights that reduce the error of Eq. (2), different methods are normally used. These methods can been categorized as local or global paradigms. Gradient descent based methods [29,31,30,15] have a local behavior and separately trains each instance, only modifying weighs of the nearest RBFs. Global linear regression methods [27,12,32,13,31] consider all the data-set instances at the same time, solving the above equation by means of linear least squares in matrix forms.

LMS was selected, as representative of local weights training algorithms, because it is the reference algorithm within the gradient descent based methods. With respect to global weights training algorithms there are several well known techniques to solve the matrix that results when linear regression methods are applied, for instance: the Cholesky decomposition [25], the orthogonal least squares (OLS) method [38], or the Singular Value Decomposition (SVD) [25]. SVD, the selected global training method for the experimentation, is one of the most widely used because it avoids possible ill-conditioning in the resolution matrix.

3.1. LMS method

The Least Mean Square (LMS) algorithm [24] can be used to calculate the RBF weights. This technique exploits the local information that can be obtained from the behavior of each RBFs. Eq. (4) shows the update of the weights.

$$\overline{w}_{k+1} = \overline{w}_k + \alpha \frac{e_k \overline{x}_k}{|\overline{x}_k^2|} \tag{4}$$

where k is the number of iterations, $\overline{w}_{k+1}$ is the next value of the weight vector, $\overline{w}_k$ is the present value of the weight vector and $\overline{x}_k$ is the value of the actual input pattern vector. The present linear error, e_k, is defined as the difference between the desired output and the output network before adaptation. The α value is the *speed of learning*, which measures the size of the adjustment to be made. The choice of α controls stability and speed of convergence.

3.2. SVD method

This weight training algorithm is easier to explain if the RBFN performance is expressed in a matrix form: $\overline{y} = \Theta\overline{w}$, where $\overline{y}$ is the vector of the desired training outputs, $\overline{w}$ is the weight vector, and Θ is a non-square matrix with components which are equal to the output of the radial basis functions evaluated in the training points.

Minimization of the sum-squared error function yields the well-known least-squares solution for the weights: $A\overline{w} = \Theta^T\overline{y}$, from this equation we can extract the weight vector: $\overline{w} = A^{-1}\Theta^T\overline{y}$. A^{-1}, the variance matrix, is: $A^{-1} = \Theta^T\Theta^{-1}$.

Thus, the network weights can be computed by fast linear matrix inversion techniques using SVD.

SVD is chosen because it avoids possible ill-conditioning of A. This method provides a decomposition of A that allows the calculation of $\overline{w}$ and an optimal solution to the least mean squares equation.

4. RBFN design algorithms

This section describes the algorithms which represent some of the most significant paradigms in the RBFN design: a method based on the clustering technique (Clustering), a procedure based on the incremental technique (Incremental), an evolutionary algorithm based on the Pittsburgh representation scheme (Genetic), and CO^{2}RBFN, an evolutionary cooperative–competitive algorithm. All the algorithms were implemented by the authors of this paper.

4.1. Clustering method

Traditionally, clustering has been associated with classification tasks. The objective of the cluster analysis is to capture the natural structure of the data by dividing them into groups (clusters) that are meaningful, useful or both. These meaningful groups have been assigned to classes and often a representative or prototype of each group is often determined. As has been previously mentioned, clustering techniques have been the first stage in the classical design of RBFNs, as they relate each cluster to an RBF. Therefore, the RBFN will have the same number of RBFs as the calculated clusters and the geometric center of the clusters are established as the center of the RBFs. RBF widths are typically set to the width of the previously calculated clusters or may be established as the average distance between RBFs.

One of the most well-known clustering techniques is *K*-means [53]. This technique defines a prototype in terms of a centroid, i.e., the mean of a group of points. K initial centroids are chosen, where K is a user specified parameter and represents the desired number of clusters. Each point is then assigned to the closest centroid, and each collection of points assigned to a centroid is a cluster. The centroid of each cluster is then updated using the points assigned to the cluster. The assignment and update steps are repeated until the centroids remain the same, i.e., no point changes clusters.

Following this paradigm, the Clustering method for RBFN design was implemented. The structure of this algorithm is shown in Algorithm 1. Thus, given the input data vector $X = \vec{x}_i : i = 1, \ldots, n$, the objective was to obtain K clusters with their corresponding prototype. In order to calculate the cluster to which each vector belongs, it was necessary to define a distance measure and to minimize the objective function D (Eq. (5)):

$$D = \sum_{j=1}^{c} \sum_{i=1}^{n} \| \vec{x}_i - \vec{p}_j \| \tag{5}$$

where $\vec{x}_1, \ldots, \vec{x}_n$ are the vectors to group, $\vec{p}_1, \ldots, \vec{p}_c$ are the prototypes for each group and $\|\|$ is the Euclidean distance. Moreover, membership function $\mu_{p_j}(\vec{x}_i)$ of the input vector $\vec{x}_i$ to the cluster represented by the prototype $\vec{p}_j$ is defined in Eq. (6). The clustering achieved by the *K*-means algorithm can be described by the partition matrix U shown in Eq. (7).

$$\mu_{p_j}(\vec{x}_i) = \begin{cases} 1 & \text{if} \quad \| \vec{x}_i - \vec{p}_j \|^2 < \| \vec{x}_i - \vec{p}_l \|^2 \quad \forall l \neq j \\ 0 & \text{in another case} \end{cases} \tag{6}$$

$$U = \left\{ \mu \in \{0,1\} | \sum_{i=1}^{c} \mu_{P_i}(\vec{x}_k) = 1 \quad \forall k \quad \text{and} \quad 0 < \sum_{k=1}^{n} \mu_{P_i}(\vec{x}_k) < n \quad \forall i \right\} \tag{7}$$

The Clustering method, described in Algorithm 1, begins with random initialization of the prototypes $\vec{p}_j$ to existing input vector and sets the distortion δ to infinite where δ measures changes in the clusters. Following the main loop will determine the clusters. The stop condition of this loop is to check, whether in this iteration the distortion has changed over a threshold, ϵ, with respect to the distortion of the previous iteration δ_{prev}. The two main steps of the loop are: first, determining to which partition each vector belongs by calculating its memberships function $\mu_{p_j}(\vec{x}_i)$ (Eq. (6)). Second, recalculating prototypes of each partition using Eq. (8). In the last step of the main loop, δ is calculated using Eq. (9) where δ_j is distortion calculated for cluster j. Finally, once the clusters are determined, the weights are trained using LMS or SVD.

Algorithm 1. Main steps of the Clustering method

```
δ =∞
Random initialization of the first set of prototypes p_j
while |δ_ant − δ|/δ >= ε do
  δ_ant = δ
  Calculate membership functions μ_{p_i}(x_k)
  Calculate prototypes p_j
  Calculate δ
end while
Train Weights (LMS or SVD)
```

$$\vec{p}_j = \frac{\sum_{i=1}^{n} \mu_{p_j}(\vec{x}_i)\vec{x}_i}{\sum_{i=1}^{n} \mu_{p_j}(\vec{x}_i)} \tag{8}$$

$$\delta = \sum_{j=1}^{c} \delta_j \tag{9}$$

$$\delta_j = \sum_{\vec{x}_i \in P_j} \| \vec{x}_i - \vec{p}_j \|^2 \tag{10}$$

4.2. Incremental method

Incremental algorithms are a traditional paradigm for RBFN design [42,54,41]. One of the first and most well-known algorithms is the RAN (Random Allocation Network) algorithm [42]. This algorithm starts with an empty RBFN (without RBFs) and, iteratively, identifies patterns that are not currently well represented by the network, allocating new units that memorize these patterns. Thus, in each iteration of the main loop, a random pattern of the training set is analyzed in order to check whether two conditions are fulfilled. These conditions are firstly whether the pattern is placed at a distance of more than a δ threshold from any RBF, and secondly whether the difference for this pattern, between the desired output and the output of the network, is larger than an ϵ threshold. If both conditions are met, a new RBF is inserted, where the RBF center is set to the pattern, and the RBF width is set to the minimum distance between the new RBF and the other RBFs in the RBFN. If these conditions are not fulfilled, the weights of this RBF are adapted using the LMS algorithm.

To test this important paradigm, the Incremental method was implemented. This method includes two versions of the RAN algorithm, one for each weights training algorithm (LMS and SVD). The main steps of the algorithm are detailed in Algorithm 2.

The main loop of the implemented algorithm begins with the random determination of the instance I from the training data-set (TD). Then, the output of the RBFN, $f(x)$, and the error committed for this instance, e, are calculated. Next, the distance, d, from I to the nearest RBF in the RBFN is determined. Having e and d, the typical conditions of the RAN algorithm are checked, i.e., If e is higher than ϵ And d is higher than δ Then a new RBF is placed on I. As in the original RAN algorithm, the RBF center is set to I, and the RBF width is set to the minimum distance between the new RBF and the other RBFs. For the first RBF, the RBF width is set to δ. After that, weights are trained using SVD or LMS.

This is the main difference with regards to the original RAN algorithm, because RAN only trains the weight of the instance I when the conditions are not met. This change was introduced for two reasons: SVD cannot be applied only for a weight of one RBF, and most algorithms for RBFN design train the whole set of the weights of the RBFN. Finally, the stop condition of the loop checks the number of times that RAN conditions are not met. If this number *cont* is higher than the total number of instances of the training data-set, then the algorithm ends.

Algorithm 2. Main steps of Incremental method

```
cont = 0
while cont < nInstances(TD) do
  I = random(TD)
  Evaluate the output of the RBFN f(x)
  Compute error e = |y − f(x)|
  Find distance, d, from I to nearest RBF
  if (e > ε)and(d > δ) then
    Insert a new RBF with center = I
    if is the first RBF then
      RBF width = delta
    else
      RBF width = d
    end if
    Train Weights (LMS or SVD)
    cont = 0
  else
    cont = cont + 1
  end if
end while
```

4.3. Genetic method

Evolutionary computation is a widely used paradigm in optimization and particularly in RBFN design [9]. Thus, the Genetic method has been implemented following the design lines of genetic algorithms for RBFNs learning [46].

This method follows the traditional Pittsburgh evolutionary approach for the design of RBFNs. In this approach each individual is a whole network and therefore contains the coordinates of the center, widths and weights for each RBF. A real codification of the individuals was used considering a variable number of RBFs. The objective of the evolutionary process is to minimize the classification error. The best individual will be the final solution.

Algorithm 3. Main steps of Genetic method

```
Initialize RBFN
while Number of Generations not Achieved do
  Selection
  Recombination
  Mutation
  Train Weights (LMS or SVD)
  Evaluation RBFN
end while
```

The main steps of this algorithm are shown in Algorithm 3.

As an initialization step, for each individual, a random number of neurons is allocated to the different patterns of the training set. Thus, each RBF center, $\vec{c}_i$, is randomly established to a pattern of the training set. The RBF widths, d_i, will be set to half of the average distance between the centers. Finally, the RBF weights, w_{ij}, are set to zero.

A tournament selection mechanism is applied to the whole group in order to determine the new population. The diversity of the population is promoted by using a low value for the tournament size ($k = 3$).

