Boosting for class-imbalanced datasets using genetically evolved supervised non-linear projections

Nicolás García-Pedrajas · César García-Osorio

Received: 17 September 2012 / Accepted: 25 September 2012 / Published online: 17 October 2012
© Springer-Verlag Berlin Heidelberg 2012

Abstract It has repeatedly been shown that most classification methods suffer from an imbalanced distribution of training instances among classes. Most learning algorithms expect an approximately even distribution of instances among the different classes and suffer, to different degrees, when that is not the case. Dealing with the class-imbalance problem is a difficult but relevant task, as many of the most interesting and challenging real-world problems have a very uneven class distribution. In this paper we present a new approach for dealing with class-imbalanced datasets based on a new boosting method for the construction of ensembles of classifiers. The approach is based on using the distribution of the weights given by a given boosting algorithm for obtaining a supervised projection. Then, the supervised projection is used to train the next classifier using a uniform distribution of the training instances. We tested our method using 35 class-imbalanced datasets and two different base classifiers: a decision tree and a support vector machine. The proposed methodology proved its usefulness achieving better accuracy than other methods both in terms of the geometric mean of specificity and sensibility and the area under the ROC curve.

Keywords Data mining · Class-imbalanced problems · Boosting · Real-coded genetic algorithms

1 Introduction

A classification problem of K classes and n training observations consists of a set of instances whose class membership is known. Let $S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$ be a set of n training samples where each instance $\mathbf{x}_i$ belongs to a domain X . Each label is an integer from the set $Y = \{1, \dots, K\}$. A multiclass classifier is a function $f : X \rightarrow Y$ that maps an instance $\mathbf{x} \in X \subset R^D$ into an element of Y .

The task is to find a definition for the unknown function, $f(\mathbf{x})$, given the set of training instances. In a classifier ensemble framework we have a set of classifiers $F = \{f_1, f_2, \dots, f_T\}$, each classifier performing a mapping of an instance vector $\mathbf{x} \in R^D$ into the set of labels $Y = \{1, \dots, K\}$. The design of classifier ensembles must face two main tasks: constructing the individuals classifiers, f_k , and developing a combination rule that finds a class label for $\mathbf{x}$ based on the outputs of the classifiers $\{f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_T(\mathbf{x})\}$.

One of the distinctive features of many common problems in current data-mining applications is the uneven distribution of the instances of the different classes. In extremely active research areas, such as artificial intelligence in medicine, bioinformatics or intrusion detection, two classes are usually involved: a class of interest, or positive class, and a negative class that is overrepresented in the datasets. This is usually referred to as the class-imbalance problem [28]. In highly imbalanced problems, the ratio between the positive and negative classes can be as high as 1:1,000 or 1:10,000. In recent years much attention has also been given to class-imbalanced multiclass problems [13].

It has repeatedly been shown that most classification methods suffer from an imbalanced distribution of training instances among classes [5]. Many algorithms and methods have been proposed to ameliorate the effect of class

N. García-Pedrajas (✉)
Computational Intelligence and Bioinformatics Research Group,
University of Córdoba, Córdoba, Spain
e-mail: npedrajas@uco.es

C. García-Osorio
Advanced Data Mining Research and Bioinformatics Learning
Research Group, University of Burgos, Burgos, Spain
e-mail: cgosorio@ubu.es

imbalance on the performance of learning algorithms. There are three main approaches to these methods:

- Internal approaches acting on the algorithm. These approaches modify the learning algorithm to deal with the imbalance problem. They can adapt the decision threshold to create a bias toward the minority class or introduce costs in the learning process to compensate the minority class.
- External approaches acting on the data. These algorithms act on the data instead of the learning method. They have the advantage of being independent from the classifier used. There are two basic approaches: oversampling the minority class and undersampling the majority class.
- Combined approaches that are based on ensembles of classifier, and most commonly *boosting*, accounting for the imbalance in the training set. These methods modify the basic boosting method to account for minority class underrepresentation in the dataset.

There are two principal advantages of choosing sampling over cost-sensitive methods. First, sampling is more general, as it does not depend on the possibility of adapting a certain algorithm to work with classification costs. Secondly, the learning algorithm is not modified, which can cause difficulties and add additional parameters to be tuned.

Data-driven algorithms can be broadly classified into two groups: those that undersample the majority class and those that oversample the minority class. There are also algorithms that combine both processes. Both undersampling and oversampling can be achieved randomly or through a more complicated process of searching for least or most useful instances. Previous works have shown that undersampling the majority class usually leads to better results than oversampling the minority class [5] when oversampling is performed using sampling with replacement from the minority class. Furthermore, combining undersampling of the majority class with oversampling of the minority class has not yielded better results than undersampling of the majority class alone [41]. One of the possible sources of the problematic performance of oversampling is the fact that no new information is introduced in the training set, as oversampling must rely on adding new copies of minority class instances already in the dataset. Sampling has proven a very efficient method of dealing with class-imbalanced datasets [11,56].

Removing instances only from the majority class, usually referred to as one-sided selection [34], has two major problems. Firstly, reduction is limited by the number of instances of the minority class. Secondly, instances from the minority class are never removed, even when their contribution to the models performance is harmful.

The third approach has received less attention. However, we believe that the approaches based on boosting are more promising because these methods have proven their ability to improve the results of many base learners in balanced datasets. Furthermore, when the imbalance ratio is high it is likely that the only way of obtaining good performance is by means of combining many classifiers.

An ensemble of classifiers consists of a combination of different classifiers, homogeneous or heterogeneous, to perform a classification task jointly. Ensemble construction is one of the fields of machine learning that is receiving more research attention, mainly due to the significant performance improvements over single classifiers that have been reported with ensemble methods [1,3,25,32,55].

Techniques using multiple models usually consist of two independent phases: model generation and model combination [44]. Most techniques are focused on obtaining a group of classifiers which are as accurate as possible but which disagree as much as possible. These two objectives are somewhat conflicting, since if the classifiers are more accurate, it is obvious that they must agree more frequently. Many methods have been developed to enforce diversity on the classifiers that form the ensemble [8].

In this paper we propose a new ensemble for class-imbalanced datasets based on boosting. Boosting trains a classifier on a biased distribution of the training set to optimize weighted training error. However, for some problems, optimizing this weighted error may harm the overall performance of the ensemble, because too much relevance is given to incorrectly labelled instances and outliers. Our approach is based on optimizing the weighted error in a less aggressive way. We use the biased distribution of instances given by boosting algorithm to obtain a supervised projection of the original data into a new space where the weighted error is improved. Then, the classifier is trained using this projection and with an uniform distribution of the training instances. In this way, the view of the data the classifier sees is biased towards difficult instances, as the supervised projection is obtained using the biased distribution of instances given by the boosting algorithm, but without putting too much pressure on correctly classifying misclassified instances. Informally, a supervised projection is a projection constructed using both the inputs and the label of the patterns and with the aim of improving the classification accuracy of any given learner.

The projections are constructed using a real-coded genetic algorithm (RCGA). The use of such a method allows a more flexible approach than other boosting methods. Standard boosting methods focus on optimizing weighted accuracy. However, as it is well known, accuracy is not an appropriate method for class-imbalanced datasets. In our approach, the RCGA evolves using as fitness value a measure that is specific of class-imbalanced problems, the geometric mean

of specificity and sensitivity (the so-called *G*-mean). The presented method is named Genetically Evolved Supervised Projection Boosting, GESUPERPBOOST, algorithm.

A final feature is introduced to account for the class-imbalanced nature of the datasets. Before obtaining the supervised projection by means of the RCGA, we randomly undersample the majority class. With this process we obtain three relevant benefits. Firstly, we make the algorithm faster; secondly, we obtain better performance due to the balanced datasets used for the RCGA; thirdly, we introduce diversity in the ensemble from the use of the random undersampling.

Unsupervised [49] and supervised projections [22, 23] has been used before for constructing ensembles of classifiers. However, the use of genetically evolved supervised projections is new, and most of the previous approaches were devoted to indirect methods to obtain what we could call *semi-supervised* projections.

This paper is organized as follows: Sect. 2 describes in detail the proposed methodology and its rationale; Sect. 3 explains the experimental setup; Sect. 4 shows the results of the experiments performed; and finally Sect. 5 states the conclusions of our work.

