Master's thesis

(Trabajo de Fin de Máster)

Pruning boosting ensembles

Sergio García-Moratilla
sergio.garcia@uam.es
Advisor: Alberto Suárez
alberto.suarez@uam.es

8th October 2009

Abstract

A classifier is a system that discriminates among the instances in a dataset on the basis of their attributes and assigns them a class label [1, 2]. An ensemble is a set of classifiers whose individual decisions are combined to produce a final classification [2]. Ensemble learning is an important area of research because ensembles are often more accurate than the individual classifiers that compose them. Specifically, if the classifiers of the ensemble are complementary, that is, if their errors are uncorrelated, then the generalization error of the ensemble is lower than the error of the base classifiers [2, 3].

Well-known ensemble methods are bagging [4] and boosting [5, 6]. Bagging builds classifiers from bootstrap samples of the training dataset. Bagging is a parallel ensemble method. Given a training set, the classifiers built by bagging are generated independently. Boosting constructs classifiers by modifying the weights of the training instances, so that the new classifiers progressively focus on instances that are difficult to classify by previous members of the ensemble. Boosting is a sequential algorithm, in which each classifier depends on the previous classifiers that have been generated.

In most cases, specially for bagging, the generalization performance improves with the size of the ensemble. Therefore, large ensembles are used. This leads to large memory requirements and slow speeds of training and classification. To alleviate these shortcomings, there are techniques that attempt to reduce the size of the ensemble without

deteriorating its performance. These techniques are called ensemble pruning methods. An example of these methods is ordered aggregation [3]. Ordered aggregation is based on modifying the order in which the classifiers are aggregated in the ensemble and stopping when the number of classifiers in the subensemble is sufficient [3]. The order is modified so that the classifiers which are expected to perform better when combined are aggregated first. Most of the measures used in ordered aggregation methods are based on errors of pairs of classifiers or measures of complementarity between classifiers. While these techniques generally perform well in bagging ensembles [3], where the order of aggregation is left unspecified, they are not expected to work as well in boosting, where the order of the classifiers is prescribed by the ensemble learning algorithm. Furthermore, boosting often drives the training error of the ensemble to 0 in the first few iterations [6], therefore the measures used in ordered aggregation, which are often based on estimates of the error on the training set cannot distinguish among the different classifiers [3].

Another method for pruning ensembles is based on semidefinite programming (SDP) [7]. SDP problems are convex optimization problems defined by a semidefinite positive matrix with a quadratic target function and linear restrictions [8]. In SDP pruning, ensemble pruning is redefined as a minimization problem using a measure $\tilde{g}$ that takes into account both the classification errors of the individual ensemble classifiers and the pairwise complementarity of these classifiers [7]. This measure $\tilde{g}$ can be written in terms of a semidefinite positive matrix whose diagonal elements are the errors of each classifier and whose off-diagonal elements are the common errors for pairs of classifiers. Even if the subensembles have 0 error, the measure $\tilde{g}$ will generally be different from 0. [7]. In this work, we analyze pruning methods based on this measure. Specifically, we propose using ordered aggregation (greedy search), a genetic algorithm [9] and simulated annealing [10] to obtain pruned subensembles from a boosting ensemble and compare them to SDP pruning and to early stopping. There are different implementations of the genetic algorithm that can be used [11]. Specifically, we use a genetic algorithm with set encoding and RAR crossover operator [12, 13]. This operator has the property that it preserves the cardinality of the solutions.

The performance of subensembles of sizes 20%, 30% and 50% of the original size is evaluated on benchmark classification problems from UCI [14]. The optimal size for pruned subensembles generated by bagging is between 20% and 40% of the original ensemble size [3]. Our experiments show that boosting cannot be pruned by a large amount significant deterioration of performance. Pruned ensembles whose size

is 20% of the original ensemble are slightly worse than complete boosting, but outperform boosting ensembles of the same size obtained by early stopping. For larger sizes, our experiments show that the pruned ensembles obtain a generalization error which is lower or comparable to the complete ensemble generated by boosting in most of the problems investigated. A Nemenyi's test based on average ranks [15] indicates that the differences between the complete ensemble generated by boosting and pruned subensembles whose size are 30%-50% of the complete ensemble are not statistically significant. Therefore, boosting ensembles can be pruned to smaller sizes without deterioration of their performance. The experiments also show that there are no significant differences among the pruning strategies considered. In terms of computational cost, pruning based on ordered aggregation should be preferred. This greedy method obtains subensembles for which the estimate of $\tilde{g}$ on the training set is worse, but whose generalization performance is equivalent or better than the other methods.

Contents

1 Introduction

A classifier is a system that predicts a class label for the elements of an unlabeled dataset [2]. Classifiers are built from a set of labeled instances, which is called the training dataset. The objective is to identify regularity patterns based on the information present in the training dataset to classify unseen instances [1]. An ensemble is a collection of classifiers whose final prediction is obtained by combining the predictions of the individual classifiers [2]. Ensembles are useful because their predictions can be more accurate than the individual classifiers that compose them. Specifically, the generalization error of the ensemble is lower than the error of the ensemble members when the errors of the base classifiers are uncorrelated [2, 3].

