Expert Systems with Applications 36 (2009) 10570–10582

Contents lists available at ScienceDirect

Expert Systems with Applications

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

# Boosting *k*-nearest neighbor classifier by means of input space projection

Nicolás García-Pedrajas [*], Domingo Ortiz-Boyer

*Department of Computing and Numerical Analysis, University of Córdoba, Campus de Rabanales, 14071 Córdoba, Spain*

ARTICLE INFO

*Keywords:*
*k*-Nearest neighbors
Boosting
Subspace methods

ABSTRACT

The *k*-nearest neighbors classifier is one of the most widely used methods of classification due to several interesting features, such as good generalization and easy implementation. Although simple, it is usually able to match, and even beat, more sophisticated and complex methods. However, no successful method has been reported so far to apply boosting to *k*-NN. As boosting methods have proved very effective in improving the generalization capabilities of many classification algorithms, proposing an appropriate application of boosting to *k*-nearest neighbors is of great interest.

Ensemble methods rely on the instability of the classifiers to improve their performance, as *k*-NN is fairly stable with respect to resampling, these methods fail in their attempt to improve the performance of *k*-NN classifier. On the other hand, *k*-NN is very sensitive to input selection. In this way, ensembles based on subspace methods are able to improve the performance of single *k*-NN classifiers. In this paper we make use of the sensitivity of *k*-NN to input space for developing two methods for boosting *k*-NN. The two approaches modify the view of the data that each classifier receives so that the accurate classification of difficult instances is favored.

The two approaches are compared with the classifier alone and bagging and random subspace methods with a marked and significant improvement of the generalization error. The comparison is performed using a large test set of 45 problems from the UCI Machine Learning Repository. A further study on noise tolerance shows that the proposed methods are less affected by class label noise than the standard methods.

© 2009 Elsevier Ltd. All rights reserved.

## 1. Introduction

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

*k*-Nearest neighbors (*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. To classify a new instance $\mathbf{x}$ the $k$ prototypes that are nearest to $\mathbf{x}$ are obtained, *the k-nearest neighbors*, and $\mathbf{x}$ is classified into the class most frequent in this set of $k$ neighbors. *k*-NN method is used mainly because of its simplicity and its ability to achieve error results comparable with much more complex methods.

It is noticeable that the performance *k*-NN has not been able to be improved by ensemble methods as much as for other classifiers. An ensemble of classifiers consists of a combination of different classifiers, homogeneous or heterogeneous, to jointly perform a classification task.

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 $Y = \{1, \ldots, K\}$. The design of classifier ensembles must perform two main tasks: constructing the individuals classifiers, $F_i$, and developing a combination rule that finds a class label for $\mathbf{x}$ based on the outputs of the classifiers $\{F_1(\mathbf{x}), F_2(\mathbf{x}), \ldots, F_m(\mathbf{x})\}$.

Among the different methodologies for constructing ensembles of classifiers, boosting (Freund & Schapire, 1996) is one of the most widely used. Boosting methods adaptively change the distribution of the training set based on the performance of the previous classifiers. The most widely used boosting method is AdaBoost and its numerous variants. 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, $f_i(\mathbf{x})$:

$$F(\mathbf{x}) = \sum_{i=1}^{M} \alpha_i f_i(\mathbf{x}), \tag{1}$$

[*] Corresponding author. Tel.: +34 957211032; fax: +34 957218630.
*E-mail addresses:* npedrajas@uco.es (N. García-Pedrajas), dortiz@uco.es (D. Ortiz-Boyer).
*URLs:* http://cibrg.org (N. García-Pedrajas), http://cibrg.org (D. Ortiz-Boyer).

0957-4174/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2009.02.065

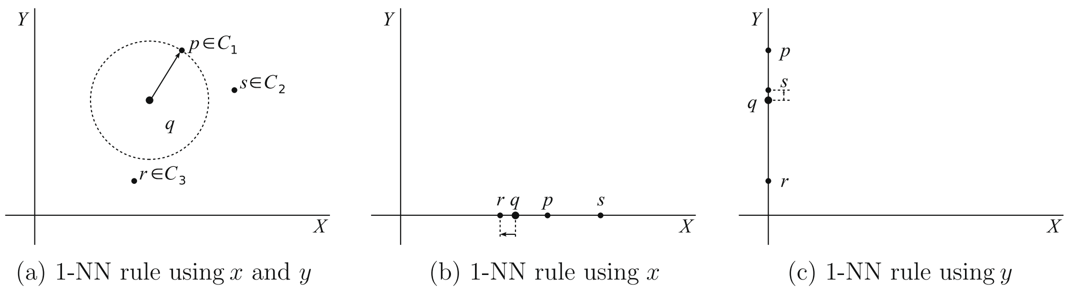

**Fig. 1.** Sensitivity to subspace selection. We have a test instance, $q$, and three training instances, $p$, $s$ and $r$, belonging respectively to classes 1, 2 and 3. Using a 1-NN rule, $q$ can be classified either into class 1 (a), using variables $x$ and $y$, class 2 (c), using variable $y$, or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

where the $\alpha_i$ are appropriately defined. The basis of boosting is assigning a different weight to each training instance depending on how difficult it has been for the previous classifiers to classify it. Thus, for AdaBoost, each instance $j$ receives a weight $w_j^k$ for training the $k$ classifier. Initially all the instances are weighted equally $w_j^1 = 1/N, \forall j$. Then, for classifier $k+1$ the instance is weighted following:

$$w_j^{k+1} = \frac{w_j^k \exp(-\alpha_k [[f_k(\mathbf{x}_j) = y_j]])}{\sum_{i=1}^N w_i^k (1-\beta_k)}, \tag{2}$$

where $[[\pi]]$ is 1 if $\pi$ is true and 0 otherwise, $\beta_k = \frac{\epsilon_k}{(1-\epsilon_k)}$, and $\epsilon_k$ is the weighted error of classifier $k$ when the weight vector is normalized $\sum_{j=1}^N w_j^k = 1$.

An alternative method, the random subspace method (RSM) was proposed by Ho (1998b). This method trains each classifier in a random subspace of the original input space. Combination methods, such as voting or bagging, are usually not useful when applied to $k$-NN, as this method is fairly stable with respect to modification of the training set. On the other hand, $k$-NN is very sensitive to input perturbations, such as subspace selection or non-linear projections. Fig. 1 illustrates with a simple example the sensitivity to subspace selection. The test pattern, $q$, can be classified in three different classes depending on the subspace considered. In this way, results using random subspaces and $k$-NN have been reported (Ho, 1998a) showing improved performance on hand-written digit recognition. In a further step, Tremblay, Sabourin, and Maupin (2004) used a multiobjective genetic algorithm to obtain a set of $k$-NN in different subspaces that jointly maximized individual accuracy and ambiguity.

Although boosting is a very successful method for improving the performance of any classifier, its application to $k$-NN methods is troublesome. The reweighting approach is not useful, as weighting an instance affects the classification of its neighbors, but not the classification of the instance itself. If we use resampling, we can sample the instances using the distribution given by the boosting method. However, this approach is not likely to produce an improvement in the testing error.[1]

Our methodology is based on previous works (García-Pedrajas, in press; García-Pedrajas & Ortiz-Boyer, 2008), where we have shown that the distribution of instances given by boosting can be used in different ways to improve the performance of the ensemble. One of the side effects of this approach is that boosting can be applied to classifiers, as $k$-NN, to which the standard methodology of boosting is not easily introduced. Following the same underlying ideas we develop two methods for boosting $k$-NN classifiers. Both methods are based on changing the view that each classifier of the ensemble has of the instances, modifying the input space in a way that favors the accurate classification of difficult instances.

Thus, the basic idea of the two methods is modifying the view of the data each classifier sees in a way that improves the weighted accuracy over the instances. That is, if for $i$th boosting step we have a weight assigned to each instance given by vector $\mathbf{w}$ we can define the weighted error $\epsilon = E[\mathbf{w}_{(y \neq f(\mathbf{x}))}]$. The objective is to provide the classifier with a view of the data that is more likely to minimize $\epsilon$. Given a training set we can act either on the rows, selecting a certain subset of instances, or on the columns, selecting certain inputs or transforming the inputs. As we have stated that $k$-NN is hardly sensitive to modifications in the training set rows, we focus on the latter approach. Thus, we present here two methods, both of them based on the second alternative: (i) selecting a subset of the inputs; and (ii) transforming the inputs by means of a non-linear projection.

This rest of this paper is organized as follows: Section 2 surveys some related work; Section 3 explains in depth the proposed methods; Section 4 shows the experimental setup and Section 5 shows the results of experiments carried out; and finally Section 6 states the conclusions of our work.

## 2. Related work

The interest on how boosting can be used with $k$-NN rule is not new. Bauer and Kohavi (1999) cited it as one of the interesting research lines in voting classifier. They also noticed that the standard interpretation of counting a highly weighted instance more would not work, as increasing the weight of an instance helps to classify its neighbors, not to classify itself.

It has been shown that $k$-NN classifiers are not effective when used in bagging type methods (Breiman, 1996a). The reason is the stability of $k$-NN with respect to variations on the training instances. As the different bagged training sets have a large overlapping, the errors of the different classifiers are highly correlated and voting them is inefficient.