Recombination and mutation operators are applied to the new RBFNs population. With the crossover (recombination) operator two individuals (RBFNs) parents are chosen to obtain an RBFN offspring. The number of RBFs of the new individual will be delimited between a minimum and a maximum value. The minimum value is set to the number of RBFs of the parent with fewest RBFs. In the same way, the maximum value is set to the number of RBFs of the parent with most RBFs. In order to generate the offspring, RBFs will be chosen from the parents at random.

Six mutation operators, usually considered in the specialized bibliography [46], were implemented. They can be classified as random operators or biased operators. The random operators are:

- DelRandRBFs: randomly eliminates k RBFs, where k is a pm percent of the total number of RBFs in the RBFN.
- InsRandRBFs: randomly aggregates k RBFs, where k is a pm percent of the total number of RBFs in the RBFN.
- ModCentRBFs: randomly modifies the center of k RBFs, where k is a pm percent of the total number of RBFs in the RBFN. The center of the basis function will be modified in a pr percent of its width.
- ModWidtRBFs: randomly modifies the center of k RBFs, where k is a pm percent of the total number of RBFs in the RBFN. The width of the basis function will be modified in a pr percent of its width.

Biased operators which exploit local information are:

- DelInfRBFs: deletes the k RBFs of the RBFN with a lower weight. k is a pm percent of the total number of RBFs in the RBFN.
- InsInfRBFs: inserts the k RBFs in the RBFN outside the width of any RBF present in the RBFN. k is a pm percent of the total number of RBFs in the RBFN.

After applying mutation operators, weights are trained using the LMS or the SVD algorithm. The fitness for each individual/RBFN is defined as the geometric mean, see Eq. (15), classification error.

The search space of this method was limited in order to establish similar operating conditions for the algorithms. As it is well known, with Pittsburgh genetic algorithms, where the only objective to optimize is the classification error, the complexity of the individuals (i.e. number of RBFs) grows in an uncontrolled way (because an RBFN with more RBFs usually gives a lower error percentage than an RBFN with fewer RBFs). In this way a maximum complexity (chromosome size) was established.

4.4. CO²RBFN method

CO²RBFN [10], is an evolutionary cooperative–competitive hybrid algorithm for the design of RBFNs. In this algorithm each individual of the population represents, with a real representation, an RBF and the entire population is responsible for the final solution. The individuals cooperate toward a definitive solution, but they must also compete for survival. In this environment, in which the solution depends on the behavior of many components, the fitness of each individual is known as its "credit assignment". In order to measure the credit assignment of an individual, three factors were proposed: the RBF contribution to the network output, the error in the basis function radius, and the degree of overlapping among RBFs.

There are four evolutionary operators that can be applied to an RBF: an operator that eliminates the RBF, two operators that mutate the RBF, and finally, an operator that maintains the RBF parameters in order to explore and exploit the search space and to preserve the best RBF, respectively.

The application of the operators is determined by an fuzzy rule base system. The inputs of this system are the three parameters used for credit assignment and the outputs are the operators' application probability.

The main steps of CO²RBFN, explained in the following subsections, are shown in the pseudocode in Algorithm 4.

Algorithm 4. Main steps of CO²RBFN method

```
Initialize RBFN
while Not Stop do
  Train RBFN (LMS or SVD)
  Evaluate RBFs
  Apply operators to RBFs
  Substitute the eliminated RBFs
  Select the best RBFs
end while
```

In the RBFN initialization step, to define the initial network, a specified number m of neurons (i.e. the size of population) is randomly allocated to the different patterns of the training set. To do so, each RBF center, $\vec{c}_i$, is randomly assigned to a pattern of the training set. The RBF widths, d_i, will be set to half of the average distance between the centers. Finally, the RBF weights, w_{ij}, are set to zero.

In the RBFN training step, LMS or SVD training algorithm is used.

For the RBF evaluation, a credit assignment mechanism is required in order to evaluate the role of each RBF ϕ_i in the cooperative–competitive environment. For an RBF, three parameters, a_i, e_i, o_i are defined:

- The contribution, a_i, of the RBF ϕ_i, $i = 1, \ldots, m$, is determined by considering the weight, w_i, and the number of patterns of the training set inside its width, npi_i. An RBF with a low weight and few patterns inside its width will have a low contribution:

$$a_i = \begin{cases} |w_i| & \text{if } npi_i > q \\ |w_i| * (npi_i/q) & \text{otherwise} \end{cases} \quad (11)$$

  where q is the average of the npi_i values minus the standard deviation of the npi_i values.
- The error measure, e_i, for each RBF ϕ_i, is obtained by counting the wrongly classified patterns inside its radius:

$$e_i = \frac{npibc_i}{npi_i} \quad (12)$$

  where $npibc_i$ and npi_i are the number of wrongly classified patterns and the number of all patterns inside the RBF width respectively.
- The overlapping of the RBF ϕ_i and the other RBFs is quantified by using the parameter o_i. This parameter is computed by taking into account the fitness sharing methodology [44], whose aim is to maintain the diversity in the population. This factor is expressed as:

$$o_i = \sum_{j=1}^{m} o_{ij} \quad o_{ij} = \begin{cases} (1 - \| \phi_i - \phi_j \| / d_i) & \text{if } \| \phi_i - \phi_j \| < d_i \\ 0 & \text{otherwise} \end{cases} \quad (13)$$

  where o_{ij} measures the overlapping of the RBF ϕ_i y ϕ_j $j = 1 \ldots m$.

In CO²RBFN four operators have been defined in order to be applied to the RBFs:

- Operator *Remove*: eliminates an RBF.
- Operator *Random Mutation*: randomly modifies the coordinates of the center and the width of an RBF.
- Operator *Biased Mutation*: modifies the width and the coordinates of the center using local information of the RBF environment.
- Operator *Null*: in this case all the parameters of the RBF are maintained.

The operators are applied to the whole population of RBFs. The probability of choosing an operator is determined by means of a Mandani-type fuzzy rule based system [55]. This system represents expert knowledge about the operator application in order to obtain a simple and accurate RBFN.

The inputs of this system are parameters a_i, e_i and o_i used for defining the credit assignment of the RBF ϕ_i. These inputs are considered linguistic variables va_i, ve_i and vo_i. The outputs, p_{remove}, p_{rm}, p_{bm} and p_{null}, represent the probability of applying Remove, Random Mutation, Biased Mutation and Null operators, respectively.

The rule base system aims to evolve RBFs with a good behavior (high contribution, low error and low overlapping) and to eliminate

RBFs with a bad behavior (low contribution, high error and high overlapping).

In the introduction of new RBFs step, the eliminated RBFs are substituted by new RBFs. The new RBF is located in the center of the area with maximum error or in a randomly chosen pattern with a probability of 0.5 respectively.

The width of the new RBF will be set to the average of the RBFs in the population plus half of the minimum distance to the nearest RBF. Its weights are set to zero.

The replacement scheme determines which new RBFs (obtained before the mutation) will be included in the new population. To do so, the role of the mutated RBF in the net was compared with the original one to determine the RBF with the best behavior in order to include it in the population.

5. Imbalanced data-sets

Firstly, this section introduces the problem of imbalanced data-sets in classification. Secondly, it describes the pre-processing technique that was applied in order to deal with the imbalanced data-sets: the SMOTE algorithm [56].

5.1. The problem of imbalanced data-sets

The number of instances of each class can be very different in imbalanced classification problems [57]. Standard classifier algorithms usually have a bias toward the majority class during the learning process in favor of the standard accuracy rate metric, which does not take into account the class distribution of the data. Consequently, the instances belonging to the minority class are misclassified more often than those belonging to the majority class.

We used the imbalance ratio (IR), defined as the ratio of the number of instances of the majority class to the minority class, to organize the different data-sets. In the bibliography [58], research usually focuses on binary imbalanced data-sets, where there is only one positive and one negative class. In the same way, in order to make the analysis of the results easier, data-sets were categorized into two groups according to the IR level. The first group, called "high", comprises data-sets with IR levels higher than 9, where there are no more than 10% of positive instances in the whole data-set compared to the negative ones. The second sub-group is "low" and it is composed of data-sets where the instances of the positive class are between 10% and 40% of the total instances (IR between 1.5 and 9).

The measures of the quality of classification were defined from a confusion matrix (shown in Table 1), which recorded correctly and incorrectly recognized examples for each class.

The most extensively used empirical classification measure, accuracy rate (Eq. (14)), does not distinguish between the number of correct labels of different classes. This fact may lead to erroneous conclusions in the context of imbalanced problems. For example, a classifier that obtains an accuracy of 99% in a data-set with a distribution of 1:100 might not be accurate if it does not extensively cover any minority class instance.

$$Acc = \frac{TP + TN}{TP + FN + FP + TN} \tag{14}$$

Table 1
Confusion matrix for a two-class problem.

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

A widely used metric for imbalanced data-sets, also used in this study, is the geometric mean of the true rates [59], which can be defined as:

$$GM = \sqrt{\frac{TP}{TP + FN} \cdot \frac{TN}{FP + TN}} \tag{15}$$

This metric attempts to maximize the accuracy of each of the two classes with a good balance. It is a performance metric that links both objectives.

As already mentioned, the approaches previously proposed for dealing with the class-imbalance problem can be categorized into three groups [1]: resampling methods which pre-process or resample the data in order to diminish the effects of their class imbalance [60,61,2]; algorithmic adaptations which create new algorithms or modify existing ones to take the class-imbalance problem into consideration [59,62,63,3]; and cost-sensitive learning solutions [4] which incorporate the data and algorithmic level approaches and which assume higher misclassification costs with samples in the minority class and seek to minimize the high cost errors [64–66].

The advantage of the approaches based on pre-processing the data is that they are independent of the classifier used. In [60] these methods are classified into three groups:

- Under-sampling methods that eliminates some of the examples of the majority class in order to decrease the imbalance.
- Over-sampling methods that creates new synthetic examples of the minority class.
- Hybrid methods that combine the two previous methods.

In this study the choice was an over-sampling method which is widely-used in this area: the SMOTE algorithm [56].

5.2. SMOTE

With this approach, the positive class is over-sampled by taking each minority class sample and introducing synthetic examples along the line segments joining any/all of the k minority class nearest neighbors. Depending upon the amount of over-sampling required, neighbors from the k nearest neighbors are randomly chosen. This process is illustrated in Fig. 2, where x_i is the selected point, x_{i1} to x_{i4} are some selected nearest neighbors and r_1 to r_4 the synthetic data points created by the randomized interpolation. The implementation of this method uses only one nearest neighbor with the Euclidean distance, and balances both classes to 50% distribution.

In short, the main idea is to form new minority class examples by interpolating between several minority class examples that lie together. More specifically, synthetic samples are generated in the following way: Take the difference between the feature vector

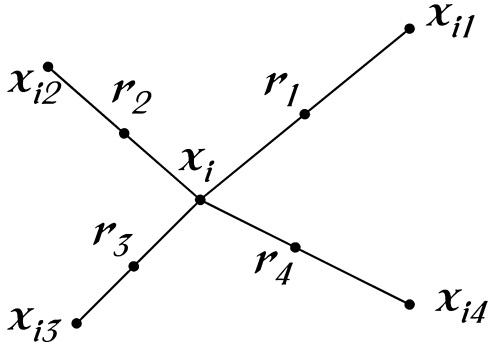

Fig. 2. An illustration of how to create the synthetic data points in the SMOTE algorithm.

