Constructing Ensembles of Classifiers by Means of Weighted Instance Selection

Nicolás García-Pedrajas, *Member, IEEE*

***Abstract*—In this paper, we approach the problem of constructing ensembles of classifiers from the point of view of instance selection. Instance selection is aimed at obtaining a subset of the instances available for training capable of achieving, at least, the same performance as the whole training set. In this way, instance selection algorithms try to keep the performance of the classifiers while reducing the number of instances in the training set. Meanwhile, boosting methods construct an ensemble of classifiers iteratively focusing each new member on the most difficult instances by means of a biased distribution of the training instances. In this work, we show how these two methodologies can be combined advantageously. We can use instance selection algorithms for boosting using as objective to optimize the training error weighted by the biased distribution of the instances given by the boosting method. Our method can be considered as boosting by instance selection. Instance selection has mostly been developed and used for k-nearest neighbor (k-NN) classifiers. So, as a first step, our methodology is suited to construct ensembles of k-NN classifiers. Constructing ensembles of classifiers by means of instance selection has the important feature of reducing the space complexity of the final ensemble as only a subset of the instances is selected for each classifier. However, the methodology is not restricted to k-NN classifier. Other classifiers, such as decision trees and support vector machines (SVMs), may also benefit from a smaller training set, as they produce simpler classifiers if an instance selection algorithm is performed before training. In the experimental section, we show that the proposed approach is able to produce better and simpler ensembles than random subspace method (RSM) method for k-NN and standard ensemble methods for C4.5 and SVMs.**

***Index Terms*—Boosting, decision trees, ensembles of classifiers, instance selection, k-nearest neighbors (k-NN), support vector machines (SVMs).**

I. Introduction

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

Manuscript received October 07, 2007; revised June 23, 2008 and August 06, 2008; accepted August 24, 2008. First published January 27, 2009; current version published February 06, 2009.

The author is with the Department of Computing and Numerical Analysis, University of Córdoba, Córdoba 14071, Spain (e-mail: npedrajas@uco.es).

Digital Object Identifier 10.1109/TNN.2008.2005496

[1]There are other classification tasks where the instances to be classified are not represented as vectors. The proposed methodology can be applied to any of these problems provided that a learning algorithm and an instance selection method are available.

The task is to find a definition for the unknown function $\mathcal{F}(\mathbf{x})$, given the set of training instances. In a classifier ensemble framework, we have a set of classifiers $\mathbb{F} = \{f_1, f_2, \ldots, f_M\}$, each classifier performing a mapping of an instance vector $\mathbf{x} \in \mathbb{R}^D$ into the set of labels $Y = \{1, \ldots, K\}$, $f_t : \mathbb{R}^D \rightarrow Y$. The design of classifier ensembles consists of constructing the individual classifiers f_t and combining them to obtain a rule that finds a class label for $\mathbf{x}$ based on the outputs of the classifiers $\{f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_M(\mathbf{x})\}$. In this paper, we will use f to refer to any member of the ensemble, and F to refer to the ensemble.

Techniques using multiple models usually consist of two independent phases: model generation and model combination [1]. 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 [2] on the classifiers that form the ensemble [3]. Kuncheva [4] identifies four fundamental approaches: 1) using different combination schemes, 2) using different classifier models, 3) using different feature subsets, and 4) using different training sets. Perhaps the latter is the most commonly used. The algorithms in this last approach can be divided into two groups: algorithms that adaptively change the distribution of the training set based on the performance of the previous classifiers, and algorithms that do not adapt the distribution. Boosting methods are the most representative of the first group. The most widely used boosting method is ADABOOST [5] and its numerous variants. All of them are 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 [1], [3], [6], [7], or [8].

Bagging [9] is the most representative algorithm of the second group. Bagging (after *B*ootstrap *agg*regat*ing*) simply generates different bootstrap samples from the training set. Several empirical studies have shown that ADABOOST is able to reduce both bias and variance components of the error [10]–[12]. On the other hand, bagging seems to be more efficient in reducing bias than ADABOOST [12].

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 M individual classifiers $f_t(\mathbf{x})$

$$F(\mathbf{x}) = \arg\max_{y \in Y} \sum_{t=1}^{M} \alpha_t \left[f_t(\mathbf{x}) = y \right] \quad (1)$$

where the α_t are appropriately defined, and $[\pi]$ 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 the instance correctly. Thus, for ADABOOST, each instance $\mathbf{x}_j$ receives a weight w_j^t for training the tth classifier. Initially, all the instances are weighted equally $w_j^1 = 1/n, \forall j$. Then, for classifier $(t+1)$th the instance is weighted as follows:

$$w_i^{t+1} = w_i^t \beta_t^{(1-[f_t(\mathbf{x}_i)=y_i])} \tag{2}$$

where

$$\beta_t = \frac{\epsilon_t}{(1-\epsilon_t)} \tag{3}$$

being ϵ_t the weighted error of classifier t when the weight vector is normalized $\sum_{j=1}^n w_j^t = 1$. Once the classifiers are trained, the function $F(\mathbf{x})$ is given by (1), with the weight of each classifier given by

$$\alpha_t = \ln\left(\frac{1}{\beta_t}\right). \tag{4}$$

In a widely used variant [12], when the weighted error of a classifier ϵ_t is greater than 0.5, the classifier is assigned a 0 weight and the distribution is reinitialized taking a bootstrap sample from the training set. This is the method used in all our experiments when the classifier has an error above 0.5.

Instance selection [13] consists of choosing a subset of the total available data to achieve the original purpose of the data mining application as if the whole data are used. Different variants of instance selection exist. We can distinguish two main models [14]: instance selection as a method for prototype selection for algorithms based on prototypes (such as k-nearest neighbors (k-NN) [15]) and instance selection for obtaining the training set for a learning algorithm that uses this training set (such as decision trees or neural networks). In this paper, we make use of both options.

The idea underlying this work is that we can use instance selection as a tool for constructing ensembles of classifiers. The basic aim of boosting is trying to improve, at each step, the classification rate of a certain subset of instances found difficult by the previous classifiers. An alternative to the use of instance weights, as in boosting, would be using just the most difficult instances to train the next classifier. It is likely that such a method would bias the learned rule so much as to achieve poor overall results. However, if we are able to obtain a subset of instances that, used as training set, are able to improve the classification rate of those difficult instances, we can achieve two advantages. First, we develop a new approach to boosting classifiers, and second, we can obtain simpler classifiers for those learners, such as k-NN, C4.5, and support vector machines (SVMs), for which the space complexity of the induced rule for the same problem depends, at least partially, on the number of instances. Thus, the use of instance selection for constructing ensembles of classifiers is a natural choice. Moreover, as the space complexity is decreased, also the time complexity of classifying an instance is reduced once the ensemble is constructed.

The general idea of the proposed method is the following: Classifier f_0 is obtained using all the available data. Then, to train classifier f_{t+1}, first we obtain the distribution of weights given by a certain boosting algorithm,[2] $\mathbf{w}^{t+1}$, and then we use an instance selection algorithm to obtain a subset of instances S^{t+1} aimed at improving the weighted error $\epsilon_w = E[\mathbf{w}^{t+1}_{(y \neq f(\mathbf{x}))}]$. With this subset S^{t+1}, the classifier f_{t+1} is trained with a uniform distribution of the instances.

This method follows the philosophy of standard boosting with a different approach. In boosting, difficult instances are given higher weights that are used for resampling or used by the classifier when possible. In this way, the learning algorithm is focused on these difficult instances. However, this is not the only way to improve the classification of difficult instances. The one presented here is another one. To try to classify difficult instances, we search the subset of training instances that must be fed to the induction algorithm to learn those difficult instances. So, we have the same objective as boosting, improving weighted error, using a different approach. Once the classifiers are learned on the reduced training sets, the same additive model of (1) is constructed. Our approach has the additional important advantage of reducing the space complexity of the ensemble.

This paper is organized as follows. Section II reviews some related work. Section III explains the proposed method applied to k-NN and Section IV shows the application to C4.5 and SVM. Section V shows the experimental setup and Section VI shows the results of experiments carried out. Finally, Section VII states the conclusions of our work.

II. Related Work

To the best of our knowledge no previous work has tried to construct ensembles using instance selection. Only the work of Freund and Schapire [5] shares some of the ideas underlying our work. Freund and Schapire developed a boosting version of nearest neighbor classifier, but with the goal of speeding it up and not improving its accuracy. In their method, each weak classifier is defined by a subset P of the training set and a 1-NN rule. On each boosting round, a set P is created for the classifier adding instances in a stepwise manner. Initially, $P = \emptyset$. At each boosting step, ten random candidates are selected according to the current distribution of instances given by the boosting algorithm. The candidate that causes the largest decrease in the pseudoloss is added to the set P. This process is repeated until P reaches a prespecified size.

III. Constructing Ensembles of k-NNs by Weighted Instance Selection

The k-NN rule is a well-known and widely used method for classification. The method consists of storing a set of prototypes that must represent the knowledge of the problem. This set of prototypes can be the whole training set, a subset obtained using instance selection [16], or different combinations of training instances [17].

The k-NN method is used mainly because of its simplicity and its ability to achieve error results comparable with much more complex methods. For instance, in computer vision, it has been applied to a wide range of problems successfully, such as

[2]The method can be used with any of the existing boosting methods, as it only needs a vector of weights.

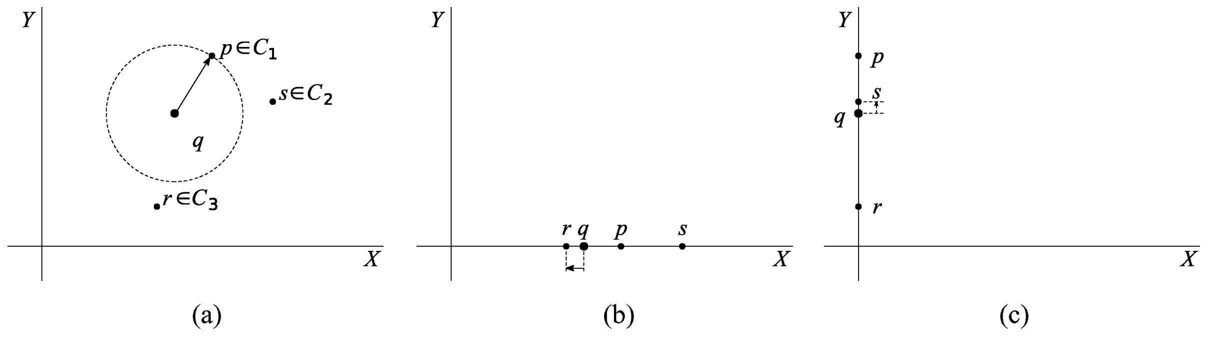

Fig. 1. Sensitiveness to subspace selection. We have a test instance q and three training instances p, s, and r belonging, respectively, to classes 1, 2, and 3. Using a 1-NN rule, q can be classified either into (a) class 1 using variables x and y, (b) class 3 using variable x, or (c) class 2 using variable y.

face recognition [18], articulated pose estimation [19], and character recognition [20]. On many occasions, the reported results are able to improve the errors obtained with more sophisticated methods [21].

Combination methods, such as voting or bagging, are not usually useful when applied to k-NN, as this method is fairly stable with respect to modification of the training set. Furthermore, error correcting output codes [22] are also unsuccessful with k-NN. It has been shown that error correcting output codes are not useful with classifiers that use local information due to high error correlation [23]. On the other hand, k-NN is very sensitive to input perturbations, such as subspace selection or nonlinear projections. Fig. 1 illustrates with a simple example the sensitiveness to subspace selection. The test pattern can be classified in three different classes depending on the subspace considered. In this way, subspace methods have shown good results when applied to k-NN [24], [25]. This property will be used in our method.

As we have stated, the basic idea is using the distribution given by the boosting algorithm to optimize a weighted training error using an instance selection algorithm. The outline of this procedure is given in Algorithm 1. The different versions of the method are given by the different ways of implementing the instance selection algorithm and the different learning algorithms to be used. In this section, we focus on using the proposed methodology to "boost" k-NN classifier, while Section IV is devoted to its application to other classifiers.

Algorithm 1: Outline of the proposed methodology for constructing ensembles using weighted instance selection.

```
    Data: A training set $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$,
          $\mathbf{x}_j \in \mathbb{R}^D$, a learning algorithm $\mathbb{L}$, and the number
          of iterations $M$.

    Result: The final classifier: $F(\mathbf{x}) = \arg\max_{y \in Y} \sum_{t: f_t(\mathbf{x}) = y} \alpha_t$.

1.  $f_0 = \mathbb{L}(T)$
    for $t = 1$ to $M$ do
2.      Obtain vector $\mathbf{w}^t$ (†) of weights for each instance using
        a boosting method
3.      Obtain an appropriate subset $S^t \subset T$ aimed at
        optimizing $\epsilon_t = E[\mathbf{w}^t_{(y \neq f(\mathbf{x}))}]$
4.      $f_t = \mathbb{L}(S^t)$
    end

    †The exact way of obtaining $\mathbf{w}^t$ depends on the boosting
    algorithm used.
```

The key point of Algorithm 1 is how to modify instance selection procedures to optimize $\epsilon_t = E[\mathbf{w}^t_{(y \neq f(\mathbf{x}))}]$ instead of $\epsilon = E[\mathbf{1}_{(y \neq f(\mathbf{x}))}]$. Within the field of instance selection, we can consider two basic groups: "classical" algorithms, in the sense that they are aimed at looking for and removing useless and noisy instances, and evolutionary algorithms, which try to optimize a fitness function without taking into account the instances that are removed by the procedure. The modification of evolutionary algorithms to consider instance weights $\mathbf{w}$ is straightforward. We only need to modify the fitness function. On the other hand, not all classical algorithms can be adapted to use $\mathbf{w}$. In the following sections, we show how two classical and one evolutionary procedures can be modified. Many other methods can be modified to our methodology as well.

A. Using Standard Instance Selection Methods

As we have said, classical instance selection methods try to detect noisy and redundant instances considering the behavior of k-NN classifier. In this way, these algorithms are specifically designed for NN-like rules and are not easily applied to other classifiers. Among the most widely used instance selection methods, many of them can be conveniently modified to use a vector of weights that assigns a weight to each instance showing the stress that must be put on keeping the classification of that instance correct. As an example we show the modification for two widely used methods: reduced nearest neighbor rule and decremental reduction optimization procedure 3 algorithm. The former is chosen because it is one of the simplest methods so it can be easily implemented and used, and the latter is chosen because it is one of the top performing instance selection algorithms.

One of the possible problems with our approach is the stability of k-NN rule with respect to instance selection. However,

k-NN is unstable with respect to subspaces, as we explained above. Thus, in order to avoid problems derived from the stability with respect to instance selection, standard instance selection algorithms are coupled with the random subspace method (RSM) proposed by Ho [26] to add the required diversity to the obtained classifiers. RSM [27] consists of training each classifier on a different subspace randomly sampled from the whole input space. It is rooted in the theory of stochastic discrimination [28], and it has common points with bagging, but instead of sampling instances, it samples subspaces [29]. RSM has been successfully applied to different problems [30], [31]. For instance, in selection based on genetic algorithms, we can evolve the input subspace instead of using RSM, which is a better solution for the stability of k-NN rule with respect to sampling (see Section III-B).

Some instance selection methods, such as decremental reduction optimization procedure 3 (Drop3) [32] and iterative case filtering (ICF) [16], include an initial step of noise removal based on Wilson's edited nearest neighbor rule [33]. This step removes instances that are misclassified by its neighbors. As we are precisely interested in those instances, this initial step is not considered in our modified version of instance selection algorithms.

1) Reduced Nearest Neighbor Rule (RNN): Hart [34] introduced the concept of a minimally consistent subset of the training set. A consistent subset of a training set T is some subset S of T that correctly classifies every case in T with the same accuracy as T itself. A consistent subset is therefore likely to preserve the classification accuracy achieved in the testing set. He developed an attempt to find such a subset called condensed nearest neighbor rule (CNN). CNN removes instances whose removal does not result in an increase in the misclassification rate.