Grabowski (2002) constructed an ensemble of $k$-NN classifiers using cross-validated values of $k$ for each member which were trained in a random partition of the whole dataset in the same way as bagging. The results reported are slightly above a single $k$-NN classifier. Athitsos and Sclaroff (2005) used AdaBoost algorithms for learning a distance measure for multiclass $k$-NN classifier, but the $k$-NN classifier itself is not boosted in the standard sense.

Bay (1999) applied RSM to nearest neighbor classifier, calculating the closest neighbor to the test instances in different subsets of the input space. Zhou and Yu (2005a) developed an approach to apply bagging to nearest neighbor classifier. In order to add instability to the learning process to favor diversity of classifier, they used Minkowsky distance with a different randomly selected value

[1] As a matter of fact, this approach is tested in the experiments, as a control method, with very poor results.

of p for each classifier. In a subsequent work (Zhou & Yu, 2005b) this method was coupled with boostrap sampling and attribute filtering, and tested on 20 datasets with improved performance.

To avoid the damaging effect of the use of random subspaces when the selected subspace lacks the necessary discriminant information, Domeniconi and Yan (2004) first estimated the relevance of each feature and then sampled the features using that distribution. In this way, relevant inputs are less likely to be overlooked. Viswanath, Murty, and Bhatnagar (2004) developed an ensemble of approximate nearest neighbor classifiers by majority voting. This model is a hybrid with features of bagging and subspace methods, as each nearest neighbor classifier obtains an approximate neighbor depending on the partition chosen for each class and the ordering of features within each block.

François, Grandvalet, Denœux, and Roger (2003) developed ensembles using bagging and evidential $k$-NN and reported improved performance. Altınçay (2007) used a genetic algorithm to evolve an ensemble of evidential $k$-NN classifiers introducing multimodal perturbation. In his approach each chromosome codified a complete ensemble.

Very few previous works have undertaken the task of introducing boosting into $k$-nearest neighbor classifiers. Lazarevic, Fiez, and Obradovic (2000) used principal components analysis (Jolliffe, 1986) applied to the inputs weighted by means of a neural network at each boosting step. The authors coupled this method with spatial data block resampling using the distribution given by AdaBoost.

Freund and Schapire (1996) developed a boosting version of nearest neighbor classifier, but with the goal of speeding it up and not of improving its accuracy. Amores, Sebe, and Radeva (2006) developed a boosted distance function and applied it to $k$-NN. However, boosting is used for constructing the distance function and the $k$-NN algorithm is a standard one. O'Sullivan, Langford, Caruna, and Blum (2000) applied a feature boosting method called FeatureBoost to a $k$-NN classifier. The reported results showed improved performance over standard AdaBoost on three used datasets. FeatureBoost is aimed at applying boosting principles to sample features instead of instances.

$k$-NN has been used as part of an hybrid ensemble (Woods, Kegelmeyer, & Bowyer, 1997), combined with neural networks, decision trees, and Quadratic Bayes classifiers in a single ensemble. In Todorovski and Dzeroski (2003) it was combined with two algorithms for learning decision trees, a rule learning algorithm, and a naive Bayes algorithm.

## 3. Boosting *k*-nearest neighbors

As we have stated in the introduction, the direct application of boosting to $k$-NN classifiers is either not available, when using reweighting, or not useful, when using resampling. Thus, in order to "boost" $k$-NN classifiers we must develop another approach, which uses the boosting philosophy in a new way. As $k$-NN is instable with respect to inputs we will act on the input space, modifying it by means of subspace selection or projection.

The central issue in boosting algorithms is the different distribution of the training instances that receives each classifier. At each boosting step, a new distribution of the instances is given by a vector of weights, **w**. This distribution is biased towards the instances that have been more often missclassified by the previous classifiers. The rationale of our work is that this distribution can be used advantageously to improve the performance of $k$-NN, although its direct use is not available. In this way we propose two different methods, both of them based on the philosophy whose general outline is depicted in Algorithm 1. In Algorithm 1 $\alpha_t$ is obtained as in standard AdaBoost.

**Algorithm 1**: Outline of the proposed methodology

```
Data   : A training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, $\mathbf{x}_i \in \mathbb{R}^D$, a $k$-NN learning algorithm, $\mathbb{L}$, and the number of iterations $T$.
Result : The final classifier: $F^*(\mathbf{x}) = \arg\max_{y \in Y} \sum_{t:F(\mathbf{x})=y} \alpha_t$.
1 $F_0 = \mathbb{L}(S)$
  for $t = 1$ to $T$ do
2     Obtain vector $\mathbf{w}^t$ (†) of weights for each instance using a boosting method
3     Obtain an appropriate projection $S^t$ of the input space aimed at optimizing $\epsilon_t = E[\mathbf{w}^t_{(y \neq f(\mathbf{x}))}]$
4     $F_t = \mathbb{L}$(Projection of $S$ into $S^t$)
  end
†The exact way of obtaining $\mathbf{w}^t$ depends on the boosting algorithm used
```

Step 3 of the algorithm is the key point in our methodology. We must modify how the $k$-NN classifier at step $t$ views the training data, to bias that view for focusing on missclassified instances. Thus, we present here two methods that act on the columns of the training data, by means of feature selection and feature transformation. These two methods follow the general idea of Algorithm 1 and differ on how each one performs step 3.

The key idea for boosting $k$-NN is that this method is highly stable with respect to resampling, but unstable with respect to input selection. Thus, any method for boosting $k$-NN must focus on using boosting weight in a different way, not to resample or reweight instances but to modify the input space.[2]

The first of the two methods is based on searching for subspaces of the original input space where missclassified instances are more likely to be correctly classified. The second method is based on projecting the original variables into a new space using a supervised non-linear projection. The following sections explain each one of these two methods.

### 3.1. *k*-NN boosting by means of optimal subspace search

The use of different spaces for ensemble construction has been extensive in recent research. RSM has shown that the random feature selection improves accuracy without, in general, seriously affecting error rates. The experiments reported in this paper show how RSM is able to achieve good results, better than other ensemble methods such as bagging. However, RSM has some serious drawbacks. Random selection of features does not guarantee that the selected inputs have the necessary discriminant information. If such is the case, poor classifiers are obtained that damage the ensemble, especially when $k$-NN is used, as it is highly sensitive to subspace selection. In a previous work (García-Pedrajas & Ortiz-Boyer, 2008) we developed a new approach based on obtaining, before training a classifier, the subspace where the classification error weighted using the distribution obtained by boosting is minimized. In this way, we avoid training the classifier using a subspace without the needed discriminant information. Then, the new classifier added to the ensemble is trained using the obtained subspace and a new boosting step is performed. This method avoids the use of subspaces that are not able to discriminate between the classes and also obtains for each classifier the subspace that is more likely to get a good classification accuracy.

Thus, our approach is based on using different subspaces for each $k$-NN classifier, but these subspaces are not randomly chosen.

[2] $k$-NN is also sensitive to the distance function (Domeniconi, Peng, & Gunopulos, 2002; Hastie & Tibshirani, 1996), and previous works have focused in this feature to improve ensembles of $k$-NN classifiers (Bao, Ishii, & Du, 2004). This feature is not considered on this paper. Nevertheless, a similar approach to the one presented in this paper dealing with the distance function might be an interesting research line.

Instead, we select the subspace that minimizes the weighted error for each boosting step. A general outline of the method is shown in Algorithm 2. We have called this method *k-NN Not so Random Subspace Method* (*k*NN.NsRSM). The basic idea underlying the method is that for each boosting iteration, as the distribution of the instances is different, a different subsets of inputs should be relevant (Lazarevic et al., 2000). Our algorithm is aimed at finding that relevant subset of inputs.

**Algorithm 2**: Outline of *k*-NN not so random subspace method (*k*NN.NsRSM)

```
Data   : A training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, $\mathbf{x}_i \in \mathbb{R}^D$, a $k$-NN learn-
         ing algorithm, $\mathbb{L}$, and the number of iterations $T$.
Result : The final classifier: $F^*(\mathbf{x}) = \arg\max_{y \in Y} \sum_{t:F(\mathbf{x})=y} \alpha_t$.
1 $F_0 = \mathbb{L}(S)$
  for $t = 1$ to $T$ do
2     Obtain vector $\mathbf{w}_t$ (†) of weights for each instance using a boosting method
3     Obtain subspace $\mathbb{R}^{D_t}$, $D_t < D$, using weight vector $\mathbf{w}_t$
4     $F_t = \mathbb{L}$(Projection of $S$ into $\mathbb{R}^{D_t}$)
  end
†The exact way of obtaining $\mathbf{w}_t$ depends on the boosting algorithm used
```

Central to our method is how the optimal subspace is obtained. The problem can be stated as follows: *Obtain a subspace* $\mathbb{R}^{D_t}, D_t < D$, *where the weighted error* $\epsilon = E[\mathbf{w}_{(y \neq f(\mathbf{x}))}]$ *is minimized*. We may approach the problem as one of feature selection. That approach has two problems: first, most of the algorithms for feature selection do not admit weighting the instances, and second, the feature selection algorithm may not be appropriate for *k*-NN method, selecting inputs that do not improve its performance. So, we have opted for a wrapper approach, a subspace is evaluated using a *k*-NN method which considers only the inputs included in the subspace, and obtaining its weighted training error. This approach converts our problem into a combinatorial optimization problem. However, due to the large number of combinations, $2^n$ for an input space of $n$ dimensions, we cannot perform an exhaustive search. The solution must be the use of a combinatorial optimization algorithm.