(sample) which is under consideration and its nearest neighbor, multiply this difference by a random number between 0 and 1, and add it to the feature vector under consideration. This causes the selection of a random point along the line segment between two specific features. This approach effectively forces the decision region of the minority class to become more general.

6. Experimental framework

A high number of data-sets with different IR was selected to analyze the performance of training algorithms for RBFNs in an imbalanced scenario. The objective was to analyze the influence that weight training algorithms with global and local behavior has on the design of RBFNs models for imbalanced data-sets. Thus, different RBFNs design methods and two representative weight training algorithms were considered. To include in the analysis the effect on preprocessed data-sets the experimentation was divided into two subsections: performance analysis with original data-sets (without preprocessing) (Section 7.1), and the same package of data-sets but preprocessed with SMOTE (Section 7.2). As previously mentioned, SMOTE was chosen because nowadays it is one of the most widely used methods that shows the best performance.

Two versions, LMS and SVD, were implemented respectively for the RBFN design methods and therefore eight different algorithms were obtained.

First, this section describes the collection of imbalanced data-sets selected for the study (Section 6.1). Second, it shows the algorithms selected for comparison in the experimental study and the corresponding parameters (Section 6.2). Finally, it presents the statistical tests used in the analysis (Section 6.3).

6.1. Data-sets

This study used forty four binary data-sets with different IR from KEEL (Knowledge Extraction based on Evolutionary Learning) [67] repository.

Table 2 summarizes the data selected in this study and shows, for each data-set, the number of examples (#Ex.), number of attributes (#Atts.), class name of each class (minority and majority), class attribute distribution and IR. This table is ordered according to the IR, from low to highly imbalanced data-sets.

A five fold cross validation approach was used to estimate precision. This is five partitions for training and test sets, 80% for training and 20% for testing, where the five test partitions form the whole

Table 2
Description for imbalanced data-sets.

Data-sets	#Ex.	#Atts.	Class(min.; maj.)	%Class(min., maj.)	IR
Glass1	214	9	(build-win-non-float-proc; remainder)	(35.51, 64.49)	1.82
Ecoli0vs1	220	7	(im; cp)	(35.00, 65.00)	1.86
Wisconsin	683	9	(malignant; benign)	(35.00, 65.00)	1.86
Pima	768	8	(tested-positive; tested-negative)	(34.84, 66.16)	1.90
Iris0	150	4	(Iris-Setosa; remainder)	(33.33, 66.67)	2.00
Glass0	214	9	(build-win-float-proc; remainder)	(32.71, 67.29)	2.06
Yeast1	1484	8	(nuc; remainder)	(28.91, 71.09)	2.46
Vehicle1	846	18	(Saab; remainder)	(28.37, 71.63)	2.52
Vehicle2	846	18	(Bus; remainder)	(28.37, 71.63)	2.52
Vehicle3	846	18	(Opel; remainder)	(28.37, 71.63)	2.52
Haberman	306	3	(Die; Survive)	(27.42, 73.58)	2.68
Glass0123vs456	214	9	(non-window glass; remainder)	(23.83, 76.17)	3.19
Vehicle0	846	18	(Van; remainder)	(23.64, 76.36)	3.23
Ecoli1	336	7	(im; remainder)	(22.92, 77.08)	3.36
New-thyroid2	215	5	(hypo; remainder)	(16.89, 83.11)	4.92
New-thyroid1	215	5	(hyper; remainder)	(16.28, 83.72)	5.14
Ecoli2	336	7	(pp; remainder)	(15.48, 84.52)	5.46
Segment0	2308	19	(brickface; remainder)	(14.26, 85.74)	6.01
Glass6	214	9	(headlamps; remainder)	(13.55, 86.45)	6.38
Yeast3	1484	8	(me3; remainder)	(10.98, 89.02)	8.11
Ecoli3	336	7	(imU; remainder)	(10.88, 89.12)	8.19
Page-blocks0	5472	10	(remainder; text)	(10.23, 89.77)	8.77
Yeast2vs4	514	8	(cyt; me2)	(9.92, 90.08)	9.08
Yeast05679vs4	528	8	(me2; mit, me3, exc, vac, erl)	(9.66, 90.34)	9.35
Vowel0	988	13	(hid; remainder)	(9.01, 90.99)	10.10
Glass016vs2	192	9	(ve-win-float-proc; build-win-float-proc, build-win-non-float-proc, headlamps)	(8.89, 91.11)	10.29
Glass2	214	9	(Ve-win-float-proc; remainder)	(8.78, 91.22)	10.39
Ecoli4	336	7	(om; remainder)	(6.74, 93.26)	13.84
Yeast1vs7	459	8	(nuc; vac)	(6.72, 93.28)	13.87
Shuttle0vs4	1829	9	(Rad Flow; Bypass)	(6.72, 93.28)	13.87
Glass4	214	9	(containers; remainder)	(6.07, 93.93)	15.47
Page-blocks13vs2	472	10	(graphic; horiz.line, picture)	(5.93, 94.07)	15.85
Abalone9vs18	731	8	(18, 9)	(5.65, 94.25)	16.68
Glass016vs5	184	9	(tableware; build-win-float-proc, build-win-non-float-proc, headlamps)	(4.89, 95.11)	19.44
Shuttle2vs4	129	9	(Fpv Open; Bypass)	(4.65, 95.35)	20.5
Yeast1458vs7	693	8	(vac; nuc, me2, me3, pox)	(4.33, 95.67)	22.10
Glass5	214	9	(tableware; remainder)	(4.20, 95.80)	22.81
Yeast2vs8	482	8	(pox; cyt)	(4.15, 95.85)	23.10
Yeast4	1484	8	(me2; remainder)	(3.43, 96.57)	28.41
Yeast1289vs7	947	8	(vac; nuc, cyt, pox, erl)	(3.17, 96.83)	30.56
Yeast5	1484	8	(me1; remainder)	(2.96, 97.04)	32.78
Ecoli0137vs26	281	7	(pp; imL, cp, im, imU, imS)	(2.49, 97.51)	39.15
Yeast6	1484	8	(exc; remainder)	(2.49, 97.51)	39.15
Abalone19	4174	8	(19; remainder)	(0.77, 99.23)	128.87

Table 3
Parameter specification for the algorithms employed in the experimentation.

Algorithm	Parameter	Value
CO^2RBFN	Generations of the main loop	200
	Number of RBFs	5
Genetic	Generations of the main loop	200
	Individuals	40
	Chromosome length	Max = 6
	Crossover probability	0.6
	Mutation probability	0.1
	Mutation widths percent	0.2
	Mutation centers percent	0.2
	Tournament size	3
Incremental	Epsilon	0.1
	Alpha	0.3
	Delta	0.1
Clustering	Epsilon	0.001
	Alpha	0.2
	clusters	5

set. For each data-set the study considered the average results of the five partitions by five run repetitions.

6.2. Parameters of the algorithms

As stated above, alternative paradigms in the RBFN design field were selected to evaluate the training methods. The paradigms selected are: a procedure based on the incremental technique (Incremental method), a method based on the clustering technique (Clustering method), a genetic algorithm based on the Pittsburgh representation scheme (Genetic method) and the evolutionary cooperative–competitive method, CO^2RBFN.

Table 3 summarizes the parameters for the different approaches used in the experimental study. The parameters used in the experimentation are typical for these algorithms. A population, number of RBFs, of 5 was established for CO^2RBFN as in [23].

In the same way, to establish models with similar complexity, the maximum number of RBFs for RBFN (Chromosome length) for the Genetic method was established at 6. Thus, the experimentation demonstrates that the average number of RBFs of the models obtained is about 5.5.

Regarding the use of the SMOTE pre-processing method [56], only the 1-nearest neighbor (using the Euclidean distance) was considered to generate the synthetic samples. Furthermore both classes were balanced to the 50% distribution.

6.3. Statistical techniques

This study used statistical techniques for the analysis of the results, in order to provide a correct empirical study [68–70]. Specifically, non-parametric tests were considered. The reason for this was that as the initial conditions that guarantee the reliability of the parametric tests might not have been satisfied and that would decrease the credibility of the statistical analysis [68]. Specifically, a Wilcoxon signed-rank test was employed as a non-parametric statistical procedure to perform pairwise comparisons between two algorithms, the RBFN design method trained with LMS and the same method trained with SVD. This test is explained in Appendix A. Furthermore, any interested reader can find additional information on the website http://sci2s.ugr.es/sicidm/, together with the software to apply the statistical tests.

7. Results and analysis

In this section, a complete experimental analysis is carried out in order to study the performance of two different training methods over the original data-sets (Section 7.1) and over the pre-processed data-sets with SMOTE (Section 7.2).

7.1. Performance analysis with original data-sets (without preprocessing)

In Tables 4 and 5 the reader can observe the test GM accuracy results for data-sets with high IR and low IR, respectively. Results are shown with their associated standard deviation, and the data-sets of these tables are sorted according to their IR.

As discussed in Section 6.2, for CO^2RBFN and Clustering the number of neurons or clusters is fixed as a parameter to five RBFs. The number of neurons of the Genetic method was upper-limited to 6. Tables 6 and 7 show the average of number of neurons obtained by the Genetic and Incremental methods during the execution are shown. Results are shown with their associated standard deviation.

Table 4
Average GM test results for data-sets with high imbalance and without pre-processing.