Gates [35] introduced a second attempt devising the reduced nearest neighbor (RNN) rule, which extends the idea of CNN by constructing a complete set of instances to form a consistent subset. Gates uses the same criteria for case deletion as Hart, but builds the edited set of cases from opposite starting positions. The criterion used is essentially that of learning feedback: for a case to be kept, it must prove useful on the basis of classification trials.

The RNN algorithm starts with $S = T$ and removes each instance from S if such removal does not cause any other instances in T to be misclassified by the instances remaining in S. Since the instance being removed is not guaranteed to be classified correctly, this algorithm is able to remove noisy instances and internal instances while retaining border points. This simple rule can be adapted almost without modification. We can rephrase RNN in the following way: RNN starts with $S = T$ and removes each instance from S if such removal does not cause an increment in the training error in T, $\epsilon = E[\mathbf{1}_{(y \neq f(\mathbf{x}))}]$.

Following the philosophy explained, a version of RNN using a vector of weights $\mathbf{w}$ obtained using a boosting algorithm can be developed in the following way: weighted reduced nearest neighbor rule (wRNN) starts with $S = T$ and removes each instance from S if such removal does not cause an increment in the weighted training error in T, $\epsilon_w = E[\mathbf{w}_{(y \neq f(\mathbf{x}))}]$, where $\mathbf{w}$ is a vector that weights each instance. The procedure is shown in Algorithm 2.

Algorithm 2: Weighted reduced nearest neighbor (wRNN) rule

```
   Data: A training set $T = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$, a weight
         vector $\mathbf{w}$ obtained using a boosting method.
   Result: The reduced training set $S \subset T$.
1. $S = T$
   foreach Instance $P \in S$ do
2.     Evaluate weighted error with $P$ in $T$:
       $\epsilon_P = E[\mathbf{w}_{(y \neq f(\mathbf{x}))}]$
3.     Remove $P$ temporarily
4.     Evaluate weighted error without $P$ in $T$:
       $\epsilon_{!P} = E[\mathbf{w}_{(y \neq f(\mathbf{x}))}]$
       if $\epsilon_P \geq \epsilon_{!P}$ then
5.         Remove $P$ definitively from $S$
       end
   end
6. Return $S$
```

2) Decremental Reduction Optimization Procedure 3 (Drop3): Drop3 [32] (see Algorithm 3) represents one of the examples of a new generation of algorithms that were designed taking into account the effect of the order of removal on the performance of the algorithm. So, this algorithm is designed to be insensitive to the order of presentation of the instances. It includes a noise filtering step using a method similar to Wilson's *edited nearest neighbor* rule [33]. Then, the instances are ordered by the distance to their nearest neighbor. The instances are removed beginning with the instances furthest from its nearest neighbor. This tends to remove the instances furthest from the boundaries first.

Algorithm 3: Decremental reduction optimization procedure 3 (Drop3) algorithm.

```
   Data: A training set $T = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$
   Result: The reduced training set $S \subset T$.
1. Noise filtering: Remove any instance in $T$ misclassified by
   its $k$ neighbors
2. $S = T$
3. Sort instances in $S$ by distance to their nearest enemy
   foreach Instance $P \in S$ do
4.     Find $P.N_{1..k+1}$, the $k + 1$ nearest neighbors of $P$ in $S$
5.     Add $P$ to each of its neighbors' list of associates
   end
```

```
    foreach Instance $P \in S$ do
6.    Let with $= \#$ of associates of $P$ classified correctly
      with $P$ as a neighbor
7.    Let without $= \#$ of associates of $P$ classified correctly
      without $P$
      if without $\geq$ with then
8.      Remove $P$ from $S$
        foreach Associate $A$ of $P$ do
9.        Remove $P$ from $A$'s list of nearest neighbors
10.       Find a new nearest neighbor for $A$
11.       Add $A$ to its new neighbor's list of associated
        end
      end
    end
12. Return $S$
```

The adaptation of this algorithm to our purposes is also straightforward. Lines 6 and 7 of Algorithm 3 must be modified. It is also necessary to remove noise filtering, as this step would remove many of the instances we are trying to classify correctly. The modified algorithm, called *weighted decremental reduction optimization procedure 3* (wDrop3) is shown in Algorithm 4. Only modified lines are shown. The idea is, instead of considering just the number of instances classified correctly with and without P, we take into account the weights of the instances. In this way, an instance is removed if the sum of the weights of its associates correctly classified is not decreased when the instance is not considered.

Algorithm 4: Weighted decremental reduction optimization procedure 3 (wDrop3) algorithm. Only modified lines with respect to standard Drop3 (Algorithm 3) are shown.

```
    Data: A training set $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, a weight
          vector w obtained using a boosting method.
    Result: The reduced training set $S \subset T$.
1. Line removed
6. Let with $= \sum w_i$ of associates of $P$ classified correctly
   with $P$ as a neighbor
7. Let without $= \sum w_i$ of associates of $P$ classified correctly
   without $P$
```

B. Using Evolutionary Instance Selection

Genetic algorithms have been widely used for instance selection, considering this task as a search problem. The application is easy and straightforward. Each individual is a binary vector that codes a certain sample of the training set. The evaluation is usually made considering both data reduction[3] and classification accuracy. Examples of applications of genetic algorithms to instance selection can be found in [36]–[38].

One of the most interesting advantages of the application of evolutionary computation to instance selection is that evolutionary approaches do not depend on specific classifiers, and can be used with any instance-based classifier. This contrasts with most standard algorithms that are specifically designed for k-NN classifiers. For example, Reeves and Bush [38] used a genetic algorithm to select instances for radial basis function (RBF) neural networks. This possibility is used in the next section to apply our methodology to C4.5 and SVM classifiers.

An evolutionary algorithm begins with a set of randomly generated solutions, a *population*. Then, new solutions are obtained by the combination of two existing solutions: *crossover* operator and the random modification of a previous solution, *mutation* operator. All the individuals are then evaluated assigning to each one a value called *fitness*, which measures its ability to solve the problem. After this process, the best individuals, in terms of higher fitness, are selected and an evolution cycle is completed. This cycle is termed a *generation*.

Cano *et al.* [14] performed a comprehensive comparison of the performance of different evolutionary algorithms for instance selection. They compared a generational genetic algorithm [39], a steady-state genetic algorithm [40], a CHC genetic algorithm [41], and a population-based incremental learning algorithm [42]. They found that evolutionary-based methods were able to outperform classical algorithms in both classification accuracy and data reduction. Among the evolutionary algorithms, CHC was able to achieve the best overall performance, so this is the method used here. CHC [41] stands for *cross-generational elitist selection, heterogeneous recombination, and cataclysmic mutation*. The nontraditional CHC genetic algorithm differs from traditional GAs in a number of ways [43].

1) To obtain the next generation for a population of size N, the parents and the offspring are put together and the N best individuals are selected.
2) To avoid premature convergence, only different individuals, separated by a threshold Hamming distance—in our implementation 4 bits—are allowed to mate.
3) During crossover, two parents exchange exactly half of their nonmatching bits. This operator is the half uniform crossover (HUX) [41]. This operator generates two offspring from two parents. Each offspring inherits the matching bits of the two parents, and half of the nonmatching bits from each parent alternately.
4) Mutation is not used during the regular evolution. In order to avoid premature convergence or stagnation of the search, the population is reinitialized when the individuals are not diverse. In such a case, only the best individual is kept in the new population.

[3]Data reduction is measured as the percentage of instances of the training set removed by the instance selection algorithm.

Although in the standard CHC algorithm mutation is not used, we have introduced two mutation operators for the sake of performance: random and RNN mutation. Random mutation randomly modifies some of the bits of an individual; RNN mutation applies a reduced nearest neighbor algorithm as an additional method for improving the reduction abilities of the CHC algorithm. Each mutation is performed independently. The procedure for performing weighted instance selection by means of a CHC method is shown in Algorithm 5.

Algorithm 5: Outline of the CHC genetic algorithm for obtaining the instance selection given a vector of weights for each instance. The method is called weighted instance selection CHC (wIS-chc) algorithm. α is a parameter to balance reduction and weighted error in the fitness function that must be fixed by the user.

```
Data: A training set $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$,
      $\mathbf{x}_i \in \mathbb{R}^D$, a base learning algorithm $\mathbb{L}$, and a weight
      vector $\mathbf{w}$ obtained using a boosting method.
Result: The reduced training set $S \subset T$.
1.  Initialize population
    for Number of generations do
2.      Obtain new individuals using HUX crossover
3.      Apply random mutation with probability $P_{\mathrm{mut}}$
4.      Apply RNN mutation with probability $P_{\mathrm{RNN}}$
5.      // Evaluation of individuals
        foreach Individual $s_i$ in the population do
6.          Train a classifier using $s_i$: $f = \mathbb{L}(s_i)$
7.          Evaluate weighted error of $f$: $\epsilon_w = E[\mathbf{w}_{(y \neq f(\mathbf{x}))}]$
8.          Evaluate reduction, $r$, in the number of instances
            of the training set
9.          Evaluate fitness of $s_i$: $\phi = \alpha(1 - \epsilon_w) + (1 - \alpha)r$
        end
10.     Select $N$ best individuals for the next generation
    end
11. Return $S$
```

The use of a genetic algorithm for instance selection has an additional advantage. As we have said, k-NN is stable with respect to resampling. This might be a problem for our algorithm. One possible solution is using RSM as we have used in the standard instance selection algorithms. However, it is known that for RSM some of the random subspaces may lack the discriminant ability to separate the different classes. The classifiers learned using such subspaces have poor performance and may harm the whole generalization ability of the ensemble. Genetic algorithms offer a better method; the subspace itself can be learned together with the selection of instances. The individual codifies the instances selected and the subspace to be used by the learner. Moreover, this approach allows the algorithm to find solutions that obtain better weighted errors, improving the performance of the approach.

The fitness of the individuals is measured using a weighted sum of two criteria: weighted error and reduction. In this way, the fitness of an individual ϕ is given by

$$\phi = \alpha(1 - \epsilon_w) + (1 - \alpha)r \qquad (5)$$

where ϵ_w is the weighted error $\epsilon_w = \sum_{j=1}^{n} w_j [f(\mathbf{x}_j) \neq y_j]$, where $f(\mathbf{x})$ is the output of the classifier trained with the selection of instances given by the individual, $\mathbf{w}$ is the vector of weights for the instances, and r is the reduction in the number of instances given by the individual. α measures the relevance of each member of the fitness value and must be fixed by the researcher. This kind of fitness function is usually used for genetic-algorithm-based instance selection methods [14], [44]. This fitness function considers the two basic factors we are combining in our model: weighted error minimization from boosting and reduction of storage from instance selection.

Regarding computational complexity, the time needed for evolving our model is significantly higher than the use of a standard method such as RSM or AdaBoost. However, we think that this problem is compensated by the following two facts.

1) The time is incremented only in the training stage. As this stage is performed offline, training time is not so dramatic a constraint. Moreover, as most parts of the algorithm can be easily parallelized or distributed, the time can be reduced when dealing with larger problems.
2) Time complexity when using the classifier is reduced, as the final ensemble is smaller. We believe that this reduction outweighs the added complexity in training time, as a faster classifier is better, even if its training was more time consuming.

IV. Constructing Ensembles Using Other Classifiers

In the previous sections, we have developed the proposed idea for k-NN rule. However, as we have said, instance selection can be also used for obtaining the training set for other learning algorithms. Thus, the obvious question is whether the proposed method can be applied to other types of classifiers advantageously. If we are able to achieve a simplification of the ensemble obtained, as in k-NN, the approach is valuable. Several classifiers have complexities that depend on the number of instances of the training set. We have observed that for C4.5 and SVM the complexity of the classifier, for the same problem, depends on the number of instances. In this way, the number of nodes of the decision tree or the number of support vectors (SVs) decreases approximately linearly as we remove instances from the training set. For these kind of classifiers, the proposed approach can be very useful. We may be able to construct ensembles whose individual members are simpler.

There are two possible approaches for boosting these classifiers by means of instance selection following the ideas shown above.

The first one consists of using the selection performed using a method designed for k-NN classifiers. Ishibuchi *et al.* [45] showed that instance selection based on nearest neighbors can be used to improve the performance and reduce training time of a neural network. However, it is unlikely that such an approach will achieve good results, due to the differences between k-NN and the other classifiers used. It has been shown that different groups of learning algorithms need different instance selectors in order to suit their learning/search bias [46].

The second, and more promising approach, consists of using a wrapper method [47] coupled with an evolutionary instance selection method such as the one described in Algorithm 5. Similar methods have been used to train neural networks [38], [48], although without focusing on boosting the classifier. Reeves and Taylor [38] used such an approach coupled with a genetic algorithm to select instances for RBF networks. Algorithm 5 can be used with any type of classifier as the base learning algorithm $\mathbb{L}$ is a parameter of the algorithm. Both approaches were tested and the results corroborate the intuition about the performance of each one. In the experiments, we show how Algorithm 5 can be applied using as learners C4.5 algorithm and SVMs.

V. Experimental Setup

For the comparison of the different methods we selected 45 data sets from the University of California at Irvine (UCI) Machine Learning Repository [49]. A summary of these data sets is shown in Table I. We have chosen a wide variety of problems regarding number of classes, number of instances, number of inputs, and types of features.

The experiments were conducted following the 5×2 cross-validation (cv) setup [50]. We perform five replications of a twofold cross validation. In each replication, the available data are partitioned into two random equal-sized sets. Each learning algorithm is trained on one set at a time and tested on the other set.

As control method for k-NN ensembles we use RSM. In a set of preliminary experiments, this was the best performing standard method for constructing ensembles of k-NN classifiers. We also include bagging for the sake of completeness, although this method is not able to improve k-NN in a consistent way. For C4.5 and SVM learning algorithms, we use the variant of AdaBoost given by Bauer and Kohavi [12], and show also comparisons with RSM and bagging.

Following [51], we carry out as a first step an Iman–Davenport test [52], to ascertain whether there are significant differences among all the methods. Then, pairwise differences are measured using a Wilcoxon test. This test is recommended because it was found to be the best one for comparing pairs of algorithms [51]. The formulation of the test [53] is the following. Let d_i be the difference between the error values of the methods in ith data set. These differences are ranked according to their absolute values; in case of ties, an average rank is assigned. Let R^+ be the sum of ranks for the data sets on which the second algorithm outperformed the first, and let R^- be the sum of ranks where the first algorithm outperformed the second. Ranks of $d_i = 0$ are split evenly among the sums

$$R^+ = \sum_{d_i>0} \mathrm{rank}(d_i) + \frac{1}{2}\sum_{d_i=0} \mathrm{rank}(d_i) \quad (6)$$

TABLE I
Summary of Data Sets. The Inputs Column Shows the Number of Inputs to the Classifier That Depends on the Number of Input Features and Their Type

Data set	Cases	Features			Classes	Inputs
		Cont.	Binary	Nominal		
arrhythmia	452	279	–	–	13	279
audiology	226	–	61	8	24	93
autos	205	15	4	6	6	72
breast-cancer	286	–	3	6	2	15
car	1728	–	–	6	4	16
card	690	6	4	5	2	51
dermatology	366	1	1	32	6	34
ecoli	336	7	–	–	8	7
gene	3175	–	–	60	3	120
german	1000	6	3	11	2	61
glass	214	9	–	–	6	9
glass-g2	163	9	–	–	2	9
heart	270	13	–	–	2	13
hepatitis	155	6	13	–	2	19
horse	364	7	2	13	3	58
ionosphere	351	33	1	–	2	34
isolet	7797	617	–	–	26	34
letter	5000	16	–	–	26	16
liver	345	6	–	–	2	6
lrs	531	101	–	–	10	101
lymphography	148	3	9	6	4	38
mfeat-fac	2000	216	–	–	10	216
mfeat-fou	2000	76	–	–	10	76
mfeat-kar	2000	64	–	–	10	64
mfeat-mor	2000	6	–	–	10	6
mfeat-pix	2000	240	–	–	10	240
mfeat-zer	2000	47	–	–	10	47
optdigits	5620	64	–	–	10	64
page-blocks	5473	10	–	–	5	10
phoneme	5404	5	–	–	2	5
pima	768	8	–	–	2	8
primary-tumor	339	–	14	3	22	23
promoters	106	–	–	57	2	114
satimage	6435	36	–	–	6	36
segment	2310	19	–	–	7	19
sick	3772	7	20	2	2	33
sonar	208	60	–	–	2	60
soybean	683	–	16	19	19	82
vehicle	846	18	–	–	4	18
vote	435	–	16	–	2	16
vowel	990	10	–	–	11	10
waveform	5000	40	–	–	3	40
yeast	1484	8	–	–	10	8
zip (USPS)	9298	256	–	–	10	50
zoo	101	1	15	–	7	16

and

$$R^- = \sum_{d_i<0} \mathrm{rank}(d_i) + \frac{1}{2}\sum_{d_i=0} \mathrm{rank}(d_i). \quad (7)$$

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

$$z = \frac{T - \frac{1}{4}N(N+1)}{\sqrt{\frac{1}{24}N(N+1)(2N+1)}} \quad (8)$$

is distributed approximately according to $N(0,1)$. We combine these two tests to assess the differences in performance of the different algorithms. When the comparison is between two algorithms only the Wilcoxon test is used.