Among the different optimization algorithms available, we have chosen a genetic algorithm because it is easy to implement, achieves good results and the authors are familiar with it.[3] In a binary coded genetic algorithm each solution is codified as a binary vector. Thus, in our problem each subspace is codified as a binary vector, $\mathbf{s}$, where $s_i = 1$ means that the corresponding $i$th input is used. The 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*, that 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*.

We have chosen a CHC genetic algorithm (Eshelman, 1990) because it is usually able to achieve good solutions using small populations. CHC stands for *Cross generational elitist selection, heterogeneous recombination and cataclysmic mutation*. Although in the standard CHC algorithm mutation is not used, we have introduced random mutation. Random mutation randomly modifies some of the bits of an individual. The process for obtaining the optimal subspace for each *k*-NN classifier using CHC method is shown in Algorithm 3.

**Algorithm 3**: Outline of the CHC genetic algorithm for obtaining the optimal subspace given a vector of weights for each instance

```
Data   : A training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, $\mathbf{x}_i \in \mathbb{R}^D$, a $k$-NN learn-
         ing algorithm, $\mathbb{L}$, and a weight vector $\mathbf{w}$ obtained using a boosting
         method.
Result : The optimal subspace $\mathbb{R}^d \subset \mathbb{R}^D, d < D$.
1 Initialize population
  for Number of generations do
2     Obtain new individuals using HUX crossover
3     Apply random mutation with probability $P_1$
4     Evaluate individuals
5     Select $N$ best individuals for the next generation
  end
```

As stated, the evaluation of the individuals follows a wrapper approach. In our population each individual represents a subspace of the input space. This individual is represented by a vector, $\mathbf{s}$, of 1's and 0's, where a 1 indicates that the corresponding input is used. Mathematically we can represent the projection into this subspace by a matrix, $\mathbf{P}$, whose main diagonal is $\mathbf{s}$ and the rest of the elements are 0. To evaluate an individual we apply a *k*-NN[4] classifier using the projected training set $S^P = \{\mathbf{z}_i, \mathbf{z}_i = \mathbf{P}\mathbf{x}_i\}$. Then the weighted accuracy, $E[\mathbf{w}_{(y=f(\mathbf{z}))}]$, of this classifier is obtained and this value is assigned as fitness to the individual.

### 3.2. Boosting by means of supervised projections

In the previous method we selected subspaces of the original input space to favor difficult instances. A second choice is modifying the input space by means of non-linear projections. In two previous papers (García-Pedrajas, García-Osorio, and Fyfe, 2007; García-Pedrajas, in press) we have shown how non-linear projections are an efficient tool for approaching boosting of classifiers. We avoid the harmful effect of maximizing the margin of noisy instances using the adaptive weighting scheme of boosting methods not to train the classifiers but to obtain a supervised projection that is the one that actually receives the classifier. The supervised projection is aimed at optimizing the weighted error given by $\mathbf{w}$. The classifier is then trained using this supervised projection with a uniform distribution of the instances. Thus, we obtain a method that benefits from the adaptive instance weighting of boosting and that can be applied to *k*-NN classifier. As in standard boosting methods, we construct an additive model:

$$F(\mathbf{x}) = \sum_{i=1}^{M} \alpha_i f_i(\mathbf{z}_i), \tag{3}$$

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

The presented methodology for constructing an ensemble of *k*-NN classifiers can be applied to most existing boosting methods. In the reported experiments we have used the variant of the

[3] As the form of the combinatorial optimization algorithm does not affect the proposed approach, any other algorithm, such as, simulated annealing or particle swarm optimization could be used as well.

[4] $k$ is obtained by $n$-fold cross-validation as it is explained in the experimental setup.

standard AdaBoost algorithm reported in Bauer and Kohavi (1999) to obtain instance distribution vector **w**. Algorithm 4 shows a general definition of the algorithm. The algorithm is named *k-NN Boosting based on Supervised Projections* (*k*-NN.BSP).

Algorithm 4: *k*-NN Boosting based on Supervised Projection (*k*NN.BSP) algorithm

```
Data   : A training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, a $k$-NN learning algorithm, $\mathbb{L}$, and the number of iterations $T$.
Result : The final classifier: $F^*(\mathbf{x}) = \arg\max_{y \in Y} \sum_{t:F(\mathbf{x})=y} \alpha_t$.
1  $F_0 = \mathbb{L}(S)$
   for $t = 1$ to $T$ do
2      Obtain vector $\mathbf{w}_t$ (†) of weights for each instance using a boosting method
3      Obtain supervised projection $\mathbf{P}_t(\mathbf{x})$ using weight vector $\mathbf{w}_t$
4      $F_t = \mathbb{L}(\mathbf{P}_t(S))$
   end
†The exact way of obtaining $\mathbf{w}_t$ depends on the boosting algorithm used.
```

We have stated that our method is based on using a supervised projection to train the classifier at round *t*. But, what exactly do we understand as a *supervised projection*? The intuitive meaning of a supervised projection using a weight vector $\mathbf{w}_t$, is a projection into a space where the weighted achieved by a *k*-NN classifier trained using the projection is minimized. More formally we can define a supervised projection as follows:

**Definition 1.** Supervised projection. Let $S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots (\mathbf{x}_n, y_n)\}$ be a set of $n$ training samples where each instance $\mathbf{x}_i$ belongs to a domain $X$, and $\mathbf{w}$ a vector that assigns a weight $\mathbf{w}_i$ to each instance $\mathbf{x}_i$. A supervised projection $\Phi$ is a projection into a new space where the weighted error $\epsilon_k = \sum_{j=1}^{n} w_k I(f(\Phi(\mathbf{x}_k)) \neq y_k)$ for a *k*-NN classifier $f(\mathbf{x})$ is minimized.

The intuitive idea is to find a projection that improves the weighted error of the classifier. However, the problem of obtaining a supervised projection is not trivial. Methods for projecting data are focused on the features of the input space, and do not take into account the labels of the instances, as most of them are specifically useful for non-labelled data and aimed at data analysis. Our approach is based on the use of the projection carried out by the hidden layer of a multilayer perceptron neural network when it is used for classification purposes and trained using vector **w** to weight the instances.

Each node of the hidden layer of a neural network performs a non-linear projection of the input vector. So, $\mathbf{h} = \mathbf{f}(\mathbf{x})$, and the output layer obtains its output from vector **h**. This projection performed by the hidden layer of a multilayer perceptron distorts the data structure and inter-instance distances (Lerner, Guterman, Aladjem, & Dinstein, 1999) in order to achieve a better classification.

Thus, the projection performed by the hidden layer focuses on making the classification of the instances easier. The number of hidden nodes of the network is the same as the number of input variables, so the hidden layer is performing a non-linear projection into a space of the same dimension as the input space. As the network has been trained with a distribution of the training instances given by the boosting algorithm, the projection performed by the hidden layer focuses on optimizing the weighted error given by vector **w**. Once the network is trained, the projection implemented by the hidden layer is used to project the training instances, and these projections are fed to the new *k*-NN classifier added to the ensemble. The proposed method for constructing the supervised projections is shown in Algorithm 5.

**Algorithm 5**: Algorithm for obtaining a supervised projection

```
Data   : A training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, and a vector of weights $\mathbf{w}$.
Result : A supervised projection: $\mathbf{P}(S)$.
1  $S' \equiv S$, with instances weighted by $\mathbf{w}$
2  Train network $H$ with $S'$ and get projection $\mathbf{P}(\mathbf{x})$ implemented by the hidden layer of $H$
3  $\mathbf{P}(S)$ = Projection performed by the hidden layer of $H$
```

The combination of the individual *k*-NN classifiers in both methods is made using a weighted voting approach. The weight of each vote is given by standard AdaBoost. Although there are other more sophisticated methods (Kittler & Alkoot, 2002) they are not able to consistently beat simple voting.

## 4. Experimental setup

For the comparison of the different methods we selected 45 datasets from the UCI Machine Learning Repository (Hettich, Blake, & Merz, 1998). A summary of these datasets is shown in Table 1.

**Table 1**
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 |

The experiments were conducted following the 5 × 2 cross-validation set-up (Dietterich, 1998). We perform five replications of a two-fold cross-validation. In each replication the available data is partitioned into two random equal-sized sets. Each learning algorithm is trained on one set at a time and tested on the other set.