Data-set	Clustering-LMS	Clustering-SVD	CO^2RBFN-LMS	CO^2RBFN-SVD	Genetic-LMS	Genetic-SVD	Incremental-LMS	Incremental-SVD
yeast2vs4	85.57 ± 5.36	79.26 ± 11.4	86.87 ± 4.9	81.87 ± 7.15	88.76 ± 4.29	83.94 ± 4.74	85.42 ± 7.7	76.62 ± 7.83
yeast05679vs4	67.98 ± 17.9	0 ± 0	77.06 ± 8.2	55.6 ± 17.5	75.38 ± 9.3	70.64 ± 10.71	72.05 ± 10.21	57.48 ± 14.67
vowel0	10.87 ± 16.73	6.4 ± 15.01	87.03 ± 5.86	70.26 ± 11.41	87.53 ± 6.52	83.72 ± 8.45	90.79 ± 6.36	90.25 ± 8.39
glass016vs2	26.57 ± 25.16	0 ± 0	47.27 ± 19.5	7.44 ± 20.33	45 ± 24.04	30.02 ± 32.55	34.51 ± 34.62	2 ± 9.8
glass2	0 ± 0	0 ± 0	57.23 ± 15.11	9.02 ± 20.94	48.17 ± 25.61	31.94 ± 30.06	12.92 ± 26.31	0 ± 0
ecoli4	39.09 ± 44.24	50.65 ± 45.94	90.96 ± 6.23	87.67 ± 9.44	91.44 ± 6.22	90.62 ± 6.42	75.3 ± 13.75	83.95 ± 9.55
shuttlec0vsc4	99.6 ± 0.81	99.6 ± 0.81	69.69 ± 12.5	99.67 ± 0.74	99.66 ± 0.74	99.59 ± 0.81	98.71 ± 2.69	97.11 ± 4.67
yeast1vs7	4.76 ± 12.9	0 ± 0	99.67 ± 0.8	27.48 ± 25.14	58.81 ± 18.44	48.24 ± 23.74	54.49 ± 19.9	27.29 ± 25.31
glass4	11.94 ± 24	18.31 ± 29.7	81.84 ± 14.07	73.27 ± 20.79	79.28 ± 14.16	79.84 ± 13.07	39.85 ± 40.81	43.72 ± 34.36
pageblocks13vs4	55.38 ± 21.61	24.15 ± 24.72	90.15 ± 7.7	73.7 ± 16.2	92.88 ± 8.15	92.44 ± 11.57	86.32 ± 11.53	63.55 ± 16.57
abalone918	3.41 ± 11.75	2.81 ± 9.52	75.7 ± 9.52	44.97 ± 11.01	75.1 ± 10.28	51.34 ± 18.55	36.29 ± 15.62	40.03 ± 14.3
glass016vs5	4 ± 19.6	9.46 ± 26	62.4 ± 40	61.7 ± 36.71	68.23 ± 36.44	70.82 ± 33.57	43.72 ± 43.33	38.99 ± 38.24
shuttlec2vsc4	86.14 ± 27.92	64.49 ± 45.18	93.59 ± 11.7	93.82 ± 11.58	93.98 ± 11.66	94.06 ± 11.68	62.14 ± 44.03	50.99 ± 46.22
yeast1458vs7	5.33 ± 14.8	0 ± 0	55.02 ± 14.7	0 ± 0	55.16 ± 20.39	28.56 ± 25.21	23.28 ± 25.89	0 ± 0
glass5	2.83 ± 13.86	0 ± 0	57.3 ± 43.95	50.9 ± 42.88	52.51 ± 44.21	68.98 ± 36.6	32.49 ± 44.1	46.04 ± 45.45
yeast2vs8	72.83 ± 13.39	72.83 ± 13.39	71.88 ± 14.13	66.18 ± 17.43	73 ± 13.44	70.89 ± 12.7	51.93 ± 24.53	51.53 ± 25.31
yeast4	2.51 ± 12.31	0 ± 0	77.33 ± 10.86	29.14 ± 24.44	78.47 ± 8.03	56.23 ± 11.58	44.32 ± 11.91	46.17 ± 13.55
yeast1289vs7	34.31 ± 25.98	0 ± 0	55.19 ± 22.5	6.53 ± 14.96	62.28 ± 9.91	32.29 ± 25.66	33.43 ± 26.34	4.89 ± 13.25
yeast5	9.32 ± 25.42	5.61 ± 15.2	94.12 ± 4.4	72.88 ± 19.36	94.83 ± 4.72	87.46 ± 10.94	77.83 ± 12.82	75.1 ± 7.89
ecoli0137vs26	73.68 ± 38.56	73.83 ± 38.66	70.5 ± 29.5	69.62 ± 40.68	72.59 ± 37.85	65.54 ± 42.45	35.75 ± 41.41	43.31 ± 46.09
yeast6	0 ± 0	0 ± 0	83.27 ± 10.45	66.82 ± 15.13	84.21 ± 10.39	79.2 ± 14.01	58.74 ± 12.69	61.35 ± 16.56
abalone19	0 ± 0	0 ± 0	50.12 ± 21.81	0 ± 0	66.66 ± 14.5	19.79 ± 27	4.34 ± 14.74	0 ± 0
Mean	**31.64 ± 16.92**	23.06 ± 12.52	**74.28 ± 14.93**	52.21 ± 17.45	**74.72 ± 15.42**	65.28 ± 18.73	**52.48 ± 22.33**	45.47 ± 18.09

The bold font is used to highligth the best result between the two versions (LMS or SVD) for each RBFN desing algorithm.

Table 5
Average GM test results for data-sets with low imbalance and without pre-processing.

Data-set	Clustering-LMS	Clustering-SVD	CO²RBFN-LMS	CO²RBFN-SVD	Genetic-LMS	Genetic-SVD	Incremental-LMS	Incremental-SVD
glass1	30 ± 25.25	49.95 ± 19.89	69.3 ± 6.16	68.64 ± 6.19	72.41 ± 6.16	69.21 ± 8.03	72.97 ± 6.9	72.93 ± 7.17
ecoli0vs1	95.02 ± 3.44	94.58 ± 3.29	97.03 ± 2.8	96.35 ± 3.24	95.89 ± 3.24	96.97 ± 2.83	96.29 ± 4.05	96.33 ± 3.71
wisconsin	97.45 ± 0.69	96.9 ± 0.96	97.29 ± 0.77	97.54 ± 0.64	97.37 ± 0.79	97.4 ± 0.88	95.48 ± 2.15	95.6 ± 1.48
pima	59.68 ± 6.56	61.22 ± 4.6	71.73 ± 4.3	71.93 ± 3.68	73.57 ± 3.27	72.87 ± 2.99	65.59 ± 6.86	67.37 ± 4.42
iris0	100 ± 0	100 ± 0	99.79 ± 1.01	99.59 ± 1.39	100 ± 0	99.9 ± 0.5	98.03 ± 3.25	99.28 ± 1.7
glass0	46.07 ± 21.93	57.49 ± 5.97	75.69 ± 7.12	73.31 ± 7.33	76.94 ± 8.47	75.71 ± 8.61	76.76 ± 6.98	81.04 ± 5.97
yeast1	48.73 ± 14.08	44.42 ± 16.02	70.77 ± 3.58	60.88 ± 4.29	70.08 ± 3.23	67.37 ± 3.61	62.51 ± 3.64	63 ± 2.87
vehicle1	48.05 ± 17.48	22.2 ± 20.85	65.57 ± 2.98	55.82 ± 7.78	66.75 ± 3.4	62.18 ± 4.86	61.48 ± 6.94	60.43 ± 4.72
vehicle2	31.36 ± 23.47	36.17 ± 21.67	83.37 ± 3.83	82.12 ± 4.79	78.51 ± 5.51	79.6 ± 5.44	93.27 ± 2.85	93.21 ± 3.21
vehicle3	21.85 ± 26.09	15.01 ± 17.5	66.97 ± 3.46	53.67 ± 5.71	65.58 ± 3.14	64.27 ± 4.3	60.38 ± 6.43	62.71 ± 3.93
haberman	19.73 ± 18.34	20.73 ± 19.07	61.21 ± 7.26	45.52 ± 11.07	59.14 ± 7.86	54.32 ± 7.5	42.98 ± 9.75	43.72 ± 6.7
glass0123vs456	86.49 ± 5.87	85.54 ± 6.03	92.27 ± 3.27	92.56 ± 3.41	94.02 ± 3.84	92.18 ± 3.9	93.27 ± 4.59	87.86 ± 6.67
vehicle0	37.85 ± 25.38	27.68 ± 23.57	89.12 ± 4.55	88.08 ± 5.04	87.29 ± 3.68	87.15 ± 4.51	92.14 ± 3.12	90.47 ± 2.46
ecoli1	77.64 ± 6.94	80.23 ± 4.94	88.65 ± 4.42	87.66 ± 4.77	87.49 ± 5.28	87.57 ± 4.36	82.47 ± 8.35	84.5 ± 6.7
newthyroid2	94.32 ± 5.94	89.43 ± 8.38	98.4 ± 3.72	97.19 ± 4.62	96.82 ± 4.31	96.72 ± 5.37	88.23 ± 11.09	90.49 ± 8.96
newthyroid1	92.96 ± 8.21	89.06 ± 8.15	98.02 ± 3.05	96.1 ± 4.64	96.87 ± 3.93	96.11 ± 4.78	91.72 ± 8.41	93.51 ± 6.71
ecoli2	90.5 ± 5.25	89.47 ± 5.75	92.02 ± 3.4	92.92 ± 4.65	92.35 ± 4.8	91.34 ± 5.5	91.25 ± 5.16	89.13 ± 7.51
segment0	25.9 ± 28.94	29.98 ± 32.76	96.05 ± 2.2	96.78 ± 1.96	95.79 ± 1.72	96.05 ± 1.74	95.27 ± 4.82	98.33 ± 1.22
glass6	83.32 ± 7.74	86.57 ± 8.07	87.07 ± 7.38	87.12 ± 7.8	88.9 ± 7.39	89.83 ± 6.54	84.54 ± 6.82	83.22 ± 6.19
yeast3	39.12 ± 36.99	35.96 ± 38.75	89.51 ± 2.58	82.27 ± 5.32	89.89 ± 2.76	87.85 ± 3.13	79.6 ± 5.7	78.97 ± 4.16
ecoli3	72.27 ± 22.99	60.58 ± 32.26	87.02 ± 7.65	78.96 ± 11.91	85.89 ± 8.26	86.66 ± 6.54	70.47 ± 13.27	68.47 ± 13.05
pageblocks0	65.48 ± 3.9	61.85 ± 3.81	86.07 ± 2.4	74.66 ± 3.63	85.73 ± 2.27	81.12 ± 2.96	81.06 ± 3.36	81.19 ± 2.73
Mean	**61.99 ± 14.34**	60.68 ± 13.74	**84.68 ± 4**	80.89 ± 5.18	**84.42 ± 4.24**	83.29 ± 4.49	80.72 ± 6.11	**80.99 ± 5.1**

The bold font is used to highligth the best result between the two versions (LMS or SVD) for each RBFN desing algorithm.

At first glance, if we examine the full results (Tables 4 and 5) with original imbalanced data-sets, the training method LMS outperforms SVD. This can be observed from the average value for GM metric. However, this in not the case for the Incremental method, where the SVD training algorithm slightly outperforms the LMS training algorithm, for the group of data-sets with lower IR, Table 5. In this case, the number of neurons created by LMS Incremental version method is higher than the SVD Incremental version method. As a result, there may be certain overlearning.

We can also appreciate that the differences between pair algorithms (LMS–SVD) are higher for high IR data-sets (Table 4), where all RBFN design paradigms trained with LMS outperform the SVD version paradigm. However, in Table 5 with the results of low IR data-sets, the difference between any paradigm trained with LMS and trained with SVD is small.

For the analysis of the number of RBFs in the models obtained, first it must be remembered that, for the CO²RBFN and Clustering methods, the number of RBFs was set to 5 and, for the Genetic method, the maximum number of nodes was fixed to 6 (it obtains an average of 5.4 nodes). So, this analysis is focused on the Incremental method.

The Incremental method, with no upper limit, obtained much more complex models with a much higher number of nodes. In fact, Table 7 shows that for the Incremental paradigm trained with LMS algorithm the average number of nodes was 76.57. Moreover, the number of RBFs for LMS Incremental was clearly higher than for the Incremental-SVD method for high and low IR data-sets (see Tables 6 and 7). In addiction, the number of RBFs introduced in models which predict low IR data-sets was higher than the number of RBFs introduced in models which predict high IR data-sets. This fact can be explained from the fundamentals of how classical classifiers (an so Incremental) work with an imbalanced dataset. When the IR of the datasets grows, there are less instances of the minority class, and almost all are of the majority class.

Table 6
Average number of nodes for data-sets with high imbalance and without pre-processing.

