A general framework for boosting feature subset selection algorithms

Javier Pérez-Rodríguez[a], Aida de Haro-García[a], Juan A. Romero del Castillo[a], Nicolás García-Pedrajas[a,*]

[a] *University of Córdoba, Campus de Rabanales, 14011 Córdoba (Spain)*

Abstract

Feature selection is one of the most important tasks in many machine learning and data mining problems. Due to the increasing size of the problems, removing useless, erroneous or noisy features is frequently an initial step that is performed before other data mining algorithms are applied. The aim is to reproduce, or even improve, the performance of the data mining algorithm when all the features are used. Furthermore, the selection of the most relevant features may offer the expert valuable information about the problem to be solved.

Over the past few decades, many different feature selection algorithms have been proposed, each with its own strengths and weaknesses. However, as in the case of classification, it is unlikely that a single feature selection algorithm would be able to achieve good results across many different datasets and application fields. Furthermore, when we are dealing with thousands of features, the most powerful feature selection methods are frequently too time consuming to be applied. In classification, one of the most successful ways of consistently improving the performance of a single weak learner is to construct ensembles using boosting methods. In this paper, we propose a general framework for feature selection boosting in the same way boosting is applied to classification.

[☆] This work was supported in part by Project TIN2015-66108-P of the Spanish Ministry of Science and Innovation and Project P09-TIC-4623 of the Junta de Andalucía.

[*] Corresponding author

Email addresses: javier.perez@uco.es (Javier Pérez-Rodríguez), adeharo@uco.es (Aida de Haro-García), aromero@uco.es (Juan A. Romero del Castillo), npedrajas@uco.es (Nicolás García-Pedrajas)

The proposed approach opens a new field of research in which to apply the many techniques developed for boosting classifiers. Using 120 datasets, the experiments reported show a clear improvement in several state-of-the-art feature selection algorithms using the proposed methodology.
Keywords: Feature selection; Boosting; Classifier ensembles.

---

1. Introduction

Feature selection [1] is one of the most important and frequently used techniques in data mining [2, 3, 4, 5, 6]. Feature selection preserves the original semantics of the variables, hence offering the advantage of interpretability by a domain expert [7]. The problem of feature selection for feature-based learning can be stated as "the isolation of the smallest set of features that enable us to predict the class of a query feature with the same (or higher) accuracy than the original set" [8].

Feature selection has been a fertile field of research and development since the 1970's in statistical pattern recognition [9], [10], machine learning [2], [11], and data mining [12], [13], and it has been widely applied to many fields, such as text categorization [14], [15], image retrieval [16], [17] customer relationship management [18], intrusion detection [19], and genomic analysis [20].

Feature selection can be defined as the selection of a subset of m features from a set of $M = \{\phi_1, \phi_2, \ldots, \phi_M\}$ features, $m < M$, such that the value of a criterion function is optimized over all subsets of size m [21]. The objectives of feature selection are manifold, the most important ones being [7]:

- To avoid over-fitting and improve model performance, i.e., prediction performance in the case of supervised classification and better cluster detection in the case of clustering.
- To provide faster and more-cost-effective models.
- To gain deeper insight into the underlying processes that generated the data.

Another common task in data mining is classification. A classification problem of K classes and N training observations consists of a set of features whose class membership, or label, is known. Each label is an integer from the set $Y = \{1, \ldots, K\}$. A multi-class classifier is a function that maps a feature to an element of Y. The task is to find a definition for the unknown function that correctly maps each feature to its label. In a classifier ensemble framework, we have a set of classifiers instead of just one. One of the most successful methodologies for constructing ensembles of classifiers is boosting [22]. In boosting, new classifiers are trained by taking into account how difficult it is to accurately classify every pattern of the dataset.

In this paper, we propose that feature selection may be approached as a classification problem of two classes. A feature selection algorithm can be considered a classifier that maps a feature into one of two classes: "selected" or "unselected". With this view of feature selection, we can apply the philosophy of boosting and construct *ensembles* of feature selectors. We present the framework for adapting boosting to feature selection and constructing ensembles of feature selectors, and we show how this framework can be used with several common boosting methods. In classifier boosting, an ensemble of classifiers is constructed iteratively. Each new classifier focuses on the patterns that previous classifiers have found more difficult. Our approach constructs a combination of feature selection steps in a similar way. A feature selection algorithm is repeatedly applied. Each round focuses on the patterns found more difficult by the previous feature selection algorithms, thus searching for the subset of features able to correctly classify those difficult patterns. The different steps are combined by voting, which is common in boosting.

Informally, our approach proceeds as follows. First, a feature selection algorithm is applied using all available data with a uniform distribution of patterns. As a result of this algorithm, we obtain a subset of selected features. That is, every feature is classified into one of two classes, *selected* or *unselected*. Using this *classification* of the features, we perform an evaluation of the given selection using a certain classifier of our choice. The accurate or wrong classification of

each pattern modifies the weight of the corresponding pattern, as is the usual way in boosting. Then, a new sample is obtained from the training set and the feature selection process is repeated with this new sample. After the given number of iterations is performed, each step has classified every feature into selected or unselected. This result of each step is considered a vote, and as in boosting, this voting is the final result of the algorithm. Instead of a simple majority voting approach, to select a feature, we obtain a dynamical threshold optimizing a certain criterion using cross-validation.

This approach opens a new field of research in which all of the methods developed for constructing ensembles of classifiers can be applied. Furthermore, it allows for the incorporation of one of the most successful classification techniques, boosting, into another relevant task, feature selection.

This paper is organized as follows: Section 2 explains the proposed methodology; Section 3 reviews some related work; Section 4 details the experimental setup; Section 5 presents and discusses the results, and, finally, Section 6 presents the conclusions of our work and some future lines of research.

2. Boosting feature selection approach

A classification problem of K classes and N training observations consists of a set of features 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 patterns. Each label is an integer from the set $Y = \{1, \ldots, K\}$. A multi-class classifier is a function $f : X \to Y$ that maps a feature $\mathbf{x} \in X \subset \mathbb{R}^D$ to an element of Y.

The task is to find a definition for the unknown function $f(\mathbf{x})$ given the set of training features. In a classifier ensemble framework, we have a set of classifiers $F = \{f_1, f_2, \ldots, f_T\}$, each classifier performing a mapping of a feature vector $\mathbf{x} \in \mathbb{R}^D$ to the set of labels $Y = \{1, \ldots, K\}$.

As a general rule, boosting methods iteratively construct an ensemble of classifiers by modifying the distribution of patterns in the dataset. A weight vector, $\mathbf{w}$, is used, where w_i is a measure of how difficult accurate classification

of the pattern $\mathbf{x}_i$ is. Initially, all patterns receive the same weight, $w_i = 1/N$. Along the boosting process, weights are adapted by increasing the values of incorrectly classified patterns and decreasing the values of correctly classified patterns. Many different boosting methods exist. These differ in, among other aspects, the way $\mathbf{w}$ is updated. Once the process is finished and the T classifiers are constructed, a final ensemble of classifiers is obtained:

$$F(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t f_t(\mathbf{x}), \tag{1}$$

where α_t is a weight associated with the t-th classifier. This weight, which may be constant for all members of the ensemble, usually depends on the achieved classification accuracy of f_t.

Our approach to boosting feature selection is based on considering feature selection as a two-class classification problem and constructing an ensemble of these *classifiers* to perform the feature selection process. Proceeding as in classifier boosting, weights are initialized to a uniform distribution. Then, the feature selection algorithm is run with this uniform selection, and the set of selected features is recorded.

After each round of boosting feature selection, a classifier of our choice is trained using a non-uniform random distribution of the patterns, where difficult patterns receive more attention, and the features selected in the corresponding step. After the classifier is trained, the distribution of the patterns is updated by considering the error of the last classifier added, or the last few in some boosting methods, and a new round is performed. In each round, a feature selection algorithm is applied using the non-uniform distribution of patterns given by the boosting weights. After the selection process, the algorithm must update the distribution of patterns. To update the distribution, the method uses the subset of selected features obtained by the last feature selection process and classifies all of the training patterns using this subset and the given classification method. With the error obtained by this classification, the weights are updated, and a new round is performed.

Table 1 shows a summary of the comparison of our proposal with standard classifier ensemble construction using boosting. The table shows the similarities and differences between boosting classifier ensembles and "boosting" feature selectors. One major difference is the final model that is obtained. In classifier ensembles, we obtain as the final result a combination of different classifiers that have been constructed stepwise. However, in boosting feature selection, the final result is a subset of selected features that is obtained from the (possibly weighted) votes cast by the selection performed at each round.

Table 1: Step-by-step comparison of standard classifier boosting and feature selection boosting for T rounds, where S is the training set. Boosting uses the reweighting scheme.

	Classifier boosting
Step 1	Weights are initialized $w_i = 1/N$
Step 2	$t = 1$
Step 3	Train the classifier using the distribution given by $\mathbf{w}$
Step 4	Add the trained classifier to the ensemble
Step 5	Update weights using the scheme proposed by the boosting algorithm
Step 6	$t = t + 1$, if $t < T$ goto to Step 3; otherwise goto to Step 7
Step 7	Return ensemble of classifiers

	Feature selection boosting
Step 1	Weights are initialized $w_i = 1/N$
Step 2	$S' = S, t = 1$
Step 3	Perform the feature selection algorithm using the set S' to obtain $m_t \subset M$
Step 4	Register the selection performed by the feature selection algorithm
Step 5	Update weights using the scheme proposed by the boos ting algorithm and a given classifier with m_t, the subset of features selected in Step 3
Step 6	Sample S' from S with replacement using the distribution given by $\mathbf{w}$ and remove repeated features
Step 7	$t = t + 1$, if $t < T$ goto to Step 3; otherwise goto Step 8
Step 8	Obtain threshold of votes for keeping the features
Step 8	Perform feature selection using the obtained threshold of votes and return the set of selected features

Once the feature selection boosting process is finished, we have an ensemble of feature selectors similar to the ensembles of classifiers. One major difference

is that our ensemble "classifies" features into two classes—selected, class 1, or unselected, class 0—instead of classifying patterns, as in the case of classifier ensembles.

For the classification of a feature as either selected or unselected, at every boosting step, the selection made is recorded by casting a vote for each selected feature. These votes are weighted when the corresponding boosting algorithm uses weighted classifiers as members of the ensemble. Once the T rounds are finished, we have a vector of votes that records how many times every feature has been selected, and we must obtain a final selection using that vector. In boosting, majority voting is usually used, but in boosting feature selection, a fixed majority threshold did not achieve good results for all problems, so we opted for a dynamic threshold. This threshold is equivalent to the threshold used to distinguish between the positive and negative class in an ensemble of classifiers. The task of finding this threshold of votes is described in the next section.

2.1. Dynamic threshold of votes

The last step of our algorithm performs the feature selection that is the result of the process. After the T boosting steps, we have a vector of (possibly weighted) votes that records for each feature how often it has been selected. In ensembles of classifiers, the usual way of obtaining the class of a new feature is majority voting. By adapting this method to our framework, we could select features whose count of votes indicated that they had been selected more often than not. Thus, the output of the boosted selector using voting would be:

$$F^*(\phi_i) = \arg \max_{y \in \{0,1\}} \sum_{t:f_t(\phi_i)=y} \alpha_t. \tag{2}$$

where ϕ_i is the i-th feature, y is 1 if the feature selector decided in favor of the feature for round t-th, and α_t is the corresponding weight of the feature selector in the ensemble.

Because we are considering the classification of every feature into one of two

classes, selected (1) or unselected (0), we define for every feature two values, F_1^* and F_0^*, defined as follows:

$$F_1^*(\phi_i) = \sum_{t:f_t(\phi_i)=1} \alpha_t, \tag{3}$$

and

$$F_0^*(\phi_i) = \sum_{t:f_t(\phi_i)=0} \alpha_t. \tag{4}$$

Using the standard majority voting approach of boosting, a feature ϕ_i would be selected if $F_1^*(\phi_i) > F_0^*(\phi_i)$. Although this method is simple and fast, it does not achieve good results, as it is very unlikely that any static threshold would be appropriate for all datasets. Thus, we have opted to use a dynamic threshold where the decision of keeping a feature is given by $F_1^*(\phi_i) - F_0^*(\phi_i) > \Theta$, where Θ is obtained from the training set.

The procedure for obtaining the optimal value of Θ is as follows. Given a final vector of votes $\mathbf{v}$, which records the votes obtained for every feature $v_i = F_1^*(\phi_i)$, a certain threshold, Θ, selects the features whose records of votes are above this threshold: $v_i > \Theta$, which is equivalent to $F_1^*(\phi_i) - F_0^*(\phi_i) > \Theta$. To obtain the best threshold, we define a criterion, $J(\Theta)$, and evaluate all of the possible thresholds.

The process is as follows. To evaluate a certain threshold Θ, we first select the subset, $m_\Theta \subset M$, of features from the training set using Θ: $m_\Theta = \{\phi_i \in M : v_i > \Theta\}$. Then, the criterion $J(\Theta)$ is evaluated using m_Θ to learn a certain classifier; in the reported experiments we used a decision tree and a support vector machine to test our approach in two different scenarios. For J, we have chosen the most common criterion in feature selection, that is, the goodness of a certain selection is determined by two factors: its classification performance and its reduction ability. Our criterion is composed of these two factors:

$$J(\Theta) = \alpha \cdot \text{“classification performance”} + (1-\alpha) \cdot \text{“reduction”}. \tag{5}$$

Classification performance is composed of two terms, accuracy and κ. These two terms are summed, so the classification performance factor has twice the weight of the reduction factor. In the experiments, we will show the effect of the value of α on the behavior of the method. Reduction is measured as the percentage of removed features. In most cases, the classification performance is more desirable than reduction; therefore, in the reported experiments, a value of $\alpha = 0.5$ was used. The procedure used to obtain the threshold of votes is given by Algorithm 1.

Algorithm 1: Algorithm to obtain the vote threshold Θ.

Data : A training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, a vector of votes, $\mathbf{v}$, the set of possible thresholds $\{\Theta_1, \Theta_2, \ldots, \Theta_n\}$ and a classifier C.

Result : The threshold Θ with the best evaluation.

$J_{\text{best}} = 0$

for $\theta = \Theta_1$ *to* Θ_n **do**

  $m_\theta = \{\phi_i \in M : v_i > \theta\}$

  /* Classifier performance */

  c = Classifier performance of C using m_θ as the subset of selected features

  $\kappa = \kappa$measure of C using m_θ as the subset of selected features

  /* Reduction performance */

  $r = 1 - \frac{|m_\theta|}{M}$

  $J(\theta) = \alpha(c + \kappa) + (1 - \alpha)r$

  if $J(\theta) \geq J_{best}$ **then**

    $J_{\text{best}} = J(\theta)$

    $\Theta = \theta$

Return Θ

The set of possible thresholds, $\{\Theta_1, \Theta_2, \ldots, \Theta_n\}$, includes all the different values in the vector of votes. However, to save execution time, a subset of this set of possible values may be considered because thresholds with similar values will obtain very similar evaluations in terms of both classification performance and reduction. Furthermore, the number of votes is limited by the number of boosting rounds T, so it is always a small value.

This evaluation step is similar to the wrapper philosophy of feature selection. Thus, in addition to incorporating boosting into feature selection, we add the advantages of wrapper algorithms for feature selection to any other method of feature subset selection.

2.2. Boosting feature selection algorithms

In this section, we present a detailed description of the boosting methods used in the experiments. Because we are boosting feature selection algorithms and not classification methods, not all boosting algorithms developed for classification can be adapted. Usable boosting algorithms must fulfill two conditions:

1. The input space should not be modified. The final selection of features will be performed in the space of the original variables; methods of constructing ensembles that modify the input space should not be used.
2. The outputs should not be modified. The learning process is one of feature selection and not of classification. Thus, any method modifying the target output will make no sense.

Among the many boosting methods that can be used, we have selected five different algorithms: AdaBoost, FloatBoost, GentleBoost, MultiBoost and ReweightBoost.

All of the boosting algorithms must evaluate the error at every step using the vector of weights $\mathbf{w}$ and the current subset of selected features m_t. This evaluation is performed using a given classifier C trained using the selection performed in the corresponding step t. In the algorithms below, such a classifier is represented using $C(m_t)$.