Following Demšar (2006) we carry out in a first step an Iman–Davenport test, 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

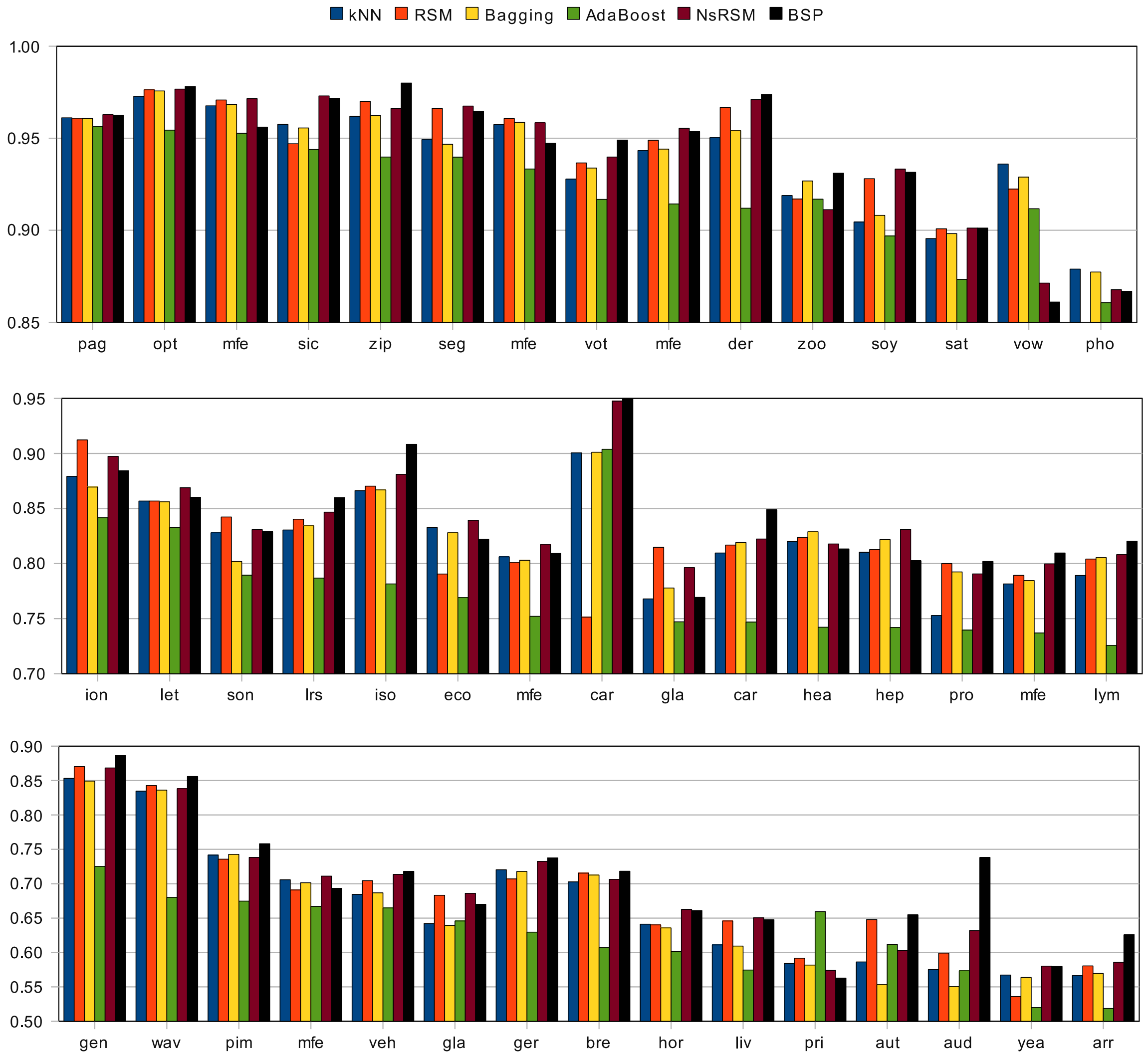

**Fig. 2.** Testing accuracy results for the standard and proposed methods. Datasets are ordered with respect to accuracy to allow a better plot.

**Table 2**
Comparison of results in terms of generalization error. 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 $p_w$).

| | | $k$-NN | $k$-NN RSM | $k$-NN Bagging | $k$-NN AdaBoost | $k$-NN.NsRSM | $k$-NN.BSP |
|---|---|---|---|---|---|---|---|
| Mean all | | 0.1942 | 0.1900 | 0.1939 | 0.2344 | 0.1810 | 0.1756 |
| $k$-NN | $s$ | | 34/0/11 | 17/1/27 | 3/0/42 | 40/0/5 | 37/0/8 |
| | $p_s$ | | 0.0008✗ | 0.1742 | 0.0000✗ | 0.0000✗ | 0.0000✗ |
| | $p_w$ | | 0.0002✗ | 0.1810 | 0.0000✗ | 0.0000✗ | 0.0000✗ |
| $k$-NN RSM | $s$ | | | 11/0/34 | 2/0/43 | 32/0/13 | 35/0/10 |
| | $p_s$ | | | 0.0008✗ | 0.0000✗ | 0.0066✗ | 0.0002✗ |
| | $p_w$ | | | 0.0002✗ | 0.0000✗ | 0.0155✗ | 0.0019✗ |
| $k$-NN Bagging | $s$ | | | | 4/0/41 | 37/0/8 | 36/0/9 |
| | $p_s$ | | | | 0.0000✗ | 0.0000✗ | 0.0001✗ |
| | $p_w$ | | | | 0.0000✗ | 0.0000✗ | 0.0000✗ |
| $k$-NN AdaBoost | $s$ | | | | | 42/0/3 | 44/0/1 |
| | $p_s$ | | | | | 0.0000✗ | 0.0000✗ |
| | $p_w$ | | | | | 0.0000✗ | 0.0000✗ |
| $k$-NN.NsRSM | $s$ | | | | | | 26/0/19 |
| | $p_s$ | | | | | | 0.3713 |
| | $p_w$ | | | | | | 0.2123 |

(Demšar, 2006). Iman–Davenport test is based on $\chi_F^2$ Friedman test (Friedman, 1940), which compares the average ranks of $k$ algorithms.

In all the tables the $p$-values of the corresponding tests are shown. The error measure is $\epsilon = E[1_{(y \neq f(\mathbf{x}))}]$. As a general rule, we consider a confidence level of 0.05. As further information of the

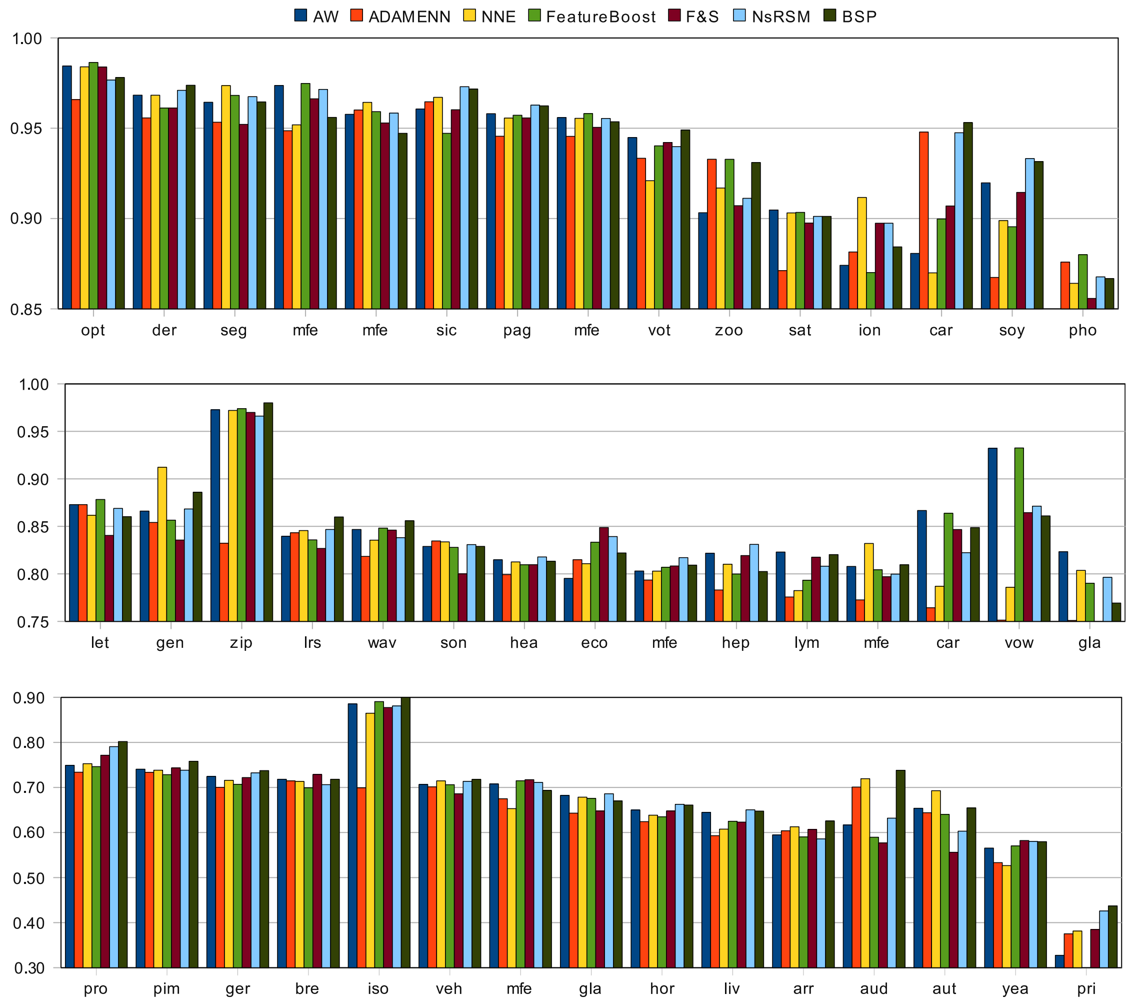

**Fig. 3.** Testing accuracy results for the ensemble and subspace based methods and proposed ones. Datasets are ordered with respect to accuracy to allow a better plot.