Data-set	Genetic-LMS	Genetic-SVD	Incremental-LMS	Incremental-SVD
yeast2vs4	5.52 ± 0.5	5.44 ± 0.7	28.04 ± 5.83	21.84 ± 2.56
yeast05679vs4	5.44 ± 0.57	5.6 ± 0.63	63.28 ± 7.12	37.16 ± 3.06
vowel0	5.52 ± 0.7	5.48 ± 0.76	31.28 ± 2.41	27.84 ± 2.66
glass016vs2	5.76 ± 0.43	5.76 ± 0.43	24.16 ± 5.07	12.08 ± 1.65
glass2	5.72 ± 0.45	5.68 ± 0.61	17.6 ± 8.19	11.88 ± 1.34
ecoli4	5.28 ± 0.72	5.2 ± 0.75	11.32 ± 2.91	10.4 ± 2.42
shuttlec0vsc4	4.32 ± 1.09	4.6 ± 1.13	10.16 ± 8.41	5.48 ± 1.94
yeast1vs7	5.68 ± 0.55	5.48 ± 0.7	50.68 ± 8.54	21 ± 2.19
glass4	5.4 ± 0.63	5.76 ± 0.43	10.08 ± 1.7	9.72 ± 1.99
pageblocks13vs4	5.72 ± 0.45	5.64 ± 0.56	16.28 ± 4.61	12.44 ± 2.37
abalone918	5.4 ± 0.85	4.96 ± 1.11	21.52 ± 4.28	20.36 ± 2.17
glass016vs5	5.36 ± 0.69	5.4 ± 0.63	8.64 ± 2.68	7.16 ± 1.49
shuttlec2vsc4	4.6 ± 1.06	4.72 ± 1.04	4.84 ± 0.92	4.68 ± 1.01
yeast1458vs7	5.52 ± 0.57	5.28 ± 0.92	86.92 ± 20.74	24.08 ± 1.6
glass5	5.44 ± 0.85	5.68 ± 0.55	7.36 ± 2.02	6.64 ± 1.72
yeast2vs8	5.64 ± 0.56	4.8 ± 1.1	13.48 ± 1.9	12.8 ± 2.12
yeast4	5.44 ± 0.57	5.48 ± 0.76	59.44 ± 39.03	36.68 ± 2.46
yeast1289vs7	5.6 ± 0.49	5.44 ± 0.75	75.96 ± 19.24	23.4 ± 1.81
yeast5	5.12 ± 0.86	5.04 ± 1.15	39.64 ± 13.65	24.28 ± 2.99
ecoli0137vs26	5.4 ± 0.63	5.24 ± 0.76	6.32 ± 1.67	5.84 ± 0.97
yeast6	5.68 ± 0.47	5.28 ± 0.96	36.12 ± 24.63	23 ± 2.3
abalone19	5.28 ± 0.67	4.72 ± 1.11	27.6 ± 21.52	15.52 ± 1.36
Mean	5.4 ± 0.65	5.3 ± 0.8	29.58 ± 9.41	17.01 ± 2.01

Table 7
Average number of nodes for data-sets with low imbalance and without pre-processing.

Data-set	Genetic-LMS	Genetic-SVD	Incremental-LMS	Incremental-SVD
glass1	5.6 ± 0.49	5.96 ± 0.2	41.6 ± 2.81	36.16 ± 3.17
ecoli0vs1	5.32 ± 0.61	5.4 ± 0.69	11.2 ± 4.1	10 ± 2.64
wisconsin	4.92 ± 0.63	5.04 ± 0.96	27.72 ± 5.22	24.04 ± 3.4
pima	5.52 ± 0.57	5.88 ± 0.33	219.56 ± 12.33	163.72 ± 7.59
iris0	4.56 ± 0.98	3.84 ± 1.12	5.76 ± 2.72	4.32 ± 0.68
glass0	4.96 ± 0.82	5.84 ± 0.37	31.04 ± 5.05	27.08 ± 1.85
yeast1	5.44 ± 0.64	5.84 ± 0.46	260 ± 18.58	193.68 ± 9.49
vehicle1	5.52 ± 0.57	5.52 ± 0.85	215.84 ± 10.52	150.96 ± 6.87
vehicle2	5.76 ± 0.51	5.8 ± 0.4	65.52 ± 5.16	52.68 ± 4.89
vehicle3	5.32 ± 0.79	5.6 ± 0.69	201.88 ± 7.7	158.24 ± 5.49
haberman	5.56 ± 0.57	5.36 ± 0.89	57.68 ± 3.08	49.56 ± 2.89
glass0123vs456	5.32 ± 0.73	5.6 ± 0.57	19.24 ± 3.42	15.48 ± 3.45
vehicle0	5.64 ± 0.48	5.84 ± 0.46	76.16 ± 7.92	60.84 ± 6.12
ecoli1	5.68 ± 0.55	5.48 ± 0.64	37.64 ± 4.17	29.12 ± 3.72
newthyroid2	5.04 ± 0.96	5.32 ± 0.73	12.4 ± 4.85	8.16 ± 1.91
newthyroid1	5 ± 0.94	4.96 ± 0.87	11.2 ± 3.56	8.4 ± 2.83
ecoli2	5.36 ± 0.69	5.52 ± 0.85	24.36 ± 6.12	16.36 ± 3.03
segment0	5.8 ± 0.49	5.88 ± 0.33	152.72 ± 174.54	23.32 ± 2.89
glass6	5.44 ± 0.57	5.32 ± 0.84	13.68 ± 3.94	10.28 ± 2.07
yeast3	5.36 ± 0.69	5.64 ± 0.79	76.36 ± 14.51	57.72 ± 3.7
ecoli3	5.32 ± 0.73	4.68 ± 0.88	23.88 ± 3.55	21.52 ± 2.47
pageblocks0	5.6 ± 0.63	4.92 ± 0.98	99.12 ± 5.29	94.8 ± 3.94
Mean	5.37 ± 0.67	5.42 ± 0.68	76.57 ± 14.05	55.29 ± 3.87

Table 8
Wilcoxon test of original data-sets with high IR. R^+ corresponds to LMS and R^- to SVD.

Comparison	R^+	R^-	p-Value
Clustering-LMS vs Clustering-SVD	177.0	54.0	0.031164
CO^2RBFN-LMS vs. CO^2RBFN-SVD	238.0	15.0	0.000277
Genetic-LMS vs Genetic-SVD	223.0	30.0	0.001637
Incremental-LMS vs Incremental-SVD	190.0	63.0	0.037728

In these situations the classifiers obtained are simpler (with a lower number of RBFs) because the classification space is easier to represent. On the other hand, if the imbalance level of the dataset is low, there is a more similar number of instances of the majority and minority classes. As result difficult classification tasks and more complex (with more RBFs) models are defined.

Here follows the analysis of the results obtained by the two training methods of the different algorithms using statistical tests are analyzed. The aim is to determine whether the differences between the two versions (LMS or SVD) of the same algorithm are statistically significant. To do so, a Wilcoxon signed-ranks test is applied to compare the results of each version and to detect significant differences between the behavior of pairs of algorithms.

From the above results statistical tests were applied to data-sets with high IR, Table 8, and to data-sets with low IR, Table 9. In Table 8, R^+ corresponds to LMS and is the sum of ranks for the data-sets with high IR on which the SVD algorithm is outperformed, and R^- is the sum of ranks for the opposite (see Appendix A). In this table a p-value below 0.05 indicates that the null hypothesis of equality of means is rejected with a 95% confidence level, which implies that LMS outperforms SVD with significant differences for all RBFN design paradigms.

Table 9
Wilcoxon test of original data-sets with low IR. R^+ corresponds to LMS and R^- to SVD.

Comparison	R^+	R^-	p-Value
Clustering-LMS vs Clustering-SVD	141.0	90.0	0.366155
CO^2RBFN-LMS vs. CO^2RBFN-SVD	224.0	29.0	0.001464
Genetic-LMS vs Genetic-SVD	200.0	53.0	0.016285
Incremental-LMS vs Incremental-SVD	96.0	157.0	1

Now, Table 9 analyzes the performance of the training algorithms in imbalanced data-sets with low IR. In this table R^+ corresponds to LMS and R^- to SVD. A p-value below 0.05 indicates that the null hypothesis of equality of means is rejected with a 95% confidence level. Therefore, LMS outperforms SVD with significant differences for the CO^2RBFN and Pittsburgh based evolutionary paradigms.

From these results, it can be concluded that, with original imbalanced data-sets and their performance measure (GM metric), it is more appropriate to use LMS as a training algorithm for RBFN design, especially when the IR of these data-sets is higher. This can be explained by the fact that SVD is a more general training optimization method, with internal mechanisms that promote a bias toward the majority class and that disfavors the minority class. On the other hand, LMS has a more localized operation mode which, for each given instance, applies corrections in weights that only affect the nearest RBFs to this instance. Therefore, minority class instances can receive adequate training if they are isolated enough.

7.2. Performance analysis with preprocessed data-sets

The results obtained for preprocessed data-sets with the SMOTE technique are shown in Tables 10 and 11, for high and low IR respectively.

As in the previous subsection, the number of neurons obtained by the Genetic and the Incremental paradigm, achieved in the experimentation, are shown in Table 12 for high IR data-sets and in Table 13 for low IR data-sets.

From the results obtained by the algorithms for the preprocessed data-sets with high IR, Table 10, it can be observed that paradigms trained with LMS and paradigms trained with SVD achieved similar results to the ones obtained by original data-sets. The LMS algorithm outperforms the SVD algorithm with CO^2RBFN and Incremental paradigms, and the SVD algorithm outperforms the LMS algorithm for Genetic and Clustering. In Table 11, for data-sets with low IR, SVD paradigms outperform LMS paradigms for all the methods.

Regarding the number of RBFs achieved by the models, the conclusions were similar to those obtained in the previous section. From Tables 12 and 13, it can observed that the Genetic method obtained models with an approximate number of nodes in the

Table 10
Average GM test results for data-sets with high imbalance and with pre-processing (SMOTE).