2 Supervised projection approach for boosting classifiers

In order to avoid the stated harmful effect of maximizing the margin of noisy instances, we do not use the adaptive weighting scheme of boosting methods to train the classifiers, but to obtain a supervised projection. The supervised projection is aimed at optimizing the weighted error given by the boosting algorithm. However, instead of optimizing accuracy, we use the *G*-mean measure common in class-imbalanced datasets. Other mechanisms were used to account for the class-imbalance nature of the problems that are described below.

The classifier is trained using the supervised projection obtained with the RCGA and with an uniform distribution of the instances. Thus, we obtain a method that benefits from the adaptive instance weighting of boosting but that is also able to improve its drawbacks. We are extending the philosophy of a previous paper [23] where we showed that the performance of ensembles of classifiers can be improved using non-linear projections constructed with only misclassified instances.

The most widely used boosting method is ADABOOST [15] and its numerous variants. It is based on adaptively increasing the probability of sampling the instances that are not classified correctly by the previous classifiers. For more detailed descriptions of ensembles the reader is referred to [8, 10, 44, 55], or [12]. Several empirical studies have shown that ADABOOST is able to reduce both bias and variance components of the error [1, 2, 50]. Boosting methods

“boost” the accuracy of a *weak classifier* by repeatedly resampling the most difficult instances. Boosting methods construct an additive model. In this way, the classifier ensemble $F(\mathbf{x})$ is constructed using T individual classifiers, $f_k(\mathbf{x})$:

$$F(\mathbf{x}) = \arg \max_{y \in Y} \sum_{k=1}^T \alpha_k \mathbb{I}[f_k(\mathbf{x}) = y], \quad (1)$$

where the α_k are appropriately defined, and $\mathbb{I}[\pi]$ is 1 if π is true and 0 otherwise. The basis of boosting is assigning a different weight to each training instance depending on how difficult it has been for the previous classifiers to classify it. Thus, for ADABOOST, each instance $\mathbf{x}_i$ receives a weight w_i^k for training the k th classifier.¹ Initially, all the instances are weighted equally $w_i^1 = 1/n, \forall i$. Then, for classifier $(k+1)$ th the instance is weighted following:

$$w_i^{k+1} = w_i^k \beta_k^{(1 - \mathbb{I}[f_k(\mathbf{x}_i) = y_i])}, \quad (2)$$

where

$$\beta_k = \frac{\epsilon_k}{(1 - \epsilon_k)}, \quad (3)$$

ϵ_k being the weighted error of classifier k when the weight vector is normalized $\sum_{i=1}^n w_i^k = 1$. Once the classifiers are trained, the function $F(\mathbf{x})$ is given by (1) with the weight of each classifier given by $\alpha_k = \ln \left(\frac{1}{\beta_k} \right)$.

The popularity of boosting methods is mainly due to the success of ADABOOST. However, ADABOOST tends to perform very well for some problems but can also perform very poorly for other problems. One of the sources of the bad behavior of ADABOOST is that although it is always able to construct diverse ensembles, in some problems the individual classifiers tend to have large training errors [9]. Moreover, ADABOOST usually performs poorly on noisy problems [1, 9]. Schapire and Singer [51] identified two scenarios where ADABOOST is likely to fail: (i) When there is insufficient training data relative to the “complexity” of the base classifiers; and (ii) when the training errors of the base classifiers become too large too quickly. Unfortunately, these two scenarios are likely to occur in real-world problems. Several papers have attributed the failure of boosting methods, especially in the presence of noise, to the fact that the skewed data distribution produced by the algorithm tends to assign too much weight to hard instances [9]. In class-imbalanced datasets this problem may be even harder because the distribution of the patterns in the classes is uneven.

As we have stated, in a general classification problem, we have a training set $S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$ of n training samples where each instance $\mathbf{x}_i$ belongs to a

¹ We are using the multiclass variant of Freund and Schapire [16] called ADABOOST.M1 which essentially is a multiclass version of Discrete ADABOOST [17]. However, as our methodology works by projecting the inputs, it can be used with any boosting algorithm.

domain X . Let us assume that we also have a vector $\mathbf{w}$ that assigns a weight w_i to each instance $\mathbf{x}_i$. In a general sense, a supervised projection is a projection constructed using both the inputs and the label of the patterns. More specifically, in our framework, a supervised projection, Φ , is a projection into a new space where the weighted error $\epsilon = \sum_{i=1}^n w_i \llbracket f(\Phi(\mathbf{x}_i)) = y_i \rrbracket$ for a certain classifier $f(\mathbf{x})$ trained on the projected space is improved with respect to the error using the original variables. Thus, the idea is searching for a projection of the original variables that is able to improve the weighting error even when the learner is trained using a uniform distribution of the patterns. This supervised projection, which is calculated using inputs and pattern labels, leads to an *informed* or *biased* feature space, which will be more relevant to the particular supervised learning problem [58].

As it is the case for ADABOOST, the constructed model using this supervised projection is also additive:

$$F(\mathbf{x}) = \sum_{k=1}^T \alpha_k f_k(\mathbf{z}_k), \quad (4)$$

where $\mathbf{z}_k = \mathbf{P}_k(\mathbf{x})$ and $\mathbf{P}_k$ is a non-linear projection constructed using the weights of the instances given by the boosting algorithm. In this way, classifier k is constructed using the instances projected using $\mathbf{P}_k$ and all of them equally weighted.

To illustrate the method, let us explain the differences with a boosting algorithm for step k . In a boosting algorithm after adding $k - 1$ classifiers, we obtain the instance weight vector for training k th classifier, $\mathbf{w}^k$. Then, k th classifier is trained associating to each instance i a weight w_i^k , which is used directly by the classifier, if it admits instance weights, or used to obtain a biased sample from the training set if not. The aim is obtaining a classifier that optimizes the weighted error $\epsilon_k = \sum_{i=1}^n w_i^k \llbracket f_k(\mathbf{x}_i) = y_i \rrbracket$. Once trained, classifier k th is added to the ensemble, and the process is repeated for adding classifier $(k + 1)$ th. In our method, after adding $k - 1$ classifiers, we obtain the instance weight vector for training k th classifier, $\mathbf{w}^k$, using the weighting scheme of a certain boosting algorithm of our choice. However, this weight vector is not used to train the new classifier. Instead, a supervised projection, P_k , of the inputs is constructed with the objective of minimizing a certain accuracy criterion that considers that the problems is class-imbalanced. Then, the original instances $\mathbf{x}_i$ are projected using this supervised projection to obtain a new training set $Z_k = \{\mathbf{z}_i : \mathbf{z}_i = P_k(\mathbf{x}_i)\}$. Classifier k is trained on Z_k using a uniform distribution. Once trained, it is added to the ensemble and the process is repeated.

The proposed method shares the philosophy of our previous method based on non-linear projections [23]. In that method new classifiers were added to the ensemble considering only the instances misclassified by the previous classifier.

To avoid the large bias of using only those instances to learn the classifier, they were not used for training the classifier but for constructing a projection where their separation by the classifier was easier. In the present method, we do not use misclassified instances, but the distribution of instances given by a boosting method. This method has several advantages over the previous one. We can use the theoretical result about convergence of training error of boosting to assure the convergence to perfect classification with similar conditions as for ADABOOST. The weights assigned by boosting and used for constructing the non-linear projection summarizes the difficulty in classifying the instance as the ensemble grows, instead of using just the last classifier. The method in [23] does not use instance weights, and it is inspired more in random subspace method (RSM) [30], using non-linear projections instead of subspace projection. The difference in the philosophy is that, while RSM uses random projections, the method in [23] uses non-linear projections using only misclassified instances. Furthermore, part of the theory developed for boosting is applicable to the new approach but not for the previous one. In addition, the proposed model constructs an additive model as boosting, and the previous one used a simple voting scheme to obtain the final output of the ensemble.

2.1 Constructing supervised projections using a RCGA

We have stated that our method is based on using a supervised projection to train the classifier at round k . But, what exactly do we understand by a *supervised projection*? The intuitive meaning of a supervised projection using a weight vector $\mathbf{w}_k$, is a projection into a space where the weighted error achieved by any classifier trained using the projection is minimized. In a previous work [21] we defined the concept of supervised projection as follows:²

Definition 1 *Supervised projection.* Let $S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$ be a set of n training samples where each instance $\mathbf{x}_i$ belongs to a domain X , and $\mathbf{w}$ a vector that assigns a weight w_i to each instance $\mathbf{x}_i$. A supervised projection Φ is a projection into a new space where the weighted error $\epsilon = \sum_{i=1}^n w_i \llbracket f(\Phi(\mathbf{x}_i)) = y_i \rrbracket$ for a certain classifier $f(\mathbf{x})$ trained on the projected space is improved with respect to the error using the original variables $\epsilon_o = \sum_{i=1}^n w_i \llbracket f_o(\mathbf{x}_i) = y_i \rrbracket$ for training a classifier f_o .