We used AdaBoost version [23] shown in Algorithm 2. This is a standard extension of the original AdaBoost algorithm to multi-class problems.

The FloatBoost [24] algorithm is shown in Algorithm 3. FloatBoost tests each member of the ensemble after adding a new one. Members that are no longer useful are removed. This process is continued until the ensemble is completed or after a certain performance threshold is reached. However, we did not use a predefined threshold because it is not possible to set a predefined performance threshold for every dataset. Instead, we set a maximum number of attempts; if, after that number of attempts, the ensemble is not completed, the algorithm exits with the best current ensemble.

Algorithm 2: AdaBoost.FS algorithm.

Data : A training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, a set of features $M = \{\phi_1, \phi_2, \ldots, \phi_M\}$, a feature selection algorithm $FS(S, M)$ and a classifier C.

Result : The subset of selected features $m \subset M$.

$S' = S$, with all pattern weights set to $w_i = 1/N$
$\mathbf{v} = \{0, 0, \ldots, 0\}$
for $t = 1$ *to* T **do**
    Apply feature selection algorithm: $m_t = FS(S', M)$
    $v_i = v_i + 1$ if feature $\phi_i \in m_t$
    /* $C(m_t)$ is the classifier trained using only the set of features m_t */
    Store $\epsilon_t = \sum_{\mathbf{x}_j \in T: C(m_t)(\mathbf{x}_j) \neq y_j} w_j$
    if $\epsilon_t > 0.5$ **then**
        Set $\alpha_t = 0$
        Set S' to a bootstrap sample from S and reset $\mathbf{w}$ such that $w_i = 1/N$
        Remove repeated patterns from S'
    else if $\epsilon_t = 0$ **then**
        Set $\alpha_t = 10$
        Set S' to a bootstrap sample from S and reset $\mathbf{w}$ such that $w_i = 1/N$
        Remove repeated patterns from S'
    else
        Set $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
        foreach $\mathbf{x}_j \in S$ **do**
            if $C(m_t)(x_j) = y_j$ **then**
                $w_j = w_j / 2(1 - \epsilon_t)$
            else
                $w_j = w_j / 2\epsilon_t$
        Normalize $\mathbf{w}$, such as $\sum w_i = 1$
        Obtain S' as a sample with replacement from S using weights $\mathbf{w}$
        Remove repeated patterns from S'
Obtain threshold Θ that maximizes criterion $J(\Theta)$ (see Algorithm 1)
Obtain $m = \{\phi_i \in M : v_i > \Theta\}$

FloatBoost uses a backtrack mechanism after each iteration of AdaBoost to remove weak classifiers, which cause higher error rates. The resulting floatboosted ensemble consists of fewer weak classifiers, yet it usually achieves lower error rates than AdaBoost in both training and testing.

Algorithm 3: FloatBoost.FS algorithm.

Data : A training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, a set of features $M = \{\phi_1, \phi_2, \ldots, \phi_M\}$, a feature selection algorithm $FS(S, M)$ and a classifier C.

Result : The subset of selected features $m \subset M$.

1 $S' = S$, with all feature weights $w_i = 1/c \cdot N_{y_i}$, where c is the number of classes and N_j the features in class j
2 $H_0 = \emptyset$
3 $\epsilon^{min} = \{1, 1, \ldots, 1\}$
4 $\mathbf{v} = \{0, 0, \ldots, 0\}$
5 **for** $t = 1$ *to* T **do**
6 Apply feature selection algorithm: $m_t = FS(S', M)$
7 $v_i = v_i + 1$ if feature $\phi_i \in m_t$
/* $C(m_t)$ is the classifier trained using only the set of features m_t */
8 $H_t = H_{t-1} + C(m_t)$
9 $\epsilon_t = \sum_{\mathbf{x}_j \in S: H(\mathbf{x}_j) \neq y_j} w_j$
10 Update weights $w_i = \exp(-y_i H(\mathbf{x}_i))$ and $\sum w_i = 1$
11 **if** $\epsilon_t^{min} > \epsilon_t$ **then**
12 $\epsilon_t^{min} = \epsilon_t$
13 /* Conditional exclusion */
14 $h' = \arg\min_{h \in H_t} \epsilon(H_t - h)$
15 **if** $\epsilon(H_t - h') < \epsilon_{m-1}^{min}$ **then**
16 $H_{m-1} = H_t - h'$
17 $\epsilon_{m-1}^{min} = \epsilon(H_t - h')$
18 $t = t - 1$
19 goto 14
20 **else**
21 **if** *Maximum number of attempts* **then** goto 26
/* margin($\mathbf{x}$, H) is the margin of feature $\mathbf{x}$ when classified with ensemble H */
22 $w_i = \exp(-\text{margin}(\mathbf{x}_i, H_t))$
23 Normalize $\mathbf{w}$, such as $\sum w_i = 1$
24 Obtain S' as a sample with replacement from T using weights $\mathbf{w}$
25 Remove repeated patterns from S'
26 Obtain threshold Θ that maximizes criterion $J(\Theta)$ (see Algorithm 1)
27 Obtain $m = \{\phi_i \in M : v_i > \Theta\}$

The GentleBoost or Gentle AdaBoost algorithm [25] modifies the update of the pattern weights of AdaBoost. Lines 19 and 21 in Algorithm 2 are substituted by $w_j = w_j exp(-y_i C(m_t)(\mathbf{x}_j))$.

The MultiBoost [23] method is shown in Algorithm 4. MultiBoost periodically reinitializes the weights of the features using a Poisson distribution. To set the steps when the weights are reinitialized, a vector of integers, I, is constructed. The vector values are $I_i = i \times M/n$ for $i = 1, 2, \ldots, n-1$ and $I_i = M$ for $i = n, n+1, \ldots, \infty$, where $n = \lfloor \sqrt{M} \rfloor$. Thus, MultiBoost can be regarded

as a combination of AdaBoost and Bagging.

Algorithm 4: MultiBoost.FS algorithm.

```
Data   : A training set S = {(x_1, y_1), ..., (x_N, y_N)}, a set of features M = {φ_1, φ_2, ..., φ_M}, a
         feature selection algorithm FS(S, M) and a classifier C.
Result : The subset of selected features m ⊂ M.
S' = S, with all feature weights w_i = 1/N
Set k = 1
for t = 1 to T do
    if I_k = t then
        Reset w to random weights from a continuous Poisson distribution and normalize ∑ w_i = 1
        Set k = k + 1
    Apply feature selection algorithm: m_t = FS(S', M)
    v_i = v_i + 1 if feature φ_i ∈ m_t
    /* C(m_t) is the classifier trained using only the set of features m_t                    */
    ε_t = ∑_{x_j ∈ S : C(m_t)(x_j) ≠ y_j} w_j
    if ε_t > 0.5 then
        Set α_t = 0
        Reset w to random weights from a continuous Poisson distribution and normalize ∑ w_i = 1
        Set k = k + 1
    else if ε_t = 0 then
        Set α_t = 10
        Reset w to random weights from a continuous Poisson distribution and normalize ∑ w_i = 1
    else
        Set α_t = (1/2) ln (1 − ε_t)/ε_t
        foreach x_j ∈ S do
            if C(m_t)(x_j) = y_j then
                w_j = w_j / 2(1 − ε_t)
            else
                w_j = w_j / 2ε_t
            if w_j < 10^{-8} then w_j = 10^{-8}
        Normalize w such as ∑ w_i = 1
    Obtain S' as a sample with replacement from T using weights w
    Remove repeated features from S'
Obtain threshold Θ that maximizes criterion J(Θ) (see Algorithm 1)
Obtain m = {φ_i ∈ M : v_i > Θ}
```

Finally, we used the ReweightBoost method, which is shown in Algorithm 5. The main novelty of ReweightBoost is that instead of considering only the last added classifier as the weak learner to obtain the weights for the next step, it considers the ensemble formed by the last r classifiers, where r must be fixed by the user. However, this modification has the problem of making the algorithm more time-consuming because at each step, instead of checking the last classifier, we must consider the subensemble formed by the last r classifiers.

Similar to classifier boosting, feature selection boosting requires a feature selection algorithm as the base method for obtaining the selected features in each round. In the experiments reported, we will present the results for five

Algorithm 5: ReweightBoost.FS algorithm.

Data : A training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, a set of features $M = \{\phi_1, \phi_2, \ldots, \phi_M\}$, a feature selection algorithm $FS(S, M)$ and a classifier C.

Result : The subset of selected features $m \subset M$.

$S' = S$, with all feature weights $w_i = 1/N$

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

  Apply feature selection algorithm: $m_t = FS(S', M)$

  $v_i = \mathbf{v}_i + 1$ if feature $\phi_i \in m_t$

  /* $C(m_t)$ is the classifier trained using only the set of features m_t */

  /* r is a parameter that must be fixed by the user */

  Obtain combined classifier $H_t^r = C^{m_t} + C^{m_{t-1}} + \ldots + C_{\max(t-r,1)}$

  $\epsilon_t = \sum_{\mathbf{x}_j \in T : C(m_t)(\mathbf{x}_j) \neq y_j} {}^{w_j}$

  Set $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$

  foreach $\mathbf{x}_j \in T$ **do**

    /* $I(\cdot)$ is an indicator function, $I(\texttt{true}) = 1, I(\texttt{false}) = 0$ */

    $w_j = w_j \exp(-\alpha_t I(C(m_t)(x_j) \neq y_j))$

  Normalize $\mathbf{w}$, $\sum w_i = 1$

  Obtain S' as a sample with replacement from S using weights $\mathbf{w}$

  Remove repeated features from S'

Obtain threshold Θ that maximizes criterion $J(\Theta)$ (see Algorithm 1)

Obtain $m = \{\phi_i \in M : v_i > \Theta\}$

different feature subset selection algorithms. As a naming convention, we will use the name of the boosting algorithm and feature selection algorithm together to designate the use of a boosting method with a feature selection algorithm. For instance, AdaBoost.X indicates the use of the AdaBoost algorithm to boost the X feature selection method.

2.3. Feature selection methods

Our approach has two basic components, the boosting scheme and the base feature selection algorithm to be used. To allow for a fair assessment of the validity of our proposal, we must compare our approach with state-of-the-art methods for feature selection. We have chosen the following methods:

1. FAST (fast clustering-based feature selection algorithm) [26]: The FAST algorithm works in two steps. In the first step, features are divided into clusters by using graph-theoretic clustering methods. In the second step, the most representative feature that is strongly related to target classes is selected from each cluster to form a subset of features. Features in different clusters are relatively independent; the clustering-based strategy of FAST has a high probability of producing a subset of useful and independent

features. To ensure the efficiency of FAST, the efficient minimum-spanning tree (MST) clustering method is used.

2. FCBF (Fast Correlation-Based Filter) [27]: Existing feature selection methods mainly focus on finding relevant features. The authors showed that feature relevance alone was insufficient for efficient feature selection of high-dimensional data and they defined feature redundancy and proposed to perform explicit redundancy analysis in feature selection. A new framework was introduced that decoupled relevance analysis and redundancy analysis. They developed a correlation-based method for relevance and redundancy analysis.
3. LVF (Las Vegas Filter) [28]: The LVF algorithm generates subsets, S, from N features in every round. If the number of features is less than the current best, then the training set D with the features in S is checked against an inconsistency criterion. If the inconsistency criterion is below a certain threshold, then S becomes the new current best feature subset.
4. ReliefF [29]: The relief family of algorithms constitutes general, efficient and successful attribute estimators that do not assume the independence of the attributes, and their quality estimates have a natural interpretation. A key idea of the original Relief [30] algorithm is to estimate the quality of attributes according to how well their values distinguish among patterns that are near each other. Features whose weights exceed a user-determined threshold are selected in designing the classifier. The ReliefF algorithm is not limited to two class problems, it is more robust, and it can deal with incomplete and noisy data.
5. SetCover [31]: Class separability is normally used as the basic feature selection criterion. Instead of maximizing the class separability, SetCover adopts a criterion aiming to maintain the discriminating power of the data describing its classes. In other words, the problem is formalized as that of finding the smallest set of features that is "consistent" in describing classes. The new feature selection algorithm is based on Johnson's [32] algorithm for set covering. Johnson's analysis implies that this algorithm

runs in polynomial time, and it outputs a consistent feature set whose size is within a log factor of the best possible.

3. Related work

We must bear in mind that we are not using boosting as a method for feature selection but are developing a general framework for feature selection boosting. Previous works [33, 34] have used different variants of boosting for feature selection purposes. Yin et al. [35] also used the principles of boosting to improve feature combination. At each round of their variant of boosting method, some weak classifiers are built on different feature sets, one of which is trained on one feature set. Then, these classifiers are combined by weighted voting into a single output classifier of this round. Choi et al. [36] combined feature selection with boosting for face recognition.

Miao et al. [37] developed a method called BoostFS with combined AdaBoost with decision stumps to obtain a subset of selected features. BoostFS maintains a distribution over training samples that is initialized from the uniform distribution. In each iteration, a decision stump is trained under the sample distribution, and then the sample distribution is adjusted so that it is orthogonal to the classification results of all the generated stumps. Because a decision stump can also be regarded as one selected feature, BoostFS is capable of selecting a subset of features that are irrelevant to each other, as much as possible.

In the same way, feature selection and boosting have also been combined several times with the aim of improving the performance of the resulting classifier ensemble, as in the work of Redpath and Lebart [38], the FeatureBoost algorithm [39] or robust twin boosting feature selection (RTBFS) [40]. Bailly and Milgram [41] combined a fuzzy feature selection criterion with the instance weights given by a neural network classification error to construct a feature selection method using those two modules. The major difference with other approaches is the evaluation of the fuzzy criterion using a non-uniform distribution of the instances in the same way as boosting. García-Pedrajas and

Ortiz-Boyer [42] developed an ensemble method, based on the Random Subspace Method [43], that combined feature selection and the construction of the ensemble by means of an evolutionary algorithm.

Liu et al [44] developed a combination of vector boosting [45] and feature selection methods to tackle the task of adaptive compressive tracking. Another boosting and feature selection combination included the use of the Brownboost [46] loss function as a criterion for feature selection [47]. Yu et al. [48] proposed the Progressive Subspace Ensemble Learning approach (PSEL). PSEL adopts the random subspace technique to generate a set of subspaces. Each subspace is used to train a classifier in the original ensemble. Then, a progressive selection process is adopted to select the classifiers based on two cost functions which incorporate current and long-term information. Finally, a weighted voting scheme is used to combine the predicted results from individual classifiers of the ensemble, and generate the final result. Yu et al. [49] designed a general hybrid adaptive ensemble learning framework (HAEL) for constructing ensembles based on the use of different subspaces. HAEL adopts a hybrid adaptive approach to adjust the weights of the base classifiers and also further improves the performance by optimizing the random subspace set. In that way, it combines the concepts of feature selection and ensemble construction. However, there is a fundamental difference between all these approaches and our proposal, while these methods construct ensembles of classifiers where each classifier may use a different subset of features, our method obtains a selection of the most relevant features.

Other previous works have used classifier boosting concepts to improve the performance of feature selection. Liu et al. [50] developed a filter method for feature selection that used the weighting scheme of AdaBoost to improve the performance. They used the information evaluation criterion at each step of the selection algorithm, but instead of considering a uniform distribution of instances, each instance has its weight updated after each feature is removed from the pool of features by the selection process.

In the area of clustering there have been also some papers combining the

concepts of clustering and subspace selection. Yu et al. [51] constructed a feature selection based semi-supervised cluster ensemble framework (FS-SSCE) for tumor clustering from bio-molecular data. The method combines clustering performed on different subspaces which are obtained using a feature selection algorithm. As in the previous case, the difference is that the task carried out is one of clustering not of feature selection.

To the best of our knowledge, no previous work has developed a general framework to apply boosting to feature selection, such as the framework proposed in this paper.

4. Experimental setup

To make a fair comparison between standard feature selection algorithms and our proposed approach, we have selected a set of 120 datasets with a wide ranging number of patterns, features and classes. A summary of these datasets is provided in Table 2. To estimate the storage reduction and classifier performance, we used 10-fold cross-validation.