**Table 3**
Comparison of results in terms of generalization error. 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 $p_w$).

| | | $k$-NN. NsRSM | $k$-NN. BSP | AW | Adamenn | NNE | Feature Boost | F&S |
|---|---|---|---|---|---|---|---|---|
| Mean all | | 0.1810 | 0.1756 | 0.1849 | 0.2070 | 0.1888 | 0.1906 | 0.1923 |
| $k$-NN. NsRSM | $s$ | | 22/0/23 | 18/0/27 | 10/0/35 | 18/0/27 | 19/0/26 | 12/1/32 |
| | $p_s$ | | 1.0000 | 0.2327 | 0.0002 ✗ | 0.2327 | 0.3713 | 0.0037 ✗ |
| | $p_w$ | | 0.2293 | 0.2041 | 0.0000 ✗ | 0.0463 ✗ | 0.0324 ✗ | 0.0005 ✗ |
| $k$-NN. BSP | $s$ | | | 14/0/31 | 4/0/41 | 14/0/31 | 15/0/303 | 10/0/35 |
| | $p_s$ | | | 0.0161 ✗ | 0.0000 ✗ | 0.0161 ✗ | 0.0357 ✗ | 0.0002 ✗ |
| | $p_w$ | | | 0.0251 ✗ | 0.0000 ✗ | 0.0039 ✗ | 0.0073 ✗ | 0.0001 ✗ |
| AW | $s$ | | | | 11/3/31 | 17/0/28 | 17/0/28 | 13/0/32 |
| | $p_s$ | | | | 0.0029 ✗ | 0.1352 | 0.1352 | 0.0066 ✗ |
| | $p_w$ | | | | 0.0009 ✗ | 0.3941 | 0.0416 ✗ | 0.0463 ✗ |
| Adamenn | $s$ | | | | | 35/0/10 | 32/0/13 | 31/0/14 |
| | $p_s$ | | | | | 0.0002 ✗ | 0.0066 ✗ | 0.0161 ✗ |
| | $p_w$ | | | | | 0.0001 ✗ | 0.0021 ✗ | 0.0170 ✗ |
| NNE | $s$ | | | | | | 21/0/24 | 22/0/23 |
| | $p_s$ | | | | | | 0.7660 | 1.0000 |
| | $p_w$ | | | | | | 0.8700 | 0.9865 |
| Feature Boost | $s$ | | | | | | | 19/1/25 |
| | $p_s$ | | | | | | | 0.4514 |
| | $p_w$ | | | | | | | 0.3879 |

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 datasets. 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 learning task used in the experiments.

The source code, in C and licensed under the GNU General Public License, used for all methods as well as the partitions of the datasets are freely available upon request to the authors.

## 5. Experimental results

As we are proposing two boosting methods for *k*-NN classifiers, we must test these methods against known algorithms for constructing ensembles of *k*-NN classifiers as well as a *k*-NN classifier alone. Thus, we have chosen as base algorithms to compare our method a *k*-NN alone, ensembles of *k*-NN classifiers using RSM and bagging and an implementation of AdaBoost by means of resampling. As we have said, it is very unlikely that this last algorithm would be able to obtain good results, and it is included here for the sake of completeness.

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 dataset and for every classifier, so each time a *k*-NN algorithm is used, the value of *k* is previously obtained by 10-fold cross-validation of the training set.

Fig. 2 shows a bar plot of the results for the 45 datasets of the four standard methods and the two proposed ones in terms of testing accuracy. Table 2 shows the comparison among the 6 methods. For the four standard methods the Iman–Davenport test has a *p*-value of 0.000, showing significant differences between them. In all the comparison tables significant differences are marked with a ✗, 95% confidence, or ✓, 90% confidence.

The first noticeable fact is the poor performance of boosting *k*-NN by means of a resampling version of AdaBoost. This algorithm performs significantly worse than the rest of algorithms. Bagging performs just as well as *k*-NN alone. As we have said, this is a known fact, as *k*-NN is fairly stable to resampling of instances, ensembling bagged classifiers does not improve its performance. The best performing standard method is RSM, which is able to significantly outperform all the other three standard algorithms. Similar behavior of RSM and *k*-NN has been reported before (Bay, 1999).

Considering the best two standard algorithms and the two proposed approaches, Iman–Davenport test shows a *p*-value of 0.0000. The comparison in Table 2 shows that both methods, *k*NN.NsRSM and *k*NN.BSP are able to significantly outperform *k*-NN and RSM. The differences with the standard methods are very marked with win/loss records of 32/13 and 35/10 respectively for the worst case. These results show a marked advantage of the two proposed methods. The differences are all significant at a confidence level above 98%.

The previous results show the comparison between our methods and the most common ensemble methods. However, in the previous sections we have shown alternative ways of trying to improve *k*-NN performance. We performed an additional comparison between our methods and those intended to boost *k*-NN and those based on subspace methodologies. Namely, we have tried the attribute weighting (AT) method of Lazarevic et al. (2000), the Adamenn algorithm (Domeniconi et al., 2002), the nearest neighbor ensemble (NNE) based on Adamenn (Domeniconi & Yan, 2004), FeatureBoost algorithm (O'Sullivan et al., 2000), and the combination of AdaBoost and *k*-NN proposed by Freund and Schapire (1996) (F&S). The parameters chosen for these experiments were the suggested in the papers by the authors. As for the other methods, *k* was always obtained by cross-validation. Fig. 3 shows the testing accuracy of these methods together with the two proposed ones. Table 3 shows the comparison among these methods.

The comparison shows that *k*NN.BSP is still able to achieve a better performance than these more sophisticated methods. Furthermore, we must take into account that these methods, with the exception of AW, are more computationally demanding than *k*NN.BSP. With respect to *k*-NN.NsRSM, it is able to perform significantly better than all the methods, excepting AW. The differences

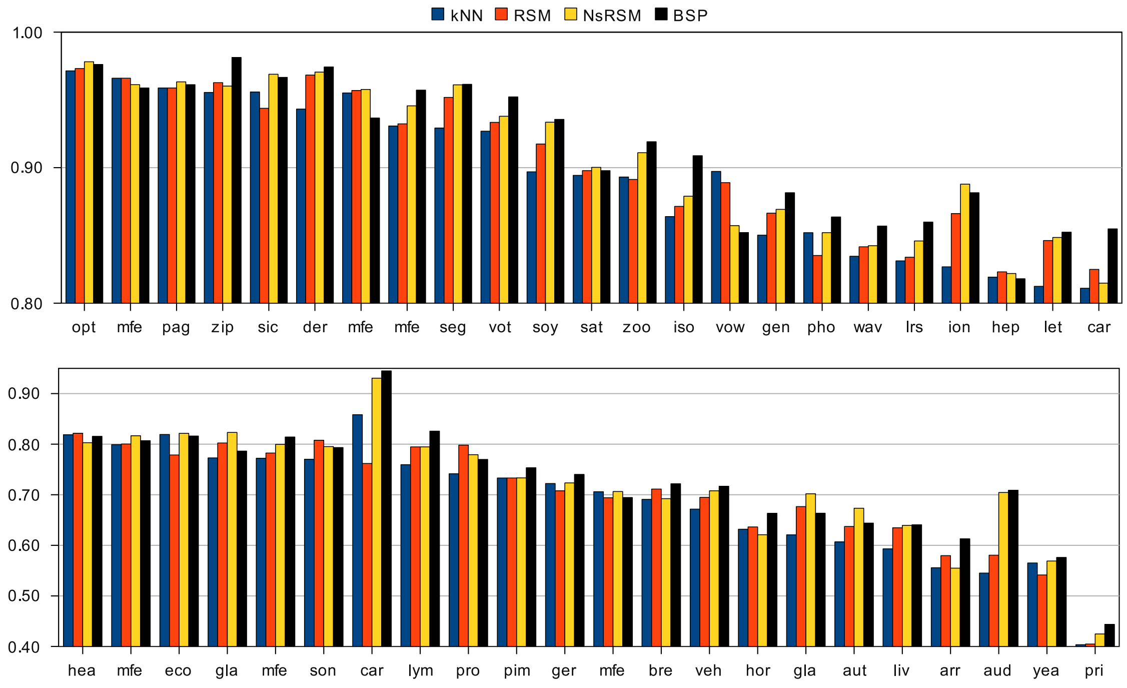

**Fig. 4.** Testing accuracy results for the best two standard methods and the two proposals for boosting *k*-NN for a noise level of 5%. Datasets are ordered with respect to accuracy to allow a better plot.

between *k*-NN.NsRSM and AW are not significant according to Wilcoxon test, although the win/loss record, 28/17, is favorable to *k*-NN.NsRSM.