The definition is general, as we do not restrict ourselves to any particular type of classifier. The intuitive idea is to find a projection that improves the weighted error of the classifier.

² The given definition is an description of what we understand by “supervised projection”, but it is not intended as an strict mathematical definition. More formal definitions of this concept can be found in papers devoted to this topic [27,37,39,58,59].

We can consider this problem similar to feature extraction. In feature extraction a mapping g transform the original variables $x_1, \dots, x_d$ of a d -dimensional input space into a new set of variables $z_1, \dots, z_m$ of a m -dimensional projected space [38], in such a way that a certain criterion J is optimized. The mapping g is chosen among all the available transformations G as the one that optimizes J . Considering a projection that depends on a vector of parameters Θ the problem can be stated as minimizing the following cost function:

$$J(\Theta) = \sum_{i=1}^N w_i \mathbb{I}(f(\Phi_{\Theta}(\mathbf{x}_i)) \neq y_i) \quad (5)$$

where classifier f is trained on the projection $\Phi_{\Theta}(\mathbf{x})$, and Θ is the vector of values to optimize.

Although most methods for projecting data are focused on the features of the input space, and do not take into account the labels of the instances, as most of them are specifically useful for non-labelled data and aimed at data analysis, methods for supervised projection, in the sense that class labels are taken into account, do exist in the literature. For instance, S-Isomap [27] is a supervised version of Isomap [53] projection algorithm which uses class labels to guide the manifold learning. Projection pursuit, a technique initially developed for exploratory data analysis [33], has been also applied to supervised projections [46] where projections are chosen to minimize estimates of the expected overall loss in each projection pursuit stage. Lee et al. [36] introduced new indexes derived from linear discriminant analysis that can be used for exploratory supervised classification. However, all of these methods do not contemplate the possibility of assigning a weight to each instance measuring its relevance.

As we have said, developing a method for obtaining a supervised projection, we face a similar problem as feature selection/extraction. There are two basic approaches for feature selection/extraction: The *filter approach* separates feature selection/extraction and the learning of the classifier. Measures that do not depend on the learning method are used. The *wrapper approach* considers that the selection of inputs (or extraction of features) and the learning algorithm cannot be separated. Both approaches are usually developed as an optimization process where certain objective function must be optimized.

Filter methods are based on performance evaluation metrics calculated directly from the data, without direct reference to the results of any induction algorithm. Such algorithms are usually computationally cheaper than those that use a wrapper approach. However, one of the disadvantages of filter approaches is that they may not be appropriate for a given problem. In previous works we have used indirect methods [23] or supervised linear projections [22]. However, none of these methods is able to obtain a supervised projec-

tion aimed at optimizing the weighted error given by the boosting resampling scheme.

Thus, in this work we propose the use of a real-coded genetic algorithm to directly optimize the weights of the non-linear projection. The process of obtaining the supervised projection is shown in Algorithm 1. Thus, our solution is based on a wrapper approach by means of an evolutionary computation method.

The input vector $\mathbf{x}$ is projected into a vector $\mathbf{z}$ using a non-linear projection of the form:

$$z_i = \tanh \left(\sum c_{ij} x_j + b \right). \quad (6)$$

The whole dataset is then transformed according to:

$$\mathbf{Z} = \tanh(\mathbf{C}\mathbf{X} + \mathbf{B}). \quad (7)$$

The RCGA must evolve the coefficient matrix, $\mathbf{C}$, and constants $\mathbf{B}$. Our aim is obtaining the coefficients that produce the optimum accuracy when used to train a classifier using an uniform distribution. For the initialization of individuals the values of the coefficients are randomly distributed in $[-1, 1]$.

Algorithm 1: RCGA for obtaining the supervised projection.

Data : A training set $S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$, a base learning algorithm, L , a vector of pattern weights $\mathbf{w}$, and the number of generations G .

Result : The final non-linear projection coefficient matrix, $\mathbf{C}$, and coefficients $\mathbf{B}$ (see eq: 6).

```

1 Initialize population
  for  $k = 1$  to  $G$  do
2   Evaluate individuals using weighted G-mean
3   Apply BLX- $\alpha$  crossover
4   Apply non-uniform mutation
5   Select best individuals for next generation
  end
  Return best individual

```

To avoid over-fitting we introduce additional diversity into the ensemble using random subspaces. Before obtaining the supervised projection using the RCGA the input space is divided into subspaces of the same size and each subspace is projected separately. In our experiments a subspace size of five features is used. We evolve the whole projection with all the subspaces using the same RCGA. From the practical point of view the only difference when using subspaces is that many of the elements of matrix $\mathbf{C}$ are fixed to zero during the evolution.

The main problem of the method described above is the scalability [24]. When we deal with a large dataset, the cost of the RCGA is high. To partially avoid this problem we combine our method with undersampling. Thus, before the construction of each supervised projection is started we undersample the majority class to obtain a balanced distribution of both classes. Then we construct the ensemble as it has been described above. The final version of the method is shown in Algorithm 2.

Algorithm 2: Genetically Evolved Supervised Projection Boosting (GESUPERPBOOST) algorithm.

Data : A training set $S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$, a base learning algorithm, L , and the number of iterations T .

Result : The final classifier: $F^*(\mathbf{x}) = \arg \max_{y \in Y} \sum_{k: f_k(\mathbf{x})=y} \alpha_k$.

```

1  $C_0 = L(S)$ 
  for  $k = 1$  to  $T$  do
2   Obtain vector  $\mathbf{w}_k$  (†) of weights for each instance using a boosting method
3   Randomly undersample majority class
4   Obtain supervised projection  $P_k(\mathbf{x})$  using weight vector  $\mathbf{w}_k$  and a RCGA
5    $f_k = L(P_k(S))$ 
  end

```

†The exact way of obtaining $\mathbf{w}_k$ depends on the boosting algorithm to which our approach is applied. In the reported experiments ADABOOST scheme is used.

The RCGA genetic algorithm used is an standard generational genetic algorithm. Initially each individual is randomly obtained. Each individual represents the coefficient matrix, $\mathbf{C}$, of the non-linear projection plus the constants, $\mathbf{B}$. Then the evolution is carried out for a number of generations where new individuals are obtained by non-uniform mutation [45] and standard BLX- α [29] crossover with $\alpha = 0.5$.

Non-uniform mutation mutates and individual $\mathbf{X}^t$ in generation t to produce another individual $\mathbf{X}^{t+1}$. Each element of the individual, x_k^{t+1} is obtained from x_k^t using:

$$x_k^{t+1} = \begin{cases} x_k^t + \Delta(t, \text{UB} - x_k^t), & \text{if } \rho < 0.5 \\ x_k^t - \Delta(t, x_k^t - \text{LB}), & \text{if } \rho \geq 0.5 \end{cases} \quad (8)$$

where ρ is a uniform random value in $[0, 1]$, LB and UB are the lower and upper bounds of the search space, respectively, and $\Delta(t, y) = y \cdot (1 - r^{(1-t/T)^b})$, with r a uniform random number in $[0, 1]$, and T the maximum number of generations.

The fitness function should measure the performance of each projection. To obtain this fitness value we train a classifier with the projection that the individual represents and evaluate the classifier. However, accuracy is not a useful measure for imbalanced data, especially when the number of instances of the minority class is very small compared with the majority class. If we have a ratio of 1:100, a classifier that assigns all instances to the majority class will have 99 % accuracy. Several measures have been developed to take the imbalanced nature of the problems into account. Given the number of true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN) we can define several measures. Perhaps the most common are the true positive rate (TP_{rate}), recall (R) or sensitivity (Sn):

$$\text{TP}_{\text{rate}} = R = \text{Sn} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \quad (9)$$

which is relevant if we are only interested in the performance on the positive class and the true negative rate (TN_{rate}) or specificity (Sp), as:

$$\text{TN}_{\text{rate}} = \text{Sp} = \frac{\text{TN}}{\text{TN} + \text{FP}}. \quad (10)$$

From these basic measures, others have been proposed, such as the F -measure or, if we are concerned about the performance of both negative and positive classes, the G -mean measure: $G - \text{mean} = \sqrt{\text{Sp} \cdot \text{Sn}}$. However, this measure does not take into account patterns weights. Thus we modified the standard specificity and sensibility measures using the patterns weights instead of just counting the numbers of true positive, true negatives, false positives and false negatives. This *weighted G-mean* is used as the fitness function of the individuals.