Table 2: Summary of datasets characteristics.

	Dataset	Cases	Features	Classes
1	abalone	4,177	10	29
2	ads	3,279	1,558	2
3	adult	48,842	105	2
4	all-aml	72	7,129	2
5	anneal	898	59	5
6	arabidopsis	33,971	4,016	2
7	arrhythmia	452	279	13
8	audiology	226	93	24
9	autos	205	72	6
10	balance	625	4	3
11	barleyblumeria	4,340	64	2
12	basehock	1,993	4,862	2
13	biodeg	1,055	41	2
14	breast-cancer	286	15	2
15	breast-cancer-ma	97	24,481	2
16	cancer	699	9	2
17	car	1,728	16	4
18	card	690	51	2
19	ccds	36,449	4,016	2
20	census	29,926	409	2
21	central-nervous-system	60	7,129	2
22	chrom21	12,677	1,612	2
23	cmc	1,473	9	3
24	cnae-9	1,080	856	9
25	colon-tumor	62	2,000	2
26	cottonmeloidogyne	4,156	64	2
27	cyclopentane	271	128	2
28	dbworld	64	7,402	2
29	dermatology	366	34	6
30	dexter	600	20,000	2
31	dlbcl-harvard-outcome	58	7,129	2
32	dlbcl-harvard-tumor	77	7,129	2
33	dlbcl-nih	240	7,399	2
34	dlbcl-stanford	47	4,026	2
35	dna	50,000	800	2
36	ecoli	336	7	8
37	eeg	59,069	72	16
38	euthyroid	3,163	44	2
39	gene	3,175	120	3
40	german	1,000	61	2
41	gina	3,468	970	2
42	gisette	7,000	5,000	2
43	glass	214	9	6
44	glass-g2	163	9	2
45	heart	270	13	2
46	heart-c	302	22	2
47	hepatitis	155	19	2
48	hERG-121inputs	65	121	2
49	hERG-64inputs	65	64	2
50	hERG-9inputs	130	9	2
51	hiva	4,229	1,617	2
52	horse	364	58	3
53	hypothyroid	3,772	29	4
54	ionosphere	351	34	2
55	iris	150	4	3
56	isolet	7,797	617	26
57	kddcup98	9,541	23,549	2
58	krkopt	28,056	6	18
59	krvskp	3,196	38	2
60	labor	57	29	2

	Dataset	Cases	Features	Classes
61	led24	200	24	10
62	lenses	24	6	3
63	letter	20,000	16	26
64	leukemia	72	12,582	3
65	liver	345	6	2
66	lrs	531	101	10
67	lung-cancer-harvard1	203	12,600	5
68	lung-cancer-harvard2	181	12,533	2
69	lung-cancer-michigan	96	7,129	2
70	lung-cancer-ontario	39	2,880	2
71	lymphography	148	38	4
72	madelon	2,600	500	2
73	magic04	19,020	10	2
74	mammography	961	18	2
75	mfeat-fac	2,000	216	10
76	mfeat-fou	2,000	76	10
77	mfeat-kar	2,000	64	10
78	mfeat-mor	2,000	6	10
79	mfeat-pix	2,000	240	10
80	mfeat-zer	2,000	47	10
81	mushroom	8,124	117	2
82	musk	6,598	166	2
83	new-thyroid	215	5	3
84	nursery	12,960	23	5
85	optdigits	5,620	64	10
86	ovarian-cancer	253	15,154	2
87	ozone1hr	2,536	72	2
88	ozone8hr	2,534	72	2
89	page-blocks	5,473	10	5
90	pcmac	1,943	3,289	2
91	pediatric-leukemia	327	12,558	7
92	pendigits	10,992	16	10
93	phoneme	5,404	5	2
94	pima	768	8	2
95	polya	9,255	169	2
96	post-operative	90	20	3
97	primary-tumor	339	23	22
98	promoters	106	114	2
99	prostate	136	12,600	2
100	rcv1	5,346	47,236	53
101	reuters21578	8,293	18,933	65
102	satimage	6,435	36	6
103	segment	2,310	19	7
104	shuttle	58,000	9	7
105	sick	3,772	33	2
106	sonar	208	60	2
107	soybean	683	82	19
108	texture	5,500	40	11
109	tic-tac-toe	958	9	2
110	tis	13,375	927	2
111	titanic	2,201	8	2
112	ustilago	6,141	4,016	2
113	vehicle	846	18	4
114	vote	435	16	2
115	vowel	990	10	11
116	waveform	5,000	40	3
117	wine	178	13	3
118	yeast	1,484	8	10
119	zip	9,298	256	10
120	zoo	101	16	7

Regarding the hyperparameters of our approach, each algorithm has the same parameters as its standard version. We chose an ensemble size of $M = 30$, which is a fairly common value. For ReweightBoost, we chose $r = 5$, following the recommendations of the authors. For FloatBoost, we set 50 as the maximum number of attempts. The feature selection algorithms used the hyperparameters recommended by the authors. The value of α, see eq. 5, is set to $\alpha = 0.5$ because improving accuracy is the major aim of our method. As explained in Section 2, the classification performance part of the threshold of vote evaluation has two terms. Thus, $\alpha = 0.5$ means more stress on classification performance than in the reduction of the number of features selected. In the experimental results, we will also show a study of the effect of this value on the performance of the proposed method.

For setting the hyper-parameters of the classifiers used to evaluate the final subset of selected features we employed 10-fold cross-validation. For each of the classifiers, we obtained the best hyper-parameters from a set of different values. For SVM, we utilized a linear kernel with $C \in \{0.1, 1, 10\}$, and a Gaussian kernel with $C \in \{0.1, 1, 10\}$ and $\gamma \in \{0.0001, 0.001, 0.01, 0.1, 1, 10\}$, testing all 21 possible combinations. For C4.5 we tested 1 and 10 trials and the option of softening of thresholds trying all four possible combinations. The hyper-parameter setting process is repeated each time that a classifier is trained for both our approach and the standard feature selection methods. Although this method does not assure an optimum set of hyper-parameters, it guarantees that a good set of hyper-parameters will be obtained in a reasonable amount of time.

Because we are evaluating feature selection algorithms, we considered for evaluation purposes testing accuracy and reduction. Section 4.2 explains the measures used in the experiments.

The source code, written 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 from the authors.

4.1. Statistical tests

We used the Wilcoxon test [52] as the main statistical test for comparing pairs of algorithms. In our experiments, we will also compare groups of methods. In such cases, it is not advisable to use only pairwise statistical tests such as the Wilcoxon test. Instead, we first apply an Iman-Davenport test to ascertain whether there are significant differences among the methods. If the Iman-Davenport test rejects the null hypothesis, we proceed with a post hoc Nemenyi test [53]. The performance of two classifiers is significantly different if the corresponding average ranks differ by at least the critical difference:

$$CD = q_\alpha \sqrt{k(k+1)\frac{6}{N}} \tag{6}$$

where critical values q_α are based on the Studentized range statistic divided by $\sqrt{2}$, N is the number of datasets, and k is the number of compared methods. As a graphical representation of the Nemenyi test, we use the plots described by Demšar [52]. When comparing all the algorithms against each other, we connect the groups of algorithms that are not significantly different with a horizontal line. We also show the critical difference above the graph.

4.2. Evaluation measures

Reduction is measured as the percentage of features removed by any algorithm. Thus, if we have M features and a certain algorithm selects m features, the reduction is $r = 1 - m/M$. We can use the standard measure of accuracy as the percentage of features correctly classified. However, recent works have shown that misclassification rates may be biased because they contain substantial randomness [54]. Furthermore, when the number of patterns among the classes is not evenly distributed, the accuracy may give a poor evaluation of the performance, as the performance over classes with few patterns is almost disregarded if we use the accuracy over the entire test set. Thus, as a performance measure, we used Cohen's κ measure, which is a method that compensates for random hits. Its original purpose was to measure the degree of agreement. However, κ can also be adapted to measure classification accuracy, and its use is

recommended because it takes random successes into consideration [54]. The value of κ can be computed from the confusion matrix in a classification task as follows:

$$\kappa = \frac{n \sum_{i=1}^{C} x_{ii} - \sum_{i=1}^{C} x_{i.} x_{.i}}{n^2 - \sum_{i=1}^{C} x_{i.} x_{.i}}, \tag{7}$$

where x_{ii} is the cell count on the main diagonal, n is the number of examples, C is the number of classes, and $x_{i.}$ and $x_{.i}$ are the column and row total counts, respectively. The value of κ ranges from -1 (total disagreement) to 1 (perfect agreement). For multi-class problems, κ is a very useful yet simple metric for measuring the accuracy of the classifier while compensating for random successes.

5. Experimental results

In this section, we present the results for the set of experiments carried out. Our first set of experiments compared the performance of the five standard feature selection methods, which were used as base feature selectors, and the performance of the five boosting methods. The tables show the comparison of the boosting approach against the basic feature selection method alone for all the selection methods in terms of κ measure and reduction. The tables show the average κ value, the Friedman ranks [55], the win/loss record of the boosting method against the basic method and the p-value of the Wilcoxon test of the boosting method against the standard method. The table also shows the p-value of the Iman-Davenport for the comparison of the five methods.

We were interested in comparing the performance of the different boosting methods; thus, our aim was to test whether AdaBoost.FS, FloatBoost.FS, MadaBoost.FS, MultiBoost.FS or ReweightBoost.FS achieved better performance than the FS algorithm alone, where FS represents each of the methods described in Section 2.3. The performance is determined by two factors, classification performance and reduction. However, because we were proposing a method for

boosting feature selection algorithms, our major aim was improving classification performance. Thus, we considered our algorithm better than the standard method when the classification performance was significantly better, even if the reduction was the same. Furthermore, we may risk a small decrease in reduction if the method achieved significantly better classification performance.

To perform an experimental analysis that was as comprehensive as possible, we tested our approach with two different classification methods: a decision tree using the C4.5 method [56] and a support vector machine [57].

5.1. Results using a decision tree

The comparison using a decision tree as a base classifier is shown in Table 3 for κ and in Table 4 for reduction. For FAST, the boosting approach beat FAST alone for four of the five methods. MultiBoost.FAST was no better than FAST in terms of classification performance. The boosting approach achieved a better classification performance without losing its reduction ability. In fact, ReweightBoost.FAST improved FAST in both κ and reduction.

For FCBF, the results are similar. The five boosting methods improved the classification performance of FCBF alone. This improvement was achieved while keeping the reduction power of the method, with the exception of MultiBoost.FCBF, which achieved a worse reduction than FCBF.

The behavior of LVF was different. In terms of κ, the boosting performed worse than LVF for all five methods. However, this is because LVF achieved poor reduction. Our method, although performing slightly worse in terms of κ, was able to improve the reduction ability by more than 30% on average for most boosting methods. These results suggested that the use of $\alpha = 0.5$ was not an appropriate value for an algorithm that has such poor reduction ability. To check this possibility, we repeated the experiments for AdaBoost using $\alpha = 0.75$. The results are shown in Table 5. This new experiment showed that with the appropriate value of α, AdaBoost.LVF significantly improved both the classification performance and the reduction results of LVF. AdaBoost.LVF improved the accuracy by almost 3% and at the same time improved the reduction ability

Table 3: Comparison of the five boosting methods in terms of κ using the five feature selection algorithms against the feature selection method alone for C4.5 as a classification method. The table shows the win/loss record of every boosting method against the feature selection method alone and the p-value of the Wilcoxon test. Significant differences in favor of our method are marked with a ✓; significant differences against our approach are marked with a ✗.

	FAST	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.7055	0.7928	0.7937	0.7925	0.7648	0.7791
Rank	4.2000	2.8708	2.8417	3.3208	4.2875	3.4792
Win/draw/loss		79/33	79/32	78/33	58/59	67/36
p-value		0.0000✓	0.0000✓	0.0000✓	0.0789	0.0000✓
Iman-Davenport text p-value:	0.0000					
	FCBF	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.6966	0.7897	0.7882	0.7817	0.7465	0.7698
Rank	4.1833	2.6958	2.8667	3.6333	4.0375	3.5833
Win/draw/loss		78/33	78/36	74/39	66/51	68/41
p-value		0.0000✓	0.0000✓	0.0000✓	0.0154✓	0.0000✓
Iman-Davenport text p-value:	0.0000					
	LVF	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.8364	0.8162	0.8205	0.8023	0.8110	0.8108
Rank	2.1958	3.4958	3.3542	4.1667	3.9542	3.8333
Win/draw/loss		30/80	29/80	20/96	24/88	19/91
p-value		0.0000✗	0.0000✗	0.0000✗	0.0000✗	0.0000✗
Iman-Davenport text p-value:	0.0000					
	ReliefF	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.6343	0.7884	0.7909	0.8045	0.7714	0.8052
Rank	3.9167	3.2083	3.3875	3.4958	3.8125	3.1792
Win/draw/loss		66/39	61/48	68/43	65/51	64/43
p-value		0.0000✓	0.0001✓	0.0000✓	0.0021✓	0.0000✓
Iman-Davenport text p-value:	0.0060					
	SetCover	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.7879	0.8044	0.8035	0.7854	0.7661	0.7815
Rank	3.1500	2.9125	3.0958	3.8167	4.1125	3.9125
Win/draw/loss		51/54	50/55	48/64	38/69	38/67
p-value		0.4991	0.7732	0.1535	0.0018✗	0.0145✗
Iman-Davenport text p-value:	0.0000					

by more than 20%.

For ReliefF, the boosting feature selection achieved a remarkably good performance. In terms of classification performance, boosting was able to beat the ReliefF algorithm for all five methods. In the best case, ReweightBoost.ReliefF, the average improvement was 17%. Furthermore, the improvement was achieved while keeping the reduction power, as boosting was not worse than ReliefF in this aspect for the five methods.

The results for SetCover shared some similarities with the LVF case. In terms of performance, AdaBoost.SetCover, FloatBoost.SetCover and GentleBoost.SetCover were no worse than SetCover, while MultiBoost.SetCover and ReweightBoost.SetCover were significantly worse. However, in terms of reduc-

Table 4: Comparison of the five boosting methods in terms of reduction using the five feature selection algorithms against the feature selection method alone for C4.5 as a classification method. The table shows the win/loss record of every boosting method against the feature selection method alone and the p-value of the Wilcoxon test. Significant differences in favor of our method are marked with a ✓; significant differences against our approach are marked with a ✗.

	FAST	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.8404	0.8077	0.7978	0.8026	0.6939	0.8116
Rank	3.8708	2.9750	3.5125	3.2750	4.2833	3.0833
Win/Loss		70/46	69/47	70/48	48/70	78/35
p-value		0.3882	0.9093	0.8341	0.0002✗	0.0354✓
Iman-Davenport text p-value: 0.0000						
	FCBF	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.8023	0.8056	0.7849	0.8055	0.7386	0.7971
Rank	3.3375	3.2792	3.4625	3.3917	4.3917	3.1375
Win/Loss		54/60	61/57	56/62	42/77	61/57
p-value		0.9509	0.6458	0.8741	0.0037✗	0.7090
Iman-Davenport text p-value: 0.0000						
	LVF	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.4072	0.7133	0.6883	0.7319	0.6777	0.7297
Rank	5.7083	2.7958	3.3125	3.1583	3.4375	2.5875
Win/Loss		109/2	106/4	112/6	109/4	111/1
p-value		0.0000✓	0.0000✓	0.0000✓	0.0000✓	0.0000✓
Iman-Davenport text p-value: 0.0000						
	ReliefF	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.6830	0.6658	0.6578	0.6843	0.6198	0.6666
Rank	3.6250	3.4375	3.4833	3.2583	3.9042	3.2917
Win/Loss		61/52	67/50	62/55	52/67	63/51
p-value		0.6004	0.5761	0.7813	0.1152	0.8865
Iman-Davenport text p-value: 0.0413						
	SetCover	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.6632	0.7656	0.7526	0.7916	0.7376	0.7723
Rank	4.2167	3.1542	3.4917	3.5458	3.9167	2.6750
Win/Loss		74/35	75/35	73/46	68/44	75/33
p-value		0.0000✓	0.0000✓	0.0000✓	0.0002✓	0.0000✓
Iman-Davenport text p-value: 0.0000						

tion, the five methods achieved better results than SetCover, with an average improvement of over 10%.