## 5.1. Noise effect

Several researchers have reported that boosting methods, among them AdaBoost, degrade their performance in the presence of noise (Quinlan, 1996; Bauer & Kohavi, 1999). Dietterich (2000) tested this effect introducing artificial noise in the class labels of different datasets and confirmed this behavior. However, RSM method has been shown to be less affected by noise. Breiman (2001) reported that random forests were less affected by noise in the class labels of the instances. As our proposed methods share common ideas with RSM and AdaBoost it is interesting to study their behavior in the presence of noise. In this section we study

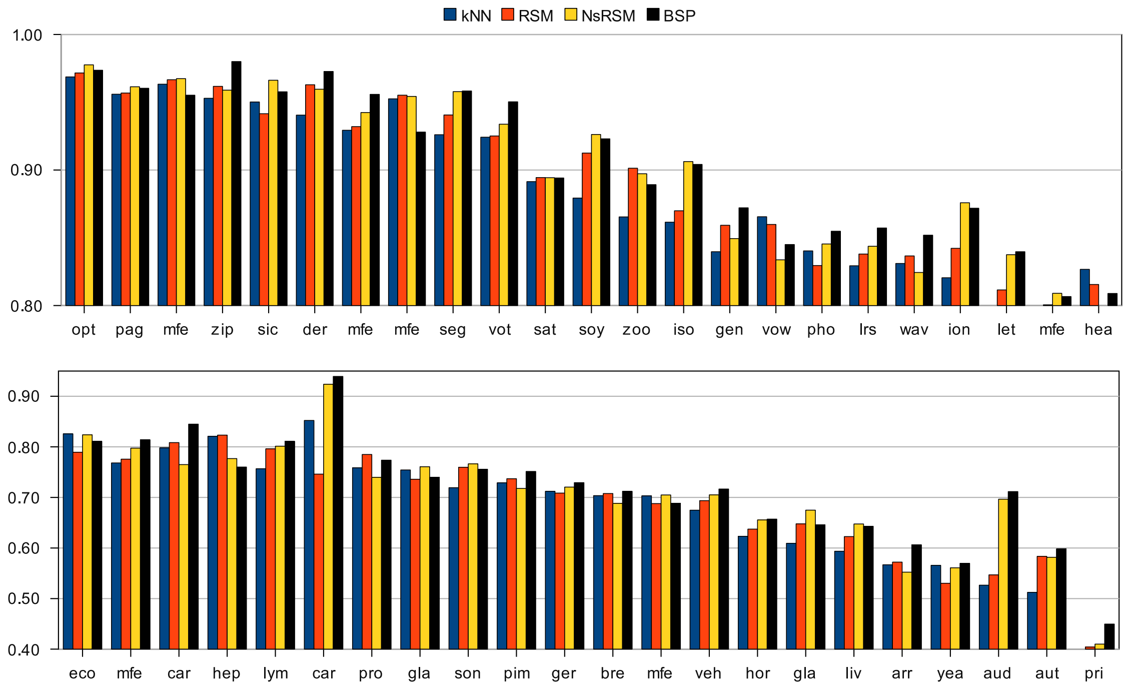

**Fig. 5.** Testing accuracy results for the best two standard methods and the two proposals for boosting *k*-NN for a noise level of 10%. Datasets are ordered with respect to accuracy to allow a better plot.

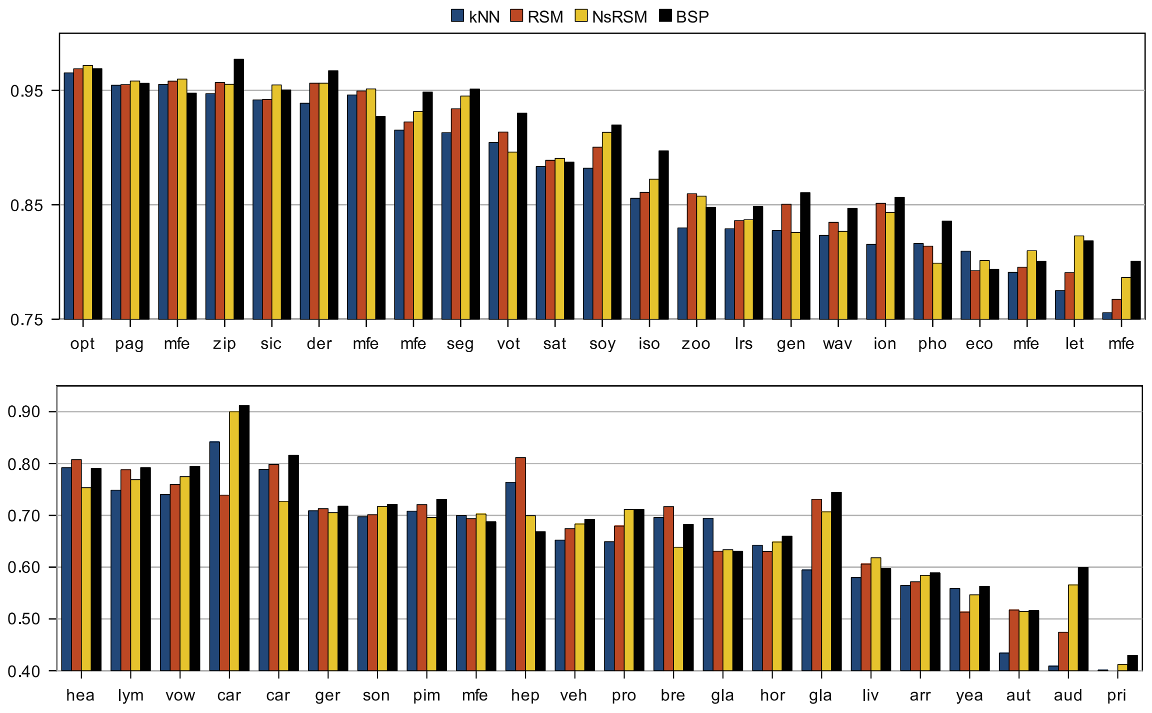

**Fig. 6.** Testing accuracy results for the best two standard methods and the two proposals for boosting *k*-NN for a noise level of 20%. Datasets are ordered with respect to accuracy to allow a better plot.

the sensitivity of our methods to noise and compared it with the best two standard algorithms.

To add noise to the class labels we follow the method of Dietterich (2000). 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 datasets and rates of noise of 5%, 10% and 20%. With these levels of noise we performed the experiments using the $5 \times 2$cv setup and the best two standard methods, *k*-NN and *k*-NN + RSM, and the two proposed methods. Figs. 4–6 show a bar plot of the results for the methods and the three levels of noise respectively, and Tables 4–6 show the comparison of the methods. Error ratio row shows the ratio of the average error using noise with respect to the same method applied to original datasets.

For a noise level of 5%, Table 4 shows that the two proposed methods are less affected by noise than *k*-NN and *k*-NN + RSM methods. The average increment of the error of *k*NN.NsRSM and *k*NN.BSP is below the increment of the two standard methods. Both proposed methods increment their error in about 2%, meanwhile *k*-NN increment its error in a 6% and *k*-NN combined with RSM almost a 4%. Furthermore, the two proposed methods also improve their win/loss record compared to standard methods.

For a noise level of 10%, Table 5, the behavior is similar but with the differences less marked. Although, the average increment of the error of *k*-NN.NsRSM and *k*NN.BSP is below the increment of the two standard methods, the differences are about 1% with RSM and 2% with *k*-NN alone. For a noise level of 20%, the standard methods behave slightly better than the proposed ones. It seems

**Table 4**
Comparison of results in terms of generalization error for a nose level of 5%. 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 $p_w$).

| | | *k*-NN | *k*-NN + RSM | *k*NN.NsRSM | *k*-NN.BSP |
|---|---|---|---|---|---|
| Mean all | | 0.2060 | 0.1971 | 0.1848 | 0.1787 |
| Error ratio | | 106.05% | 103.73% | 102.11% | 101.75% |
| *k*-NN | *s* | | 35/1/9 | 39/0/6 | 38/0/7 |
| | $p_s$ | | 0.0001 ✗ | 0.0000 ✗ | 0.0000 ✗ |
| | $p_w$ | | 0.0016 ✗ | 0.0000 ✗ | 0.0000 ✗ |
| *k*-NN+RSM | *s* | | | 33/1/11 | 36/0/9 |
| | $p_s$ | | | 0.0013 ✗ | 0.0001 ✗ |
| | $p_w$ | | | 0.0041 ✗ | 0.0001 ✗ |
| *k*NN.NsRSM | *s* | | | | 28/0/17 |
| | $p_s$ | | | | 0.1352 |
| | $p_w$ | | | | 0.0210 ✗ |

**Table 5**
Comparison of results in terms of generalization error for a noise level of 10%. 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 $p_w$).

| | | *k*-NN | *k*-NN + RSM | *k*NN.NsRSM | *k*-NN.BSP |
|---|---|---|---|---|---|
| Mean all | | 0.2140 | 0.2070 | 0.1959 | 0.1881 |
| Error ratio | | 110.18% | 108.97% | 108.24% | 107.09% |
| *k*-NN | *s* | | 35/0/10 | 34/0/11 | 37/0/8 |
| | $p_s$ | | 0.0002 ✗ | 0.0008 ✗ | 0.0000 ✗ |
| | $p_w$ | | 0.0044 ✗ | 0.0010 ✗ | 0.0000 ✗ |
| *k*-NN+RSM | *s* | | | 29/0/16 | 35/0/10 |
| | $p_s$ | | | 0.0725 ✓ | 0.0002 ✗ |
| | $p_w$ | | | 0.0383 ✗ | 0.0001 ✗ |
| *k*NN.NsRSM | *s* | | | | 27/0/18 |
| | $p_s$ | | | | 0.2327 |
| | $p_w$ | | | | 0.0223 ✗ |

**Table 6**
Comparison of results in terms of generalization error for a noise level of 20%.

| | | *k*-NN | *k*-NN + RSM | *k*-NN.NsRSM | *k*-NN.BSP |
|---|---|---|---|---|---|
| Mean all | | 0.2347 | 0.2228 | 0.2206 | 0.2093 |
| Error ratio | | 120.86% | 117.26% | 121.89% | 119.17% |
| *k*-NN | *s* | | 37/0/8 | 33/0/12 | 37/0/8 |
| | $p_s$ | | 0.0000 ✗ | 0.0025 ✗ | 0.0000 ✗ |
| | $p_w$ | | 0.0002 ✗ | 0.0069 ✗ | 0.0000 ✗ |
| *k*-NN+RSM | *s* | | | 28/1/16 | 34/1/10 |
| | $p_s$ | | | 0.0725 ✓ | 0.0004 ✗ |
| | $p_w$ | | | 0.2760 | 0.0004 ✗ |
| *k*NN.NsRSM | *s* | | | | 30/1/14 |
| | $p_s$ | | | | 0.0226 ✗ |
| | $p_w$ | | | | 0.0010 ✗ |

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 $p_w$).