The source code used for all methods is in C and is licensed under the GNU General Public License. The code, the partitions of the datasets and the detailed numerical results of all the experiments are available from the authors upon request and from <http://cib.uco.es>.

3 Experimental setup

We used a set of 35 problems to test the performance of the proposed method, which is shown in Table 1. The datasets breast-cancer, cancer, euthyroid, german, haberman, hepatitis, ionosphere, ozone1hr, ozone8hr, pima, sick and tic-tac-toe are from the UCI Machine Learning Repository [14]. The remaining datasets were created following García et al. [20] from UCI datasets. To estimate the accuracy we used tenfold cross-validation.

As base learners for the ensembles we used two classifiers: a decision tree using the C4.5 learning algorithm [47] and a support vector machine (SVM) [6] using a Gaussian kernel. The SVM learning algorithm was programmed using functions from the libsvm library [4]. We used these two methods because they are arguably the two most popular classifiers in the literature.

For the RCGA we used a population of 100 individuals evolved during 100 generations. To avoid extremely long running times, an arbitrary maximum running time of 100 seconds was imposed for the RCGA. Once this time limit was reached the best individual so far was used as the non-linear projection.

3.1 Statistical tests

We used the Wilcoxon test as the main statistical test for comparing pairs of algorithms. This test was chosen because it assumes limited commensurability and is safer than parametric tests because it does not assume normal distributions or homogeneity of variance. Furthermore, empirical results [7] show that it is also stronger than other tests. The formulation of the test [57] is the following: Let d_i be the difference between the error values of the methods in i th dataset. These differences are ranked according to their absolute values; in

Table 1 Summary of datasets

	Dataset	Cases	Attributes			Inputs	IR
			C	B	N		
1	abalone19	4,177	7	–	1	10	1:130
2	breast-cancer	286	–	3	6	15	1:3
3	cancer	699	9	–	–	9	1:2
4	carG	1,728	–	–	6	16	1:25
5	ecoliCP-IM	220	7	–	–	7	1:2
6	ecoliM	336	7	–	–	7	1:4
7	ecoliMU	336	7	–	–	7	1:9
8	ecoliOM	336	7	–	–	7	1:16
9	euthyroid	3,163	7	18	–	44	1:10
10	german	1,000	6	3	11	61	1:3
11	glassBWFP	214	9	–	–	9	1:3
12	glassBWNFP	214	9	–	–	9	1:3
13	glassContainer	214	9	–	–	9	1:16
14	glassNW	214	9	–	–	9	1:4
16	glassTableware	214	9	–	–	9	1:23
17	haberman	306	3	–	–	3	1:3
18	hepatitis	155	6	13	–	19	1:4
19	hypothyroidT	3,772	7	20	2	29	1:12
20	ionosphere	351	33	1	–	34	1:2
21	new-thyroidT	215	5	–	–	5	1:6
22	ozone1hr	2,536	72	–	–	72	1:34
23	ozone8hr	2,534	72	–	–	72	1:15
24	pima	768	8	–	–	8	1:2
25	segmentO	2,310	19	–	–	19	1:7
26	sick	3,772	7	20	2	33	1:16
27	splice-EI	3,175	–	–	60	120	1:4
28	splice-IE	3,175	–	–	60	120	1:4
29	tic-tac-toe	958	–	–	9	9	1:2
30	vehicleVAN	846	18	–	–	18	1:3
31	vowelZ	990	10	–	–	10	1:11
32	yeastCYT-POX	483	8	–	–	8	1:24
33	yeastEXC	1,484	8	–	–	8	1:42
34	yeastME1	1,484	8	–	–	8	1:33
35	yeastME2	1,484	8	–	–	8	1:29

The attributes of each dataset can be *C* (continuous), *B* (binary) or *N* (nominal) . Imbalance ratio (*IR*) is also shown

case of ties an average rank is assigned. Let R^+ be the sum of ranks for the datasets on which the second algorithm outperformed the first, and R^- the sum of ranks where the first algorithm outperformed the second. Ranks of $d_i = 0$ are split evenly among the sums:

$$R^+ = \sum_{d_i > 0} \text{rank}(d_i) + \frac{1}{2} \sum_{d_i = 0} \text{rank}(d_i), \quad (11)$$

and

$$R^- = \sum_{d_i < 0} \text{rank}(d_i) + \frac{1}{2} \sum_{d_i = 0} \text{rank}(d_i). \quad (12)$$

Let T be the smaller of the two sums and N be the number of datasets. For a small N , there are tables with the exact critical values for T . For a larger N , the statistics

$$z = \frac{T - \frac{1}{4}N(N+1)}{\sqrt{\frac{1}{24}N(N+1)(2N+1)}} \quad (13)$$

is distributed approximately according to $N(0, 1)$.

In our experiments, we will also compare groups of methods. In such a case it is not advisable to use pairwise statistical tests such as Wilcoxon test. Instead, we first carry out an Iman–Davenport test to ascertain whether there are significant differences among methods. The Iman–Davenport test is based on the χ_F^2 Friedman test, which compares the average ranks of k algorithms, but is more powerful than the Friedman test. Let r_i^j be the rank of j th algorithm on i th dataset, where in case of ties, average ranks are assigned, and let $R_j = \frac{1}{N} \sum_i r_i^j$ be the average rank for N datasets. Under the null hypothesis, all algorithms are equivalent, the statistic:

$$\chi_F^2 = \frac{12N}{k(k+1)} \left[\sum_j R_j^2 - \frac{k(k+1)^2}{4} \right] \quad (14)$$

is distributed following a χ_F^2 with $k-1$ degrees of freedom for k and N sufficiently large. In general, $N > 10$ and $k > 5$ is enough. Iman and Davenport found this statistic to be too conservative and developed a better one:

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1) - \chi_F^2} \quad (15)$$

which is distributed following a F distribution with $k-1$ and $(k-1)(N-1)$ degrees of freedom. After carrying out the Iman–Davenport test, we should not perform pairwise comparisons with the Wilcoxon test, because it is not advisable to perform many Wilcoxon tests against the control method. We can instead use one of the general procedures for controlling the family-wise error in multiple hypotheses testing. The test statistic for comparing the i th and j th classifier using these methods is as follows:

$$z = \frac{(R_i - R_j)}{\sqrt{k(k+1)/6N}}. \quad (16)$$

The z value is used to find the corresponding probability from the table of normal distribution, which is then compared with an appropriate α . Step-up and step-down procedures sequentially test the hypotheses ordered by their significance. We will denote the ordered p values by $p_1, p_2, \dots$, so that $p_1 \leq p_2 \leq \dots \leq p_{k-1}$. One of the simplest such methods was developed by Holm. It compares each p_i with $\alpha/(k-i)$. Holms step-down procedure starts with the most significant p value. If p_1 is below $\alpha/(k-1)$, the corresponding hypothesis is rejected, and we are allowed to compare p_2 with $\alpha/(k-2)$. If the second hypothesis is rejected, the test proceeds with the third, and so on. As soon as a certain null hypothesis cannot be rejected, all remaining hypotheses are retained as well. We will use for all statistical tests a significance level of 0.05.

3.2 Evaluation measures

As we have stated, accuracy is not a useful measure for imbalanced data. Thus, as a general accuracy measure we will use the G -mean defined above.

However, many classifiers are subject to some kind of threshold that can be varied to achieve different values of the above measures. For that kind of classifiers receiver operating characteristic (ROC) curves can be constructed. A ROC curve, is a graphical plot of the TP_{rate} (sensitivity) against the FP_{rate} ($1 - \text{specificity}$ or $FP_{rate} = \frac{FP}{TN+FP}$) for a binary classifier system as its discrimination threshold is varied. The perfect model would achieve a true positive rate of 1 and a false positive rate of 0. A random guess will be represented by a line connecting the points (0, 0) and (1, 1). ROC curves are a good measure of the performance of the classifiers. Furthermore, from this curve a new measure, area under the curve (AUC), can be obtained, which is a very good overall measure for comparing algorithms. AUC is a useful metric for classifier performance as it is independent of the decision criterion selected and prior probabilities.