TABLE II
SUMMARY OF TEST ERROR RESULTS FOR THE THREE STANDARD k-NN METHODS AND THE THREE PROPOSALS FOR BOOSTING k-NN USING INSTANCE SELECTION, WITH A 5 × 2 CV SETUP

Dataset	k-NN			Instance selection					
	Test error			Test error			Reduction		
	1 classifier	RSM	Bagging	wIS-chc	wRNN	wDrop3	wIS-chc	wRNN	wDrop3
arrhythmia	0.4337	0.4195	0.4305	0.3752	0.4142	0.4045	0.7984	0.5692	0.5644
audio	0.4248	0.4009	0.4496	0.2911	0.3097	0.3522	0.6393	0.6600	0.5553
autos	0.4137	0.3521	0.4467	0.3657	0.3151	0.3219	0.6261	0.5825	0.5354
breast-cancer	0.2972	0.2846	0.2874	0.3203	0.2860	0.2965	0.7619	0.9464	0.8278
car	0.0995	0.2486	0.0989	0.0302	0.1375	0.1926	0.7074	0.9766	0.9249
card	0.1904	0.1832	0.1809	0.1661	0.2090	0.1690	0.6901	0.9327	0.7765
dermatology	0.0497	0.0333	0.0459	0.0350	0.0454	0.0344	0.8111	0.8666	0.7889
ecoli	0.1673	0.2095	0.1720	0.1673	0.1959	0.1720	0.7618	0.8890	0.7896
gene	0.1468	0.1297	0.1508	0.1123	0.1703	0.1139	0.7457	0.9026	0.6684
german	0.2796	0.2930	0.2822	0.2834	0.2910	0.2840	0.7195	0.8214	0.6627
glass	0.3579	0.3168	0.3607	0.3224	0.2692	0.2879	0.6807	0.6819	0.6226
glass-g2	0.2321	0.1853	0.2222	0.1621	0.2050	0.1755	0.7530	0.8483	0.7268
heart	0.1800	0.1763	0.1711	0.1889	0.2133	0.1830	0.7460	0.9157	0.7585
hepatitis	0.1898	0.1873	0.1782	0.2038	0.2233	0.1796	0.8048	0.8714	0.8257
horse	0.3588	0.3599	0.3643	0.3478	0.3451	0.3577	0.6598	0.7871	0.6782
ionosphere	0.1208	0.0877	0.1305	0.0895	0.0940	0.0980	0.7774	0.8897	0.8563
isolet	0.1338	0.1297	0.1331	0.1288	0.1437	0.1246	0.7451	0.8647	0.5902
letter	0.1432	0.1432	0.1439	0.1176	0.1388	0.1508	0.7090	0.6765	0.6571
liver	0.3890	0.3542	0.3907	0.3316	0.3890	0.3536	0.7068	0.8814	0.6956
lrs	0.1695	0.1597	0.1657	0.1420	0.1748	0.1673	0.7216	0.8732	0.8382
lymph	0.2108	0.1959	0.1946	0.2054	0.2122	0.1919	0.7820	0.8309	0.7186
mfeat-fac	0.0426	0.0393	0.0414	0.0445	0.0545	0.0506	0.7493	0.9273	0.9112
mfeat-fou	0.2185	0.2107	0.2154	0.1810	0.2199	0.2134	0.7443	0.7538	0.7212
mfeat-kar	0.0567	0.0511	0.0559	0.0385	0.0475	0.0449	0.7333	0.8441	0.8003
mfeat-mor	0.2942	0.3091	0.2985	0.2948	0.3477	0.3109	0.7276	0.9446	0.8262
mfeat-pix	0.0324	0.0292	0.0316	0.0338	0.0346	0.0324	0.7459	0.9287	0.8995
mfeat-zer	0.1939	0.1992	0.1971	0.1783	0.2212	0.2062	0.7543	0.8722	0.7681
optdigits	0.0272	0.0237	0.0243	0.0236	0.0204	0.0199	0.7500	0.8919	0.8906
page-blocks	0.0389	0.0394	0.0393	0.0395	0.0399	0.0365	0.7611	0.9823	0.9702
phoneme	0.1211	0.1539	0.1227	0.1486	0.1292	0.1423	0.7577	0.8287	0.7679
pima	0.2583	0.2643	0.2573	0.2643	0.3130	0.2594	0.6797	0.9119	0.7820
primary	0.5841	0.5917	0.5817	0.5610	0.6342	0.6094	0.6280	0.9050	0.8193
promoters	0.2472	0.2000	0.2076	0.2208	0.2340	0.1924	0.8381	0.7351	0.5742
satimage	0.1045	0.0992	0.1018	0.1027	0.0954	0.0940	0.7552	0.8273	0.8757
segment	0.0508	0.0338	0.0533	0.0295	0.0334	0.0334	0.7392	0.8823	0.8467
sick	0.0425	0.0530	0.0444	0.0274	0.0331	0.0351	0.7503	0.9589	0.9335
sonar	0.1721	0.1577	0.1981	0.1808	0.1644	0.1558	0.7643	0.7809	0.6737
soybean	0.0954	0.0720	0.0919	0.0685	0.0794	0.0718	0.7151	0.8789	0.8163
vehicle	0.3156	0.2955	0.3132	0.2721	0.2962	0.2887	0.7272	0.8178	0.6096
vote	0.0722	0.0634	0.0662	0.0506	0.0611	0.0616	0.8006	0.9690	0.9107
vowel	0.0640	0.0776	0.0711	0.0735	0.0800	0.0760	0.6800	0.6123	0.6153
waveform	0.1654	0.1573	0.1639	0.1500	0.1913	0.1720	0.7597	0.9352	0.7195
yeast	0.4328	0.4640	0.4363	0.4191	0.4981	0.4642	0.6899	0.8723	0.6698
zip	0.0380	0.0300	0.0377	0.0357	0.0323	0.0264	0.7488	0.8560	0.8599
zoo	0.0811	0.0830	0.0732	0.0691	0.0671	0.0927	0.7305	0.8182	0.8221
Average	0.1942	0.1900	0.1939	0.1754	0.1936	0.1845	0.7350	0.8445	0.7588

In all the tables the p-values of the corresponding tests are shown. The error measure is $\epsilon = E[\mathbf{1}_{(y \neq f(\mathbf{x}))}]$. As a general rule, we consider a confidence level of 95%. As further information of the relative performance of each pair of algorithms, comparison tables also show a sign test on the win/loss record of the two algorithms across all data sets. If the probability of obtaining the observed results by chance is below 5%, we may conclude that the observed performance is indicative of a general underlying advantage of one of the algorithms with respect to the type of the learning task used in the experiments.[4]

[4]The source code, in C and licensed under the GNU General Public License Version 3 [54], used for all methods as well as the partitions of the data sets, is freely available upon request to the authors.

VI. EXPERIMENTAL RESULTS

As we are proposing a method for constructing ensembles, we must test this method against known algorithms for constructing ensembles. As a first step, we will show the results of the comparison among k-NN ensembles constructed using instance selection and some standard methods. As stated, we have chosen as control methods a k-NN classifier alone, and ensembles constructed using RSM and bagging methods. For C4.5 and SVM, we tested our methodology against ensembles constructed using ADABOOST method. All of the constructed ensembles have a size of 30 classifiers. This number is fairly common in the literature [12], [55], and [56]. Although desirable, fixing the number

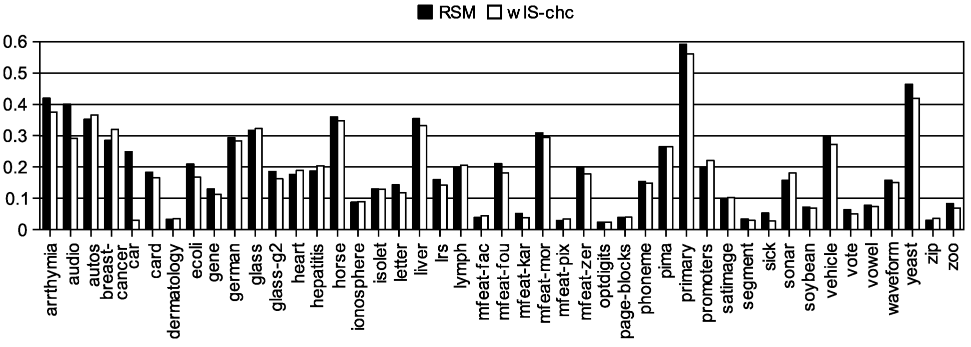

Fig. 2. Summary of test error results for the best standard k-NN method and the best of our proposals for boosting k-NN using instance selection.

TABLE III
COMPARISON OF RESULTS IN TERMS OF GENERALIZATION ERROR FOR THE STANDARD- AND INSTANCE-SELECTION-BASED METHODS FOR k-NN. WIN/DRAW/LOSS RECORD (ROW s) OF THE ALGORITHMS AGAINST EACH OTHER AND p-VALUE OF THE SIGN TEST (ROW p_s), p-VALUE OF THE WILCOXON TEST (ROW p_w). SIGNIFICANT DIFFERENCES ARE MARKED WITH AN ×, 95% CONFIDENCE, OR ✓, 90% CONFIDENCE

		k-NN			Instance selection		
		Standard	RSM	Bagging	wIS-chc	wDrop3	wRNN
Mean all		0.1942	0.1900	0.1939	0.1754	0.1845	0.1995
k-NN	s		34/0/11	26/0/19	32/0/13	29/1/15	22/1/22
	p_s		0.0008✗	0.3713	0.0066✗	0.0488✗	1.0000
	p_w		0.0002✗	0.5090	0.0001✗	0.0180✗	0.7692
k-NN RSM	s			17/0/28	30/0/15	30/0/15	17/0/28
	p_s			0.1352	0.0357✗	0.0357✗	0.1352
	p_w			0.1207	0.0038✗	0.0353✗	0.0773✓
k-NN Bagging	s				32/0/13	26/0/19	22/0/23
	p_s				0.0066✗	0.3713	1.0000
	p_w				0.0003✗	0.0529✓	0.3879
wIS-chc	s					17/0/28	10/0/35
	p_s					0.1352	0.0002✗
	p_w					0.0633✓	0.0002✗
wDrop3	s						13/0/32
	p_s						0.0066✗
	p_w						0.0052✗

Iman–Davenport test: 0.0000

of classifiers in each ensemble taking into account either the problem or the type of base learner is computationally unfeasible. Moreover, it is known [57] that the diversity and the accuracy of the ensemble usually plateau at some size between 10 and 50 members.

A. Using k-NN Classifier

One of the key parameters for any k-NN classifier is which value of k to use. The value of k greatly affects the performance of the algorithm. To avoid any bias produced by a bad choice of k, we have adjusted this value by cross validation for every partition of every data set and for every classifier, so each time a k-NN algorithm is used, the value of k is previously obtained by tenfold cross validation on the training set.

Table II shows the results for the 45 data sets of the three standard methods and the methods using instance selection described above. These results for the best standard algorithm and the best algorithm based on our approach are plotted in Fig. 2. We show the results using the three methods based on instance selection: wRNN, wDrop3, and wIS-chc. For the instance-selection-based ensembles, the reduction achieved is also shown. Table III shows the comparison among the methods. For all the methods, the Iman–Davenport test has a p-value of 0.000,

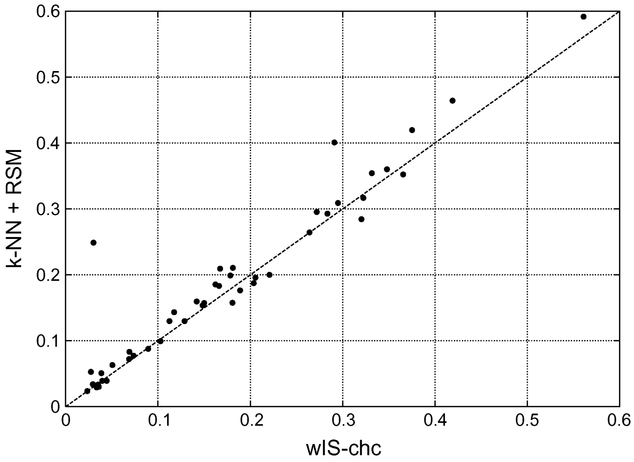

Fig. 3. Comparison of testing error for k-NN with random subspace method and the ensemble constructed using instance selection with wIS-chc algorithm.

showing significant differences among them. For wIS-chc, we used a population of 50 individuals evolved for 100 generations. During initialization, we have a probability of selecting an instance of 0.25. Random mutation was applied with a rate of $P_{\text{mut}} = 0.1$, and RNN mutation was applied with a rate of $P_{\text{RNN}} = 0.005$. Fitness value is given by (5) with $\alpha = 0.75$. The same parameters were used for C4.5 and SVM.

As for the standard methods, we see that Bagging performs just as well as k-NN alone. As we have said, this is a known fact. As k-NN is fairly stable with respect to the resampling of instances, ensembling bagged classifiers do not improve its performance. The best performing standard method is RSM, which is able to outperform the other two standard algorithms significantly. Similar behavior of RSM and k-NN has been reported before [24].