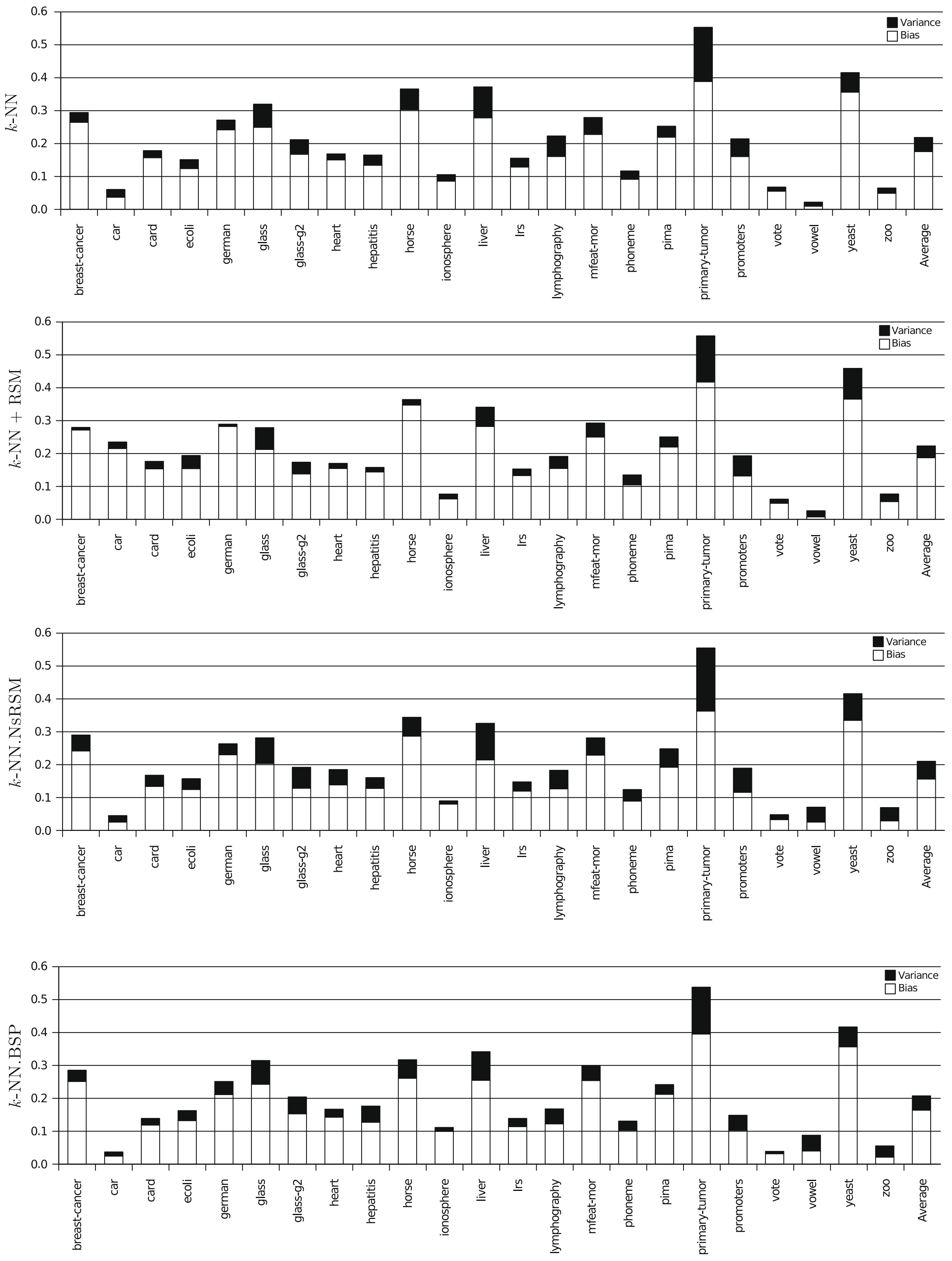

**Fig. 7.** Bias/variance decomposition of the methods for 23 datasets for *k*-NN, *k*-NN and RSM, *k*-NN.NsRSM, and *k* NN.BSP.

that at this high level of noise the boosting component of the proposed methods harms their performance. However, it is noticeable that even at this high level of noise two methods based on boosting are not significantly more damaged by noise than methods that do not use boosting. We must remark that with the exception of the comparison of *k*-NN and RSM and *k*-NN.NsRSM, all the differences between our two proposals and the standard algorithms are still significant in presence of noise at a confidence level of 95%.

## 5.2. Bias/variance decomposition

Many of the papers studying ensembles analyze error performance in terms of two factors: *bias* and *variance*. The bias of a

**Table 7**
*p*-value of the Wilcoxon test for the bias/variance terms of the error for the 23 datasets used for this experiment.

| | *k*-NN | *k*-NN+RSM | *k*-NN.NsRSM | *k*-NN.BSP |
|---|---|---|---|---|
| *Bias* | | | | |
| Mean all | 0.1757 | 0.1972 | 0.1563 | 0.1641 |
| *k*-NN | | 0.1529 | 0.0001 | 0.0285 |
| *k*-NN+RSM | | | 0.0003 | 0.0177 |
| *k*-NN.NsRSM | | | | 0.0885 |
| *Variance* | | | | |
| Mean all | 0.0430 | 0.0360 | 0.0540 | 0.0436 |
| *k*-NN | | 0.0658 | 0.0014 | 0.7843 |
| *k*-NN+RSM | | | 0.0003 | 0.0777 |
| *k*-NN.NsRSM | | | | 0.0011 |

learning algorithm is the contribution to the error of the central tendency when it is trained using different data, and the variance is the contribution to the error of the deviations from the central tendency. These two terms are evaluated with respect to a distribution of training sets $\mathscr{T}$ usually obtained by different permutations of the available data.

In addition, there is an *irreducible error* that is given by the degree to which the correct answer for a pattern can differ from that of other patterns with the same description. As this error cannot be estimated in most real world problems, the measures of bias and variance usually include part of this error.

Bias–variance decomposition is rooted in quadratic regression where averaging several independently trained regressors can never increase the expected error. In this framework, bias and variance are always nonnegative and averaging decreases variance without modifying bias. However, in classification, majority voting can yield an increase in expected error. This fact suggests that it is more difficult to obtain a bias–variance decomposition for classification as natural as for regression. In this way, several authors have suggested different proposals for estimating the decomposition of the classification error into bias and variance terms (Kong & Dietterich, 1995; Kohavi & Wolpert, 1996 & Breiman, 1996c).

Let us assume that the training pairs, $(\mathbf{x}, y)$ are drawn from a test instance distribution $X, Y$, and that the classification of pattern $\mathbf{x}$ by means of classifier $\mathscr{L}$ for a distribution of training data sets $\mathscr{T}$ is $\mathscr{L}(\mathscr{T})(\mathbf{x})$. We used the decomposition of Breiman (1996b) which is defined as:[5]

$$\begin{aligned}\text{bias}_B &= P_{(Y,X),\mathscr{T}}(\mathscr{L}(\mathscr{T}) \neq Y \wedge \mathscr{L}(\mathscr{T})(X) = C^0_{\mathscr{L},\mathscr{T}}(X)),\\ \text{variance}_B &= P_{(Y,X),\mathscr{T}}(\mathscr{L}(\mathscr{T}) \neq Y \wedge \mathscr{L}(\mathscr{T})(X) \neq C^0_{\mathscr{L},\mathscr{T}}(X)),\end{aligned} \tag{4}$$

where the central tendency, $C^0_{\mathscr{L},\mathscr{T}}(X)$, for learner $\mathscr{L}$ over the distribution of training data sets $\mathscr{T}$ is the class with the greatest probability of selection for pattern $\mathbf{x}$ by classifiers learned by $\mathscr{L}$ from training sets drawn from $\mathscr{T}$, and is defined:

$$C^0_{\mathscr{L},\mathscr{T}}(X) = \max_y P_{\mathscr{T}}(\mathscr{L}(\mathscr{T})(\mathbf{x}) = y). \tag{5}$$

For estimating these measures we have basically followed the experimental setup used in Webb (2000). We divided our data set into four randomly selected partitions. We selected each partition in turn to be used as the test set and trained the learner with the other three partitions. This method was repeated ten times with different random partitions, making a total of 40 runs of the learning algorithm.