Data-set	Clustering-LMS	Clustering-SVD	CO²RBFN-LMS	CO²RBFN-SVD	Genetic-LMS	Genetic-SVD	Incremental-LMS	Incremental-SVD
yeast2vs4	82.53 ± 6.1	85.73 ± 5.25	87.29 ± 3.6	88.33 ± 3.6	89.04 ± 3.34	88.92 ± 3.58	91.49 ± 3.89	60.85 ± 5.71
yeast05679vs4	76 ± 6.48	80.27 ± 5.44	78.22 ± 5.1	78 ± 6.03	78.39 ± 5.44	78.73 ± 5.45	78.63 ± 6.63	77.76 ± 6.22
vowel0	47.39 ± 9.87	60.36 ± 9.96	93.77 ± 3.52	94.63 ± 2.76	93.98 ± 4.26	93.81 ± 2.19	98.98 ± 0.95	99.13 ± 1.13
glass016vs2	50.23 ± 9.45	52.92 ± 21.19	56.44 ± 20.8	61.1 ± 14.69	57.77 ± 17.59	60.2 ± 15.53	59.56 ± 24.46	60.85 ± 13.85
glass2	47.15 ± 16.15	53.95 ± 9.76	58.15 ± 25.25	62.51 ± 23.49	46.59 ± 23.28	59.44 ± 18.98	63 ± 15.31	63.46 ± 16.13
ecoli4	92.81 ± 4.1	92.49 ± 5.85	89.65 ± 6.17	90.06 ± 6.29	89.81 ± 6.4	90.78 ± 6.37	90.86 ± 6.75	87.58 ± 6.21
shuttlec0vsc4	99.6 ± 0.81	99.6 ± 0.81	99.58 ± 0.8	99.59 ± 0.81	99.59 ± 0.83	99.58 ± 0.82	98.76 ± 2.52	99.57 ± 0.81
yeast1vs7	63.3 ± 9.65	70 ± 10.99	99.49 ± 0.9	70.08 ± 10.57	75.13 ± 8.88	73.63 ± 7.83	70.34 ± 10.05	73 ± 9.8
glass4	81.46 ± 9.83	83.93 ± 10.9	85.45 ± 12.62	87.23 ± 12.64	85.67 ± 14.16	88.96 ± 11.08	88.31 ± 9.78	86.39 ± 11.94
pageblocks13vs4	75.11 ± 11.34	75.74 ± 6.08	96.65 ± 3.6	96.39 ± 4.02	95.03 ± 6.69	95.92 ± 4.07	97.33 ± 3.59	95.26 ± 4.76
abalone918	39.41 ± 21.49	64.24 ± 10.04	77.41 ± 10.6	77.38 ± 14.11	76.22 ± 8.33	74.97 ± 10.05	65.82 ± 9.73	62.44 ± 13.92
glass016vs5	86.29 ± 7.95	86.62 ± 9.21	84.71 ± 31.8	80.76 ± 35.74	76.52 ± 38.65	73.59 ± 38.27	75.62 ± 38.63	76.74 ± 39.23
shuttlec2vsc4	90.25 ± 10.21	90.07 ± 27.11	99.51 ± 1	99.5 ± 1.35	95.51 ± 19.52	95.51 ± 19.53	94.06 ± 7.22	95.75 ± 19.56
yeast1458vs7	41.82 ± 16.63	47.76 ± 12.98	60.8 ± 11.2	60.53 ± 14.95	54.72 ± 19.21	55.41 ± 19.34	62.63 ± 20.28	62.33 ± 15.18
glass5	80.1 ± 24.18	87.05 ± 7.81	74.91 ± 38.45	85.59 ± 21.68	59.38 ± 42.15	60.55 ± 42.74	78.48 ± 31.26	84.17 ± 21.27
yeast2vs8	63.44 ± 20.14	75.27 ± 8.42	77.31 ± 12.23	77.99 ± 10.99	76.05 ± 12.42	75.77 ± 13.84	76.89 ± 13.4	76.11 ± 16.63
yeast4	78.94 ± 6.51	82.66 ± 3.81	78.95 ± 4.23	80.93 ± 4.33	81.95 ± 3.91	82.02 ± 4.5	77.65 ± 8.94	75.81 ± 9.78
yeast1289vs7	45.23 ± 17.24	59.46 ± 8.28	70.14 ± 7.5	71.23 ± 7.71	72.58 ± 5.12	71.14 ± 10.55	63.49 ± 9.83	61.09 ± 12.37
yeast5	95.92 ± 0.77	95.89 ± 0.51	94.69 ± 3.6	94.01 ± 4.77	93.89 ± 4.27	95.6 ± 3.16	93.85 ± 3.21	94.24 ± 2.98
ecoli0137vs26	68.13 ± 30.93	68.45 ± 31.2	70.09 ± 36.3	69.77 ± 36.14	71.1 ± 36.95	70.76 ± 36.73	72.92 ± 37.85	68.61 ± 40.01
yeast6	84.81 ± 6.28	87.65 ± 6.52	86.57 ± 8.4	86.25 ± 7.99	85.85 ± 8.75	85.75 ± 8.88	81.78 ± 11.06	82.69 ± 9.54
abalone19	56.12 ± 8.53	61.42 ± 12.14	70.18 ± 11.77	66.44 ± 17.93	64.69 ± 19.89	69.58 ± 16.84	58.57 ± 17.14	63.27 ± 9.18
Mean	70.27 ± 11.57	**75.52 ± 10.19**	**81.36 ± 11.79**	80.83 ± 11.94	78.16 ± 14.09	**79.12 ± 13.65**	**79.05 ± 13.29**	77.6 ± 13.01

The bold font is used to highligth the best result between the two versions (LMS or SVD) for each RBFN desing algorithm.

range from 5.4 to 5.8 for high and low IR. The Incremental method obtained many more complex models, even more than in the previous section. Thus Table 13 shows that for the Incremental method trained with LMS the average number of nodes is 96.73. As in the section above for the Incremental-LMS method the number of RBFs is clearly higher than for the Incremental-SVD method for low and high IR data-sets (see Tables 12 and 13). Moreover, the number of RBFs introduced in models which predict low IR data-sets is higher than the number of RBFs introduced in models which predict high IR data-sets.

Then statistical tests were applied to data-sets with high IR (Table 14), and to data-sets with low IR (Table 15). In Table 14, R^+ corresponds to SVD and R^- to LMS, as it can be seen, there are only significant differences, in favor of SVD, for the clustering paradigm.

In Table 15, data-sets with low IR, R^+ corresponds to SVD and R^- to LMS. In addition, p-values below 0.05 are observed. This indicates that SVD methods outperform LMS methods with significant differences for CO²RBFN, Clustering and Incremental methods.

As a conclusion of this study, it can be stated that results for preprocessed data-sets were coherent with those obtained with the original data-sets and validated our thesis: the use of weights training algorithms with local behavior benefits the design of RBFNs for imbalanced data-sets. Due to the use of SMOTE (an oversampling technique oriented to balance data-sets), these data-sets were more balanced than in the previous section. This implies that SVD, a more general optimization method, can improve its functioning.

Thus, with high IR data-sets, there were not significant differences between SVD and LMS, except for the Clustering design method where SVD outperforms LMS. On the other hand, for low IR data-sets SVD training method outperforms LMS for Clustering, CO²RBFN and Incremental methods with significant differences.

Table 11
Average GM test results for data-sets with low imbalance and with pre-processing (SMOTE).

Data-set	Clustering-LMS	Clustering-SVD	CO²RBFN-LMS	CO²RBFN-SVD	Genetic-LMS	Genetic-SVD	Incremental-LMS	Incremental-SVD
glass1	56.94 ± 7.16	65.84 ± 6.36	69.86 ± 6.44	69.71 ± 4.03	72.59 ± 5.38	71.72 ± 5.5	76.76 ± 5.64	78.73 ± 5.53
ecoli0vs1	95.73 ± 3.35	95.23 ± 2.9	96.18 ± 2.96	96.64 ± 2.91	96.19 ± 2.79	96.9 ± 2.68	95.84 ± 3.36	96.04 ± 3.57
wisconsin	97.47 ± 0.94	97.42 ± 0.94	97.26 ± 0.85	97.34 ± 0.7	97.56 ± 0.86	97.32 ± 0.85	96 ± 2.02	96.8 ± 1.09
pima	63.77 ± 5.55	68.83 ± 5.18	72.56 ± 3.71	73.43 ± 4.18	71.93 ± 4.9	71.01 ± 4.47	69.16 ± 4.4	68.91 ± 3.83
iris0	100 ± 0	100 ± 0	99.9 ± 0.5	99.9 ± 0.5	99.79 ± 1.01	100 ± 0	90.84 ± 10.42	99.39 ± 1.5
glass0	65.8 ± 5.1	69.58 ± 4.86	75.64 ± 6.99	77.73 ± 6.68	77.27 ± 8.8	80.12 ± 6.89	79.15 ± 5.06	80.02 ± 6.87
yeast1	56.72 ± 6.94	64.34 ± 2.54	70.08 ± 3.47	71.07 ± 3.43	70.76 ± 3.04	70.93 ± 3.22	68.55 ± 2.48	69.81 ± 2.52
vehicle1	40.74 ± 17.64	62.61 ± 4.46	69.08 ± 4.37	70.36 ± 3.42	69.26 ± 4.54	68.99 ± 3.6	69.74 ± 4.22	69.84 ± 3.51
vehicle2	50.47 ± 16.55	61.37 ± 7.75	87.24 ± 3.98	89.72 ± 2.61	81.86 ± 4.61	83.93 ± 4.13	95.8 ± 1.36	95.94 ± 2.04
vehicle3	40.88 ± 20.64	63.55 ± 3.27	69.55 ± 3.98	70.75 ± 4.55	68.42 ± 4.41	69.22 ± 3.36	74.47 ± 3.21	73.48 ± 2.94
haberman	26.72 ± 18.99	51.84 ± 8.81	60.21 ± 6.25	60.4 ± 6.72	60.76 ± 7.12	60.59 ± 6.84	53.44 ± 7.54	52.14 ± 6.71
glass0123vs456	89.41 ± 6.14	89.67 ± 4.83	93.78 ± 3.28	93.04 ± 4.22	94.2 ± 2.79	93.86 ± 3.59	92.49 ± 3.1	92.68 ± 3.94
vehicle0	67.07 ± 4.01	70.21 ± 3.58	92.15 ± 2.48	92.61 ± 1.92	87.47 ± 2.97	88.46 ± 3.11	93.3 ± 1.71	93.76 ± 1.42
ecoli1	87.26 ± 2.72	86.93 ± 2.5	87.84 ± 4.1	87.64 ± 3.8	87.04 ± 4.76	88.16 ± 4.42	87.42 ± 4.82	88.87 ± 4.31
newthyroid2	97.21 ± 2.34	98.42 ± 1.99	98.46 ± 2.22	99.04 ± 1.11	98.52 ± 2.3	98.4 ± 2.12	91.45 ± 7.56	97.25 ± 4.38
newthyroid1	97.56 ± 2.25	96.96 ± 5.68	97.54 ± 4.03	98.41 ± 2.26	98.04 ± 2.91	97.69 ± 3.07	91.79 ± 8.12	96.15 ± 4.85
ecoli2	89.68 ± 5.04	90.33 ± 4.7	93.14 ± 4.5	92.51 ± 4.19	92.84 ± 3.29	91.71 ± 4.35	90.62 ± 5.4	92.9 ± 4.6
segment0	81.1 ± 2.83	79.08 ± 7.03	97.97 ± 0.81	97.93 ± 1.06	96.57 ± 1.43	97.23 ± 1.74	98.41 ± 1.32	99.02 ± 0.62
glass6	87.73 ± 7.03	88.88 ± 6.03	85.93 ± 8.39	86.38 ± 7.87	88.77 ± 6.79	89.5 ± 8.01	87.43 ± 8.85	89.13 ± 6.66
yeast3	85.47 ± 3.21	86.76 ± 4.31	91.11 ± 2.34	91.4 ± 2.49	90.89 ± 2.34	91.08 ± 2.75	87.36 ± 3.07	88.17 ± 2.42
ecoli3	88.01 ± 4.96	88.17 ± 4.93	85.72 ± 7.9	86.24 ± 7.03	87.19 ± 6.61	84.48 ± 8.28	81.59 ± 10.08	81.64 ± 11.47
pageblocks0	74.2 ± 9.95	77.3 ± 3.75	88.6 ± 2.01	88.28 ± 1.74	86.81 ± 1.77	87.19 ± 1.89	88.84 ± 1.71	90.64 ± 1.04
Mean	74.54 ± 6.97	**79.7 ± 4.38**	85.45 ± 3.89	**85.93 ± 3.52**	85.21 ± 3.88	**85.39 ± 3.86**	84.57 ± 4.79	**85.97 ± 3.9**

The bold font is used to highligth the best result between the two versions (LMS or SVD) for each RBFN desing algorithm.