Taking into account the proposed methodology, we can see that wRNN is able to achieve the largest reduction, with an average reduction of 84.45%. However, this reduction comes at the cost of a worse testing error. In fact (see Table III), this method performs as well as standard methods. wDrop3 and wIS-chc achieved a lower reduction, on average 75.88% and 73.50%, respectively. On the other hand, their performance is better. Wilcoxon test finds significant differences with standard methods at a confidence level of 95%. Specially marked is the good behavior of wIS-chc. As genetic algorithms are usually better than standard instance selection algorithms [14], it was an

TABLE IV
SUMMARY OF TEST ERROR RESULTS FOR THE STANDARD ADABOOST AND INSTANCE BASED METHODS FOR SVM, USING A 5 × 2 CV SETUP

Dataset	AdaBoost		wIS-chc			wDrop3		
	Test error	#SV	Test error	Reduction	#SV	Test error	Reduction	#SV
arrhythmia	0.3235	133.4	0.3221	0.5057	57.5	0.3173	0.8389	34.8
audio	0.2655	79.9	0.2239	0.6128	42.0	0.3027	0.7841	22.8
autos	0.3169	58.7	0.2935	0.6071	37.7	0.3756	0.7607	22.4
breast-cancer	0.2972	124.9	0.3154	0.8057	14.2	0.2860	0.8011	17.0
car	0.0925	482.0	0.0837	0.6370	87.5	0.0979	0.7468	74.8
card	0.1667	286.1	0.1582	0.7424	39.6	0.1501	0.7808	34.5
dermatology	0.0295	47.9	0.0279	0.7442	28.6	0.0268	0.8408	20.7
ecoli	0.1673	106.9	0.1423	0.7587	23.9	0.1518	0.8046	19.4
gene	0.1361	646.8	0.0975	0.6632	144.9	0.1033	0.7514	164.2
german	0.2724	444.2	0.2622	0.6808	47.0	0.2598	0.7805	53.0
glass	0.3953	89.2	0.3327	0.7399	24.0	0.3869	0.7603	22.1
glass-g2	0.3057	68.8	0.2234	0.8302	8.5	0.3068	0.8056	11.6
heart	0.1978	105.1	0.1933	0.7743	11.8	0.1845	0.7911	12.8
hepatitis	0.2321	25.1	0.1999	0.7680	9.7	0.1949	0.8246	7.0
horse	0.3720	101.8	0.3357	0.6487	45.9	0.3418	0.8021	26.3
ionosphere	0.1629	47.5	0.1344	0.7065	18.3	0.1413	0.8141	14.4
isolet	0.0845	2785.5	0.0717	0.6847	960.9	0.0778	0.7520	755.9
letter	0.1937	1976.6	0.1651	0.6351	663.3	0.1908	0.6990	548.2
liver	0.3223	148.2	0.3130	0.8059	24.7	0.3176	0.7562	30.7
lrs	0.1168	70.7	0.1130	0.6503	52.8	0.1300	0.7658	30.7
lymph	0.1554	26.0	0.1730	0.7312	15.3	0.1851	0.7841	11.0
mfeat-fac	0.0244	259.2	0.0266	0.5868	246.6	0.0342	0.7392	156.4
mfeat-fou	0.1969	726.3	0.1691	0.5731	330.6	0.1828	0.7270	212.3
mfeat-kar	0.0404	377.8	0.0376	0.5753	304.1	0.0410	0.7003	215.7
mfeat-mor	0.2607	588.0	0.2572	0.6441	215.0	0.2669	0.7601	144.2
mfeat-pix	0.0247	331.9	0.0251	0.5892	298.7	0.0307	0.7401	188.6
mfeat-zer	0.1926	505.9	0.1839	0.6015	260.6	0.1798	0.7388	170.7
optdigits	0.0193	484.2	0.0190	0.5999	465.9	0.0232	0.7456	296.1
page-blocks	0.0355	981.7	0.0341	0.6458	124.4	0.0389	0.7430	90.3
phoneme	0.2320	2341.8	0.2238	0.5859	630.9	0.2385	0.7487	383.7
pima	0.2380	311.6	0.2357	0.7700	52.4	0.2370	0.7535	55.0
primary	0.6171	155.2	0.5858	0.6074	64.7	0.5705	0.7578	37.6
promoters	0.2057	30.3	0.1981	0.7563	12.3	0.2057	0.7879	10.7
satimage	0.1371	1506.1	0.1337	0.6631	352.0	0.1350	0.7543	256.0
segment	0.0477	405.4	0.0415	0.6439	133.7	0.0448	0.7184	107.0
sick	0.0312	1298.0	0.0246	0.6006	106.9	0.0320	0.7279	72.7
sonar	0.2615	31.6	0.2423	0.6842	19.7	0.2404	0.7354	15.8
soybean	0.0750	255.1	0.0656	0.6805	93.9	0.0671	0.6991	85.2
vehicle	0.2281	272.6	0.2140	0.7236	72.5	0.2473	0.7312	71.2
vote	0.0680	53.7	0.0501	0.7456	10.8	0.0497	0.7496	11.2
vowel	0.1796	387.4	0.1727	0.6967	119.0	0.2176	0.6312	140.2
waveform	0.1417	1496.4	0.1390	0.6859	281.9	0.1379	0.7558	218.6
yeast	0.4205	570.8	0.4131	0.6371	215.8	0.4196	0.7614	141.5
zip	0.0153	703.9	0.0171	0.6522	609.2	0.0208	0.7466	443.2
zoo	0.0633	21.8	0.0633	0.8313	8.2	0.0888	0.8067	9.1
Average	0.1858	487.8	0.1723	0.6781	164.3	0.1840	0.7601	121.5

expected result that wIS-chc was the best performing method. Additionally, wIS-chc algorithm has the advantage of also evolving the subspace. That feature is relevant due to the fact that experiments performed using the same algorithm wIS-chc, not reported here, but without evolving the subspaces obtained worse results than wIS-chc when the subspaces are evolved.

Another advantage of using wIS-chc algorithm is the possibility of weighting the two terms of the fitness functions: training error and reduction. As we have said, in the experiments reported here, the two terms had a relative weight of 75% for training error and 25% for reduction. However, depending on the specific needs of our problem, we can modify this weight to put more pressure either on training error or on storage reduction. The results in Tables II and III are illustrated in Fig. 3. The figure represents for each point the testing error of wIS-chc and the best one among the standard methods, RSM. Points above the diagonal line show a better performance of our method, and points below the diagonal line show a better performance of RSM. We can see that there are more points above the diagonal, and also that the separation of these points from the diagonal is larger.

As a conclusion, we can say that both wDrop3 and wIS-chc methods are able to significantly outperform standard methods. Moreover, this improvement in the testing error is achieved together with a very marked reduction in the storage of instances of 73.50% for wIS-chc and 75.88% for wDrop3. The good behavior of wDrop3 has the advantage of needing a very simple modification of the standard algorithm, allowing the use of the method for researchers not familiar with CHC algorithm. On the other hand, wRNN is useful for reducing the number of instances achieving a performance similar to RSM with much less complexity.

TABLE V
SUMMARY OF TEST ERROR RESULTS FOR THE STANDARD-ADABOOST- AND INSTANCE-BASED METHODS FOR A C4.5 TREE, USING A 5 × 2 CV SETUP

Dataset	ADABOOST		wIS-chc			wDrop3		
	Test error	#Nodes	Test error	Reduction	#Nodes	Test error	Reduction	#Nodes
arrhythmia	0.2995	61.4	0.2726	0.5292	22.3	0.3208	0.8440	12.4
audio	0.2699	36.6	0.2787	0.6578	22.8	0.3257	0.7644	14.3
autos	0.3149	35.0	0.2682	0.6064	23.6	0.3979	0.7468	12.6
breast-cancer	0.2853	36.5	0.3371	0.7364	17.3	0.2762	0.8045	8.6
car	0.0421	88.1	0.0337	0.6664	56.2	0.0485	0.7447	46.3
card	0.1414	62.9	0.1484	0.6655	40.1	0.1440	0.7775	21.9
dermatology	0.0246	26.7	0.0284	0.7222	15.8	0.0344	0.8427	10.6
ecoli	0.1929	25.4	0.1649	0.7091	16.8	0.1732	0.8103	9.8
gene	0.0770	245.2	0.0729	0.5593	168.4	0.0846	0.7514	95.1
german	0.2596	118.4	0.2602	0.6413	73.1	0.2654	0.7764	42.5
glass	0.3243	25.5	0.2963	0.6324	20.1	0.3654	0.7601	11.8
glass-g2	0.2088	12.7	0.1596	0.7357	8.7	0.2294	0.8099	4.5
heart	0.2096	26.1	0.1904	0.6860	16.2	0.1822	0.7943	7.4
hepatitis	0.2002	12.5	0.1987	0.7402	8.8	0.1898	0.8239	4.4
horse	0.3440	49.7	0.3182	0.5154	33.4	0.3407	0.8052	12.7
ionosphere	0.0906	14.8	0.0803	0.6800	12.5	0.0883	0.8266	6.9
isolet	0.1572	971.3	0.1850	0.5610	541.8	0.2165	0.7532	304.7
letter	0.1598	643.0	0.1750	0.5232	462.0	0.2202	0.6969	293.6
liver	0.3264	25.7	0.3066	0.7366	13.4	0.3420	0.7595	6.8
lrs	0.1514	28.7	0.1428	0.5567	24.2	0.1579	0.7571	12.8
lymph	0.1851	18.6	0.1770	0.7395	10.2	0.1838	0.7743	5.8
mfeat-fac	0.0453	148.5	0.0503	0.5679	72.6	0.0723	0.7555	40.7
mfeat-fou	0.1967	195.1	0.1920	0.5361	110.3	0.2187	0.7302	63.7
mfeat-kar	0.0777	189.4	0.0704	0.5368	103.2	0.1123	0.7159	63.9
mfeat-mor	0.2809	133.3	0.2854	0.7116	63.5	0.2911	0.7561	46.8
mfeat-pix	0.0471	159.6	0.0536	0.5659	82.0	0.0804	0.7501	47.0
mfeat-zer	0.2370	193.0	0.2393	0.5363	126.1	0.2532	0.7440	69.5
optdigits	0.0290	405.3	0.0282	0.5577	215.0	0.0524	0.7481	122.6
page-blocks	0.0300	74.4	0.0288	0.6098	52.3	0.0337	0.7393	36.2
phoneme	0.1300	182.1	0.1362	0.6001	61.1	0.1535	0.7383	39.5
pima	0.2555	45.8	0.2534	0.7043	20.9	0.2448	0.7500	8.8
primary	0.5758	68.3	0.5646	0.6638	29.4	0.6147	0.7559	19.5
promoters	0.2359	11.9	0.1623	0.7560	5.6	0.2226	0.7795	3.5
satimage	0.1041	297.8	0.1012	0.5474	233.9	0.1172	0.7529	127.6
segment	0.0308	66.9	0.0272	0.5852	54.7	0.0488	0.7143	37.9
sick	0.0165	45.4	0.0150	0.6044	32.3	0.0191	0.7245	18.3
sonar	0.2327	13.4	0.2106	0.4943	9.1	0.2414	0.7309	6.4
soybean	0.0674	63.2	0.0682	0.5781	48.7	0.0814	0.6926	40.8
vehicle	0.2653	92.6	0.2522	0.5454	55.4	0.2830	0.7298	35.5
vote	0.0510	14.6	0.0570	0.7438	10.0	0.0570	0.7552	6.0
vowel	0.1697	112.6	0.1574	0.5193	93.0	0.2586	0.6299	72.0
waveform	0.1713	283.8	0.1575	0.5565	191.4	0.1660	0.7487	108.4
yeast	0.4165	213.3	0.4202	0.5315	114.1	0.4090	0.7664	57.2
zip	0.0528	547.8	0.0616	0.5940	336.2	0.0752	0.7460	211.2
zoo	0.0831	10.1	0.0830	0.4885	9.9	0.1144	0.7813	7.7
Average	0.1793	136.3	0.1727	0.6163	83.22	0.1957	0.7591	49.70

B. Using C4.5 and SVM Classifiers

As we have stated in Section IV, different classification methods may benefit from instance selection boosting as well as k-NN. Those methods whose complexity depends, at least partially, on the number of training instances can be used with our methodology as a simpler ensemble might be obtained. We have selected two of the most used methods for classification: C4.5 algorithm and SVMs. So, in this section, we use C4.5 and SVM as weak classifiers in a boosting framework, and compare their performance with our proposal. In a first set of experiments, we observed that, for the same problem, randomly reducing the number of instances yielded an approximately linear reduction in the complexity of the classifiers. Thus, our method based on instance selection may be useful for reducing complexity as well as improving performance. We measure the complexity of C4.5 classifier as the number of nodes of the tree and the complexity of the SVM as the number of SVs.

As stated in Section IV, we have two different approaches. We can use wIS-chc algorithm and instance selection algorithm based on NN philosophy. The latter is less likely to produce good results as the selection is focused on improving NN classification. Nevertheless, the method is worth testing.

Tables IV and V show the results for SVMs and C4.5, respectively. The tables show the results for the two approaches. These testing error results for the best standard algorithm and the best algorithm based on our approach are plotted in Fig. 4. The tables show the average number of nodes/SVs for the members of the ensemble. We have used wDrop3 as standard NN instance selection algorithm as it performed better than RNN in the previous section. The first noticeable fact is the marked reduction in complexity accomplished by both methods. For SVMs, the number of SVs is reduced to a third for wIS-chc and to a fourth for wDrop3. For C4.5, the reduction is almost a half for wIS-chc and almost a third for wDrop3. This is a very marked reduction,

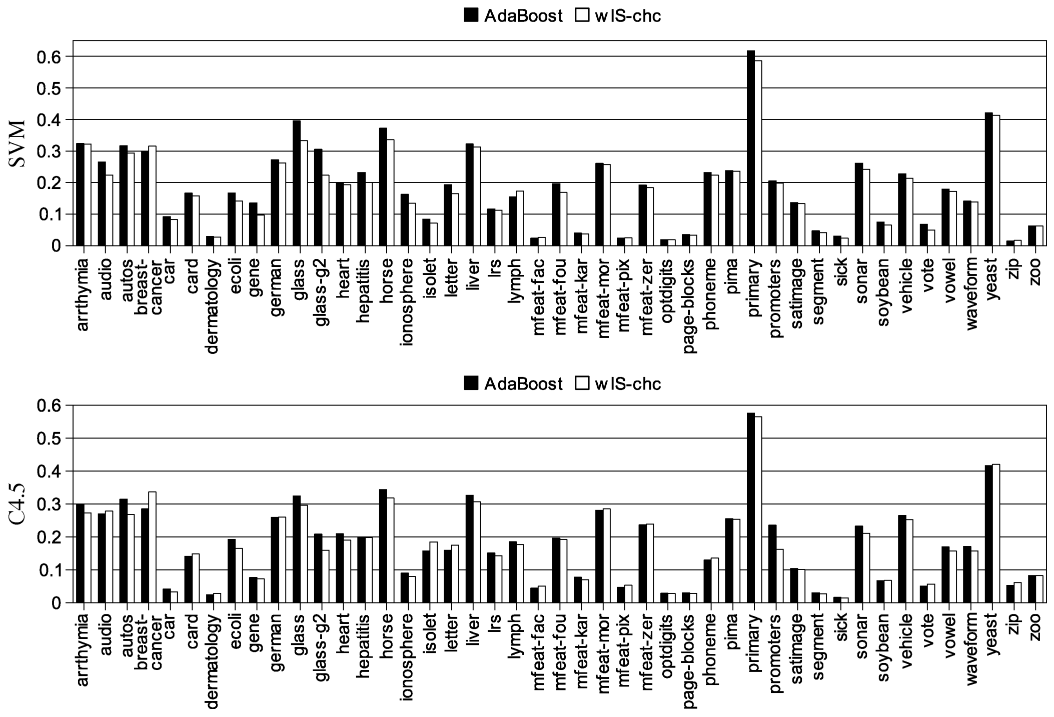

Fig. 4. Summary of test error results for the best standard method and the best of our proposals for boosting C4.5 and SVM using instance selection.

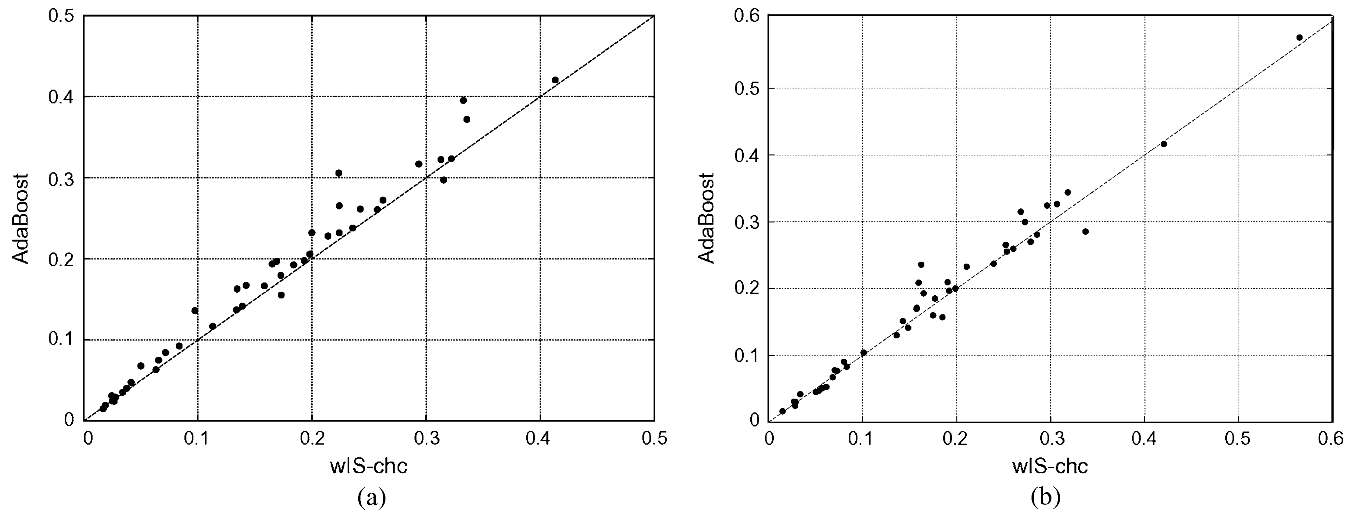

Fig. 5. Comparison of testing error for (a) SVM and (b) C4.5 using AdaBoost and wIS-chc.

especially if we take into account that recall time for both classifiers depends on the number of SVs/nodes.