In our experiments we report both G -mean, which gives a snapshot measure of the performance of each method, and AUC which gives a more general vision of its behavior.

3.3 Methods for the comparison

The first method used for comparison is undersampling the majority class until both classes have the same number of instances. We have not used oversampling methods because most previous works agree that as a common rule undersampling performs better than oversampling [40]. However, a few works have found the opposite [11]. Furthermore, methods that add new synthetic patterns, such as SMOTE [5], have also shown a good behavior. Undersampling method was used because it is very simple and offers very good performance. Thus, it is the baseline method for any comparison. Any algorithm not improving over undersampling is of very limited interest.

Random undersampling consists of randomly removing instances from the majority class until a certain criterion is reached. In most works, instances are removed until both classes have the same number of instances. Several studies comparing sophisticated undersampling methods with random undersampling [31] have failed to establish a clear advantage of the formers. Thus, in this work we consider first random undersampling. However, the problem with random undersampling is that many, potentially useful, samples from the majority class are ignored. In this way, when the majority/minority class ratio is large the performance of random undersampling degrades. Furthermore, when the

number of minority class samples is very small, we will also have problems due to small training sets.

To avoid these problems, several ensemble methods have been proposed [18]. Rodríguez et al. [48] proposed the use of ensembles of decision trees. ADABOOST [52] has been proposed as an alternative to ADABOOST for imbalanced datasets.

Liu et al. [42] proposed two ensemble methods combining undersampling and boosting to avoid that problem. This methodology is called exploratory undersampling. The two proposed methods are called EasyEnsemble and BalanceCascade. EasyEnsemble consists of applying repeatedly the standard ensemble method ADABOOST [1] to different samples of the majority class. Algorithm 3 shows EasyEnsemble method. The idea behind EasyEnsemble is generating T balanced subproblems sampling from the majority class.

Algorithm 3: EasyEnsemble method.

Data : A minority training set $\mathcal{P}$ and a majority training set $\mathcal{N}$, $|\mathcal{P}| \ll |\mathcal{N}|$, a number of subsets T to sample from $\mathcal{N}$ and s_i the number of iterations to train the ADABOOST ensemble H_i .

Result : The final ensemble:

$$H(\mathbf{x}) = \text{sgn} \left(\sum_{i=1}^T \sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(\mathbf{x}) - \sum_{i=1}^T \theta_i \right).$$

for $i = 1$ **to** T **do**

- 1: Randomly sample a subset $\mathcal{N}_i$ from $\mathcal{N}$, $|\mathcal{N}_i| = |\mathcal{P}|$
- 2: Learn H_i using $\mathcal{P}$ and $\mathcal{N}_i$. H_i is an ADABOOST ensemble with s_i weak classifiers $h_{i,j}$ and corresponding weights $\alpha_{i,j}$. The ensemble threshold is θ_i . H_i is given by:

$$H_i(\mathbf{x}) = \text{sgn} \left(\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(\mathbf{x}) - \theta_i \right).$$

end

EasyEnsemble is an unsupervised strategy to explore the set of negative instances, $\mathcal{N}$, as the sampling is made without using information from the classification performed by the previous members of the ensemble. On the other hand, BalanceCascade method explores $\mathcal{N}$ in a supervised manner, removing from the majority class those instances that have been correctly classified by the previous classifiers added to the ensemble. BalanceCascade is shown in Algorithm 4.

Algorithm 4: BalanceCascade method.

Data : A minority training set $\mathcal{P}$ and a majority training set $\mathcal{N}$, $|\mathcal{P}| \ll |\mathcal{N}|$, a number of subsets T to sample from $\mathcal{N}$ and s_i the number of iterations to train the ADABOOST ensemble H_i .

Result : The final ensemble:

$$H(\mathbf{x}) = \text{sgn} \left(\sum_{i=1}^T \sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(\mathbf{x}) - \sum_{i=1}^T \theta_i \right).$$

1: $f = T^{-1} \sqrt{|\mathcal{P}|/|\mathcal{N}|}$. f is the false positive rate that H_i should achieve.

for $i = 1$ **to** T **do**

- 2: Randomly sample a subset $\mathcal{N}_i$ from $\mathcal{N}$, $|\mathcal{N}_i| = |\mathcal{P}|$
- 3: Learn H_i using $\mathcal{P}$ and $\mathcal{N}_i$. H_i is an ADABOOST ensemble with s_i weak classifiers $h_{i,j}$ and corresponding weights $\alpha_{i,j}$. The ensemble threshold is θ_i . H_i is given by:

$$H_i(\mathbf{x}) = \text{sgn} \left(\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(\mathbf{x}) - \theta_i \right).$$

- 4: Adjust θ_i such that H_i 's false positive rate is f .
- 5: Remove from $\mathcal{N}$ all examples that are correctly classified by H_i .

end

As an additional method we also used in our experiments standard ADABOOST. Thus, our algorithm was compared against four state-of-the-art class-imbalanced methods: undersampling, EasyEnsemble, BalanceCascade and ADABOOST.

4 Experimental results

In this section we present the comparison of GESUPERPBOOST with the standard methods described above. We also present a control experiment to rule out the possibility that the good performance of GESUPERPBOOST is due to the introduction of a non-linear projection regardless how the projection is obtained.

4.1 Comparison with standard methods

The aim of these experiments was testing whether our approach based on non-linear projections was competitive when compared with the standard methods. Thus, we carried out experiments using the two base learners and the methods described in Sect. 3.3. The results, in terms of G -mean and AUC, for C4.5 are shown in Table 2. The results for SVM are shown in Table 3. The Iman–Davenport test obtained a p value of 0.0000 for the four comparisons, meaning that there were significant differences among the methods.

Figures 1 and 2 show average Friedman ranks for C4.5 and SVM respectively. These ranks are, by themselves, a good measure of the relative performance of a group of methods. In terms of rankings, using a decision tree, GESUPERPBOOST was the best method both for AUC and G -mean. Undersampling achieved good results for G -mean but its performance in terms of AUC was poor. From the standard methods, BalanceCascade achieved the most balanced behavior, with good results for both measures.

The results using a SVM as base learner had some sensible differences. In terms of AUC, GESUPERPBOOST was better than the remaining methods. However, in terms of G -mean, all methods, with the exception of ADABOOST, showed a similar behavior. ADABOOST had the worse combined behavior of the five methods. This is not an unexpected result as ADABOOST is an stable method with respect to sampling. This feature makes ADABOOST less efficient when using a SVM as base learner.

The results of the four standard methods against GESUPERPBOOST are illustrated in Figs. 3 and 4, for C4.5 and SVM, respectively. The figures show results for accuracy using both AUC and G -mean. This graphic representation is based on the κ -error relative movement diagrams [43]. However, here we use the AUC and G -mean differences instead of the κ difference value and the accuracy. These diagrams use an arrow to represent the results of two methods applied to the same dataset. The arrow starts at the coordinate origin, and the coordinates of the tip of the arrow represent the difference between the AUC and G -mean of our method and those of the standard methods. These graphs are a convenient way to summarize the results. A positive value in either AUC and G -mean means that our method performed better. Thus, arrows pointing up-right represent datasets for which our method

Table 2 Accuracy results, measured using the *G*-mean and AUC, for all the methods and C4.5 as base learner