A summary of these results is shown in Table 6. The table shows that, as a general rule, our proposal was able to beat the standard methods in terms of classification performance while keeping the reduction ability.

These results of the five methods are illustrated in Figures 1, 2, 3, 4 and 5 for FAST, FCBF, LCF, ReliefF and SetCover, respectively. The figures show the results for κ and reduction. This graphic representation is based on the κ-error relative movement diagrams [58]. These diagrams use an arrow to represent the results of the two methods applied to the same dataset. The arrow starts at

Table 5: Comparison of the AdaBoost boosting methods in terms of κ and reduction using the LVF feature selection algorithm using $\alpha = 0.75$ and $\alpha = 0.5$. The table shows the win/loss record of every boosting method against the feature selection method alone and the p-value of the Wilcoxon test. Significant differences in favor of our method are marked with a ✓; significant differences against our approach are marked with a ✗.

κ			
	LVF	AdaBoost ($\alpha = 0.5$)	AdaBoost ($\alpha = 0.75$)
Mean	0.8364	0.8162	0.8609
Rank	2.7042	4.4083	2.0125
Win/draw/loss		30/80	56/54
p-value		0.0000✗	0.0481✓

Reduction			
	LVF	AdaBoost ($\alpha = 0.5$)	AdaBoost ($\alpha = 0.75$)
Mean	0.4072	0.7133	0.6124
Rank	6.5167	2.8917	5.4333
Win/Loss		109/2	93/19
p-value		0.0000✓	0.0000✓

	AdaBoost		FloatBoost		GentleBoost		MultiBoost		ReweightBoost	
	κ	Red.	κ	Red.	κ	Red.	κ	Red.	κ	Red.
FAST	✓	—	✓	—	✓	—	—	✗	✓	✓
FCBF	✓	—	✓	—	✓	—	✓	✗	✓	—
LVF	✓	✓	✗	✓	✗	✓	✗	✓	✗	✓
ReliefF	✓	—	✓	—	✓	—	✓	—	✓	—
SetCover	—	✓	—	✓	—	✓	✗	✓	✗	✓

Table 6: Summary of results for all the feature selection methods and all the boosting algorithms using a decision tree classifier. A ✓ states that the boosting approach was better than the standard method, a ✗ that it was worse, and —that there were no significant differences according to the Wilcoxon test.

the coordinate origin, and the coordinates of the tip of the arrow represent the difference between the κ and reduction of our boosting method and those of the standard algorithm alone. These graphs are a convenient way of summarizing the results. A positive value in either reduction or κ means our method performed better.

For the FAST algorithm, Figure 1 shows that the behavior of AdaBoost.FAST, FloatBoost.FAST, GentleBoost.FAST and ReweightBoost.FAST was quite homogeneous. The figure shows a common behavior. For many datasets, boosting achieved a large improvement in terms of reduction, with a small reduction in κ (top left quadrant of the plot). For these datasets, the improvement in terms of reduction clearly outweigh the smaller κ. For another large set of problems,

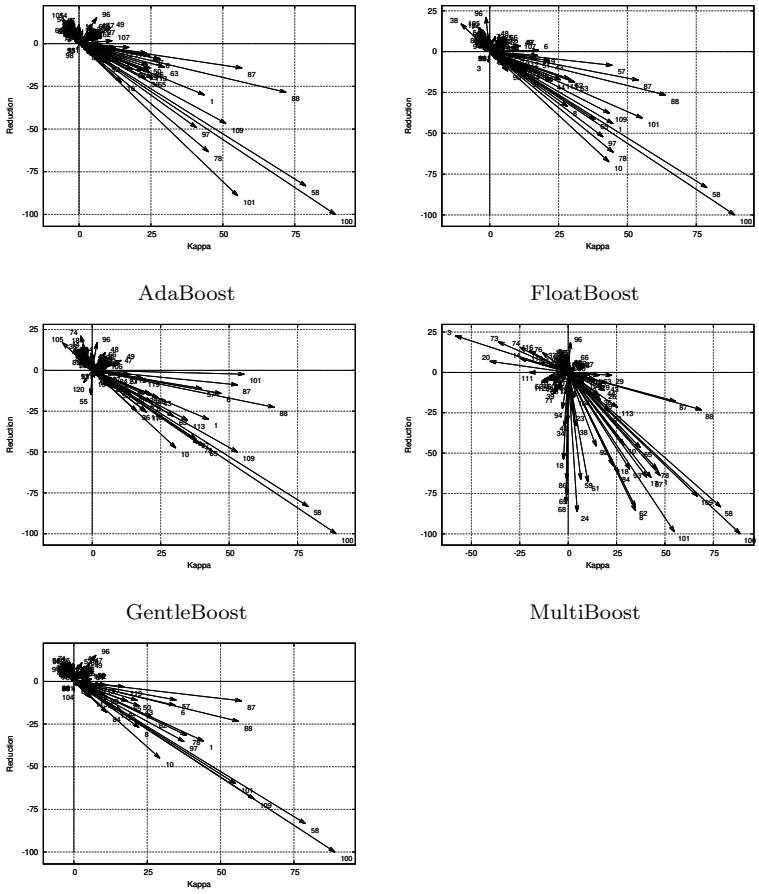


Figure 1: κ/reduction using relative movement diagrams for our proposal against the standard FAST method using the five different boosting methods and a decision tree as a classifier model. Positive values on each axis indicate better performance by our method.

boosting achieved a markedly better classification performance, with the cost of smaller reduction (bottom right quadrant of the plot). For a small subset of problems, boosting achieved both better reduction and better κ (top right quadrant). Finally, almost no dataset showed boosting that was worse in both aspects (bottom left quadrant). MultiBoost.FAST showed a different overall trend. Most datasets are in the same quadrants as the other four methods, but the decrease in terms of reduction in the bottom right quadrant and κ in the top left quadrant is greater, so the performance of MultiBoost.FAST is worse

than the performance of the other four boosting methods.

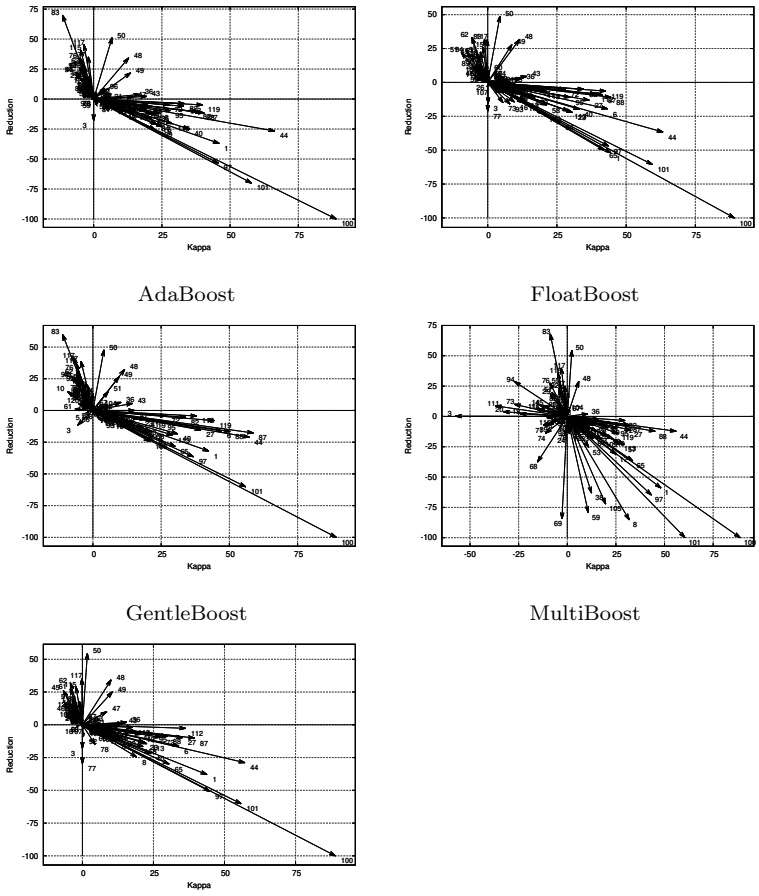


Figure 2: κ/reduction using relative movement diagrams for our proposal against the standard FCBF method using the five different boosting methods and a decision tree as a classifier model. Positive values on each axis indicate better performance by our method.

The behavior of the FCBF method (see Figure 2) was very similar to that of the FAST method for all five boosting algorithms.

Figure 3 shows the same plots for the LVF method. The results for LVF were homogeneous for the five boosting methods, as is clearly shown in the plots. The vast majority of points are in the top left quadrant of the plot, showing that for most datasets, boosting achieved a large improvement in the reduction power of the algorithm but at the cost of a lower κ. For AdaBoost.LVF and

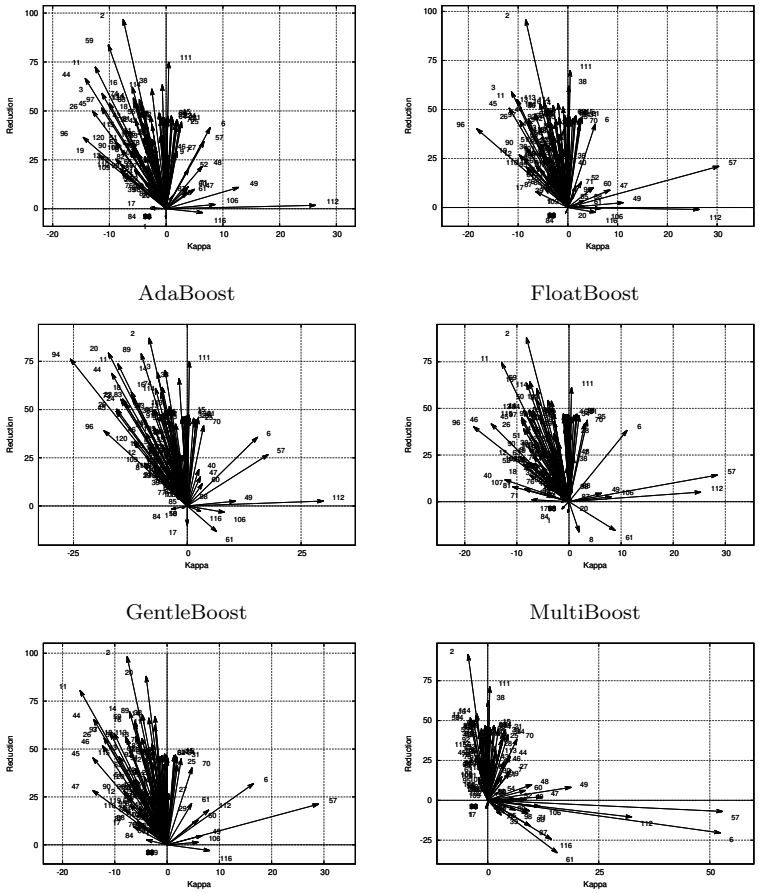


Figure 3: κ/reduction using relative movement diagrams for our proposal against the standard LVF method using the five different boosting methods and a decision tree as a classifier model. Positive values on each axis indicate better performance by our method.

$\alpha = 0.75$, the figure shows a large improvement in reduction coupled with at least a no worse performance in classification accuracy.

The results for ReliefF (see Figure 4) show a similar behavior among the five boosting methods, with small differences for the case of MultiBoost.ReliefF. The datasets are divided into two groups almost in halves. On the one hand, top left quadrant, we have datasets for which boosting achieved a significantly reduction gain with a small loss of κ. On the other hand, right bottom quadrant, we have datasets for which we have a large increment in the κ, with a decrement in the

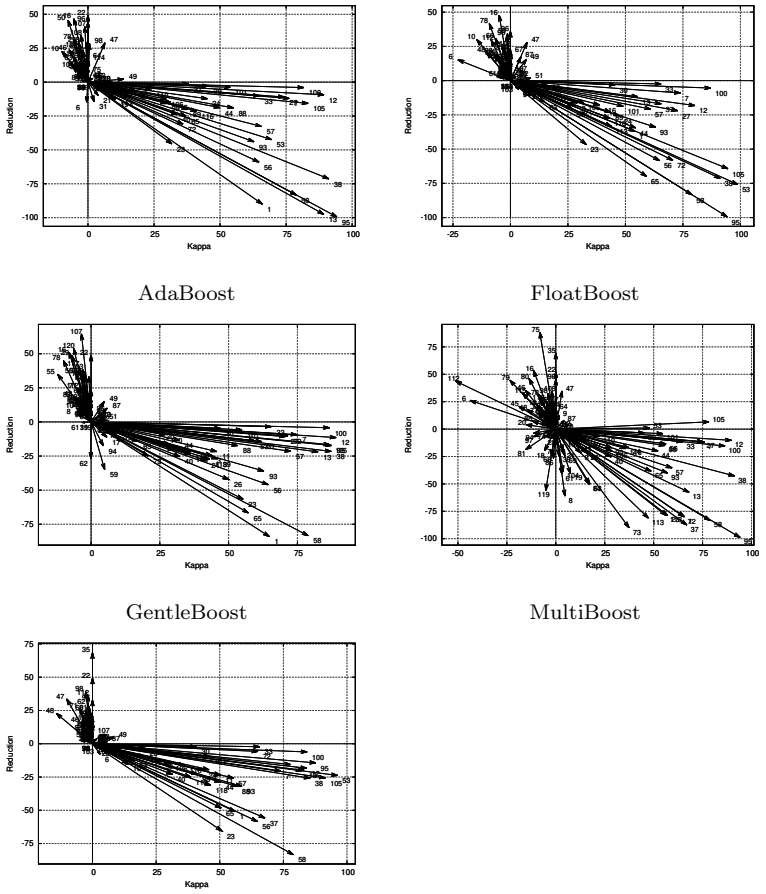


Figure 4: κ/reduction using relative movement diagrams for our proposal against the standard ReliefF method using the five different boosting methods and a decision tree as a classifier model. Positive values on each axis indicate better performance by our method.

reduction power of the method.

The results for SetCover are plotted in Figure 5. We have again two different behaviors. For a set of problems, top left quadrant, boosting achieved an important reduction increment, with a somewhat smaller decrease in κ. For another set of problems, right bottom quadrant, boosting significantly improved the κ while keeping the reduction ability of the method almost untouched.

As an additional test, we compared all the methods using a Nemenyi test. Plots of this comparison are shown for all the feature selection methods in Figure

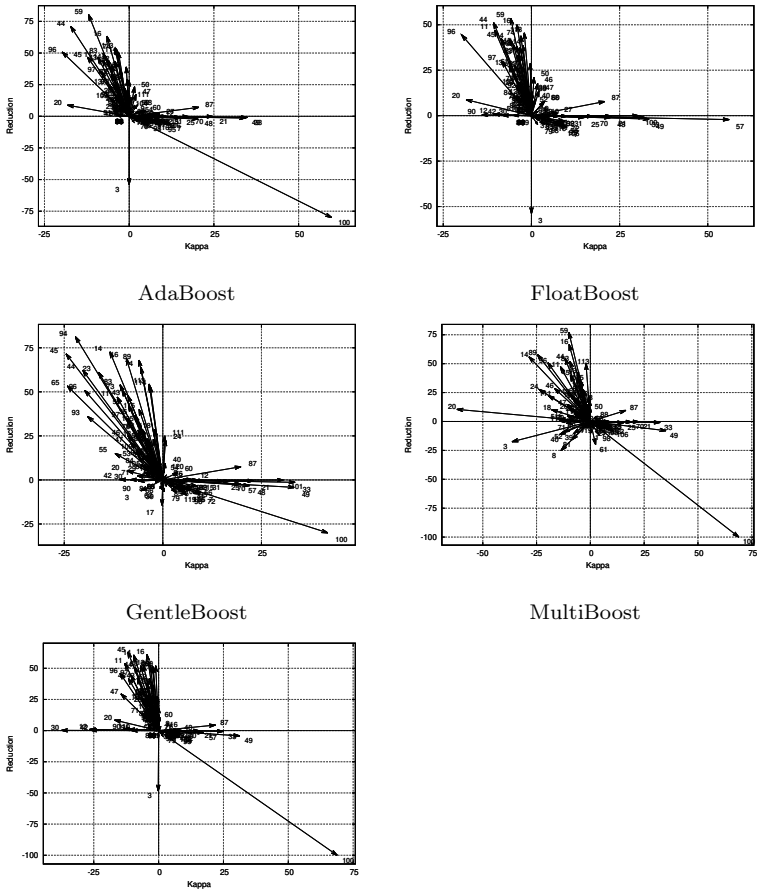


Figure 5: κ/reduction using relative movement diagrams for our proposal against the standard SetCover method using the five different boosting methods and a decision tree as a classifier model. Positive values on each axis indicate better performance by our method.