Tables VI and VII show the comparison of the three methods and also bagging and RSM.[5] For SVM, there is a clear improvement over AdaBoost, RSM, and bagging of wIS-chc. The comparison using both Wilcoxon and sign tests is significant at a confidence level of 99%. Moreover, this improvement is achieved together with a reduction in the number of SVs of 66.31%. The results using wDrop3 are also interesting. The algorithm performs better than AdaBoost, win/loss record of 27/18, although the difference is not significant. However, it is no better than RSM and significantly worse than bagging.

[5]The results of RSM and bagging are not shown in Tables IV and V for brevity's sake as they perform worse than AdaBoost.

We believe that as SVM is also based on prototypes as k-NN, methods developed for the latter may be also useful for the former. For this method, the reduction in the number of SVs is greater, 75.09%, so if we are more interested in reduction than in performance, this method can be used instead of wIS-chc. Obviously, this method is also less computationally expensive.

For C4.5, wDrop3 performs worse than standard AdaBoost, RSM, and bagging. However, wIS-chc still performs significantly better than the three standard algorithms, although the differences with AdaBoost are less marked than the case for SVM. The results in Tables IV–VII are illustrated in Fig. 5. We can see that, as it was the case for k-NN, there are more points above the diagonal, and also that the separation of these points from the diagonal is greater.

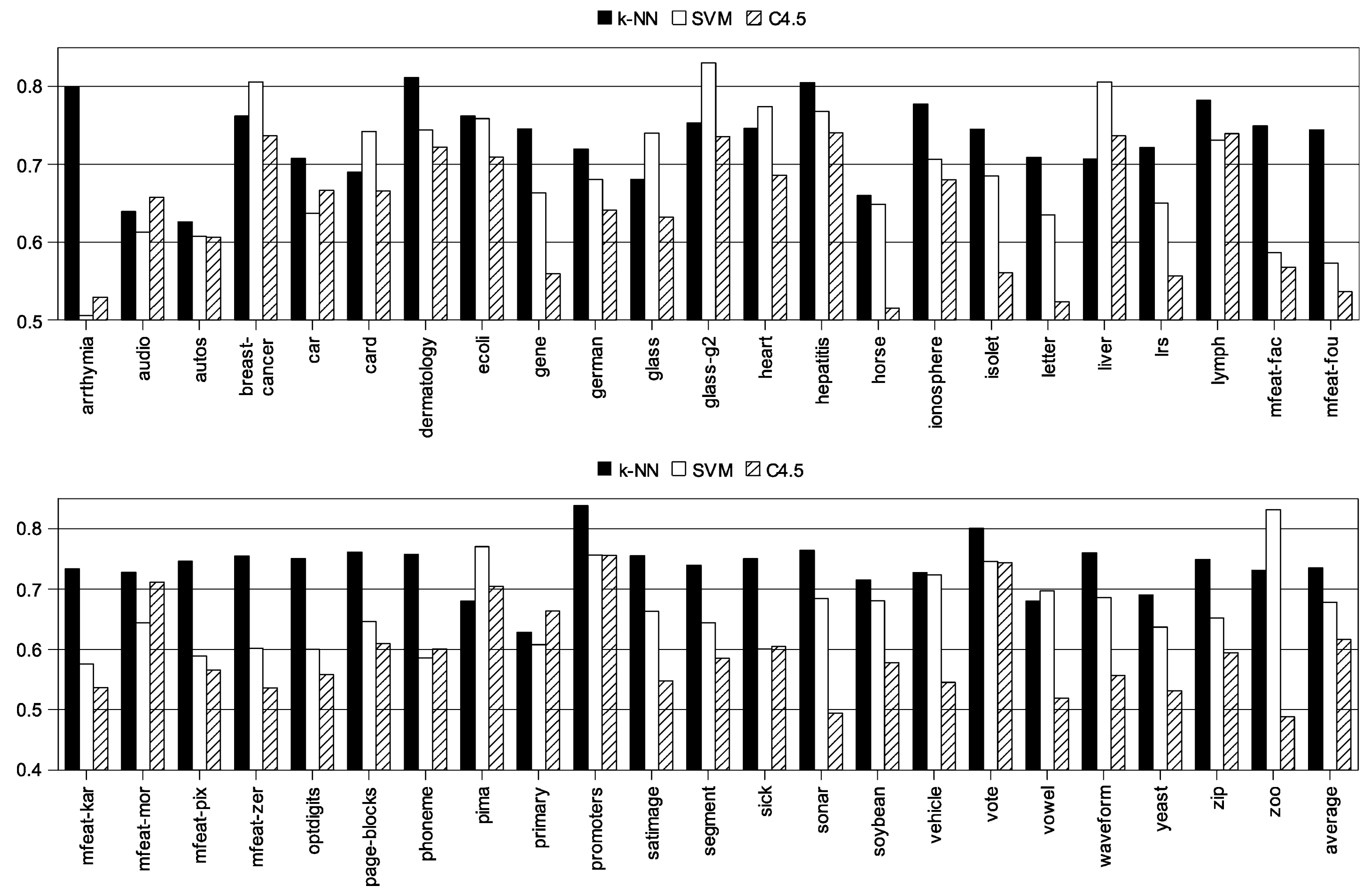

Fig. 6. Reduction achieved by wIS-chc for the three base learners. For k-NN, reduction is measured as the percentage of instances removed from the training set; for SVM, as the reduction in the number of SVs with respect to the ensemble constructed using AdaBoost; and for C4.5, as the reduction in the number of nodes of the decision trees with respect to the ensemble constructed using AdaBoost.

TABLE VI
Comparison of Results in Terms of Generalization Error Using SVM. Win/Draw/Loss Record (Row s) of the Algorithms Against Each Other and p-Value of the Sign Test (Row p_s), p-Value of the Wilcoxon Test (Row p_w). Significant Differences Are Marked With an $\mathbf{x}$, 95% Confidence, or ✓, 90% Confidence

					Instance selection	
		AdaBoost	RSM	Bagging	wIS-chc	wDrop3
Mean all		0.1858	0.2036	0.1855	0.1723	0.1840
AdaBoost	s		19/0/26	26/0/19	39/0/6	27/0/18
	p_s		0.3713	0.3713	0.0000✗	0.2327
	p_w		0.2861	0.2711	0.0000✗	0.2337
RSM	s			30/0/15	29/1/15	23/1/21
	p_s			0.0357✗	0.0488✗	0.8804
	p_w			0.0137✗	0.0022✗	0.1599
Bagging	s				29/1/15	14/0/31
	p_s				0.0488✗	0.0161✗
	p_w				0.0077✗	0.0982✓
wIS-chc	s					12/0/33
	p_s					0.0025✗
	p_w					0.0003✗

Iman–Davenport test: 0.0000

TABLE VII
Comparison of Results in Terms of Generalization Error Using a C4.5 Tree. Win/Draw/Loss Record (Row s) of the Algorithms Against Each Other and p-Value of the Sign Test (Row p_s), p-Value of the Wilcoxon Test (Row p_w). Significant Differences Are Marked With an $\mathbf{x}$, 95% Confidence, or ✓, 90% Confidence

					Instance selection	
		AdaBoost	RSM	Bagging	wIS-chc	wDrop3
Mean all		0.1793	0.1916	0.1941	0.1727	0.1957
AdaBoost	s		20/1/24	17/0/28	29/0/16	11/0/34
	p_s		0.6516	0.1352	0.0725✓	0.0008✗
	p_w		0.0792✓	0.0071✗	0.0251✗	0.0001✗
RSM	s			21/0/24	33/0/12	14/0/31
	p_s			0.7660	0.0025✗	0.0161✗
	p_w			0.8170	0.0002✗	0.0773✓
Bagging	s				34/0/11	14/0/31
	p_s				0.0008✗	0.0161✗
	p_w				0.0001✗	0.0394✗
wIS-chc	s					6/0/39
	p_s					0.0000✗
	p_w					0.0000✗

Iman–Davenport test: 0.0000

As a summary, the reduction achieved by the best method of our proposal wIS-chc is illustrated in Fig. 6 for the three base learners. As stated, for k-NN, reduction is measured as the percentage of instances removed from the training set; for SVM, as the reduction in the number of SVs with respect to the ensemble constructed using AdaBoost; and for C4.5, as the reduction in the number of nodes of the decision trees with respect to the ensemble constructed using AdaBoost. The figure shows the large reduction achieved for most data sets.

In the reported experiments, we have shown that our method is able to improve the testing error of standard methods while also reducing the complexity of the ensemble. However, the proposed method has the disadvantage of making the learning

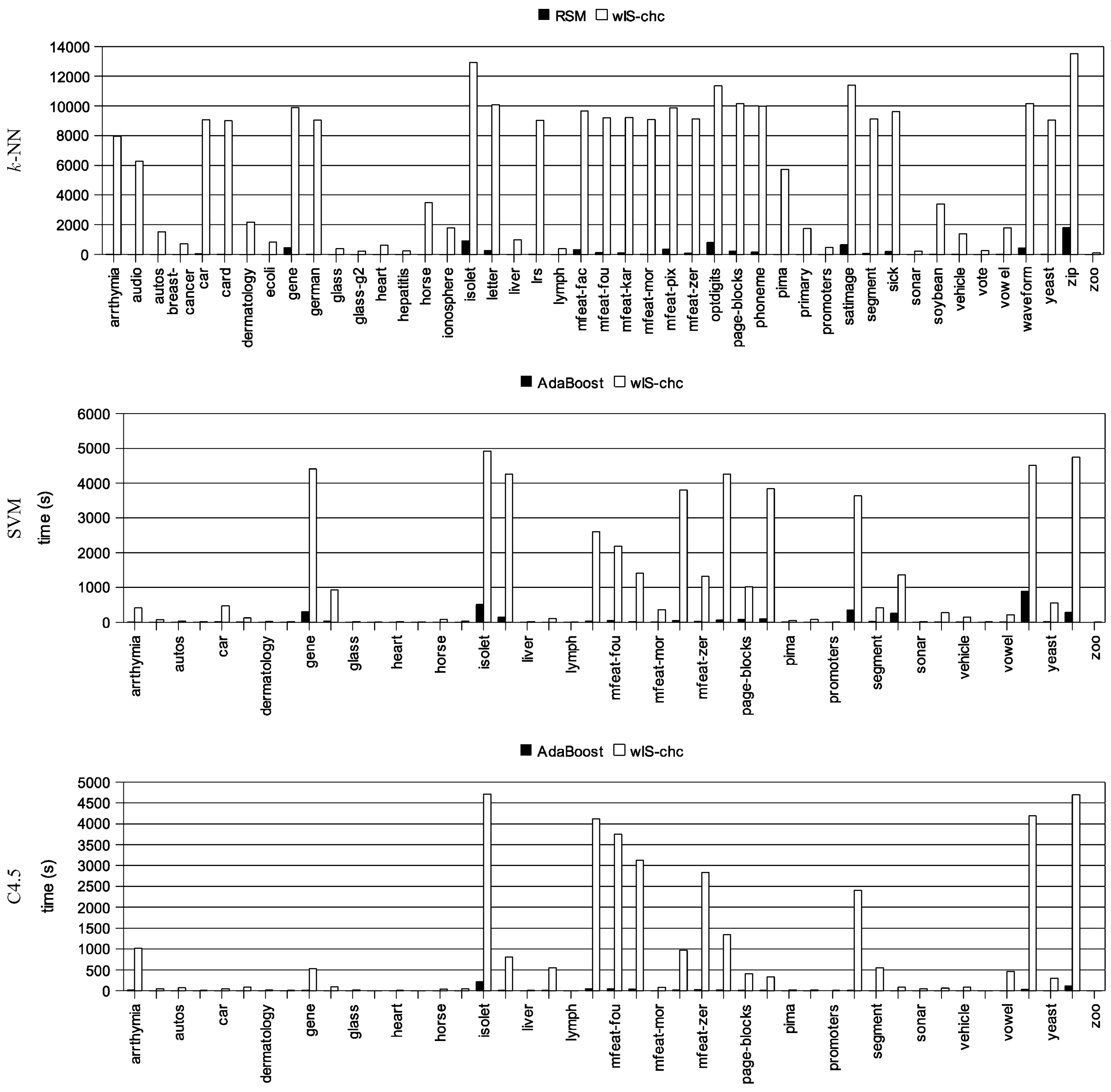

Fig. 7. Summary of average time spent by wIS-chc and the standard ensemble methods for k-NN, C4.5 and SVM learners.

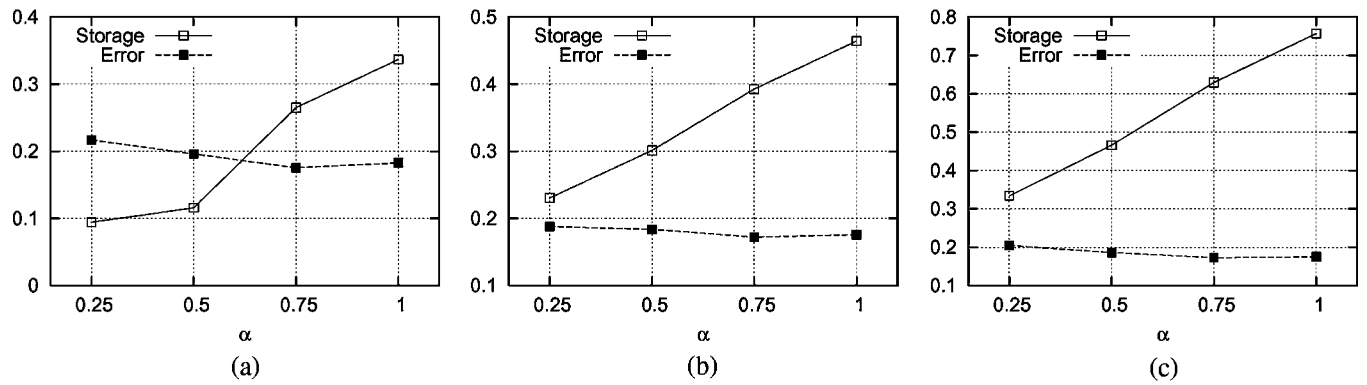

Fig. 8. Behavior in terms of testing error and storage in function of the value of α. (a) k-NN; (b) SVM; (c) C4.5.