Table 12
Average number of nodes for data-sets with high imbalance and with pre-processing (SMOTE).

Data-set	Genetic-LMS	Genetic-SVD	Incremental-LMS	Incremental-SVD
yeast2vs4	5.72 ± 0.45	5.96 ± 0.2	62.76 ± 10.18	42.16 ± 4.42
yeast05679vs4	5.72 ± 0.45	5.92 ± 0.27	101.08 ± 8.4	72.56 ± 5.78
vowel0	5.76 ± 0.51	5.92 ± 0.27	60.68 ± 6.28	38 ± 4.64
glass016vs2	5.88 ± 0.32	5.88 ± 0.32	38.4 ± 4.37	28.72 ± 3.55
glass2	5.56 ± 0.7	5.96 ± 0.2	32.68 ± 5.82	27.12 ± 5.6
ecoli4	5.68 ± 0.47	5.8 ± 0.4	27.88 ± 6.33	16.12 ± 4.03
shuttlec0vsc4	4.64 ± 1.26	4.2 ± 1.17	14.72 ± 7.74	6.6 ± 2.47
yeast1vs7	5.4 ± 0.75	5.8 ± 0.4	87.8 ± 8.84	67.12 ± 6.78
glass4	5.72 ± 0.53	5.84 ± 0.37	28.64 ± 5.84	18.6 ± 3.72
pageblocks13vs4	5.68 ± 0.55	5.88 ± 0.32	32.48 ± 6.56	20.24 ± 3.39
abalone918	5.64 ± 0.62	5.92 ± 0.27	52.08 ± 7.23	38.52 ± 4.07
glass016vs5	5.68 ± 0.47	5.64 ± 0.56	20 ± 6.27	16.68 ± 3.79
shuttlec2vsc4	5.52 ± 0.64	5.52 ± 0.57	9.56 ± 1.9	9.64 ± 1.96
yeast1458vs7	5.84 ± 5.84	6 ± 0	116.52 ± 15.93	87.6 ± 7.69
glass5	5.8 ± 0.4	5.84 ± 0.46	20.6 ± 5.87	15.56 ± 3.6
yeast2vs8	5.56 ± 0.57	5.84 ± 0.37	46.64 ± 15.64	26.56 ± 8.39
yeast4	5.56 ± 0.7	5.92 ± 0.27	121.4 ± 13.64	95.12 ± 6.02
yeast1289vs7	5.8 ± 0.4	5.68 ± 0.55	117.6 ± 10.36	80.8 ± 8.16
yeast5	5.08 ± 0.74	5.52 ± 0.75	60.48 ± 7.26	46 ± 5.97
ecoli0137vs26	5.72 ± 0.45	5.88 ± 0.32	32.12 ± 15.72	17.44 ± 5.52
yeast6	5.32 ± 0.55	5.88 ± 0.32	77.2 ± 15.05	57 ± 9.34
abalone19	5.64 ± 0.56	5.84 ± 0.37	84.36 ± 15.3	68.32 ± 9.68
Mean	5.59 ± 0.81	5.76 ± 0.4	56.62 ± 9.12	40.75 ± 5.39

Table 13
Average number of nodes for data-sets with low imbalance and with pre-processing (SMOTE).

Data-set	Genetic-LMS	Genetic-SVD	Incremental-LMS	Incremental-SVD
glass1	5.72 ± 0.45	5.84 ± 0.37	46.36 ± 4.45	38.64 ± 3.96
ecoli0vs1	5.16 ± 0.78	4.84 ± 1.05	13.84 ± 5.5	9.92 ± 1.49
wisconsin	5.2 ± 0.57	5.16 ± 0.83	42.64 ± 14.33	26.64 ± 2.71
pima	5.56 ± 0.57	5.92 ± 0.27	271.16 ± 11.84	178.72 ± 7.07
iris0	4.04 ± 1.31	4.4 ± 1.41	19.72 ± 11.92	4.84 ± 0.88
glass0	5.52 ± 0.64	5.8 ± 0.4	35.48 ± 3.47	30.96 ± 2.47
yeast1	5.48 ± 0.64	5.92 ± 0.27	326.48 ± 16.92	222.08 ± 6.79
vehicle1	5.84 ± 0.37	5.84 ± 0.37	281.8 ± 14.4	190.12 ± 7.88
vehicle2	5.84 ± 0.37	6 ± 0	93.84 ± 10.62	58.16 ± 8.22
vehicle3	5.72 ± 0.53	5.96 ± 0.2	280.52 ± 16.85	194.2 ± 8.41
haberman	5.52 ± 0.7	5.8 ± 0.4	66.56 ± 4.13	54.68 ± 4.37
glass0123vs456	5.64 ± 0.48	5.64 ± 0.48	22.84 ± 4.48	17.4 ± 1.65
vehicle0	5.88 ± 0.32	5.96 ± 0.2	108.48 ± 9.59	76.08 ± 6.85
ecoli1	5.56 ± 0.5	5.72 ± 0.53	43.4 ± 6.1	31.64 ± 4.44
newthyroid2	5 ± 0.98	5.4 ± 0.57	23.84 ± 8.59	9 ± 2.32
newthyroid1	5.12 ± 0.82	5.28 ± 0.83	19.64 ± 8.92	8.56 ± 1.75
ecoli2	5.48 ± 0.7	5.56 ± 0.57	31.68 ± 5.64	20.72 ± 3.34
segment0	5.8 ± 0.4	5.92 ± 0.27	81.44 ± 53.71	46.32 ± 10.93
glass6	5.92 ± 0.27	5.72 ± 0.53	24.56 ± 5.46	16.72 ± 3.79
yeast3	5.52 ± 0.57	5.88 ± 0.32	126.36 ± 11.9	85.32 ± 5.68
ecoli3	5.64 ± 0.56	5.76 ± 0.51	37.16 ± 5.42	30.2 ± 3.76
pageblocks0	5.68 ± 0.47	5.92 ± 0.27	130.24 ± 16.45	96.32 ± 4.86
Mean	5.49 ± 0.59	5.65 ± 0.48	96.73 ± 11.39	65.78 ± 4.71

Table 14
Wilcoxon test of preprocessed data-sets with high IR. R^+ corresponds to SVD and R^- to LMS.

Comparison	R^+	R^-	p-Value
Clustering-SVD vs Clustering-LMS	225.0	6.0	0.000132
CO^2RBFN-SVD vs. CO^2RBFN-LMS	150.0	103.0	0.435876
Genetic-SVD vs Genetic-LMS	155.0	98.0	0.346447
Incremental-SVD vs Incremental-LMS	109.0	144.0	1

Table 15
Wilcoxon test of preprocessed data-sets with low IR. R^+ corresponds to SVD and R^- to LMS.

Comparison	R^+	R^-	p-Value
Clustering-SVD vs Clustering-LMS	204.0	27.0	0.001979
CO^2RBFN-SVD vs CO^2RBFN-LMS	203.0	50.0	0.012424
Genetic-SVD vs Genetic-LMS	149.0	104.0	0.455239
Incremental-SVD vs Incremental-LMS	221.0	32.0	0.002041

8. Conclusions

Nowadays, research on imbalanced data-sets is receiving more attention because they are present in many real applications. However, traditional classifiers are often unable to handle these kinds of data-sets adequately. This is because they are biased toward a correct classification of the majority classes and leave aside minority classes.

The present study identifies the most suitable RBFN weights training methodology to avoid the above mentioned bias toward the majority class. To do so, the study considered the two main weights training paradigms (local and global) used in the RBFN design. For these typical training paradigms used to calculate output weights, two representative algorithms were chosen: LMS, a gradient descent-based algorithm that makes more local transformations; and SVD, a matrix computation algorithm that achieves a global solution. In order to test these training algorithms, different

RBFN design methods belonging to the main paradigms in the field were chosen.

The results, statistically validated, show:

- For data-sets with the highest IR, i.e., data-sets with high IR and without preprocessing, all the RBFN design methods trained with the LMS algorithm outperformed, with significant differences, RBFN design methods trained with the SVD algorithm.
- For data-sets with low IR and also without preprocessing, RBFN design methods trained with the LMS algorithm often outperformed RBFN design methods trained with the SVD algorithm, but not always with significant differences.
- For data-sets with an original high IR, but rebalanced with SMOTE, SVD methods outperformed LMS methods. However, it was difficult to find significant differences.
- Finally, for the data-sets with the lowest level of IR, i.e., data-sets with low IR and preprocessed with SMOTE, SVD methods outperformed LMS methods, even with frequently significant differences.

A general conclusion is that the higher the data-set IR, the better the results are achieved by a local weights training methods (such as LMS). This fact can be explained because SVD obtains more global optimization models, which improves the classification performance of the majority class and disfavors the minority class. Nevertheless, LMS with a more local (per instance) optimization is able to achieve best results. When the IR of the data-set is lower or the data-set is balanced with methods like SMOTE, the SVD algorithm improves its performance.

Acknowledgement

This work was supported by the Spanish Ministry of Science and Technology under Project TIN2012-33856.

Appendix A. Pairwise comparisons: Wilcoxon signed-ranks test

This is the analog of the paired t-test in non-parametrical statistical procedures; therefore, it is a pairwise test that aims to detect significant differences between the behavior of two algorithms.

Let d_i be the difference between the performance scores of the two classifiers on i-th out of N_{ds} data-sets. The differences are ranked according to their absolute values; average ranks are assigned in the case of ties. Let R^+ be the sum of ranks for the data-sets in which the second algorithm outperformed the first, and R^- the sum of ranks for the opposite. Ranks of $d_i = 0$ are split evenly among the sums; if there is an odd number in them, one is ignored:

$$R^+ = \sum_{d_i>0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i) \tag{A.1}$$

$$R^- = \sum_{d_i<0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i) \tag{A.2}$$

Let T be the smallest of the sums, $T = min(R^+, R^-)$. If T is less than or equal to the value of the distribution of Wilcoxon for N_{ds} degrees of freedom (Table B.12 in [71]), the null hypothesis of equality of means is rejected.