6 for κ and reduction. The aim of using the Nemenyi test is to confirm the improvement of our approach, as many paired comparisons using the Wilcoxon test might accumulate a certain test error.

For the FAST method, the Nemenyi test found significant differences for all the boosting algorithms with the exception of MultiBoost.FAST. Regarding κ, we observe that FCBF was beat by AdaBoost.FCBF, FloatBoost.FCBF and ReweightBoost.FCBF. Although MultiBoost.FCBF and GentleBoost.FCBF achieved a better average rank than FCBF, the Nemenyi test did not find these differences

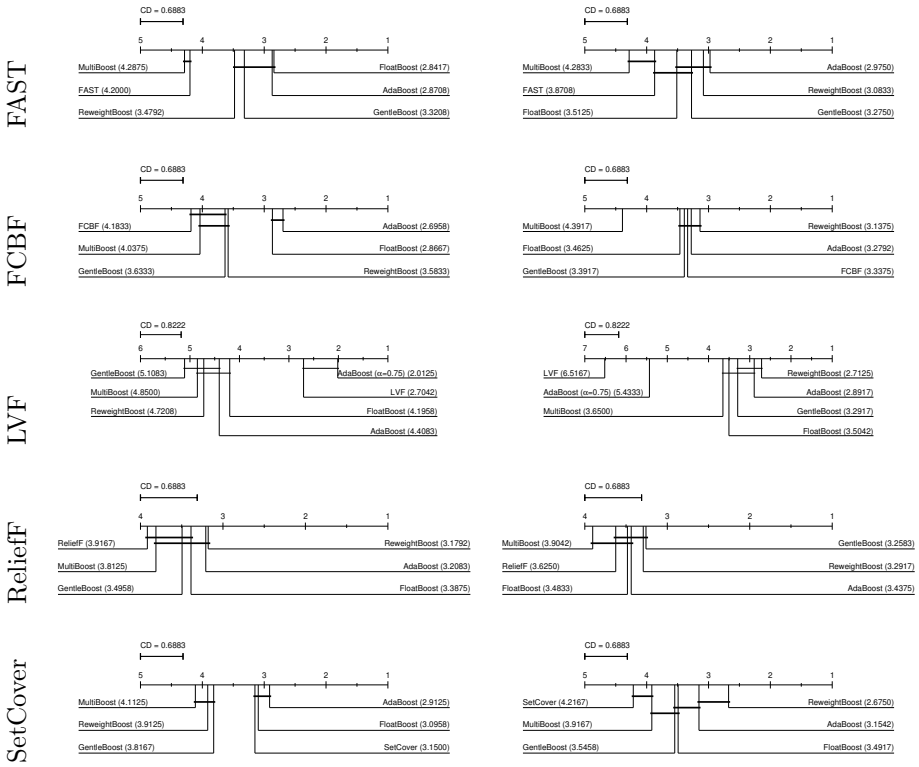


Figure 6: Nemenyi test for C4.5 as classification method for κ and reduction.

significant.

As shown in the previous results, LVF obtained better results than the boosting method, although its reduction ratio was inferior by approximately 30% as a general rule. For ReliefF, ReweightBoost.ReliefF and AdaBoost.ReliefF showed significantly better performance than ReliefF. SetCover showed a similar behavior, although FloatBoost.SetCover and AdaBoost.SetCover were no worse than SetCover alone.

In terms of reduction power, the Nemenyi test corroborates the results of the Wilcoxon test. The significant differences found with this test are also found with the Nemenyi test for almost all cases.

5.2. Results using a SVM

The comparison using an SVM as a base classifier is shown in Table 7 for κ and in Table 8 for reduction. For FAST, the boosting approach beat FAST alone for four of the five methods. MultiBoost.FAST was better than FAST in terms of performance, but the differences were not significant. Again, the boosting approach achieved a better performance without losing its reduction ability. In terms of reduction, only MultiBost.FAST performed worse than FAST.

Table 7: Comparison of the five boosting methods in terms of κ using the five feature selection algorithms against the feature selection method alone for an SVM as a classification method. The table shows the win/loss record of every boosting method against the feature selection method alone and the p-value of the Wilcoxon test. Significant differences in favor of our method are marked with a ✓; significant differences against our approach are marked with a ✗.

	FAST	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.6396	0.7307	0.7074	0.7265	0.6897	0.7263
Ranks	4.0792	3.0667	3.2125	3.3542	3.9792	3.3083
Win/loss		67/28/25	68/24/28	64/25/31	50/21/49	57/29/34
p-value		0.0000✓	0.0000✓	0.0000✓	0.3864	0.0001✓
Iman-Davenport text p-value: 0.0000						
	FCBF	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.6297	0.7106	0.7161	0.7182	0.6805	0.7080
Ranks	4.1417	3.1833	3.0958	3.4167	3.7583	3.4042
Win/loss		67/21/32	70/19/31	66/20/34	60/13/47	67/21/32
p-value		0.0000✓	0.0000✓	0.0000✓	0.0726	0.0000✓
Iman-Davenport text p-value: 0.0001						
	LVF	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.7551	0.7518	0.6992	0.7356	0.7364	0.7468
Ranks	2.5167	3.0208	4.2042	3.8208	3.9000	3.5375
Win/loss		28/32/60	22/18/80	20/27/73	24/29/67	21/28/71
p-value		0.0063✗	0.0000✗	0.0000✗	0.0000✗	0.0000✗
Iman-Davenport text p-value: 0.0000						
	ReliefF	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.5930	0.7232	0.7193	0.7239	0.7006	0.7204
Ranks	3.6375	3.4292	3.6042	3.1458	3.6083	3.5750
Win/loss		53/18/49	54/15/51	57/18/45	57/16/47	53/18/49
p-value		0.0099✓	0.0181✓	0.0021✓	0.0214✓	0.0181✓
Iman-Davenport text p-value: 0.0298						
	SetCover	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.6065	0.7318	0.7403	0.7432	0.7009	0.7178
Ranks	3.6083	3.0750	2.9042	3.2667	4.1708	3.9750
Win/loss		63/9/48	64/8/48	59/7/54	52/7/61	57/5/58
p-value		0.0007✓	0.0001✓	0.0026✓	0.2040	0.0606
Iman-Davenport text p-value: 0.0000						

For FCBF, the boosting approach beat FCBF alone for four of the five methods. MultiBoost.FCBF was better than FCBF in terms of performance,

but the differences were significant only at a 90% level of confidence. Again, the boosting approach achieved a better classification performance without losing its reduction ability. In terms of reduction, only MultiBoost.FCBF performed worse than FCBF.

Table 8: Comparison of the five boosting methods in terms of reduction using the five feature selection algorithms against the feature selection method alone for an SVM as a classification method. The table shows the win/loss record of every boosting method against the feature selection method alone and the p-value of the Wilcoxon test. Significant differences in favor of our method are marked with a ✓; significant differences against our approach are marked with a ✗.

	FAST	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.8404	0.8059	0.7940	0.7859	0.5860	0.7975
Ranks	3.5042	2.9583	3.5000	3.3917	4.4292	3.2167
Win/draw/loss		61/9/50	63/3/54	61/3/56	33/3/84	71/5/44
p-value		0.9041	0.2955	0.1119	0.0000✗	0.2771
Iman-Davenport text p-value: 0.0000						
	FCBF	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.8023	0.8024	0.7873	0.7710	0.6574	0.7884
Ranks	3.3000	3.0000	3.2000	3.7750	4.4875	3.2375
Win/draw/loss		58/4/58	63/2/55	49/3/68	38/0/82	62/3/55
p-value		0.5531	0.9969	0.0561	0.0000✗	0.6877
Iman-Davenport text p-value: 0.0000						
	LVF	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.4072	0.6552	0.6379	0.6677	0.6470	0.6810
Ranks	5.2208	3.1167	3.3083	3.1583	3.4417	2.7542
Win/draw/loss		96/11/13	93/9/18	100/7/13	94/10/16	102/6/12
p-value		0.0000✓	0.0000✓	0.0000✓	0.0000✓	0.0000✓
Iman-Davenport text p-value: 0.0000						
	ReliefF	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.6830	0.7489	0.7397	0.6994	0.6741	0.7403
Ranks	3.4750	3.0292	3.3375	3.6625	4.1375	3.3583
Win/draw/loss		63/4/53	60/1/59	58/3/59	48/2/70	60/6/54
p-value		0.0504	0.1252	0.7119	0.2818	0.1025
Iman-Davenport text p-value: 0.0002						
	SetCover	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.6632	0.7107	0.6830	0.6937	0.6639	0.7287
Ranks	3.4917	3.2375	3.8667	3.9125	3.9375	2.5542
Win/draw/loss		58/8/54	55/8/57	52/7/61	47/7/66	70/4/46
p-value		0.0134✓	0.1418	0.2623	0.9384	0.0002✓
Iman-Davenport text p-value: 0.0000						

The case for the LVF method was similar to the results for decision trees. LVF performed poorly in terms of classification performance. Thus, its boosting counterpart achieved a markedly better performance in terms of reduction, but at the cost of a significantly poorer performance in terms of κ. As with the previous case, this result suggested that our selection of the value of α was not

appropriate. We repeated the experiments for AdaBoost.LVF using $\alpha = 0.75$. The comparison is shown in Table 9. The results show that with this α value, AdaBoost.LVF improved both the κ and the reduction of LVF.

Table 9: Comparison of the AdaBoost boosting methods in terms of κ and reduction using the LVF feature selection algorithm using $\alpha = 0.75$ and $\alpha = 0.5$. The table shows the win/loss record of every boosting method against the feature selection method alone and the p-value of the Wilcoxon test. Significant differences in favor of our method are marked with a ✓; significant differences against our approach are marked with a ✗

κ			
	LVF	AdaBoost ($\alpha = 0.5$)	AdaBoost ($\alpha = 0.75$)
Mean	0.7551	0.7518	0.8063
Rank	3.0667	3.8417	2.2750
Win/draw/loss		28/60	50/38
p-value		0.0000✗	0.0063✓
Iman-Davenport text p-value: 0.0000			
Reduction			
	LVF	AdaBoost ($\alpha = 0.75$)	AdaBoost ($\alpha = 0.75$)
Mean	0.4072	0.6552	0.5349
Rank	5.8958	3.3250	5.1833
Win/Loss		96/13	76/34
p-value		0.0000✓	0.0000✓
Iman-Davenport text p-value: 0.0000			

For ReliefF, the boosting approach achieved a better κ than ReliefF alone for all five boosting methods. Furthermore, this improvement was achieved while matching the reduction ability of ReliefF. AdaBoost.ReliefF beat ReliefF in terms of κ and was also better in terms of reduction at a confidence level of 90% (p-value = 0.0504).

Finally, for SetCover, AdaBoost.SetCover, FloatBoost.SerCover and GentleBoost.SetCover improved the κ of SetCover, without losing its reduction power. In fact, AdaBoost.SetCover was better than SetCover in κ and reduction.

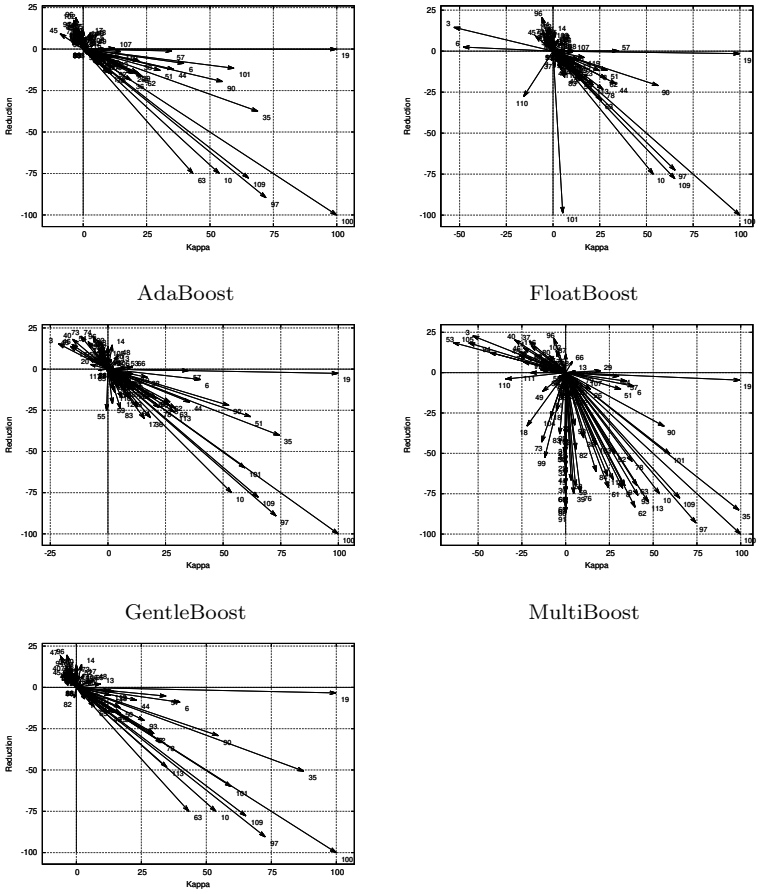
A summary of the results is shown in Table 10. As was the case for a decision tree, our proposal beat the feature selection method in terms of performance while keeping the reduction ability of the original methods.

Relative movement diagrams for κ and reduction are shown in Figures 7, 8, 9, 10 and 11 for FAST, FCBF, LVF, ReliefF and SetCover, respectively. The same behavior of decision trees is observed in these diagrams. For the five feature selection methods, the behavior of our approach was similar using decision trees

	AdaBoost		FloatBoost		GentleBoost		MultiBoost		ReweightBoost	
	κ	Red.	κ	Red.	κ	Red.	κ	Red.	κ	Red.
FAST	✓	—	✓	—	✓	—	—	✗	✓	—
FCBF	✓	—	✓	—	✓	—	—	✗	✓	—
LVF	✓	✓	✗	✓	✗	✓	✗	✓	✗	✓
ReliefF	✓	—	✓	—	✓	—	✓	—	✓	—
SetCover	✓	✓	✓	—	✓	—	—	—	✓	✓

Table 10: Summary of results for all the feature selection methods and all the boosting algorithms using an SVM. A ✓ indicates that the boosting approach was better than the standard method, a ✗ that it was worse, and —that there were no significant differences according to the Wilcoxon test.

and SVMs, showing the consistency of the proposed approach.



AdaBoost FloatBoost

GentleBoost MultiBoost

ReweightBoost

Figure 7: κ/reduction using relative movement diagrams for our proposal against the standard FAST method using the five different boosting methods and an SVM as a classifier model. Positive values on each axis indicate better performance by our method.