TABLE VIII
COMPARISON OF RESULTS IN TERMS OF GENERALIZATION ERROR FOR THE CONTROL EXPERIMENTS: INSTANCE SELECTION WITH A SINGLE CLASSIFIER (COLUMN INS. SEL.), EVOLUTION OF THE SUBSPACE WITHOUT INSTANCE SELECTION (SUBSPACE), AND INSTANCE SELECTION WITH CONSIDERING THE DISTRIBUTION OF WEIGHTS GIVEN BY BOOSTING (INS. SEL. BAGGING). WIN/DRAW/LOSS RECORD (ROW s) OF THE ALGORITHMS AGAINST EACH OTHER AND p-VALUE OF THE SIGN TEST (ROW p_s), p-VALUE OF THE WILCOXON TEST (ROW p_w). SIGNIFICANT DIFFERENCES ARE MARKED WITH AN $\mathbf{x}$, 95% CONFIDENCE, OR ✓, 90% CONFIDENCE

	k-NN				C4.5				SVM			
	wIS-chc	Ins. sel.	Subspace	Ins. sel. Bagging	wIS-chc	Ins. sel.	Subspace	Ins. sel. Bagging	wIS-chc	Ins. sel.	Subspace	Ins. sel. Bagging
Mean all	0.1754	0.2365	0.1827	0.1923	0.1727	0.2481	0.1828	0.1990	0.1723	0.2128	0.1797	0.1829
k-NN		1/0/44	20/0/25	14/0/31								
wIS-chc		0.0000✗	0.5515	0.0161✗								
		0.0000✗	0.0594✓	0.0024✗								
C4.5						3/0/42	16/0/29	6/0/39				
wIS-chc						0.0000✗	0.0725✓	0.0000✗				
						0.0000✗	0.0113✗	0.0000✗				
SVM										6/0/39	15/0/30	15/0/30
wIS-chc										0.0000✗	0.0357✗	0.0357✗
										0.0000✗	0.0128✗	0.0070✗

TABLE IX
SUMMARY OF TEST ERROR RESULTS FOR STANDARD- AND INSTANCE-BASED METHODS FOR k-NN, C4.5, AND SVM AS BASE LEARNER FOR A NOISE LEVEL OF 5%, USING A 5 × 2 CV SETUP

Dataset	k-NN			C4.5					SVM				
	RSM	wIS-chc		ADABOOST		wIS-chc			ADABOOST		wIS-chc		
	Test error	Test error	Reduction	Test error	#Nodes	Test error	Reduction	#Nodes	Test error	#SV	Test error	Reduction	#SV
arrhythmia	0.4071	0.3912	0.7722	0.3013	65.5	0.2819	0.6247	35.3	0.3491	136.8	0.3301	0.5805	92.5
audio	0.4009	0.3310	0.7072	0.2779	37.8	0.2761	0.6645	22.4	0.3141	100.6	0.2531	0.6744	35.8
autos	0.3472	0.3766	0.7205	0.3207	36.7	0.2819	0.6043	24.4	0.3316	65.6	0.3033	0.6107	37.6
breast-c.	0.2762	0.3091	0.7716	0.2867	39.8	0.3489	0.7972	15.1	0.2930	126.4	0.3245	0.8085	15.8
car	0.2269	0.0633	0.7190	0.0468	139.1	0.0539	0.7434	73.0	0.1154	542.9	0.1060	0.7408	112.0
card	0.1861	0.1945	0.7358	0.1493	75.4	0.1768	0.7355	39.2	0.1942	301.9	0.1635	0.7498	42.6
dermatol.	0.0388	0.0312	0.7533	0.0322	33.9	0.0355	0.6962	21.0	0.0738	72.3	0.0514	0.6643	45.3
ecoli	0.1988	0.1613	0.7606	0.1917	31.2	0.1851	0.6851	20.6	0.1821	113.5	0.1512	0.7515	26.4
gene	0.1245	0.1085	0.7383	0.0988	304.9	0.0929	0.6220	199.0	0.2252	1031.2	0.1106	0.7362	184.2
german	0.2842	0.2818	0.7313	0.2732	125.8	0.2784	0.6420	78.0	0.2786	459.5	0.2740	0.7611	61.7
glass	0.3234	0.3392	0.7149	0.3336	26.5	0.3028	0.6398	20.5	0.3785	91.7	0.3766	0.7377	24.3
glass-g2	0.1915	0.2063	0.7668	0.1988	11.1	0.1792	0.7397	8.5	0.3229	69.1	0.2419	0.8293	8.6
heart	0.1763	0.2015	0.7469	0.2096	29.7	0.2141	0.6716	18.8	0.2067	109.9	0.2126	0.7823	13.7
hepatitis	0.1718	0.1743	0.8029	0.2182	13.7	0.2117	0.7310	10.1	0.2349	38.6	0.2131	0.7769	9.5
horse	0.3511	0.3544	0.7475	0.3451	52.6	0.3286	0.6257	35.3	0.4137	122.6	0.3599	0.6514	46.9
ionosph.	0.1516	0.1442	0.7628	0.1145	19.5	0.1310	0.7543	13.7	0.2211	127.4	0.1875	0.7482	19.6
isolet	0.1202	0.1185	0.7419	0.1743	1045.3	0.2005	0.6924	395.0	0.1730	3024.9	0.0851	0.6828	978.7
letter	0.1550	0.1418	0.6972	0.1770	700.4	0.2043	0.6840	317.8	0.3018	2146.4	0.1858	0.7105	580.4
liver	0.3658	0.3432	0.7509	0.3310	24.7	0.3298	0.7412	13.7	0.3119	153.5	0.3240	0.8053	24.8
lrs	0.1578	0.1481	0.7575	0.1420	39.3	0.1492	0.6521	26.1	0.1747	131.5	0.1360	0.6347	72.7
lymph	0.1838	0.1703	0.7768	0.2041	19.3	0.2068	0.7930	10.2	0.2311	35.2	0.1905	0.7046	17.8
mfeat-fac	0.0445	0.0445	0.7554	0.0578	155.0	0.0645	0.6343	77.5	0.0520	375.8	0.0521	0.6222	268.3
mfeat-fou	0.2121	0.1928	0.7301	0.2077	224.0	0.2081	0.6024	123.0	0.2648	759.8	0.1983	0.6123	318.1
mfeat-kar	0.0569	0.0514	0.7245	0.0882	221.3	0.0954	0.6830	103.8	0.0830	442.2	0.0685	0.6107	296.5
mfeat-mor	0.3086	0.2936	0.7541	0.2861	190.6	0.2897	0.7200	70.9	0.2631	667.5	0.2635	0.7570	166.8
mfeat-pix	0.0325	0.0325	0.7550	0.0608	195.3	0.0736	0.6524	77.4	0.0566	458.2	0.0487	0.6272	289.6
mfeat-zer	0.2043	0.1882	0.7418	0.2421	223.6	0.2490	0.6956	106.2	0.2602	713.8	0.1893	0.6890	223.1
optdigits	0.0235	0.0242	0.7535	0.0376	492.8	0.0515	0.7238	204.7	0.1367	1672.5	0.0360	0.7039	434.4
page-bl.	0.0395	0.0398	0.7641	0.0284	244.7	0.0320	0.7351	65.2	0.0501	1652.8	0.0431	0.7582	153.5
phoneme	0.1561	0.1452	0.7518	0.1410	190.6	0.1423	0.7602	45.2	0.2364	2309.4	0.2268	0.7679	391.0
pima	0.2617	0.2604	0.7364	0.2581	49.2	0.2651	0.7089	20.6	0.2375	321.3	0.2453	0.7759	55.0
primary	0.5823	0.5652	0.7382	0.5788	69.0	0.5681	0.6671	29.6	0.6295	156.2	0.5988	0.5964	66.7
promoters	0.2547	0.2208	0.7704	0.2849	12.6	0.2472	0.7431	6.0	0.2453	30.6	0.2472	0.7353	13.4
satimage	0.0990	0.1103	0.7625	0.1060	458.3	0.1099	0.6363	262.0	0.1489	1859.5	0.1377	0.7554	341.4
segment	0.0450	0.0501	0.7404	0.0348	126.2	0.0419	0.6924	59.7	0.0557	708.8	0.0421	0.7478	125.0
sick	0.0506	0.0304	0.7542	0.0183	171.9	0.0202	0.7221	52.0	0.0598	1604.6	0.0401	0.7715	112.9
sonar	0.1712	0.1808	0.7611	0.2673	14.3	0.2366	0.6502	10.1	0.3010	39.4	0.2798	0.6733	21.7
soybean	0.0823	0.0700	0.7080	0.0767	79.2	0.0796	0.6601	49.5	0.1400	268.9	0.0957	0.6625	105.2
vehicle	0.2974	0.2910	0.7212	0.2662	105.3	0.2631	0.6214	66.5	0.2345	293.4	0.2187	0.7312	74.7
vote	0.0735	0.0579	0.7498	0.0533	24.9	0.0653	0.7282	15.6	0.0666	160.3	0.0492	0.7644	14.3
vowel	0.1063	0.1319	0.6696	0.1988	123.7	0.1697	0.6348	89.5	0.2887	424.9	0.2160	0.7012	122.6
wavef.	0.1614	0.1500	0.7476	0.1743	322.0	0.1695	0.6315	210.5	0.1461	1827.1	0.1430	0.7539	347.2
yeast	0.4615	0.4152	0.7504	0.4206	231.3	0.4117	0.6517	109.8	0.4239	580.5	0.4174	0.7495	156.0
zip	0.0343	0.0418	0.7481	0.0619	754.6	0.0763	0.6619	333.6	0.0830	2861.1	0.0330	0.7384	534.8
zoo	0.0812	0.0948	0.8141	0.1049	10.9	0.0909	0.7703	9.2	0.0753	29.2	0.0811	0.8635	6.7
Average	0.1929	0.1838	0.7462	0.1885	168.08	0.1887	0.6873	79.91	0.2192	649.31	0.1900	0.7180	157.55

process more time consuming. As learning is usually performed offline, the added training time is not a critical issue. In any case, it is interesting to show the behavior of our proposal in terms of training time to get a clearer idea of the introduced additional cost. Fig. 7 shows the average execution time, in seconds, for our best algorithm wIS-chc and the corresponding best performing standard algorithm. For k-NN, the proposed algorithm has a higher computational cost than for C4.5 and SVM, as the instance selection algorithm needed more time to converge. For SVM and C4.5, the training time of our proposal is also clearly longer than the standard ADABOOST. Nevertheless, that cost is within reasonable limits, less than 5000 s for the worst case for C4.5 and SVM and less than 14 000 s for k-NN, and it is compensated by the reduced complexity in terms of testing time and storage requirements.

C. Effect of the Value of α

In the experiments reported previously, we have chosen a value of $\alpha = 0.75$. A sensible question might be: which is the optimal value of α? However, there is not a value of α that we can consider optimal. It depends on the necessities of the user. If we are more interested in reducing complexity, we can use a smaller value, and if we are more interested in performance, a larger value must be used. In fact, we consider this parameter an advantage of our method, as it allows the user to guide the learning process depending on the desired features of the ensemble.

In this section, we study the effect of α in testing error and storage. We have performed the same experiments as in the previous section using four different values of α, $\alpha = \{0.25, 0.5, 0.75, 1\}$. Fig. 8 shows the average testing error and complexity of the resulting ensembles for all the data sets. The complexity is measured as the number of instances for k-NN, the number of SVs for SVM, and the number of nodes of the trees for C4.5. For C4.5 and SVM, the values shown represent the ratio between the complexity using our method and the complexity constructing the ensemble using ADABOOST.

Fig. 8 shows the expected behavior. Reducing the value of α enforces complexity reduction, with a larger error. Increasing the value of α obtains better error but less reduction. However, a value of 1 achieved a worse error than a value of 0.75. This behavior might be provoked by overlearning. As a summary, $\alpha = 0.75$ shows a good compromise between reduction and accuracy. However, small changes in the value have little effect on the behavior of the algorithm.

D. Validation of the Methodology

In the previous sections, we have shown that our model is able to outperform standard ensemble methods with the additional advantage of constructing smaller ensembles. However, it is necessary to show that the good behavior is due to the proposed design for the construction of the ensemble. Thus, in this section, we perform several additional experiments to validate the model.

In the first set of experiments, we compare the obtained results with the application of instance selection for obtaining just a single classifier instead of the ensemble. This experiment is aimed at testing whether the ensemble constructed by instance selection is better than instance selection alone. The comparison between instance selection applied to a single classifier and our method wIS-chc is shown in Table VIII in the column "Ins. sel." For the three classifiers used, our method is able to beat the single classifier for almost all data sets, with differences that are significant at a confidence level of 99%. This experiment shows that using only instance selection is not enough to match our method for constructing the ensemble.

However, still there are other aspects of our method that must be checked. The genetic algorithm applied for instance selection allows selection of instances and inputs, as is common in modern instance selection algorithms [58], [59]. However, we must check that the selection of the subspace is not the reason for the good results of the algorithm. Thus, we have performed an experiment where the same algorithm is performed but without instance selection. That is, we evolve the subspace with the objective of optimizing the weighted error, but using all the instances to train the classifier. Table VIII shows the comparison under the column "Subspace." For C4.5 and SVM, the comparison is favorable to our method at a significant level of 95%, and for k-NN at a significant level of 90%.

Finally, to further validate our model, we must check that instance selection using the weights given by boosting is responsible for its good behavior, and not just the repeated use of instance selection, regardless of the optimization of the weighted error. So, we construct the ensemble using the same algorithm, but without using the weights given by boosting. That is, we modify the fitness value of our algorithm, substituting (5) by

$$\phi = \alpha(1-\epsilon) + (1-\alpha)r \tag{9}$$

where ϵ is the standard error $\epsilon = (1/n)\sum_{j=1}^{n}[f(\mathbf{x}_j) \neq y_j]$. For α, we use the same value as for the previous experiments $\alpha = 0.75$. To avoid performing the instance selection on the same set of instances, a method that will probably achieve poor results, we add bagging sampling to introduce diversity in the ensemble creation. Table VIII shows the comparison in the column "Ins. sel. Bagging." The comparison is favorable to our method with a difference that is significant at a 95% confidence level for the three classifiers.

As a summary, we can say that with the experiments reported in this section, we have shown that the performance of our method is only achievable if all its parts are used together. We have shown that either instance selection, the evolution of the subspaces, or weighted instance selection coupled with bagging separately is not able to obtain the same testing error as our method.

E. Noise Effect

Several researchers have reported that boosting methods, among them ADABOOST, degrade their performance in the presence of noise [12], [60]. Dietterich [61] tested this effect introducing artificial noise in the class labels of different data sets and confirmed this behavior. However, RSM method has been shown to be less affected by noise. Breiman [62] reported that random forests were less affected by noise in the class labels of the instances. In this section, we study the sensitivity of our method to noise compared with the best standard algorithm.

To add noise to the class labels, we follow the method of Dietterich [61]. To add classification noise at a rate r, we chose a fraction r of the training instances and changed their class labels to be incorrect choosing uniformly from the set of incorrect labels. We chose all the data sets and rates of noise of 5%, 10%, and 20%. With these levels of noise, we performed the experiments using the 5×2 cv setup and the best standard methods, RSM for k-NN and ADABOOST for C4.5 and SVM, and wIS-chc as the best proposed method. Tables IX–XI show the results for the methods and the three levels of noise, and Tables XII–XIV show the comparison of the methods for the three levels of noise.