Dataset	Undersampling		AdaBoost		EasyEnsemble		BalanceCascade		GESUPERPBOOST	
	AUC	<i>G</i> -mean	AUC	<i>G</i> -mean	AUC	<i>G</i> -mean	AUC	<i>G</i> -mean	AUC	<i>G</i> -mean
abalone9–18	0.7139	0.6963	0.8234	0.2391	0.7845	0.7111	0.7640	0.6743	0.8477	0.7788
breast-cancer	0.6052	0.5729	0.6222	0.4543	0.6149	0.5824	0.6362	0.5778	0.6362	0.6047
cancer	0.9598	0.9496	0.9860	0.9686	0.9856	0.9580	0.9816	0.9709	0.9876	0.9750
carG	0.9397	0.9447	0.9965	0.9006	0.9768	0.9596	0.9959	0.9705	0.9962	0.9833
ecoliCP-IM	0.9743	0.9738	0.9940	0.9809	0.9816	0.9571	0.9816	0.9704	0.9964	0.9741
ecoliM	0.8955	0.8692	0.9457	0.8266	0.9313	0.9088	0.9524	0.8866	0.9455	0.8737
ecoliMU	0.8673	0.8452	0.9052	0.6884	0.9026	0.8618	0.9262	0.8942	0.9107	0.8440
ecoliOM	0.8772	0.8660	0.9412	0.8074	0.9403	0.8813	0.9611	0.9084	0.9873	0.8946
euthyroid	0.9534	0.9445	0.9812	0.9294	0.9751	0.9479	0.9821	0.9443	0.9847	0.9444
german	0.6371	0.6443	0.7589	0.5941	0.7340	0.6725	0.7462	0.6741	0.7702	0.6875
glassBWFP	0.7741	0.7820	0.8959	0.8248	0.8791	0.8191	0.8990	0.8054	0.9182	0.8525
glassBWNFP	0.7905	0.7468	0.9090	0.8070	0.8197	0.7316	0.8630	0.7665	0.8874	0.7783
glassContainers	0.8501	0.7531	0.9775	0.6365	0.9508	0.8428	0.9214	0.8205	0.9551	0.8612
glassNW	0.9227	0.8816	0.9734	0.9041	0.9413	0.8846	0.9688	0.9272	0.9753	0.8990
glassTableware	0.9583	0.9565	0.9950	0.7000	0.9777	0.9625	0.9850	0.9562	0.9902	0.8618
glassVWFP	0.5477	0.4485	0.6516	0.1422	0.6075	0.5742	0.6915	0.5013	0.7434	0.6878
haberman	0.6412	0.5879	0.6931	0.3922	0.6496	0.5754	0.6511	0.6198	0.7007	0.5284
hepatitis	0.7197	0.6697	0.8664	0.5605	0.8110	0.6245	0.7972	0.6814	0.8505	0.7726
hypothyroidT	0.9893	0.9879	0.9942	0.9919	0.9930	0.9872	0.9938	0.9902	0.9982	0.9905
ionosphere	0.8844	0.8812	0.9723	0.9076	0.9649	0.9221	0.9720	0.9050	0.9850	0.9394
new-thyroidT	0.9338	0.9302	0.9985	0.9124	0.9812	0.9306	0.9751	0.9617	1.0000	0.9729
ozone1hr	0.7709	0.7384	0.8644	0.0000	0.8520	0.7532	0.8745	0.7841	0.8972	0.8272
ozone8hr	0.7564	0.7406	0.8839	0.2997	0.8772	0.7986	0.8704	0.7743	0.9175	0.8465
pima	0.7242	0.7008	0.8103	0.7060	0.7869	0.7296	0.7982	0.7205	0.8101	0.7322
segmentO	0.9906	0.9903	0.9994	0.9929	0.9978	0.9938	0.9977	0.9889	0.9999	0.9919
sick	0.9711	0.9680	0.9928	0.9157	0.9864	0.9658	0.9945	0.9757	0.9949	0.9710
splice-EI	0.9641	0.9626	0.9914	0.9528	0.9817	0.9616	0.9866	0.9616	0.9899	0.9646
splice-IE	0.9244	0.9242	0.9776	0.9051	0.9721	0.9346	0.9767	0.9385	0.9824	0.9404
tic-tac-toe	0.9490	0.9128	0.9997	0.9873	0.9937	0.9735	0.9964	0.9723	0.9997	0.9906
vehicleVAN	0.9404	0.9419	0.9955	0.9554	0.9834	0.9531	0.9881	0.9381	0.9933	0.9651
vowelZ	0.9113	0.9102	0.9901	0.8269	0.9580	0.9153	0.9802	0.9349	0.9978	0.9678
yeastCYT-POX	0.7220	0.6029	0.8378	0.5818	0.8089	0.5744	0.8005	0.6290	0.8502	0.6453
yeastEXC	0.8213	0.7999	0.8893	0.7131	0.8812	0.8309	0.9121	0.8575	0.9210	0.8238
yeastME1	0.9371	0.9365	0.9808	0.7924	0.9657	0.9222	0.9699	0.9455	0.9897	0.9423
yeastME2	0.8592	0.8229	0.8535	0.2085	0.9143	0.8389	0.9345	0.8114	0.9327	0.8641
Average	0.8479	0.8253	0.9137	0.7145	0.8961	0.8412	0.9064	0.8468	0.9241	0.8622

outperformed the standard algorithm in both AUC and *G*-mean. Arrows pointing up-left indicate that our algorithm improved *G*-mean but had worse AUC, whereas arrows pointing down-right indicate that our algorithm improved AUC but had a worse *G*-mean. Arrows pointing down-left indicate that our algorithm performed worse in both values.

If we inspect the results, we see that most of the arrows are pointing up-right. This behavior is specially common when

using C4.5 as base learner. For SVM, there is a clear advantage in terms of AUC, most arrows are pointing right, but the differences in terms of *G*-mean are most homogeneously distributed.

However, the differences showed in the figures must be corroborated by statistical tests to assure the advantage of GESUPERPBOOST. We performed a Holm procedure to ascertain the differences between our methods and the four

Table 3 Accuracy results, measured using the *G*-mean and AUC, for all the methods and a SVM as base learner

Dataset	Undersampling		AdaBoost		EasyEnsemble		BalanceCascade		GESUPERPBOOST	
	AUC	<i>G</i> -mean	AUC	<i>G</i> -mean	AUC	<i>G</i> -mean	AUC	<i>G</i> -mean	AUC	<i>G</i> -mean
abalone9–18	0.7297	0.6751	0.8128	0.4443	0.8161	0.7677	0.8708	0.7384	0.8177	0.7502
breast-cancer	0.5558	0.5500	0.5351	0.3536	0.6469	0.6022	0.6626	0.6343	0.6531	0.5733
cancer	0.9868	0.9672	0.9841	0.9551	0.9750	0.9676	0.8097	0.9683	0.9909	0.9736
carG	0.9968	0.9789	0.9675	0.8775	0.9713	0.9523	0.9419	0.9495	0.9941	0.9866
ecoliCP-IM	0.9901	0.9809	0.9864	0.9608	0.9832	0.9826	0.5000	0.9730	0.9985	0.9756
ecoliM	0.9335	0.8855	0.9245	0.7723	0.9089	0.8927	0.8985	0.8564	0.9564	0.8561
ecoliMU	0.9326	0.8861	0.8963	0.5473	0.8740	0.8630	0.9099	0.8733	0.9148	0.8738
ecoliOM	0.9811	0.9127	0.9176	0.8746	0.9819	0.9029	0.8500	0.9015	0.9843	0.9591
euthyroid	0.9175	0.8343	0.8976	0.7517	0.9016	0.8845	0.9463	0.8925	0.8976	0.8292
german	0.5235	0.1470	0.5230	0.0554	0.7440	0.7094	0.7726	0.7102	0.7634	0.3930
glassBWFP	0.8781	0.8002	0.8400	0.2904	0.7599	0.6803	0.8395	0.7238	0.8624	0.7993
glassBWNFP	0.8147	0.6925	0.8160	0.6726	0.6411	0.5197	0.6825	0.5717	0.8299	0.7628
glassContainers	0.9700	0.9094	0.9112	0.6605	0.8905	0.8352	0.8890	0.8309	0.9390	0.7421
glassNW	0.9798	0.9430	0.9706	0.9025	0.9406	0.9175	0.9190	0.9267	0.9675	0.9285
glassTableware	0.9667	0.8108	0.7832	0.5926	0.9121	0.8881	0.8675	0.7955	0.9783	0.9377
glassVWFP	0.6387	0.5590	0.7970	0.3406	0.5388	0.5111	0.6003	0.4353	0.6561	0.6007
haberman	0.7090	0.6003	0.6761	0.4330	0.6480	0.5463	0.7096	0.6395	0.6703	0.5292
hepatitis	0.6172	0.4788	0.4958	0.0000	0.7870	0.7324	0.8061	0.7266	0.8431	0.5829
hypothyroidT	0.7728	0.6985	0.7858	0.6894	0.8154	0.7883	0.8909	0.7863	0.8329	0.7454
ionosphere	0.9789	0.9078	0.9764	0.9382	0.8854	0.8643	0.8716	0.8600	0.9795	0.9450
new-thyroidT	1.0000	0.9858	0.9969	0.9499	0.9916	0.9752	0.7700	0.9752	1.0000	0.9754
ozone1hr	0.7802	0.6929	0.6397	0.3537	0.8450	0.8237	0.8949	0.8448	0.8505	0.7265
ozone8hr	0.8322	0.7280	0.7798	0.4646	0.8444	0.8060	0.8954	0.8096	0.8634	0.7664
pima	0.8191	0.7283	0.7379	0.5857	0.7541	0.7380	0.8320	0.7411	0.8253	0.7325
segmentO	0.9997	0.9902	0.9981	0.9936	0.9917	0.9909	0.5000	0.9912	0.9999	0.9968
sick	0.9125	0.8165	0.9012	0.7248	0.8878	0.8816	0.9416	0.8828	0.9274	0.8626
splice-EI	0.5997	0.4194	0.5891	0.4153	0.9669	0.9488	0.5000	0.9524	0.9646	0.4891
splice-IE	0.6082	0.4404	0.6061	0.4404	0.9401	0.9121	0.5000	0.9199	0.9287	0.5459
tic-tac-toe	0.9999	0.9749	0.9754	0.9749	0.9754	0.9749	0.9678	0.9749	0.9986	0.9797
vehicleVAN	0.9889	0.9571	0.9962	0.9672	0.9705	0.9649	0.5906	0.9606	0.9896	0.9650
vowelZ	0.9952	0.9685	0.9172	0.8832	0.8949	0.8772	0.9072	0.8715	0.9881	0.9656
yeastCYT-POX	0.7610	0.6204	0.7847	0.5062	0.7675	0.6073	0.7822	0.6070	0.8027	0.6483
yeastEXC	0.9284	0.8888	0.9109	0.6013	0.8798	0.8701	0.9221	0.8913	0.9413	0.8643
yeastME1	0.9843	0.9637	0.9842	0.7957	0.9634	0.9560	0.9866	0.9576	0.9842	0.9516
yeastME2	0.8917	0.8163	0.8596	0.3465	0.8563	0.8415	0.8836	0.8174	0.9032	0.8134
Average	0.8564	0.7774	0.8335	0.6319	0.8615	0.8279	0.8032	0.8283	0.8999	0.8008