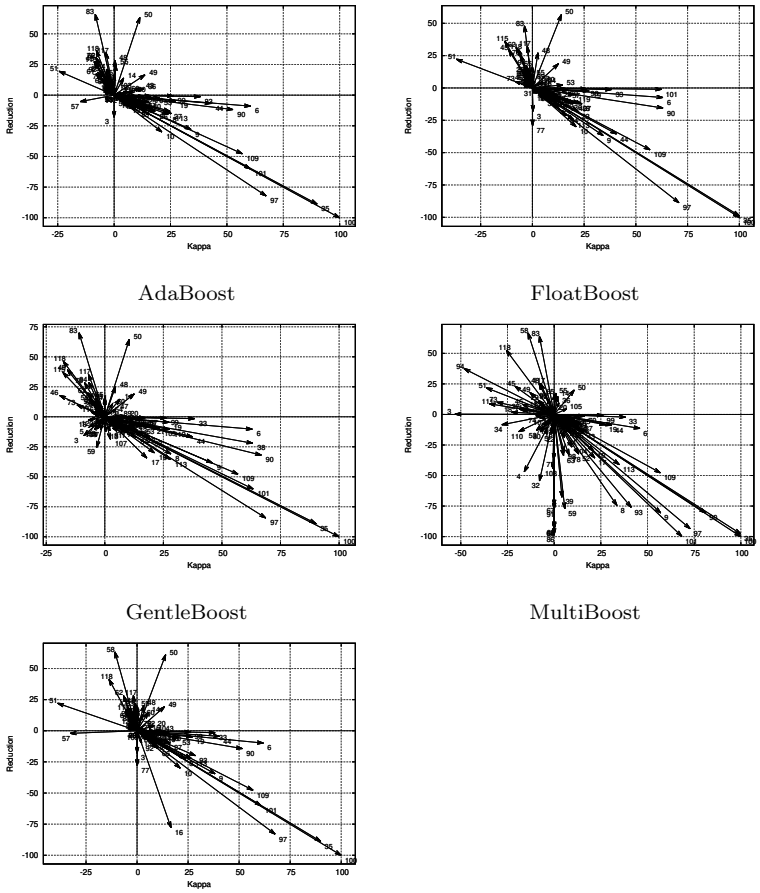


Figure 8: κ/reduction using relative movement diagrams for our proposal against the standard FCBF method using the five different boosting methods and an SVM as a classifier model. Positive values on each axis indicate better performance by our method.

These observed differences are corroborated by the Nemenyi test shown in Figure 12 for κ and reduction.

The most remarkable result is that the experiments support the validity of the proposed approach. In 19 out of 25 cases, boosting was able to significantly improve the testing κ of the method alone. Furthermore, in none of these 19 cases did the improvement have the negative side effect of decreasing the reduction ability of the method.

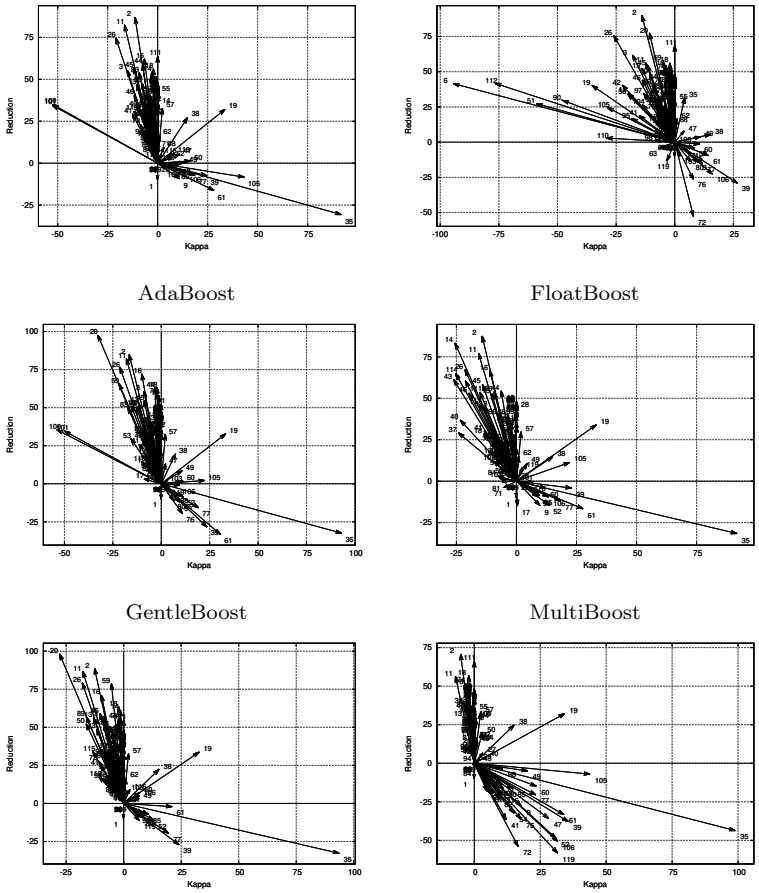


Figure 9: κ/reduction using relative movement diagrams for our proposal against the standard LVF method using the five different boosting methods and an SVM as a classifier model. Positive values on each axis indicate better performance by our method.

5.3. Comparison for datasets with more than 1000 features

In addition of the previous comparison, it is interesting to know the behavior of the different methods when the dataset has many features. In many current data mining tasks is very common to have thousands of features, so the ability of any method to deal with a large number of inputs is a relevant issue. In this section we show the comparison when the datasets have at least 1,000 features. Thus, we restrict the study to the 30 datasets with more than 1,000 features (see Table 2).

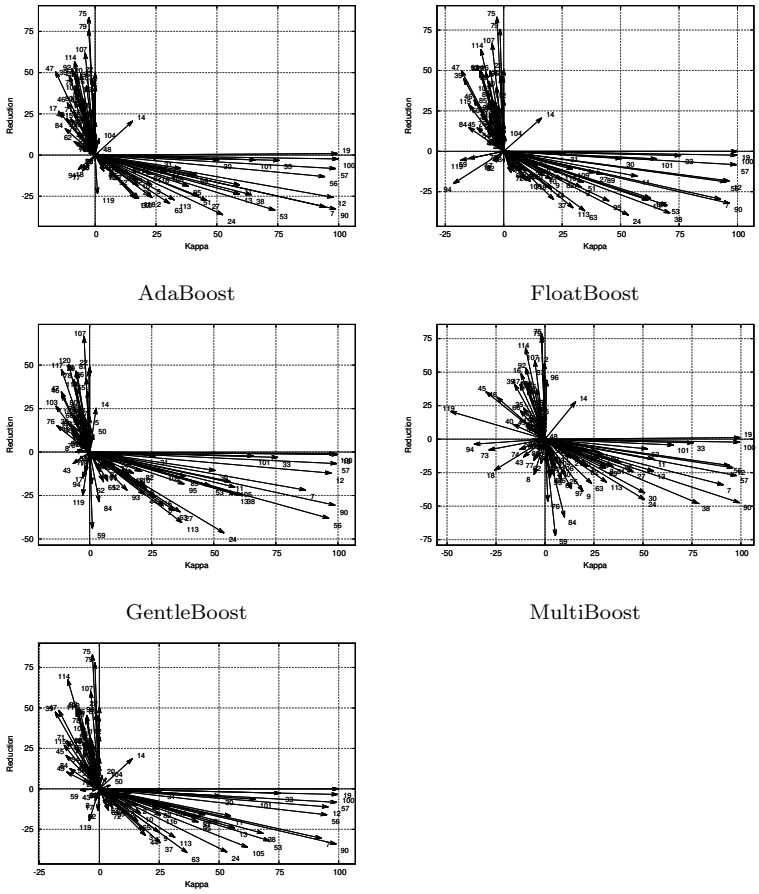


Figure 10: κ/reduction using relative movement diagrams for our proposal against the standard ReliefF method using the five different boosting methods and an SVM as a classifier model. Positive values on each axis indicate better performance by our method.

Tables 11 and 12 show the comparison for the five feature selection algorithms and the five boosting methods for κ and reduction respectively using a C4.5 decision tree as classifier. In terms of classification performance the behavior of our proposal was similar to the previous results using all the datasets. For FAST and FCBF methods boosting improved the method alone with the exception of MultiBoost. For LVF the performance was better than using all the datasets, as boosting was no worse than LVF for the five boosting methods. For ReliefF the behavior was also similar again with the exception of MultiBoost.

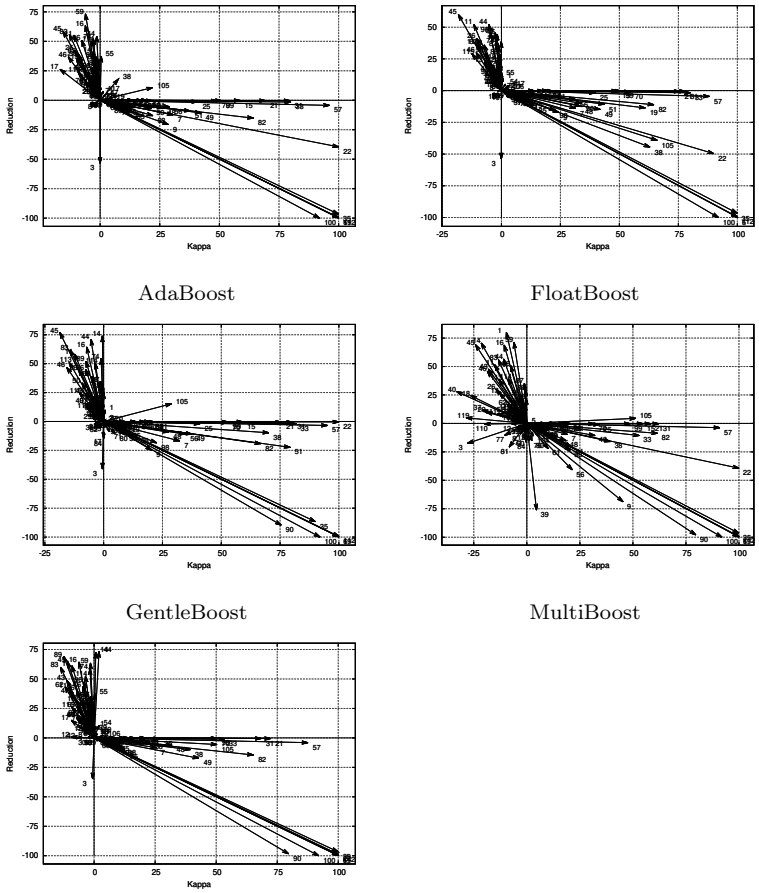


Figure 11: κ/reduction using relative movement diagrams for our proposal against the standard SetCover method using the five different boosting methods and an SVM as a classifier model. Positive values on each axis indicate better performance by our method.

Finally, for SetCover the results improved with respect to using all datasets, as {AdaBoost, FloatBoost, GentleBoost}.SetCover were better than SetCover and the other two were no worse than SetCover. As a summary, boosting performed better with many inputs for LVF and SetCover and similarly for FAST, FCBF and ReliefF.

Regarding reduction, see Table 12, the performance of boosting feature selection was also coherent with the results using all the datasets. For FAST, FCBF and ReliefF the comparison between the base method and the five boosting al-

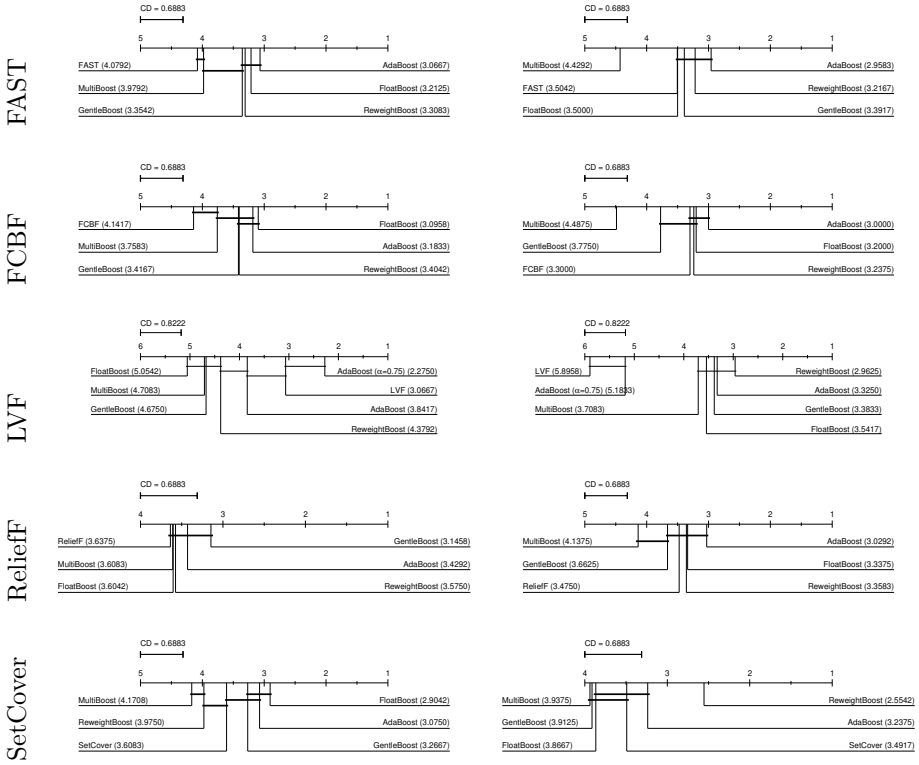


Figure 12: Nemenyi test for a SVM as classification method for κ and reduction.

Table 11: Comparison for datasets with more than 1000 features of the five boosting methods in terms of κ using the five feature selection algorithms against the feature selection method alone for a C4.5 tree as classification method. The table shows the win/loss record of every boosting method against the feature selection method alone and the p-value of the Wilcoxon test. Significant differences in favor of our method are marked with a ✓; significant differences against our approach are marked with a ✗.

	FAST	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.7441	0.8344	0.8348	0.8391	0.7961	0.8305
Ranks	4.9333	2.2333	2.6333	2.8500	5.0333	3.3167
Win/draw/loss		25/3/2	24/4/2	23/4/3	14/2/14	22/7/1
p-value		0.0000✓	0.0000✓	0.0000✓	0.3547	0.0000✓
	FCBF	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.7083	0.8170	0.8175	0.8176	0.7726	0.8070
Ranks	5.1000	2.1000	2.4833	3.2500	4.3333	3.7333
Win/draw/loss		26/3/1	25/4/1	24/4/2	18/2/10	21/5/4
p-value		0.0000✓	0.0000✓	0.0000✓	0.1332	0.0001✓
	LVF	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.8079	0.8038	0.8099	0.8112	0.8132	0.8113
Ranks	3.1500	3.3333	3.7333	3.7333	3.6833	3.3667
Win/draw/loss		14/1/15	13/1/16	11/1/18	12/1/17	12/1/17
p-value		0.4466	0.2667	0.2579	0.3877	0.3991
	ReliefF	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.6888	0.8462	0.8326	0.8481	0.8075	0.8370
Ranks	4.4500	2.8833	3.1500	3.0333	4.1167	3.3667
Win/draw/loss		20/4/6	18/5/7	20/4/6	17/1/12	19/5/6
p-value		0.0008✓	0.0051✓	0.0018✓	0.0804	0.0009✓
	SetCover	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.6595	0.7373	0.7237	0.7334	0.7096	0.6764
Ranks	4.4167	2.3000	3.3833	2.9167	3.3833	4.6000
Win/draw/loss		20/6/4	18/5/7	20/6/4	16/5/9	15/5/10
p-value		0.0002✓	0.0152✓	0.0012✓	0.1330	0.3035

gorithms obtained the same results. For LVF the reduction ability of boosting with respect to the base method was even improved with a average different above the 40%. For SetCover the behavior of boosting was slightly worse than the case for all datasets, but still the reduction was not worse than the use of SetCover alone excepting GentleBoost.SetCover.

Tables 13 and 14 show the comparison for the five feature selection algorithms and the five boosting methods for κ and reduction respectively using a SVM as classifier. In terms of κ there were no large differences with when all the datasets were used for the comparison. For FAST, FCBF and ReliefF the comparison between boosting and the method alone is the same as the previous case. For LVF there was a slight improvement over using all the datasets and

Table 12: Comparison for datasets with more than 1000 features of the five boosting methods in terms of reduction using the five feature selection algorithms against the feature selection method alone for C4.5 tree as classification method. The table shows the win/loss record of every boosting method against the feature selection method alone and the p-value of the Wilcoxon test. Significant differences in favor of our method are marked with a ✓; significant differences against our approach are marked with a ✗.