TABLE X
SUMMARY OF TEST ERROR RESULTS FOR STANDARD- AND INSTANCE-BASED METHODS FOR k-NN, C4.5 AND SVM AS BASE LEARNER FOR A NOISE LEVEL OF 10%, USING A 5×2 CV SETUP

Dataset	k-NN			C4.5					SVM				
	RSM	wIS-chc		ADABOOST		wIS-chc			ADABOOST		wIS-chc		
	Test error	Test error	Reduction	Test error	#Nodes	Test error	Reduction	#Nodes	Test error	#SV	Test error	Reduction	#SV
arrhythmia	0.4115	0.3969	0.7733	0.3208	71.2	0.2801	0.6149	39.0	0.3739	139.6	0.3602	0.5698	95.3
audio	0.4301	0.3257	0.7126	0.2761	41.1	0.2814	0.7045	24.5	0.3345	106.1	0.2796	0.6644	37.1
autos	0.3921	0.4000	0.7312	0.3443	38.0	0.3384	0.5991	25.9	0.4105	85.9	0.3803	0.6058	38.5
breast-c.	0.2797	0.3678	0.8088	0.3238	41.1	0.3608	0.7874	15.8	0.3259	130.2	0.3245	0.8095	14.9
car	0.2271	0.0680	0.7256	0.0533	177.2	0.0551	0.7413	84.9	0.1373	545.7	0.1186	0.7459	118.5
card	0.2035	0.2247	0.7844	0.1603	88.5	0.1890	0.7313	42.2	0.1930	305.1	0.1867	0.7595	48.1
dermatol.	0.0399	0.0339	0.7471	0.0454	42.0	0.0382	0.6893	23.5	0.1416	91.0	0.0896	0.6496	50.5
ecoli	0.1988	0.1625	0.7707	0.1923	41.9	0.1887	0.6775	23.2	0.1792	125.4	0.1625	0.7498	29.6
gene	0.1342	0.1204	0.7541	0.1156	343.9	0.1096	0.6160	217.7	0.2011	1066.5	0.1243	0.7465	201.3
german	0.2830	0.3104	0.7810	0.2788	138.7	0.3008	0.7188	65.8	0.2930	453.9	0.2738	0.7608	67.4
glass	0.3355	0.3421	0.7465	0.3421	34.5	0.3383	0.6357	24.6	0.4430	91.9	0.3710	0.7254	26.2
glass-g2	0.2420	0.2493	0.7817	0.2639	11.5	0.2381	0.7560	8.3	0.3411	72.1	0.2725	0.8360	8.5
heart	0.1785	0.2296	0.7837	0.2326	31.7	0.2378	0.6753	18.4	0.2163	112.4	0.2193	0.7859	13.3
hepatitis	0.1679	0.2438	0.8026	0.2375	18.8	0.2826	0.7545	11.1	0.3109	60.9	0.2826	0.7603	11.0
horse	0.3588	0.3412	0.7522	0.3511	55.2	0.3549	0.6206	36.2	0.4269	132.5	0.3626	0.6468	48.0
ionosph.	0.1555	0.1397	0.7563	0.1253	20.3	0.1231	0.6650	15.6	0.2006	131.5	0.1629	0.7428	20.7
isolet	0.1222	0.1233	0.7442	0.1808	1324.7	0.2076	0.7001	421.8	0.2684	3323.1	0.1039	0.6988	971.8
letter	0.1704	0.1534	0.7017	0.1862	826.2	0.2081	0.6831	318.6	0.2381	1925.7	0.2064	0.7173	593.0
liver	0.3797	0.3478	0.7521	0.3542	24.8	0.3339	0.7491	12.5	0.3363	154.9	0.3154	0.8160	23.7
lrs	0.1552	0.1533	0.7642	0.1469	53.4	0.1560	0.6886	39.2	0.2339	171.1	0.1627	0.6167	86.2
lymph	0.1797	0.1919	0.7746	0.2149	19.8	0.1933	0.7090	12.2	0.1865	30.8	0.2014	0.6860	19.4
mfeat-fac	0.0524	0.0484	0.7658	0.0610	205.4	0.0680	0.6324	95.2	0.0912	507.4	0.0675	0.6049	312.3
mfeat-fou	0.2140	0.1967	0.7336	0.2228	264.6	0.2182	0.5962	136.6	0.3082	786.0	0.2140	0.5943	341.8
mfeat-kar	0.0578	0.0513	0.7329	0.0948	257.5	0.0930	0.5956	132.0	0.1245	521.5	0.0796	0.6025	315.5
mfeat-mor	0.3058	0.2978	0.7628	0.2951	228.2	0.2960	0.7169	78.3	0.2628	719.8	0.2600	0.7592	176.1
mfeat-pix	0.0372	0.0361	0.7623	0.0660	243.1	0.0810	0.6447	96.8	0.0767	513.2	0.0671	0.6015	336.4
mfeat-zer	0.2098	0.1898	0.7465	0.2430	257.8	0.2569	0.6816	117.6	0.3196	797.7	0.1991	0.7031	225.5
optdigits	0.0254	0.0285	0.7546	0.0431	634.8	0.0556	0.7180	239.5	0.1446	1932.8	0.0480	0.7325	451.9
page-bl.	0.0413	0.0428	0.7719	0.0327	388.0	0.0333	0.7398	94.9	0.0550	1691.2	0.0487	0.7604	210.7
phoneme	0.1666	0.1568	0.7643	0.1506	170.0	0.1520	0.7652	37.0	0.2370	2383.1	0.2269	0.7697	415.9
pima	0.2696	0.2667	0.7351	0.2557	49.4	0.2857	0.7766	22.2	0.2419	327.1	0.2466	0.7759	57.5
primary	0.5870	0.5864	0.7474	0.5952	72.1	0.5711	0.6620	30.3	0.6608	156.7	0.6118	0.5890	67.9
promoters	0.2661	0.2264	0.7679	0.2472	12.2	0.2434	0.7587	5.7	0.2340	30.1	0.2566	0.7391	13.2
satimage	0.1029	0.1126	0.7602	0.1092	601.4	0.1123	0.6297	333.4	0.1563	1917.9	0.1447	0.7574	405.9
segment	0.0490	0.0534	0.7894	0.0384	186.7	0.0365	0.7199	104.5	0.0620	719.8	0.0484	0.7567	144.1
sick	0.0557	0.0312	0.7598	0.0224	262.7	0.0222	0.7292	72.7	0.0612	1695.4	0.0612	0.7843	166.8
sonar	0.2211	0.2510	0.8264	0.2894	14.2	0.2846	0.6498	10.6	0.3548	55.2	0.3000	0.6687	22.5
soybean	0.0978	0.0855	0.7126	0.0858	89.8	0.0919	0.6577	50.2	0.1918	291.5	0.1280	0.6391	114.1
vehicle	0.3104	0.3012	0.7249	0.2773	114.5	0.2603	0.6137	70.7	0.2333	307.9	0.2244	0.7313	77.6
vote	0.0726	0.0653	0.7403	0.0529	33.7	0.1034	0.7946	18.2	0.0791	170.9	0.0557	0.7718	17.1
vowel	0.1408	0.1738	0.7053	0.2115	135.7	0.2248	0.5857	94.8	0.3057	417.8	0.2305	0.6977	125.8
wavef.	0.1646	0.1509	0.7552	0.1810	394.8	0.1775	0.6132	248.4	0.1532	1998.2	0.1444	0.7559	413.5
yeast	0.4492	0.4197	0.7520	0.4222	255.0	0.4151	0.6499	116.6	0.4286	580.2	0.4234	0.7446	162.8
zip	0.0360	0.0449	0.7509	0.0675	1020.4	0.0794	0.6695	386.8	0.0501	2745.0	0.0373	0.7498	568.9
zoo	0.0910	0.1085	0.8069	0.0888	12.9	0.1184	0.7585	9.5	0.1525	35.2	0.1366	0.8181	9.0
Average	0.2015	0.1967	0.7573	0.1978	207.76	0.2021	0.6861	90.75	0.2395	679.07	0.2049	0.7156	170.57

For k-NN, wIS-chc suffers from noise presence, increasing its testing error as the noise is introduced in an amount close to the level of noise. On the other hand, RSM is less affected by noise. This is an expected result as RSM does not modify the data set in any way to adapt to the misclassified instances. However, the degradation of the proposed algorithm is not dramatic; for a noise level of 5%, wIS-chc is still significantly better than RSM, and for noise levels of 10% and 20%, it is still able to achieve a favorable win/loss record with RSM although the differences are no longer significant.

For C4.5, the results are somewhat worse, as the proposed method is more affected by noise than ADABOOST, although the differences are not marked. For SVM, the differences are marked with a clear advantage for our method whose degradation is clearly less than the degradation of ADABOOST. In this way, the proposed method achieves a win/loss record of 37/8, 40/4, and 37/7 for noise levels of 5%, 10%, and 20%, respectively.

Nevertheless, for C4.5 and SVM, wIS-chc has a very good behavior in terms of complexity of the classifiers. As the noise level increases, the classifiers need to be more complex to cope with the added noise. However, wIS-chc is able to reduce this growing complexity. In this way, without noise, the reduction in the number of nodes for C4.5 was of 38.94%; with a noise level of 20%, this reduction raises to 54.27%, and for SVM, the reduction in the number of SVs was of 66.32% and with a noise level of 20% of 72.98%.

VII. CONCLUSION AND FUTURE WORK

In this paper, we have proposed a new approach for constructing ensembles of classifiers using weighted instance selection. The distribution of weights given by any boosting algorithm is used to perform an instance selection process where

TABLE XI
SUMMARY OF TEST ERROR RESULTS FOR STANDARD- AND INSTANCE-BASED METHODS FOR k-NN, C4.5 AND SVM AS BASE LEARNER FOR A NOISE LEVEL OF 20%, USING A 5 × 2 CV SETUP

Dataset	k-NN			C4.5					SVM				
	RSM	wIS-chc		ADABOOST		wIS-chc			ADABOOST		wIS-chc		
	Test error	Test error	Reduction	Test error	#Nodes	Test error	Reduction	#Nodes	Test error	#SV	Test error	Reduction	#SV
arrhythmia	0.4203	0.4075	0.7811	0.3438	78.1	0.2925	0.5834	47.9	0.3978	142.4	0.3956	0.5421	102.5
audio	0.5009	0.3832	0.7309	0.3416	48.6	0.3159	0.6755	21.4	0.5487	108.9	0.4195	0.5853	46.3
autos	0.4809	0.4731	0.7432	0.4536	41.9	0.4496	0.5918	27.1	0.5375	93.2	0.4683	0.6061	39.3
breast-c.	0.2902	0.3713	0.8125	0.3273	45.1	0.3804	0.8020	14.9	0.3413	132.2	0.3839	0.8134	16.3
car	0.2270	0.1002	0.7416	0.0838	233.6	0.0842	0.7042	89.0	0.1717	604.7	0.1501	0.7568	144.4
card	0.2267	0.2867	0.7958	0.1942	103.6	0.2635	0.7291	44.4	0.2194	316.1	0.2215	0.7701	53.9
dermatol.	0.0492	0.0350	0.7521	0.0530	55.6	0.0426	0.6688	29.8	0.2399	115.3	0.1295	0.6154	59.6
ecoli	0.2054	0.1744	0.7747	0.2060	49.4	0.1952	0.6620	27.3	0.1911	127.0	0.1625	0.7489	32.1
gene	0.1517	0.1385	0.7556	0.1846	397.2	0.1412	0.6026	252.9	0.2137	1146.0	0.1551	0.7517	254.1
german	0.2842	0.3350	0.7824	0.3054	146.0	0.3392	0.7179	67.7	0.3036	470.4	0.2984	0.7644	74.6
glass	0.3654	0.3785	0.7521	0.3654	32.9	0.3654	0.6012	25.3	0.4514	91.9	0.3823	0.7356	25.5
glass-g2	0.2898	0.2763	0.8092	0.3242	10.1	0.2860	0.7741	7.8	0.3437	74.6	0.3045	0.8310	8.8
heart	0.1896	0.2978	0.7943	0.2711	36.8	0.3148	0.7280	18.2	0.2445	119.7	0.2882	0.8000	14.8
hepatitis	0.2027	0.3198	0.8146	0.2996	22.6	0.3573	0.7507	11.2	0.3714	67.7	0.3883	0.7650	10.8
horse	0.3566	0.3626	0.8026	0.3632	59.1	0.3912	0.6777	34.3	0.4956	144.4	0.3879	0.6408	50.3
ionosph.	0.1687	0.1555	0.7694	0.1572	21.4	0.1715	0.7402	14.6	0.2416	148.5	0.2233	0.7700	21.9
isolet	0.1299	0.1261	0.7476	0.2062	1654.7	0.2148	0.6285	598.0	0.3415	3447.9	0.1322	0.7249	939.4
letter	0.1838	0.1724	0.7065	0.2154	939.8	0.2228	0.6196	424.0	0.2512	1870.2	0.2338	0.7144	631.2
liver	0.3907	0.3803	0.7641	0.3767	21.5	0.3762	0.7980	12.4	0.3710	158.1	0.3629	0.8167	23.7
lrs	0.1612	0.1571	0.7673	0.1654	68.8	0.1586	0.6104	44.1	0.3382	231.9	0.1959	0.5896	99.9
lymph	0.1986	0.2230	0.8046	0.2365	23.2	0.2689	0.7158	14.7	0.2635	36.6	0.2635	0.6383	23.7
mfeat-fac	0.0661	0.0539	0.7696	0.0719	258.3	0.0788	0.6079	131.2	0.1931	675.2	0.0994	0.5747	367.3
mfeat-fou	0.2268	0.2096	0.7437	0.2532	319.7	0.2269	0.5638	173.1	0.4143	876.0	0.2593	0.5779	374.9
mfeat-kar	0.0733	0.0612	0.7442	0.1313	347.8	0.1194	0.5652	172.0	0.3312	812.1	0.1360	0.5873	352.6
mfeat-mor	0.3033	0.3050	0.7632	0.3094	292.1	0.2996	0.7097	101.2	0.2608	759.0	0.2614	0.7599	196.0
mfeat-pix	0.0499	0.0433	0.7665	0.0880	316.7	0.0842	0.6270	131.4	0.1433	613.3	0.1025	0.5716	388.3
mfeat-zer	0.2158	0.1928	0.7490	0.2593	314.2	0.2596	0.5771	174.4	0.4046	861.5	0.2176	0.7097	235.0
optdigits	0.0295	0.0313	0.7644	0.0543	866.0	0.0532	0.6138	406.5	0.1057	1903.6	0.0588	0.7435	489.4
page-bl.	0.0437	0.0464	0.7745	0.0374	598.2	0.0387	0.7359	155.5	0.0725	1696.2	0.0569	0.7641	305.5
phoneme	0.1761	0.1773	0.7606	0.1842	84.0	0.1899	0.7527	15.0	0.2378	2434.4	0.2334	0.7708	470.4
pima	0.2760	0.3284	0.7890	0.2906	49.7	0.3214	0.7898	20.7	0.2458	344.9	0.2550	0.7819	63.9
primary	0.5893	0.5806	0.7420	0.5931	73.2	0.5894	0.6657	30.0	0.6938	161.2	0.6247	0.5880	68.7
promoters	0.2887	0.2774	0.7847	0.3679	13.1	0.3264	0.7263	6.4	0.3076	30.7	0.3095	0.7055	14.7
satimage	0.1092	0.1195	0.7747	0.1153	846.1	0.1170	0.5903	497.9	0.1696	2132.3	0.1549	0.7585	504.0
segment	0.0610	0.0572	0.8006	0.0510	291.1	0.0436	0.6704	121.5	0.0818	798.1	0.0602	0.7591	187.5
sick	0.0585	0.0351	0.7675	0.0323	346.1	0.0230	0.7348	98.1	0.0614	1759.0	0.0612	0.7852	238.5
sonar	0.2644	0.2740	0.8391	0.3260	13.7	0.3163	0.7532	8.6	0.3808	56.1	0.3337	0.6698	22.5
soybean	0.1116	0.0826	0.7233	0.1075	108.7	0.0902	0.6493	53.4	0.2814	310.4	0.1924	0.6165	125.4
vehicle	0.3180	0.3135	0.7306	0.3038	132.1	0.2986	0.5952	84.2	0.2567	312.5	0.2468	0.7362	82.8
vote	0.0786	0.1393	0.7991	0.0965	50.6	0.1683	0.7985	19.8	0.1122	189.4	0.0869	0.7830	26.1
vowel	0.1919	0.2428	0.7136	0.2422	168.5	0.2463	0.5865	116.7	0.3568	399.8	0.2986	0.7063	129.4
wavef.	0.1660	0.1523	0.7629	0.2058	485.0	0.1894	0.5860	310.3	0.1709	2116.8	0.1545	0.7577	495.9
yeast	0.4558	0.4229	0.7518	0.4472	289.1	0.4284	0.6314	134.4	0.4324	558.5	0.4229	0.7476	167.7
zip	0.0423	0.0501	0.7518	0.0828	1359.3	0.0824	0.6356	563.2	0.0431	2704.1	0.0422	0.7499	707.6
zoo	0.1325	0.1325	0.7957	0.1484	14.5	0.1425	0.7590	10.2	0.2115	39.9	0.1721	0.7976	10.0
Average	0.2178	0.2196	0.7681	0.2283	260.62	0.2303	0.6735	119.17	0.2855	715.39	0.2397	0.7107	193.26

TABLE XII
COMPARISON OF RESULTS IN TERMS OF GENERALIZATION ERROR FOR A k-NN CLASSIFIER AND NOISE LEVELS OF 5%, 10%, AND 20%. WIN/DRAW/LOSS RECORD (ROW s) OF THE ALGORITHMS AGAINST EACH OTHER AND p-VALUE OF THE SIGN TEST (ROW p_s), p-VALUE OF THE WILCOXON TEST (ROW p_w). SIGNIFICANT DIFFERENCES ARE MARKED WITH AN **x**, 95% CONFIDENCE, OR ✓, 90% CONFIDENCE. ERROR RATIO ROW SHOWS THE INCREMENT OF ERROR WITH RESPECT TO THE SAME METHOD APPLIED TO ORIGINAL DATA SETS