References

[1] 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, Inform. Sci. 250 (2013) 113–141.
[2] V. García, J. Sánchez, R. Mollineda, On the effectiveness of preprocessing methods when dealing with different levels of class imbalance, Knowl.-Based Syst. 25 (1) (2012) 13–21.
[3] F. Fernández, V. López, M. Galar, M. del Jesus, F. Herrera, Analysing the classification of imbalanced data-sets with multiple classes: binarization techniques and ad-hoc approaches, Knowl.-Based Syst. 42 (2013) 97–110.
[4] F. Provost, T. Fawcett, Robust classification for imprecise environments, Mach. Learn. 42 (3) (2001) 203–231.
[5] D. Broomhead, D. Lowe, Multivariable functional interpolation and adaptive networks, Complex Syst. 2 (1988) 321–355.
[6] J. Jang, C. Sun, Functional equivalence between radial basis functions and fuzzy inference systems, IEEE Trans. Neural Netw. 4 (1993) 156–158.
[7] Y. Jin, B. Sendhoff, Extracting interpretable fuzzy rules from RBF networks, Neural Process. Lett. 17 (2) (2003) 149–164.
[8] J. Park, I. Sandberg, Universal approximation using radial-basis function networks, Neural Comput. 3 (1991) 246–257.
[9] O. Buchtala, M. Klimek, B. Sick, Evolutionary optimization of radial basis function classifiers for data mining applications, IEEE Trans. Syst. Man Cybern. B 35 (5) (2005) 928–947.
[10] M. Pérez-Godoy, A. Rivera, M. del Jesus, F. Berlanga, CO^{2}RBFN: an evolutionary cooperative–competitive RBFN design algorithm for classification problems, Soft Comput. 14 (9) (2010) 953–971.
[11] S. Qasem, S. Shamsuddin, Memetic elitist pareto differential evolution algorithm based radial basis function networks for classification problems, Appl. Soft Comput. 11 (8) (2011) 5565–5581.
[12] H. Chen, L. Kong, W. Leng, Numerical solution of PDEs via integrated radial basis function networks with adaptive training algorithm, Appl. Soft Comput. 11 (2011) 856–860.
[13] A. Alexandridis, E. Chondrodima, H. Sarimveis, Radial basis function network training using a nonsymmetric partition of the input space and particle swarm optimization, IEEE Trans. Neural Netw. Learn. Syst. 24 (2) (2013) 219–230.
[14] H. Du, N. Zhang, Time series prediction using evolving radial basis function networks with new enconding scheme, Neurocomputing 71 (2008) 1388–1400.
[15] H. Niu, J. Wang, Financial time series prediction by a random data-time effective RBF neural network, Soft Comput. 18 (3) (2014) 497–508.
[16] K. Aydin, O. Kisi, Damage detection in timoshenko beam structures by multilayer perceptron and radial basis function networks, Neural Comput. Appl. 24 (3–4) (2014) 583–597.
[17] C. Chen, A review of intelligent algorithm approaches and neural-fuzzy stability criteria for time-delay tension leg platform systems, J. Vib. Control 20 (4) (2014) 561–575.
[18] W. Yao, X. Chen, Y. Huang, M.V. Tooren, A surrogate-based optimization method with RBF neural network enhanced by linear interpolation and hybrid infill strategy, Optim. Methods Softw. 29 (2) (2014) 406–429.
[19] I. Maglogiannis, H. Sarimveis, C. Kiranoudis, A. Chatziioannou, N. Oikonomou, V. Aidinis, Radial basis function neural networks classification for the recognition of idiopathic pulmonary fibrosis in microscopic images, IEEE Trans. Inf. Technol. Biomed. 12 (1) (2008) 42–54.
[20] S. Jenifer, S. Parasuraman, A. Kadirvel, An efficient biomedical imaging technique for automatic detection of abnormalities in digital mammograms, J. Med. Imaging Health Inform. 4 (2) (2014) 291–296.
[21] J. Tian, M. Gao, Z. Zhang, Web text mining based on improved genetic algorithm and radial basis function neural network, J. Comput. Inform. Syst. 8 (3) (2012) 1195–1202.
[22] C. Dash, A. Behera, M. Pandia, S. Dehuri, Neural networks training based on differential evolution in radial basis function networks for classification of web logs, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), LNCS 7753 (2013) 183–194.
[23] M. Pérez-Godoy, A. Fernández, A. Rivera, M. del Jesus, Analysis of an evolutionary RBFN design algorithm CO^{2}RBFN for imbalanced data sets, Pattern Recogn. Lett. 31 (15) (2010) 2375–2388.
[24] B. Widrow, M. Lehr, 30 years of adaptive neural networks: perceptron madaline and backpropagation, Proc. IEEE 78 (9) (1990) 1415–1442.
[25] G. Golub, C. Van Loan, Matrix Computations, 3rd ed., Hopkins University Press, 1996.
[26] I. Rojas, O. Valenzuela, A. Prieto, Statistical analysis of the main parameters in the definition of radial basis function networks, LNCS 1240 (1997) 882–891.
[27] C. Lin, J. Wang, C. Chen, C. Chen, C. Yen, Improving the generalization performance of RBF neural networks using a linear regression technique, Exp. Syst. Appl. 36 (10) (2009) 12049–12053.
[28] H. Huan, D. Hien, H. Tue, Efficient algorithm for training interpolation RBF networks with equally spaced nodes, IEEE Trans. Neural Netw. 22 (6) (2011) 982–988.
[29] S. Su, C. Chuang, C. Tao, J. Jeng, C. Hsiao, Radial basis function networks with linear interval regression weights for symbolic interval data, IEEE Trans. Syst. Man Cybern. 42 (1) (2012) 69–80.
[30] Y. Yang, T. Sun, C. Huo, Y. Yu, C. Liu, C. Tsai, A novel self-constructing radial basis function neural-fuzzy system, Appl. Soft Comput. 13 (2013) 2390–2404.
[31] G. Kayhan, A. Ozdemir, I. Eminoglu, Review and designing pre-precessing units for RBF network: initial structure identification and coarse-tuning of free parameters, Neural Comput. Appl. 22 (2013) 1655–1666.
[32] T. Xie, H. Yu, J. Hewlett, P. Różycki, B. Wilamowski, Fast and efficient second-order method for training radial basis function networks, IEEE Trans. Neural Netw. Learn. Syst. 23 (4) (2012) 609–619.

[33] H. Han, L. Wang, J. Qiao, Hierarchical extreme learning machine for feedforward neural network, Neurocomputing 128 (0) (2014) 128–135.
[34] W. Pedrycz, Conditional fuzzy clustering in the design of radial basis function neural networks, IEEE Trans. Neural Netw. 9 (4) (1998) 601–612.
[35] R. Duda, P. Hart, Pattern Classification and Scene Analysis, Wiley, New York, 1973.
[36] J. Moody, C. Darken, Fast learning in networks of locally-tuned processing units, Neural Comput. 1 (1989) 281–294.
[37] M. Orr, Regularization on the selection of radial basis function centers, Neural Comput. 7 (1995) 606–623.
[38] S. Chen, C. Cowan, P. Grant, Orthogonal least squares learning algorithm for radial basis function networks, IEEE Trans. Neural Netw. 2 (1991) 302–309.
[39] R. Neruda, P. Kudová, Learning methods for radial basis function networks, Future Gener. Comput, Syst. 21 (7) (2005) 1131–1142.
[40] N. Ampazis, S. Perantonis, Two highly efficient second-order algorithms for training feedforwards networks, IEEE Trans. Neural Netw. 13 (3) (2002) 1064–1074.
[41] J. Peng, k. Li, D. Huang, A hybrid forward algorithm for RBF neural network construction, IEEE Trans. Neural Netw. 17 (6) (2006) 1439–1451.
[42] J. Plat, A resource allocating network for function interpolation, Neural Comput. 3 (2) (1991) 213–225.
[43] J. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, 1975.
[44] D. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, Reading, MA, 1989.
[45] T. Bäck, U. Hammel, H. Schwefel, Evolutionary computation: comments on the history and current state, IEEE Trans. Evol. Comput. 1 (1) (1997) 3–17.
[46] C. Harpham, C. Dawson, M. Brown, A review of genetic algorithms applied to training radial basis function networks, Neural Comput. Appl. 13 (2004) 193–201.
[47] E. Lacerda, A. Carvalho, A. Braga, T. Ludermir, Evolutionary radial functions for credit assessment, Appl. Intell. 22 (2005) 167–181.
[48] B. Whitehead, T. Choate, Cooperative–competitive genetic evolution of radial basis function centers and widths for time series prediction, IEEE Trans. Neural Netw. 7 (4) (1996) 869–880.
[49] M. Potter, K. De Jong, Cooperative coevolution: an architecture for evolving coadapted subcomponents, Evol. Comput. 8 (1) (2000) 1–29.
[50] C. Chen, P. Chen, Ga-based adaptive neural network controllers for nonlinear systems, Int. J. Innov. Comput. Inform. Control 6 (4) (2010) 1793–1803.
[51] C. Chen, P. Chen, W. Chiang, Stabilization of adaptive neural network controllers for nonlinear structural systems using a singular perturbation approach, J. Vib. Control 17 (8) (2011) 1241–1252.
[52] C. Chen, P. Chen, W. Chiang, Modified intelligent genetic algorithm-based adaptive neural network control for uncertain structural systems, J. Vib. Control 19 (9) (2013) 1333–1347.
[53] J. Hartigan, M. Wong, Algorithm AS 136: a *K*-means clustering algorithm, J. R. Stat. Soc. Ser. C (Appl. Stat.) 28 (1) (1979) 100–108.
[54] N. Sundararajan, P. Saratchandran, L. Yingwei, Radial basis function neural network with sequential learning: MRAN and its application, 1999.
[55] E. Mandani, S. Assilian, An experiment in linguistic synthesis with a fuzzy logic controller, Int. J. Man–Mach. Stud. 7 (1) (1975) 1–13.
[56] N. Chawla, K. Bowyer, L. Hall, W. Kegelmeyer, SMOTE: synthetic minority oversampling technique, J. Artif. Intell. Res. 16 (2002) 321–357.
[57] N. Chawla, N. Japkowicz, A. Kolcz, Special issue on learning from imbalanced data sets, SIGKDD Explor. Newslett. 6 (1) (2004) 1–6.
[58] A. Fernández, M. del Jesus, F. Herrera, Hierarchical fuzzy rule based classification system with genetic rule selection for imbalanced data-set, Int. J. Approx. Reason. 50 (2009) 561–577, http://dx.doi.org/10.1016/j.ijar.2008.11.004.
[59] R. Barandela, J. Sánchez, V. García, E. Rangel, Strategies for learning in class imbalance problems, Pattern Recogn. 36 (3) (2003) 849–851.
[60] G. Batista, R. Prati, M. Monard, A study of the behaviour of several methods for balancing machine learning training data, SIGKDD Explor. 6 (1) (2004) 20–29.
[61] A. Estabrooks, T. Jo, N. Japkowicz, A multiple resampling method for learning from imbalanced data-sets, Comput. Intell. 20 (1) (2004) 18–36.
[62] G. Wu, E. Chang, KBA: Kernel boundary alignment considering imbalanced data distribution, IEEE Trans. Knowl. Data Eng. 17 (6) (2005) 786–795.
[63] L. Xu, M. Chow, L. Taylor, Power distribution fault cause identification with imbalanced data using the data mining-based fuzzy classification e-algorithm, IEEE Trans, Power Syst. 22 (1) (2007) 164–171.
[64] P. Domingos, Metacost: a general method for making classifiers cost sensitive, in: Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining, 1999, pp. 155–164.
[65] Z. Zhou, X. Liu, Training cost-sensitive neural networks with methods addressing the class imbalance problem, IEEE Trans. Knowl. Data Eng. 18 (1) (2006) 63–77.
[66] Y. Sun, M.S. Kamel, A.K. Wong, Y. Wang, Cost-sensitive boosting for classification of imbalanced data, Pattern Recogn. 40 (2007) 3358–3378.
[67] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera, Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework, J. Mult.-Valued Logic Soft Comput. 17 (2011) 255–287.
[68] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
[69] S. García, F. Herrera, An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons, J. Mach. Learn. Res. 9 (2008) 2677–2694.
[70] S. García, A. Fernández, J. Luengo, F. Herrera, A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability, Soft Comput. 13 (10) (2009) 959–977.
[71] J. Zar, Biostatistical Analysis, Prencice Hall, Upper Saddle River, NJ, 1999.