	FAST	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.9530	0.9045	0.9197	0.9315	0.7951	0.9117
Ranks	4.4000	2.8500	3.0167	2.9500	4.5667	3.2167
Win/draw/loss		21/0/9	22/0/8	22/0/8	13/0/17	24/0/6
p-value		0.2369	0.1359	0.0519	0.0368✗	0.1470
	FCBF	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.9783	0.9184	0.9189	0.9173	0.8578	0.9222
Ranks	3.0333	3.3333	3.1167	3.7000	4.7167	3.1000
Win/draw/loss		12/0/18	15/0/15	13/0/17	7/0/23	14/0/16
p-value		0.2712	0.3820	0.2989	0.0148✗	0.5440
	LVF	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.5057	0.9173	0.9104	0.9091	0.9066	0.9135
Ranks	5.9667	2.8167	3.2500	3.1500	2.9000	2.9167
Win/draw/loss		30/0/0	29/0/1	30/0/0	30/0/0	30/0/0
p-value		0.0000✓	0.0000✓	0.0000✓	0.0000✓	0.0000✓
	ReliefF	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.8907	0.8931	0.9082	0.9246	0.8716	0.9188
Ranks	3.5000	3.6333	3.6333	2.8000	4.4333	3.0000
Win/draw/loss		15/0/15	17/0/13	17/0/13	11/0/19	15/0/15
p-value		0.9099	0.4284	0.2452	0.2536	0.5304
	SetCover	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.9947	0.9694	0.9959	0.9835	0.9636	0.9632
Ranks	2.7500	3.5000	3.4000	4.7667	3.5333	3.0500
Win/draw/loss		10/2/18	10/1/19	8/0/22	11/1/18	11/1/18
p-value		0.3337	0.6509	0.0300✗	0.5786	0.6509

for SetCover there was a slight decrement of the performance.

In terms of reduction, for FAST, FCBF and LVF the behavior did not change, while for ReliefF and SetCover boosting was worse in terms of reduction ability than when using all the datasets. As a summary, we can state that the proposed boosting approach was also efficient when the datasets have a large number of features.

5.4. Alpha effect

One of the key parameters of our approach is the value of α (see eq. 5). In this section, we study the behavior of our method when this parameter is given different values. Because we are proposing a method of boosting feature selection algorithms, we are more interested in classification performance than reduction.

Table 13: Comparison for datasets with more than 1000 features of the five boosting methods in terms of κ using the five feature selection algorithms against the feature selection method alone for an SVM as a classification method. The table shows the win/loss record of every boosting method against the feature selection method alone and the p-value of the Wilcoxon test. Significant differences in favor of our method are marked with a ✓; significant differences against our approach are marked with a ✗.

	FAST	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.8374	0.9786	0.9317	0.9891	0.9751	0.9713
Ranks	4.1333	3.3167	3.4500	3.0667	3.5000	3.5333
Win/draw/loss		10/18/2	9/18/3	10/20/0	10/18/2	8/20/2
p-value		0.0226✓	0.0912	0.0065✓	0.0266✓	0.0715
	FCBF	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.8131	0.9429	0.9543	0.9670	0.9374	0.9293
Ranks	4.4500	3.3667	3.1167	2.7333	3.5667	3.7667
Win/draw/loss		14/13/3	15/13/2	15/13/2	15/9/6	14/13/3
p-value		0.0074✓	0.0021✓	0.0012✓	0.0303✓	0.0106✓
	LVF	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.9858	0.9531	0.8554	0.9523	0.9874	0.9878
Ranks	2.8167	3.3500	4.7000	3.5833	3.3000	3.2500
Win/draw/loss		2/19/9	2/10/18	2/17/11	4/18/8	3/19/8
p-value		0.0634	0.0002✗	0.0273✗	0.2542	0.1779
	ReliefF	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.7250	0.9883	0.9839	0.9760	0.9868	0.9853
Ranks	4.3000	3.3333	3.3667	3.2500	3.2500	3.5000
Win/draw/loss		14/12/4	14/11/5	13/14/3	14/12/4	14/11/5
p-value		0.0057✓	0.0074✓	0.0077✓	0.0050✓	0.0089✓
	SetCover	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.5180	0.9303	0.9139	0.9659	0.9083	0.8812
Ranks	5.7667	2.5167	2.9000	2.1500	3.4833	4.1833
Win/draw/loss		30/0/0	29/0/1	30/0/0	28/0/2	26/0/4
p-value		0.0000✓	0.0000✓	0.0000✓	0.0000✓	0.0000✓

However, it is interesting to know the behavior of our approach depending on the value assigned to α. We repeated the experiments for AdaBoost and FCBF as representatives of boosting and feature selection methods, respectively. We used five different values of α: $\alpha = \{0.1, 0.25, 0.5, 0.75, 0.9\}$.

The behavior for decision trees is shown in Figure 13, and relative movement diagrams are shown in Figure 14. The figures show that the value used for α strikes a good compromise between classification performance and reduction. Although larger increments of κ could be achieved with a greater value of α, the cost is a barrier to the reduction ability of the feature selection algorithm. These results also show the flexibility of our approach, as we can control the balance between classification performance and reduction depending on the problem we

Table 14: Comparison for datasets with more than 1000 features of the five boosting methods in terms of reduction using the five feature selection algorithms against the feature selection method alone for an SVM as a classification method. The table shows the win/loss record of every boosting method against the feature selection method alone and the p-value of the Wilcoxon test. Significant differences in favor of our method are marked with a ✓; significant differences against our approach are marked with a ✗.

	FAST	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.9530	0.9260	0.8989	0.9016	0.4912	0.9064
Ranks	3.9000	2.5333	3.0500	2.9333	5.5667	3.0167
Win/draw/loss		21/0/9	23/0/7	21/0/9	0/0/30	22/0/8
p-value		0.0978	0.0300✓	0.2210	0.0000✗	0.2536
	FCBF	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.9783	0.9208	0.9433	0.9097	0.6518	0.9216
Ranks	2.9333	2.9667	2.4167	3.9833	5.4000	3.3000
Win/draw/loss		12/0/18	15/0/15	14/0/16	2/0/28	15/0/15
p-value		0.2210	0.4528	0.1470	0.0000✗	0.1779
	LVF	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.5057	0.9201	0.9242	0.9191	0.9098	0.9081
Ranks	5.9667	3.0667	2.0667	3.0667	3.4833	3.3500
Win/draw/loss		30/0/0	29/0/1	30/0/0	30/0/0	30/0/0
p-value		0.0000✓	0.0000✓	0.0000✓	0.0000✓	0.0000✓
	ReliefF	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.8907	0.9065	0.9106	0.9036	0.8683	0.9148
Ranks	3.5833	3.5667	3.5167	3.1833	3.8667	3.2833
Win/draw/loss		17/0/13	15/0/15	16/1/13	13/0/17	16/0/14
p-value		0.4405	0.5577	0.3991	0.5170	0.4528
	SetCover	AdaBoost	FloatBoost	GentleBoost	MultiBoost	ReweightBoost
Mean	0.9947	0.8391	0.8630	0.8191	0.8030	0.7951
Ranks	1.2667	3.7000	3.8167	4.9667	3.7167	3.5333
Win/draw/loss		1/0/29	1/0/29	1/0/29	1/0/29	4/0/26
p-value		0.0000✗	0.0000✗	0.0000✗	0.0000✗	0.0010✗

are addressing.

The results for SVM are shown in Figures 15 and 16. Similar behavior is observed for this classifier. Again, $\alpha = 0.5$ is a good compromise between classification performance and reduction.

5.5. Noise effect

One of the known weaknesses of boosting is its sensitivity to noise. Because we are proposing a way of boosting feature selection, it is useful to know whether this proposal is also more sensitive to noise than standard feature selection. In this section, we show the results of the experiments we carried out to check the sensitivity to noise of our proposal compared with standard feature selection. Again, we selected as representatives the AdaBoost and FCBF feature selection

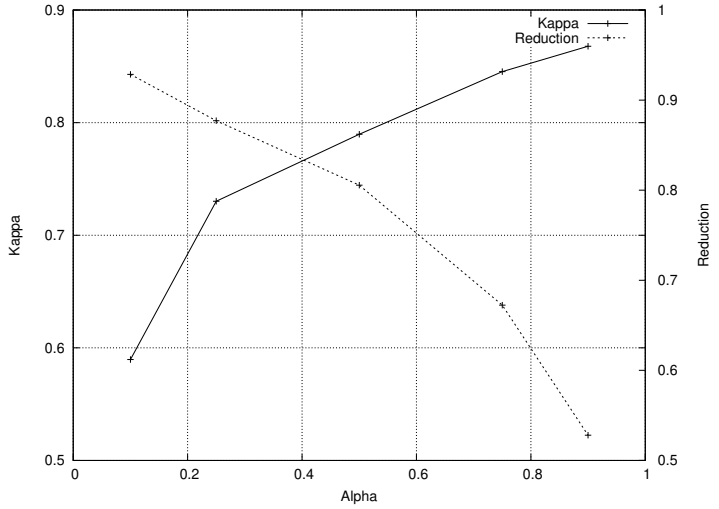


Figure 13: Behavior of κ and reduction for different values of α for AdaBoost for C4.5.

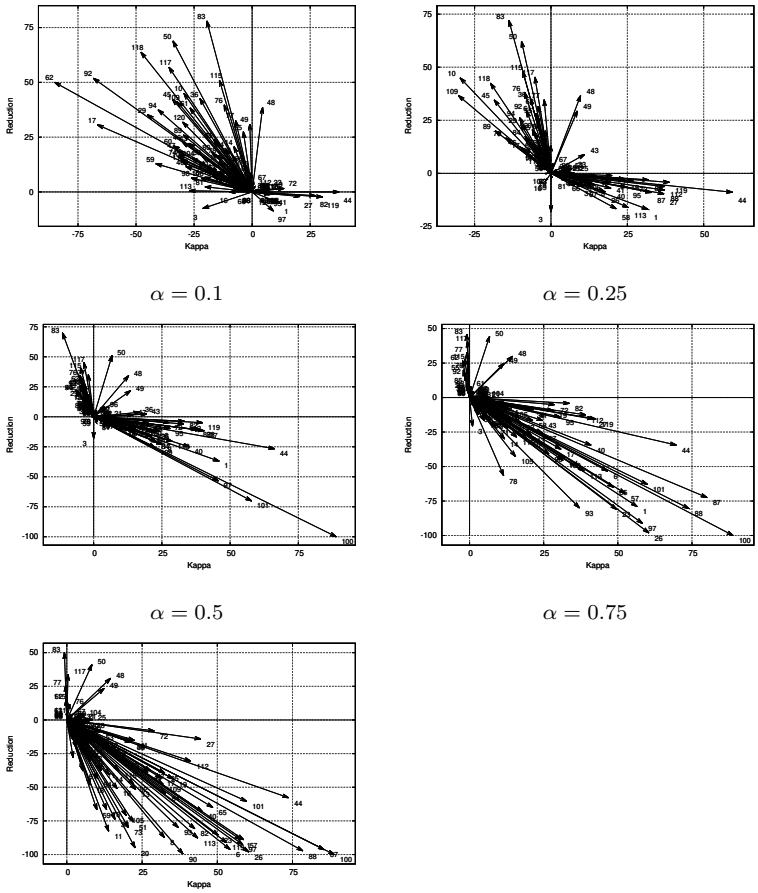
algorithms.

To add noise to the class labels, we followed the method of Dietterich [59]. To add classification noise at a rate ρ, we chose a fraction ρ of the patterns and changed their class labels to be incorrect, choosing uniformly from the set of incorrect labels. We chose all the datasets and three rates of noise: 5%, 10%, and 20%. With these three levels of noise, we performed the experiments using FCBF and AdaBoost and the two classification methods.

The behavior for decision trees is shown in Figure 17, and relative movement diagrams are shown in Figure 18. In terms of classification performance, the figure shows that boosting was more robust than FCBF. In fact, the improvement of AdaBoost.FCBF over FCBF increased with the level of noise. In terms of reduction, AdaBoost.FCBF was most conservative, as more noise was added. For a noise level of 20%, the FCBF reduction was higher than its boosted counterpart.

This behavior is clearly illustrated in Figure 18. We see that as the noise level is increased, more arrows are pointing to the bottom right part of the figure.

The results for an SVM are illustrated in Figures 19 and 16. The behavior of SVM is the same observed for decision trees in both classification performance



$\alpha = 0.1$ $\alpha = 0.25$

$\alpha = 0.5$ $\alpha = 0.75$

$\alpha = 0.9$

Figure 14: κ/reduction using relative movement diagrams for our proposal against the standard FCBF method using the AdaBoost methods and different values of α for C4.5. Positive values on each axis indicate better performance by our method.

and reduction.

6. Conclusions and future work

In this paper, we have presented a general framework of boosting for feature selection algorithms. Considering feature selection as a two-class classification process, we successfully boosted feature selection algorithms. A comprehensive set of experiments was performed using different boosting algorithms, feature selection methods and two classifiers. The boosting methods tested showed

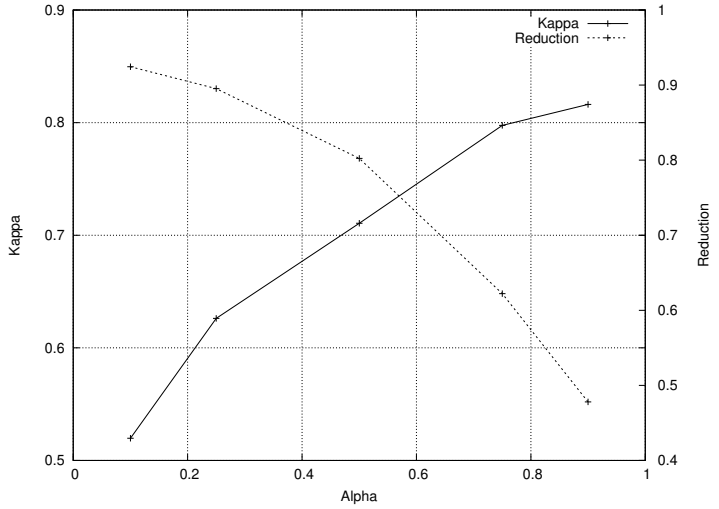
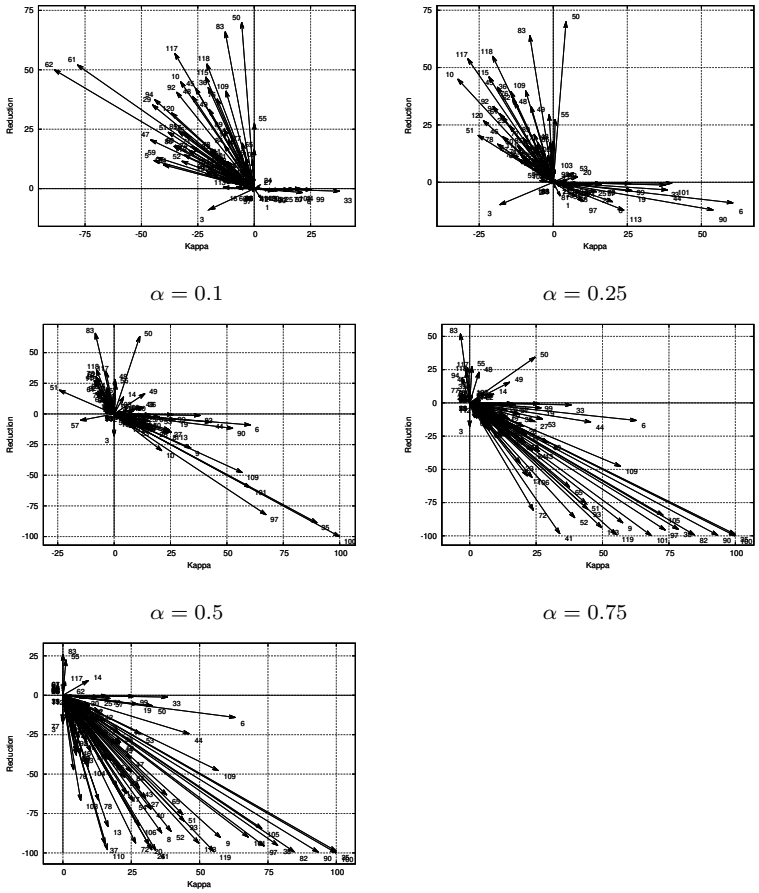


Figure 15: Behavior of κ and reduction for different values of α for AdaBoost for svm.

consistently better performance than the standard feature selection algorithms alone. Boosting improved the classifier performance of the methods in the majority of cases. These results were corroborated for the case of datasets with a large number of features.