Noise level	5%		10%		20%	
	AdaBoost	wIS-chc	AdaBoost	wIS-chc	AdaBoost	wIS-CHC
Mean all	0.2192	0.1900	0.2015	0.1967	0.2178	0.2196
Ratio	18.00%	10.30%	6.08%	12.14%	14.64%	25.22%
Reduction	649.31	157.55		75.73%		76.81%
AdaBoost		27/2/16		27/0/18		27/0/18
		0.1263		0.2327		0.2327
		0.0410✗		0.2473		0.3695

TABLE XIII
COMPARISON OF RESULTS IN TERMS OF GENERALIZATION ERROR FOR AN SVM AND NOISE LEVELS OF 5%, 10%, AND 20%. WIN/DRAW/LOSS RECORD (ROW s) OF THE ALGORITHMS AGAINST EACH OTHER AND p-VALUE OF THE SIGN TEST (ROW p_s), p-VALUE OF THE WILCOXON TEST (ROW p_w). SIGNIFICANT DIFFERENCES ARE MARKED WITH AN **x**, 95% CONFIDENCE, OR ✓, 90% CONFIDENCE. ERROR RATIO ROW SHOWS THE INCREMENT OF ERROR WITH RESPECT TO THE SAME METHOD APPLIED TO ORIGINAL DATA SETS

Noise level	5%		10%		20%	
	AdaBoost	wIS-chc	AdaBoost	wIS-chc	AdaBoost	wIS-CHC
Mean all	0.2192	0.1900	0.2395	0.2049	0.2855	0.2397
Ratio	1.51%	18.00%	28.89%	18.93%	53.65%	39.13%
#SV	649.31	157.55	679.07	170.57	715.39	193.26
AdaBoost		37/0/8		40/1/4		37/1/7
		0.0003		0.0000		0.0000
		0.0000		0.0000		0.0000

the objective is minimizing the weighted error. This method can be applied to any base learner able to benefit from a reduced learning set. In this way, we have applied the method to k-NN classifier, where an improved performance is obtained with the

TABLE XIV
COMPARISON OF RESULTS IN TERMS OF GENERALIZATION ERROR FOR A C4.5 CLASSIFIER AND NOISE LEVELS OF 5%, 10%, AND 20%. WIN/DRAW/LOSS RECORD (ROW s) OF THE ALGORITHMS AGAINST EACH OTHER AND p-VALUE OF THE SIGN TEST (ROW p_s), p-VALUE OF THE WILCOXON TEST (ROW p_w). SIGNIFICANT DIFFERENCES ARE MARKED WITH AN $\mathbf{x}$, 95% CONFIDENCE, OR ✓, 90% CONFIDENCE. ERROR RATIO ROW SHOWS THE INCREMENT OF ERROR WITH RESPECT TO THE SAME METHOD APPLIED TO ORIGINAL DATA SETS

Noise level	5%		10%		20%	
	RSM	wIS-chc	RSM	wIS-chc	RSM	wIS-CHC
Mean all	0.1885	0.1887	0.1978	0.2021	0.2283	0.2303
Ratio	5.12%	9.25%	10.30%	17.05%	27.33%	33.38%
#Nodes	168.08	79.91	207.76	90.75	260.62	119.17
RSM		18/0/27		20/0/25		24/1/20
		0.2327		0.5515		0.6516
		0.6476		0.1702		0.6721

additional advantage of a significant reduction in the complexity of the ensemble.

In a second set of experiments, we applied the method to C4.5 and SVM classifiers. Again, we are able to obtain a better performing ensemble, in terms of testing error, with a clear decrement in the complexity of the ensemble. A further study on noise effect on classifier performance has shown that the proposed method is quite robust in the presence of noise. Moreover, the reported reduction in the complexity of the classifiers increases as more noise is added to the data sets. Furthermore, the different parts of our model have been separately validated showing their usefulness in the method.

As a future research line, we are working on introducing the instance selection part within the boosting algorithm, instead of the proposed approach here where the two algorithms are coupled, but performed separately.

REFERENCES

[1] C. J. Merz, "Using correspondence analysis to combine classifiers," *Mach. Learn.*, vol. 36, no. 1, pp. 33–58, Jul. 1999.

[2] T. Windeatt, "Accuracy/diversity and ensemble MLP classifier design," *IEEE Trans. Neural Netw.*, vol. 17, no. 5, pp. 1194–1211, Sep. 2006.

[3] T. G. Dietterich, "Ensemble methods in machine learning," in *Lecture Notes in Computer Science*, J. Kittler and F. Roli, Eds. Berlin, Germany: Springer-Verlag, 2000, vol. 1857, pp. 1–15.

[4] L. I. Kuncheva, "Combining classifiers: Soft computing solutions," in *Pattern Recognition: From Classical to Modern Approaches*, S. K. Pal and A. Pal, Eds. Singapore: World Scientific, 2001, pp. 427–451.

[5] Y. Freund and R. Schapire, "Experiments with a new boosting algorithm," in *Proc. 13th Int. Conf. Mach. Learn.*, Bari, Italy, 1996, pp. 148–156.

[6] G. I. Webb, "Multiboosting: A technique for combining boosting and wagging," *Mach. Learn.*, vol. 40, no. 2, pp. 159–196, Aug. 2000.

[7] S. Dzeroski and B. Zenko, "Is combining classifiers with stacking better than selecting the best one?," *Mach. Learn.*, vol. 54, pp. 255–273, 2004.

[8] A. Fern and R. Givan, "Online ensemble learning: An empirical study," *Mach. Learn.*, vol. 53, pp. 71–109, 2003.

[9] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996.

[10] L. Breiman, "Bias, variance, and arcing classifiers," Dept. Statist., Univ. California, Berkeley, CA, Tech. Rep. 460, 1996.

[11] R. E. Schapire, Y. Freund, P. L. Bartlett, and W. S. Lee, "Boosting the margin: A new explanation for the effectiveness of voting methods," *Ann. Statist.*, vol. 26, no. 5, pp. 1651–1686, 1998.

[12] E. Bauer and R. Kohavi, "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants," *Mach. Learn.*, vol. 36, no. 1/2, pp. 105–142, Jul./Aug. 1999.

[13] H. Liu and H. Motoda, "On issues of instance selection," *Data Mining Knowl. Disc.*, vol. 6, pp. 115–130, 2002.

[14] J. R. Cano, F. Herrera, and M. Lozano, "Using evolutionary algorithms as instance selection for data reduction in KDD: An experimental study," *IEEE Trans. Evol. Comput.*, vol. 7, no. 6, pp. 561–575, Dec. 2003.

[15] C. Domeniconi, G. D. , and J. Peng, "Large margin nearest neighbor classifiers," *IEEE Trans. Neural Netw.*, vol. 16, no. 4, pp. 899–909, Jul. 2005.

[16] H. Brighton and C. Mellish, "Advances in instance selection for instance-based learning algorithms," *Data Mining Knowl. Disc.*, vol. 6, pp. 153–172, 2002.

[17] D. W. Aha, "A study of instance-based algorithms for supervised learning tasks: Mathematical, empirical, and psychological evaluations," Ph.D. dissertation, Dept. Inf. Comput. Sci., Univ. California at Irvine, Irvine, CA, 1990.

[18] L. Wiskott, J.-M. Fellous, N. Kríger, and C. von der Malsburg, "Face recognition by elastic bunch graph matching," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 7, pp. 775–779, Jul. 1997.

[19] G. Shakhnarovich, P. Viola, and T. Darrell, "Fast pose estimation with parameter-sensitive hashing," in *Proc. 9th Int. Conf. Comput. Vis.*, Nice, France, 2003, pp. 750–757.

[20] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 4, pp. 509–522, Apr. 2002.

[21] T. K. Ho and H. S. Baird, "Large-scale simulation studies in image pattern recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 10, pp. 1067–1079, Oct. 1997.

[22] T. G. Dietterich and G. Bakiri, "Solving multiclass learning problems via error-correcting output codes," *J. Artif. Intell. Res.*, vol. 2, pp. 263–286, 1995.

[23] E. B. Kong and T. G. Dietterich, "Error-correcting output coding corrects bias and variance," in *Proc. 12th Nat. Conf. Artif. Intell.*, 1996, pp. 725–730.

[24] S. D. Bay, "Nearest neighbor classification from multiple feature subsets," *Intell. Data Anal.*, vol. 3, no. 3, pp. 191–209, 1999.

[25] J. O'Sullivan, J. Langford, R. Caruna, and A. Blum, "Featureboost: A metalearning algorithm that improves model robustness," in *Proc. 17th Int. Conf. Mach. Learn.*, 2000, pp. 703–710.

[26] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 8, pp. 832–844, Aug. 1998.

[27] L. Nanni and A. Lumini, "Evolved feature weighting for random subspace classifier," *IEEE Trans. Neural Netw.*, vol. 19, no. 2, pp. 363–366, Feb. 2008.

[28] E. Kleinberg, "On the algorithmic implementation of stochastic discrimination," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 5, pp. 473–490, May 2000.

[29] M. Skurichina and R. P. W. Duin, "Bagging and the random subspace method for redundant feature spaces," in *Proc. 2nd Int. Workshop Multiple Classifier Syst.*, J. Kittler and R. Poli, Eds., Cambridge, U.K., 2001, pp. 1–10.

[30] R. Munro, D. Ler, and J. Patrick, "Meta-learning orthographic and contextual models for language independent named entity recognition," in *Proc. 7th Conf. Natural Lang. Learn.*, 2003, pp. 192–195.

[31] L. Hall, K. Bowyer, R. Banfield, D. Bhadoria, W. Kegelmeyer, and S. Eschrich, "Comparing pure parallel ensemble creation techniques against bagging," in *Proc. 3rd IEEE Int. Conf. Data Mining*, Melbourne, FL, 2003, pp. 533–536.

[32] D. R. Wilson and T. R. Martinez, "Reduction techniques for instance-based learning algorithms," *Mach. Learn.*, vol. 38, pp. 257–286, 2000.

[33] D. L. Wilson, "Asymptotic properties of nearest neighbor rules using edited data," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-2, no. 3, pp. 408–421, 1972.

[34] P. E. Hart, "The condensed nearest neighbor rule," *IEEE Trans. Inf. Theory*, vol. IT-14, no. 3, pp. 515–516, May 1968.

[35] G. W. Gates, "The reduced nearest neighbor rule," *IEEE Trans. Inf. Theory*, vol. IT-18, no. 3, pp. 431–433, May 1972.

[36] L. Kuncheva, "Editing for the k-nearest neighbors rule by a genetic algorithm," *Pattern Recognit. Lett.*, vol. 16, pp. 809–814, 1995.

[37] H. Ishibuchi and T. Nakashima, "Pattern and feature selection by genetic algorithms in nearest neighbor classification," *J. Adv. Comput. Intell. Intell. Inf.*, vol. 4, no. 2, pp. 138–145, 2000.

[38] C. R. Reeves and D. R. Bush, "Using genetic algorithms for training data selection in RBF networks," in *Instances Selection and Construction for Data Mining*, H. Liu and H. Motoda, Eds. Norwell, MA: Kluwer, 2001, pp. 339–356.

[39] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[40] D. Whitley, "The GENITOR algorithm and selective pressure," in *Proc. 3rd Int. Conf. Genetic Algorithms*, M. K. Publishers, Ed., Los Altos, CA, 1989, pp. 116–121.
[41] L. J. Eshelman*, The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination*. San Mateo, CA: Morgan Kauffman, 1990.
[42] S. Baluja, "Population-based incremental learning," Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-CS-94-163, 1994.
[43] S. J. Louis and G. Li, "Combining robot control strategies using genetic algorithms with memory," in *Lecture Notes in Computer Science, Evolutionary Programming VI*. Berlin, Germany: Springer-Verlag, 1997, vol. 1213, pp. 431–442.
[44] H. Ishibuchi and T. Nakashima, "Multi-objective pattern and feature selection by a genetic algorithm," in *Proc. Genetic Evol. Comput. Conf.*, Las Vegas, NV, 2000, pp. 1069–1076.
[45] H. Ishibuchi, T. Nakashima, and M. Nii, "Learning of neural networks with GA-based instance selection," in *Proc. IFSA World Congr. 20th NAFIPS Int. Conf.*, Jul. 2001, vol. 4, pp. 2102–2107.
[46] C. E. Brodley, "Recursive automatic bias selection for classifier construction," *Mach. Learn.*, vol. 20, no. 1/2, pp. 63–94, 1995.
[47] L. Wang, N. Zhou, and F. Chu, "A general wrapper approach to selection of class-dependent features," *IEEE Trans. Neural Netw.*, vol. 19, no. 7, pp. 1267–1278, Jul. 2008.
[48] K.-J. Kim, "Artificial neural networks with evolutionary instance selection for financial forecasting," *Expert Syst. Appl.*, vol. 30, pp. 519–526, 2006.
[49] S. Hettich, C. Blake, and C. Merz, UCI Repository of Machine Learning Databases, 1998 [Online]. Available: http://www.ics.uci.edu/mlearn/MLRepository.html
[50] T. G. Dietterich, "Approximate statistical tests for comparing supervised classification learning algorithms," *Neural Comput.*, vol. 10, no. 7, pp. 1895–1923, 1998.
[51] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, 2006.
[52] R. L. Iman and J. M. Davenport, "Approximations of the critical regions of the Friedman statistic," *Commun. Statist.*, vol. 6, pp. 571–595, 1980.
[53] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics*, vol. 1, pp. 80–83, 1945.
[54] GNU General Public License Version 3, Free Software Foundation, Inc., Jun. 2007 [Online]. Available: http://www.gnu.org/licenses/gpl-3.0-standalone.html
[55] R. Polikar, L. Udpa, S. Udpa, and V. Honavar, "Learn++: An incremental learning algorithm for supervised neural networks," *IEEE Trans. Syst. Man Cybern. C, Appl. Rev.*, vol. 31, no. 4, pp. 497–508, Nov. 2001.
[56] P. Melville and R. J. Mooney, "Creating diversity in ensembles using artificial data," *Inf. Fusion*, vol. 6, pp. 99–111, 2005.
[57] G. Zenobi and P. Cunningham, "Using diversity in preparing ensembles of classifiers based on different feature subsets to minimize generalization error," in *Lecture Notes on Artificial Intelligence*, L. de Raedt and P. Flach, Eds. Berlin, Germany: Springer-Verlag, 2001, vol. 2167, pp. 576–587.
[58] J. H. Chen, H. M. Chen, and S. Y. Ho, "Design of nearest neighbor classifiers: Multi-objective approach," *Int. J. Approx. Reason.*, vol. 40, no. 1–2, pp. 3–22, 2005.
[59] S. H. Son and J. Y. Kim, "Data reduction for instance-based learning using entropy-based partitioning," in *Lecture Notes in Computer Science*. Berlin, Germany: Springer-Verlag, 2006, vol. 3982, pp. 590–599.
[60] J. R. Quinlan, "Bagging, boosting, and c4.5," in *Proc. 13th Nat. Conf. Artif. Intell.*, 1996, pp. 725–730.
[61] T. G. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization," *Mach. Learn.*, vol. 40, pp. 139–157, 2000.
[62] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, pp. 5–32, 2001.

Nicolás García-Pedrajas (A'01) was born in Córdoba, Spain, in 1970. He received the B.S. degree in computing and the Ph.D. degree in artificial intelligence from the University of Málaga, Málaga, Spain, in 1993 and 2001, respectively.

His current research interests include neural networks, evolutionary computation, and game playing. His dissertation research involved the automatic design of artificial neural networks using both genetic algorithms and evolutionary programming. He is a Professor in the Area of Computer Science and Artificial Intelligence at the Department of Computing and Numerical Analysis, University of Córdoba. Currently, he is the Director of the Computational Intelligence and Bioinformatics Research Group of the University of Córdoba.