standard algorithms as described in Sect. 3.1. Figures 5 and 6 show the results of the Holm test using our approach as control method and a decision tree and an SVM as base learners, respectively.

For C4.5, the test shows that GESUPERPBOOST was better than all the other methods for both AUC and *G*-mean. The differences are significant at a confidence level of 95 %. These results corroborate the differences showed by the ranks

in Fig. 1. The case for a SVM as base learner is somewhat different. The differences are significant in favor of our method for all the standard methods and AUC as accuracy measure. However, Holm test fails to find significant differences between our method and undersampling, EasyEnsemble and BalanceCascade for *G*-mean measure. However, as explained above, AUC is a more reliable measure than *G*-mean.

Fig. 1 Average Friedman's ranks for GESUPERPBOOST and the four standard methods using C4.5 as base learner

Fig. 2 Average Friedman's ranks for GESUPERPBOOST and the four standard methods using a SVM as base learner

Fig. 3 AUC/G-mean accuracy using relative movement diagrams for our proposal against the four standard methods using C4.5 as base learner. Positive values on both axes show better performance by our method

Fig. 4 AUC/G-mean accuracy using relative movement diagrams for our proposal against the four standard methods using a SVM as base learner. Positive values on both axes show better performance by our method

It is noticeable that ADABOOST obtained very bad results for G -mean for both base learners. There is an explanation of these results. ADABOOST improved as a rule the specificity values while worsening sensitivity. As ADABOOST is focused on overall error it invested its effort in the negative class, because it is more numerous and thus has a major impact on the overall error. This feature, useful for balanced datasets, is a serious drawback for class-imbalanced datasets.

4.2 Control experiment

It is well known that classifier diversity is an important feature [35] for the construction of any ensemble of classifiers. It is also the case for class-imbalanced datasets [54]. It may be argued that the performance of our approach was due to the diversity introduced by the non-linear projection itself, and not by the supervised projection obtained by means of the RCGA. To test this possibility we performed a control experiments. We constructed ensembles with an additional non-linear projection, as in GESUPERPBOOST, but the coefficients of the projection were randomly distributed in the

interval $[-1, 1]$. That way, we could compared whether the mere introduction of a random projection was able to obtain the same results as our method. Table 4 shows the comparison between GESUPERPBOOST and this random projection method using Wilcoxon test.

The table shows that the source of the good performance of GESUPERPBOOST was not due to the projection of the inputs alone, as the random projection method was worse than GESUPERPBOOST for both base learners and according to both measures.

5 Conclusions and future work

In this paper we have presented a new method for constructing ensembles based on combining the principles of boosting and the construction of supervised projections by means of a real-coded genetic algorithm. The idea of using a supervised projection, instead of the standard way of resampling or reweighting of boosting, seems appropriate for class-imbalanced datasets. We combine this method with

Fig. 5 Results of the Holm test for our approach as the control method for AUC (*top*) and G-mean (*bottom*) and C4.5 as base learner. Numerical p values are shown

Fig. 6 Results of the Holm test for our approach as the control method for AUC (*top*) and G-mean (*bottom*) and a SVM as base learner. Numerical p values are shown

Table 4 Pairwise comparison of GESUPERPBOOST accuracy measured using G -mean and AUC and ensembles constructed using a random projection

	GESUPERPBOOST			
	C.45		SVM	
	AUC	G -mean	AUC	G -mean
Random projection				
w/d/l	29/1/5	24/0/11	29/0/6	28/1/6
p value	0.0001	0.0084	0.0001	0.0156

Win/draw/loss record of GESUPERPBOOST against the random projection method and the p value of the Wilcoxon test are shown

undersampling to make it more scalable and to obtain better results.

Our experiments have shown that the proposed method achieved better results than undersampling and three different boosting methods. Two of these methods are specifically designed for class-imbalanced datasets and have shown their performance in previous papers [26].

The main drawback of our method is the scalability of the approach. Although this problem is ameliorated introducing undersampling in a previous step, it may be still a serious handicap if we deal with large datasets. In this way, our current research line is focused on improving the scalability of the method by means of the paradigm of the *democratization* [19] of learning algorithms.

Acknowledgments This work was supported in part by the Grant TIN2008-03151 of the Spanish “Comisin Interministerial de Ciencia y Tecnología” and the Grant P09-TIC-4623 of the Regional Government of Andalucía.