The central tendency was evaluated as the most frequent classification for a pattern. The error was measured as the proportion of incorrectly classified patterns. This experimental setup guarantees that each pattern is selected for the test set the same number of times, and alleviates the effect that the random selection of patterns can have over the estimations.

Fig. 7 shows the bias–variance decomposition for *k*-NN alone, *k*-NN and RSM and the two proposed methods, *k*NN.BSP and *k*-NN.NsRSM. Only a subset of datasets is chosen for each base learner due to the high computational cost of estimating bias and variance, as 40 runs of the algorithms are required for each dataset. For *k*-NN classifier, we can see that the stability with respect to training set sampling directly translates into a low variance term, concentrating most of the error in the bias term. The behavior of *k*-NN combined with RSM yields a reduction of variance term, in fact, a Wilcoxon test on the bias/variance terms of the error shows that the variance term for *k*-NN and RSM is significantly smaller than the other three methods (see Table 7).

The two proposed methods behave in the same way with respect to bias term, both of them are able to significantly improve bias term of the error. However, with regard to variance term *k*NN.NsRSM obtains worse results, while *k*NN.BSP achieves a variance term similar to *k*-NN and not significantly worse than *k*-NN and RSM. Nevertheless, the bad behavior of the variance term of *k*NN.NsRSM is compensated with the best bias term of the four methods. As a summary, we may say that the good results of the two proposed methods are based on their ability to reduce bias term of error, which is the main source of error for *k*-NN based methods.

## 6. Conclusions and future work

In this paper we have proposed two approaches for boosting *k*-NN classifier. The usual method for boosting algorithms of modifying instance distribution during the learning process is not effective with *k*-NN classifiers, as this distribution is not easy to adapt to this classifier and, furthermore, *k*-NN is fairly stable with respect to instance selection.

Thus, we have proposed two methods that take advantage of the instability of *k*-NN with regard to input space modifications. The input space is modified, by input selection or projection, to favor difficult instances to be accurately classified. In this way, boosting of *k*-NN is achieved. We have shown in an extensive comparison using 45 real-world problems that the two proposed approaches are able to improve the performance of *k*-NN alone and the different ensemble methods used so far. The conclusions are based on an large comparison with 9 state-of-the-art methods.

A further study on noise effect on classifier performance has shown that the proposed methods are more robust in presence of noise at low level, 5%, and as robust as *k*-NN when the level of noise in the class labels is increased to 10% and 20%. In a final study we have shown that the proposed methods are able to improve *k*-NN performance by means of reducing bias term of the error.

[5] Although not reported here, similar conclusions about the behavior of the methods are obtained using either Kohavi and Wolpert or Kong and Dietterich measures.

## References

Altinçay, H. (2007). Ensembling evidential k-nearest neighbor classifiers through multi-modal perturbation. *Applied Soft Computing, 7*, 1072–1083.

Amores, J., Sebe, N., & Radeva, P. (2006). Boosting the distance estimation. Application to the k-nearest neighbor classifier. *Pattern Recognition Letters, 27*, 201–209.

Athitsos, V., & Sclaroff, S. (2005). Boosting nearest neighbor classiers for multiclass recognition. In *Proceedings of the IEEE computer society conference on computer vision and pattern recognition* (Vol. 3, pp. 45–52).

Bao, Y., Ishii, N., & Du, X. (2004). Combining multiple k-nearest neighbor classifiers using different distance functions. In: *Proceedings of the fifth international conference on intelligent data engineering and automated learning*. Lecture notes in computer science. UK: Springer, Exeter (Vol. 3177, pp. 634–641).

Bauer, E., & Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning, 36*(1/2), 105–142.

Bay, S. D. (1999). Nearest neighbor classification from multiple feature subsets. *Intelligent Data Analysis, 3*(3), 191–209.

Breiman, L. (1996a). Bagging predictors. *Machine Learning, 24*(2), 123–140.

Breiman, L. (1996b). *Bias, variance, and arcing classifiers*. Technical Report 460, Department of Statistics, University of California, Berkeley, CA.

Breiman, L. (1996c). Stacked regressions. *Machine Learning, 24*(1), 49–64.

Breiman, L. (2001). Random forests. *Machine Learning, 45*, 5–32.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research, 7*, 1–30.

Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation, 10*(7), 1895–1923.

Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning, 40*, 139–157.

Domeniconi, C., & Yan, B., 2004. Nearest neighbor ensemble. In: *Proceedings of the 17th international conference on pattern recognition* (ICPR'04) (Vol. 1, pp. 228–231).

Domeniconi, C., Peng, J., & Gunopulos, D. (2002). Locally adaptive metric nearest neighbor classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 24*(9), 1281–1285.

Eshelman, L. J. (1990). *The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination*. San Mateo, CA: Morgan Kauffman.

François, J., Grandvalet, Y., Denœux, T., & Roger, J. M. (2003). Resample and combine: An approach to improving uncertainty representation in evidential pattern classification. *Information Fusion, 4*, 75–85.

Freund, Y., & Schapire, R. (1996). Experiments with a new boosting algorithm. In *Proceedings of the thirteenth international conference on machine learning, Bari, Italy* (pp. 148–156).

Friedman, M. (1940). A comparison of alternative tests of significance for the problem of M rankings. *Annals of Mathematical Statistics, 11*, 86–92.

García-Pedrajas, N., García-Osorio, C., & Fyfe, C. (2007). Nonlinear boosting projections for ensemble construction. *Journal of Machine Learning Research, 8*, 1–33.

García-Pedrajas, N. (in press). Supervised projection approach for boosting classifiers. *Pattern Recognition*.

García-Pedrajas, N., & Ortiz-Boyer, D. (2008). Boosting random subspace method. *Neural Network, 21*, 1344–1362.

Grabowski, S. (2002). Voting over multiple k-NN classifiers. In *Proceedings of the international conference on modern problems of radio engineering, telecommunications and computer science* (pp. 223–225).

Hastie, T., & Tibshirani, R. (1996). Discriminant adaptive nearest neighbor classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 18*(6), 607–615.

Hettich, S., Blake, C., & Merz, C. (1998). UCI repository of machine learning databases, <http://www.ics.uci.edu/mlearn/MLRepository.html>.

Ho, T. K. (1998a). Nearest neighbors in random subspaces. In *Proceedings of the second international workshop on statistical techniques in pattern recognition, Syndey, Australia* (pp. 640–648).

Ho, T. K. (1998b). The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 20*(8), 832–844.

Jolliffe, I. T. (1986). *Principal components analysis*. New York, NY: Springer-Verlag.

Kittler, J., & Alkoot, F. M. (2002). Moderating k-NN classifiers. *Pattern Analysis & Applications, 5*(3), 326–332.

Kohavi, R. & Wolpert, D. H. (1996). Bias plus variance decomposition for zero–one loss functions. In L. Saitta (Ed.), *Machine learning: Proceedings of the thirteenth international conference* (pp. 275–283). Morgan Kaufmann.

Kong, E. B. & Dietterich, T.G. (1995). Error-correcting output coding corrects bias and variance. In: Prieditis, A., Lemmer, J.F. (Eds.), Machine Learning: Proceedings of the Twelfth International Conference. Elsevier Science Publishers, pp. 275–283.

Lazarevic, A., Fiez, T., & Obradovic, Z. (2000). Adaptive boosting for spatial functions with unstable driving attributes. In *Proceedings of the Pacific–Asia conference on knowledge discovery and data mining* (pp. 329–340). Japan: Springer-Verlag.

Lerner, B., Guterman, H., Aladjem, M., & Dinstein, I. (1999). A comparative study of neural networks based feature extraction paradigms. *Pattern Recognition Letters, 20*(1), 7–14.

O'Sullivan, J., Langford, J., Caruna, R., & Blum, A. (2000). Featureboost: A metalearning algorithm that improves model robustness. In *Proceedings of the seventeenth international conference on machine learning* (pp. 703–710).

Quinlan, J. R. (1996). Bagging, boosting, and c4.5. In *Proceedings of the thirteenth national conference on artificial intelligence* (pp. 725–730). AAAI Press and the MIT Press.

Todorovski, L., & Dzeroski, S. (2003). Combining classifiers with meta decision trees. *Machine Learning, 50*, 223–249.

Tremblay, G., Sabourin, R., & Maupin, P. (2004). Optimizing nearest neighbour in random subspaces using a multi-objective genetic algorithm. In: *Proceedings of the 17th international conference on pattern recognition* (Vol. 1, pp. 208–211).

Viswanath, P., Murty, M. N., & Bhatnagar, S. (2004). Fusion of multiple approximate nearest neighbor classifiers for fast and efficient classification. *Information Fusion, 5*, 239–250.

Webb, G. I. (2000). Multiboosting: A technique for combining boosting and wagging. *Machine Learning, 40*(2), 159–196.

Woods, K., Kegelmeyer, W., & Bowyer, K. (1997). Combination of multiple classiers using local accuracy estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 19*, 405–410.

Zhou, Z. H., & Yu, Y. (2005a). Adapt bagging to nearest neighbor classifiers. *Journal of Computer Science and Technology, 20*(1), 48–54.

Zhou, Z. H., & Yu, Y. (2005b). Ensembling local learners through multimodal perturbation. *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics, 35*(4), 725–735.