The proposed methodology opens many possibilities for future research. An obvious first line of research is related to diversity. As is well-known, diversity is one of the most important features for boosting success. Thus, using the many known diversity enforcing techniques might be one way to improve the results presented in this paper.

Another way of improving the proposed method is by addressing the scalability of the approach. Stratified sampling is a common approach for dealing with scalability problems. The adaptation of our approach to this framework would be a very promising line of future research. Furthermore, other methods commonly used for improving the scalability of feature selections algorithms, such as democratization [60], could benefit from the approach presented in this paper.



$\alpha = 0.1$ $\alpha = 0.25$

$\alpha = 0.5$ $\alpha = 0.75$

$\alpha = 0.9$

Figure 16: κ/reduction using relative movement diagrams for our proposal against the standard FCBF method using the AdaBoost methods and different values of α for SVM. Positive values on each axis indicate better performance by our method.

Bibliography

[1] V. Kumar, S. Minz, Feature selection: A literature review, Smart Computing Review 4 (2014) 211–228.

[2] A. Blum, P. Langley, Selection of relevant features and examples in machine learning, Artificial Intelligence 97 (1997) 245–271.

[3] Y. Liu, X. Yao, Q. Zhao, T. Higuchi, Evolving a cooperative population of neural networks by minimizing mutual information, in: Proc. of the

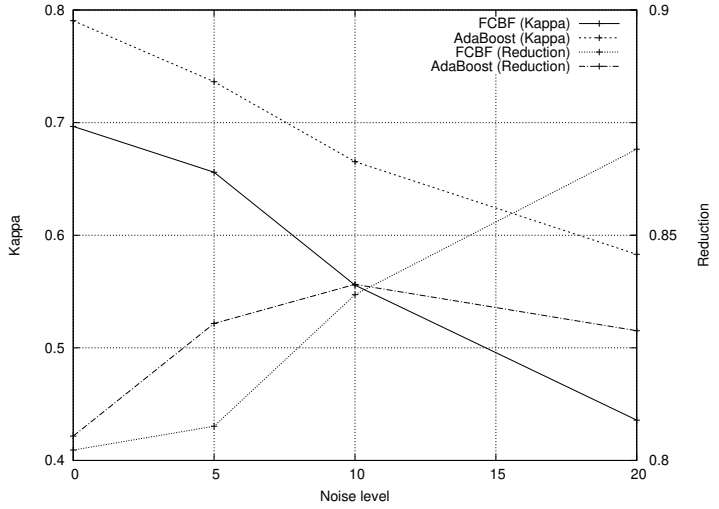
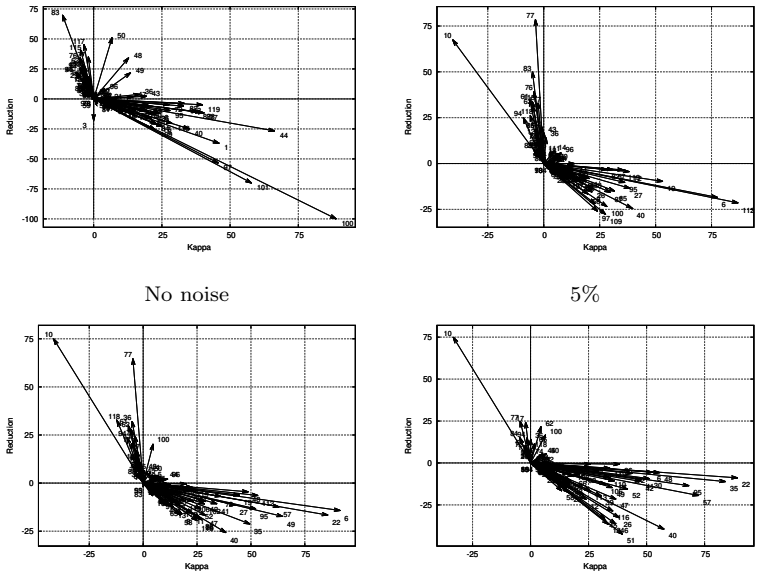


Figure 17: Behavior of κ and reduction for different levels of noise for standard FCBF and AdaBoost for C4.5.



No noise 5%

10% 20%

Figure 18: κ/reduction using relative movement diagrams for our proposal against the standard FCBF method using the AdaBoost methods and three different levels of noise for C4.5. Positive values on each axis indicate better performance by our method.

2001 IEEE Congress on Evolutionary Computation, Seoul, Korea, 2001, p. 384–389.

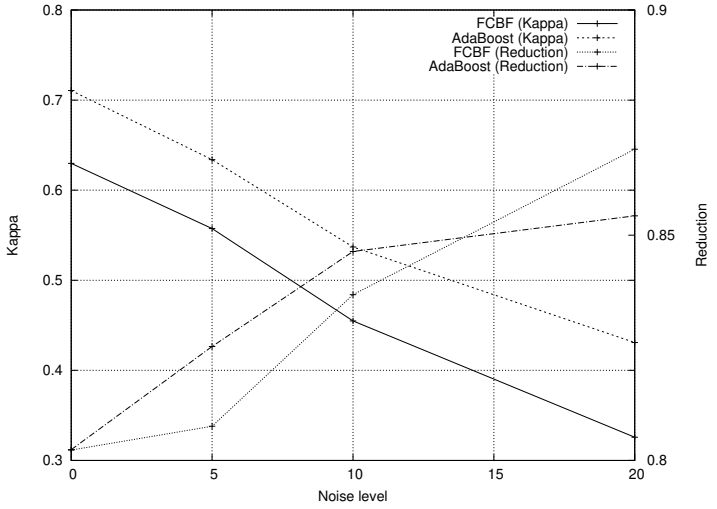


Figure 19: Behavior of κ and reduction for different levels of noise for standard FCBF and AdaBoost for SVM.

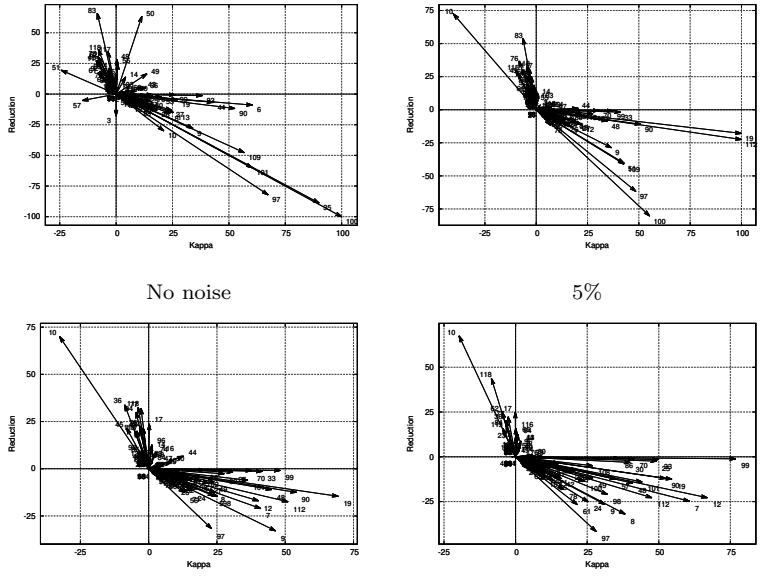


Figure 20: κ/reduction using relative movement diagrams for our proposal against the standard FCBF method using the AdaBoost methods and three different levels of noise for SVM. Positive values on each axis indicate better performance by our method.

[4] J. Lee, D.-W. Kim, Mutual information-based multi-label feature selection using interaction information, Expert Systems with Applications 42 (4)

(2015) 2013–2025.

[5] P. Moradi, M. Gholampour, A hybrid particle swarm optimization for feature subset selection by integrating a novel local search strategy, Applied Soft Computing Journal 43 (2016) 117–130.

[6] S. Li, S. Oh, Improving feature selection performance using pairwise pre-evaluation, BMC Bioinformatics 17 (1).

[7] Y. Saeys, T. Abeel, S. Degroeve, Y. V. de Peer, Translation initiation site prediction on a genomic scale: beauty in simplicity, Bioinformatics 23 (2007) 418–423.

[8] H. Brighton, C. Mellish, Advances in instance selection for instance-based learning algorithms, Data Mining and Knowledge Discovery 6 (2002) 153–172.

[9] A. Jain, D. Zongker, Feature selection: Evaluation, application, and small sample performance, IEEE Transactions on Pattern Analysis and Machine Intelligence 19 (1997) 153–158.

[10] P. Mitra, C.-A. Murthy, S.-K. Pal, Unsupervised feature selection using feature similarity, IEEE Transactions on Pattern Analysis and Machine Intelligence 24 (2002) 301–312.

[11] R. Kohavi, G.-H. John, Wrappers for feature subset selection, Artificial Intelligence 97 (1-2) (1997) 273–324.

[12] M. Dash, K. Choi, P. Scheuermann, H. Liu, Feature selection for clustering - a filter solution, in: Proceedings of the Second International Conference on Data Mining, 2002, p. 115–122.

[13] Y. Kim, W.-N. Street, F. Menczer, Feature selection in unsupervised learning via evolutionary search, in: The 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM Press, 2000, p. 365–369.

[14] E. Leopold, J. Kindermann, Text categorization with support vector machines. how to represent texts in input space?, Machine Learning 46 (1-3) (2002) 423–444.

[15] K. Nigam, A.-K. Mccallum, S. Thrun, T. Mitchell, Text classification from labeled and unlabeled documents using em, in: Machine Learning, 1999, p. 103–134.

[16] Y. Rui, T.-S. Huang, Image retrieval: Current techniques, promising directions and open numbers, Journal of Visual Communication and Image Representation 10 (1999) 39–62.

[17] D.-L. Swets, J.-J. Weng, Efficient content-based image retrieval using automatic feature selection, in: In IEEE International Symposium on Computer Vision, 1995, p. 85–90.

[18] K. Ng, H. Liu, Customer retention via data mining, AI Review 14 (1999) 590.

[19] W. Lee, S.-J. Stolfo, K.-W. Mok, Adaptive intrusion detection: a data mining approach, Artificial Intelligence Review 14 (2000) 533–567.

[20] E.-P. Xing, M.-I. Jordan, R.-M. Karp, Feature selection for high-dimensional genomic microarray data, in: The 18th International Conference on Machine Learning, Morgan Kaufmann, 2001, p. 601–608.

[21] P. Narendra, K. Fukunaga, Branch, and bound algorithm for feature subset selection, IEEE Transactions Computer C-26 (9) (1977) 917–922.

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

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

[24] S. Z. Li, Z. Zhang, Floatboost learning and statistical face recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence 26 (9) (2004) 1–12.

[25] J. H. Friedman, T. Hastie, R. Tibshirani, Additive logistic regression: A statistical view of boosting, Annals of Statistics 28 (2) (2000) 337–407.

[26] Q. Song, J. Ni, G. Wang, A fast clustering-based feature subset selection algorithm for high-dimensional data, IEEE Transactions on Knowledge and Data Engineering 25 (2013) 1–14.

[27] L. Yu, H. Liu, Efficient feature selection via analysis of relevance and redundancy, Journal of Machine Learning Research 5 (2004) 1205–1224.

[28] H. Liu, R. Setiono, Scalable feature selection for large sized databases, in: In Proceedings of the Fourth World Congress on Expert Systems, Morgan Kaufmann, 1998, p. 521—528.

[29] I. Kononenko, Estimating attributes: Analysis and extensions of relief, in: European Conference on Machine Learning, 1994, p. 171–182.

[30] K. Kira, L. A. Rendell, A practical approach to feature selection, in: Proc. 9th international workshop on Machine learning, Morgan Kaufmann, 1992, p. 249–256.

[31] M. Dash, H. Liu, Feature selection for classification, Intelligent Data Analysis 1 (1997) 131–156.

[32] D. Johnson, Approximation algorithms for combinatorial problems, Journal of Computer and Systems Sciences 9 (1974) 256–278.

[33] K. Tieu, P. Viola, Boosting image retrieval, International Journal of Computer Vision 56 (2004) 17–36.

[34] E. Tuv, A. Borisov, G. Runger, K. Torkkola, Feature selection with ensembles, artificial variables, and redundancy elimination, Journal of Machine Learning Research (2009) 1341–1366.

[35] X.-C. Yin, C.-P. Liu, Z. Han, Feature combination using boosting, Pattern Recognition Letters 26 (2005) 2195–2205.

[36] J. Y. Choi, Y. M. Ro, K. N. Plataniotis, Boosting color feature selection for color face recognition, IEEE Transactions on Image Processing 20 (5) (2011) 1425–1434.

[37] Q.-G. Miao, Y. Cao, J.-F. Song, J. Liu, Y. Quan, Boostfs: A boosting-based irrelevant feature selection algorithm, International Journal of Pattern Recognition and Artificial Intelligence 29 (2015) 1–18.

[38] D. B. Redpath, K. Lebart, Boosting feature selection, in: Procceding of the International Conference on Pattern Recognition and Image Analysis, Vol. 3686 of Lecture Notes in Computer Science, Springer, 2005, p. 305–314.

[39] J. O'Sullivan, J. Langford, R. Caruna, A. Blum, Featureboost: A metalearning algorithm that improves model robustness, in: Proceedings of the Seventeenth International Conference on Machine Learnin, 2000, p. 703–710.

[40] S. He, H. Chen, Z. Zhu, D. G. Ward, H. J. Cooper, M. R. Viant, J. K. Heath, X. Yao, Robust twin boosting for feature selection from high-dimensional omics data with label noise, Information Sciences 291 (2015) 1–18.

[41] K. Bailly, M. Milgram, Boosting feature selection for neural network based regression, Neural Networks 22 (2009) 748–756.

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

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

[44] Q. Liu, J. Yamg, K. Zhang, Y. Wu, Adaptive compressive tracking via online vector boosting feature selection, IEEE Transactions on CyberneticsIn press.

[45] C. Huang, H. Ai, Y. Li, , S. Lao, Vector boosting for rotation invariant multi-view face detection, in: Proceedings of the IEEE International Conference on Computer Vision, Vol. 1, Beijing, China, 2005, p. 446–453.

[46] Y. Freund, An adaptive version of the boost by majority algorithm, Machine Learning 43 (3) (2001) 293–318.

[47] P. Wei, Q. Hu, P. Ma, X. Su, Robust feature selection based on regularized brownboost loss, Knowledge-Based Systems 54 (2013) 180–198.

[48] Z. Yu, D. Wang, J. You, H.-S. Wong, S. Wu, J. Zhang, G. Han, Progressive subspace ensemble learning, Pattern Recognition 60 (2016) 692–705.

[49] Z. Yu, L. Li, J. Liu, G. Han, Hybrid adaptive classifier ensemble, IEEE Transactions on Cybernetics 42 177–190.

[50] H. Liu, L. Liu, H. Zhang, Boosting feature selection using information metric for classification, Neurocomputing 73 (2009) 295–303.

[51] Z. Yu, H. Chen, J. You, H.-S. Wong, J. Liu, L. Li, G. Han, Double selection based semi-supervised clustering ensemble for tumor clustering from gene expression profiles, IEEE/ACM Transactions on Computational Biology and Bioinformatics 11 727–740.

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

[53] P. B. Nemenyi, Distribution-free multiple comparisons, Ph.D. thesis (1963).

[54] A. Ben-David, A lot of randomness is hiding accuracy, Engineering Applications of Artificial Intelligence 20 (7) (2007) 875–885.

[55] M. Friedman, A comparison of alternative tests of significance for the problem of m rankings, Annals of Mathematicals Statistics 11 (1940) 86–92.

[56] J. R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, San Mateo, 1993.

[57] V. Vapnik, The nature of Statistical Learning Theory, Springer Verlag, New York, 1999.

[58] J. Maudes-Raedo, J. J. Rodríguez-Díez, C. García-Osorio, Disturbing neighbors diversity for decision forest, in: G. Valentini, O. Okun (Eds.), Workshop on Supervised and Unsupervised Ensemble Methods and Their Applications (SUEMA 2008), Patras, Grecia, 2008, p. 67–71.

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

[60] C. García-Osorio, A. de Haro-García, N. García-Pedrajas, *Democratic* instance selection: a linear complexity instance selection algorithm based on classifier ensemble concepts, Artificial Intelligence 174 (2010) 410–441.