References

- Bauer, E., Kohavi, R.: An empirical comparison of voting classification algorithms: bagging, boosting, and variants. *Mach. Learn.* **36**(1/2), 105–142 (1999)
- Breiman, L.: Bias, variance, and arcing classifiers. Tech. Rep. 460, Department of Statistics, University of California, Berkeley (1996)
- Breiman, L.: Stacked regressions. *Mach. Learn.* **24**(1), 49–64 (1996)
- Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines (2001)
- Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: SMOTE: synthetic minority over-sampling technique. *J. Artif. Intell. Res.* **16**, 321–357 (2002)
- Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines. Cambridge University Press, Cambridge (2000)
- Demšar, J.: Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.* **7**, 1–30 (2006)
- Dietterich, T.G.: Ensemble methods in machine learning. In: Kitter, J., Roli, F. (eds.) *Proceedings of the First International Workshop on Multiple Classifier Systems*, Lecture Notes in Computer Science, vol. 1857, pp. 1–15. Springer (2000)
- Dietterich, T.G.: An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. *Mach. Learn.* **40**, 139–157 (2000)
- Dzeroski, S., Zenko, B.: Is combining classifiers with stacking better than selecting the best one? *Mach. Learn.* **54**, 255–273 (2004)
- Estabrooks, A., Jo, T., Japkowicz, N.: A multiple resampling method for learning from imbalanced data sets. *Comput. Intell.* **20**(1), 18–36 (2004)
- Fern, A., Givan, R.: Online ensemble learning: an empirical study. *Mach. Learn.* **53**, 71–109 (2003)
- Fernández, A., Jesús, M.J.D., Herrera, F.: Multi-class imbalanced data-sets with linguistic fuzzy rule based classification systems based on pairwise learning. In: *Proceedings of the Computational intelligence for knowledge-based systems design, and 13th international conference on Information processing and management of uncertainty, IPMU’10*, pp. 89–98. Springer, Berlin (2010)
- Frank, A., Asuncion, A.: UCI machine learning repository (2010). <http://archive.ics.uci.edu/ml>
- Freund, Y., Schapire, R.: Experiments with a new boosting algorithm. In: *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 148–156. Bari (1996)
- Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.* **55**(1), 119–139 (1997)
- Friedman, J.H., Hastie, T., Tibshirani, R.: Additive logistic regression: a statistical view of boosting. *Ann. Stat.* **28**(2), 337–407 (2000)
- Galar, M., Fernández, A., Barrenechea, E., Bustince, H., Herrera, F.: A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *IEEE Trans. Syst. Man Cybern. C Appl. Rev.* **42**, 463–484 (2012)
- García-Osorio, C., de Haro-García, A., García-Pedrajas, N.: Democratic instance selection: a linear complexity instance selection algorithm based on classifier ensemble concepts. *Artif. Intell.* **174**, 410–441 (2010)
- García-Pedrajas, N.: Constructing ensembles of classifiers by means of weighted instance selection. *IEEE Trans. Neural Netw.* **20**(2), 258–277 (2008)
- García-Pedrajas, N.: Supervised projection approach for boosting classifiers. *Pattern Recognit.* **42**, 1741–1760 (2009)
- García-Pedrajas, N., García-Osorio, C.: Constructing ensembles of classifiers using supervised projection methods based on misclassified instances. *Expert Syst. Appl.* **38**(1), 343–359 (2010)
- García-Pedrajas, N., García-Osorio, C., Fyfe, C.: Nonlinear boosting projections for ensemble construction. *J. Mach. Learn. Res.* **8**, 1–33 (2007)
- García-Pedrajas, N., de Haro-García, A.: Scaling up data mining algorithms: review and taxonomy. *Progr. Artif. Intell.* **1**, 71–87 (2012)
- García-Pedrajas, N., Hervás-Martínez, C., Ortiz-Boyer, D.: Cooperative coevolution of artificial neural network ensembles for pattern classification. *IEEE Trans. Evol. Comput.* **9**(3), 271–302 (2005)
- García-Pedrajas, N., Pérez-Rodríguez, J., García-Pedrajas, M.D., Ortiz-Boyer, D., Fyfe, C.: Class imbalance methods for translation initiation site recognition in dna sequences. *Knowl. Based Syst.* **25**, 22–34 (2012)
- Geng, X., Zhan, D.C., Zhou, Z.H.: Supervised nonlinear dimensionality reduction for visualization and classification. *IEEE Trans. Syst. Man Cybern. B Cybern.* **35**(6), 1098–1107 (2005)
- He, H., Garcia, E.A.: Learning from imbalanced data. *IEEE Trans. Knowl. Data Eng.* **21**, 1263–1284 (2009)
- Herrera, F., Lozano, M., Verdegay, J.L.: Tackling real-coded genetic algorithms: operators and tools for behavioural analysis. *Artif. Intell. Rev.* **12**, 265–319 (1998)

30. Ho, T.K.: The random subspace method for constructing decision forests. *IEEE Trans. Pattern Anal. Mach. Intell.* **20**(8), 832–844 (1998)
31. Japkowicz, N.: The class imbalance problem: significance and strategies. In: *Proceedings of the 200 International Conference on Artificial Intelligence (IC-AI'2000): Special Track on Inductive Learning*, vol. 1, pp. 111–117. Las Vegas (2000)
32. Kohavi, R., Kunz, C.: Option decision trees with majority voting. In: *Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 161–169. Morgan Kaufman, San Francisco (1997)
33. Kruskal, J.B.: Toward a practical method which helps uncover the structure of a set of multivariate observations by finding the linear transformation which optimizes a new 'index of condensation'. In: Milton, R.C., Nelder, J.A. (eds.) *Statistical Computing*, pp. 427–440. Academic Press, London (1969)
34. Kubat, M., Matwin, S.: Addressing the curse of imbalanced training sets: One-sided selection. In: *Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 179–186. Morgan Kaufmann (1997)
35. Kuncheva, L., Whitaker, C.J.: Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Mach. Learn.* **51**(2), 181–207 (2003)
36. Lee, E., Cook, D., Klinke, S., Lumley, T.: Projection pursuit for exploratory supervised classification. *J. Comput. Graph. Stat.* **14**(4), 831–846 (2005)
37. Lee, Y., Ahn, D., Moon, K.: Margin preserving projections. *Electron. Lett.* **42**(21), 1249–1250 (2006)
38. Lerner, B., Guterman, H., Aladjem, M., Dinstein, I., Romem, Y.: On pattern classification with sammon's nonlinear mapping—an experimental study. *Pattern Recognit.* **31**(4), 371–381 (1998)
39. Li, C.J., Jansuwan, C.: Dynamic projection network for supervised pattern classification. *Int. J. Approx. Reason.* **40**, 243–261 (2005)
40. Li, X., Yan, Y., Peng, Y.: The method of text categorization on imbalanced datasets. In: *Proceedings of the 2009 International Conference on Communication Software and Networks*, pp. 650–653 (2009)
41. Ling, C., Li, G.: Data mining for direct marketing problems and solutions, *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)*, pp. 73–79. AAAI Press, New York (1998)
42. Liu, X.Y., Wu, J., Zhou, Z.H.: Exploratory undersampling for class-imbalance learning. *IEEE Trans. Syst. Man Cybern. B Cybern.* **39**(2), 539–550 (2009)
43. Maudes-Raedo, J., Rodríguez-Díez, J.J., García-Osorio, C.: Disturbing neighbors diversity for decision forest. In: G. Valentini, O. Okun (eds.) *Workshop on Supervised and Unsupervised Ensemble Methods and Their Applications (SUEMA 2008)*, pp. 67–71. Patras, Grecia (2008)
44. Merz, C.J.: Using correspondence analysis to combine classifiers. *Mach. Learn.* **36**(1), 33–58 (1999)
45. Michalewicz, Z.: *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, New York (1994)
46. Polzehl, J.: Projection pursuit discriminant analysis. *Comput. Stat. Data Anal.* **20**(2), 141–157 (1995)
47. Quinlan, J.R.: *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo (1993)
48. Rodríguez, J.J., Díez-Pastor, J.F., García-Osorio, C.: Ensembles of decision trees for imbalanced data. *Lect. Notes Comput. Sci.* **6713**, 76–85 (2011)
49. Rodríguez, J.J., Kuncheva, L.I., Alonso, C.J.: Rotation forest: a new classifier ensemble method. *IEEE Trans. Pattern Anal. Mach. Intell.* **28**(10), 1619–1630 (2006)
50. Schapire, R.E., Freund, Y., Bartlett, P.L., Lee, W.S.: Boosting the margin: a new explanation for the effectiveness of voting methods. *Ann. Stat.* **26**(5), 1651–1686 (1998)
51. Schapire, R.E., Singer, Y.: Improved boosting algorithms using confidence-rated predictions. *Mach. Learn.* **37**, 297–336 (1999)
52. Sun, Y., Kamel, M.S., Wong, A.K., Wang, Y.: Cost-sensitive boosting for classification of imbalanced data. *Pattern Recognit.* **40**, 3358–3378 (2007)
53. Tenenbaum, J.B., de Silva, V., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. *Science* **290**(5500), 2319–2323 (2000)
54. Ensemble diversity for class imbalance learning. Ph.D. thesis, University of Birmingham (2011)
55. Webb, G.I.: Multiboosting: a technique for combining boosting and wagging. *Mach. Learn.* **40**(2), 159–196 (2000)
56. Weiss, G.M., Provost, F.: The effect of class distribution on classifier learning: An empirical study. Rutgers University, Tech. Rep. TR-43, Department of Computer Science (2001)
57. Wilcoxon, F.: Individual comparisons by ranking methods. *Biometrics* **1**, 80–83 (1945)
58. Yu, S., Yu, K., Tresp, V., Kriegel, H.P.: Multi-output regularized feature projection. *IEEE Trans. Knowl. Data Eng.* **18**(12), 1600–1613 (2006)
59. Zhao, H., Sun, S., Jing, Z., Yang, J.: Local structure based supervised feature extraction. *Pattern Recognit.* **39**, 1546–1550 (2006)