There are different methods to generate ensembles: manipulating the training examples, modifying the attributes or the class of the instances, injecting randomness in the learning algorithm or using different base learning algorithms [2]. Section 2 provides a review of these types of ensemble learning algorithms. Bagging [4] and boosting [5, 6] are two well-known ensemble methods. Bagging builds a collection of classifiers from bootstrap samples of the training dataset [4]. Bootstrap samples are generated by random sampling with replacement from the original training dataset. Boosting constructs classifiers by modifying the weights of the training instances, so that classifiers progressively focus on instances that are erroneously classified by previous ensemble members [5, 6]. Boosting is a sequential algorithm, in which each generated classifier depends on the previous one. Bagging and boosting are described in detail in Section 3.

Despite their potential improvements in accuracy, the use of ensembles has several drawbacks. Specifically, they are complex classification systems whose predictions are difficult to understand in terms of simple rules. Computationally, ensembles have large memory requirements, are costly to train and slow in classification. These aspects can be critical in online applications. In order to bring down the memory requirements and to improve the classification speed, the size of the ensemble can be reduced using pruning methods without a significant deterioration of performance. These pruning methods

are based on selecting classifiers that optimize quantities that are expected to be correlated with the generalization performance of the subensemble. These quantities need to take into account the complementarity among the selected classifiers to ensure that the accuracy of the ensemble is improved [2, 3].

Ordered aggregation is an example of an ensemble pruning method [3]. Ordered aggregation modifies the order in which the classifiers are aggregated in the ensemble and then selects the first classifiers in the re-ordered ensemble. The number of selected classifiers is chosen to maximize the generalization performance of the pruned subensemble [3]. The order of aggregation is modified so that the classifiers which are expected to perform better when combined are included first. For this purpose, the process is guided by measures that favor the selection of classifiers that are complementary. They are based on pairwise errors or measures of accuracy and diversity. This pruning method is shown to perform well in bagging ensembles, where the order in which the predictions of the base classifiers are aggregated is not specified by the algorithm [3]. Ordered aggregation is not expected to work as well in boosting ensembles because the order of aggregation of the classifiers in the ensemble is prescribed by the algorithm. Furthermore, the ensembles generated by boosting often reach zero training errors in the first few iterations [6], which implies that measures based on errors on the training dataset cannot distinguish among the different subensembles [3].

SDP pruning follows a different approach [7]. This method identifies the pruned subensemble that minimizes a combined measure of accuracy and diversity, $\tilde{g}$. This measure takes into account both the individual classification errors and the pairwise complementarity of the classifiers [7]. The minimization of $\tilde{g}$ can be written as a quadratic-integer problem whose solutions can be approximated by semidefinite programming (SDP). SDP problems are convex optimization problems defined by a semidefinite positive matrix with a quadratic target function and linear restrictions [8]. In ensemble pruning, the linear restriction is the target size of the pruned subensemble. The measure $\tilde{g}$ does not have the problems that measures based on the training error have. Even when the ensemble has near zero error on the training dataset, the measure $\tilde{g}$ can be different from 0. Diagonal terms account for the errors of the individual classifiers and off-diagonal terms accounts for the common errors between classifiers. These two quantities are usually different from 0 [7]. Therefore, this measure could be used to prune boosting ensembles. $\tilde{g}$ can be optimized using SDP as in [7], or by means of a different optimization technique. Specifically, in this work we propose to use an ordered aggregation based on the minimization of $\tilde{g}$ (greedy search), a genetic algorithm [9] and simulated annealing [10] to obtain pruned subensembles from a complete ensemble generated by boosting. These pruning methods and the measures

of complementarity in which these methods are based are reviewed in Section 4.

The performance of these methods, including SDP pruning and early stopping of the boosting algorithm is analyzed on a series of experiments on benchmark classification problems. First, the optimality of the different optimization techniques is investigated by comparing the approximate solutions with the exact ones obtained by an exhaustive search in boosting ensembles of medium size. Then, the effectiveness of the methods are tested using ensembles composed of 101 classifiers. Finally, we analyze the complexity of the methods in terms of computation time. These experiments are described in Section 5.

2 Ensemble methods

A *supervised classification problem* consists in learning an unknown function $y = f(\mathbf{x})$ from a collection of labeled instances $\{(\mathbf{x}_i, y_i); i = 1, 2, \ldots, N\}$. The quantity $\mathbf{x}_i$ is the vector of *attributes* and y_i is the class label of the example i. Attributes can be discrete or continuous. The class label takes values only in a discrete set $\{c_i; i = 1, 2, \ldots, L\}$.

From a training dataset, $\mathcal{Z}_{train}$, a learning algorithm constructs a *classifier*, which is a hypothesis h of the function f. This classifier predicts the class $y = h(\mathbf{x})$ for unlabeled instances $\mathbf{x}$. *Ensemble methods* generate a collection of classifiers whose decisions are pooled to obtain a combined decision. Empirical investigations show that ensembles can be more accurate than the individual classifiers of which they are composed [2, 3, 16, 17, 18, 19].

Ensemble construction raises some questions. First, under which conditions it is possible to combine classifiers to obtain more accurate prediction. Second, how to construct complementary base classifiers. Ensembles obtained by combining similar classifiers will have a performance that is close to the individual ones. Finally, once the base classifiers are constructed, how are their outputs combined in a single prediction so that the generalization error of this consensus decision is as low as possible.

Dietterich [2] describes three mechanisms by which ensembles can improve the performance of individual predictors. The first is statistical: a learning algorithm can be viewed as a search in the space of hypotheses $\mathcal{H}$ to identify the hypothesis that is closest to the function we want to predict. The amount of training data is usually too small to accurately guide the search in the typically large space of hypotheses. Therefore, the same learning algorithm can generate different classifiers whose accuracy is similar on the training data but have different generalization accuracy on a test dataset. Instead of

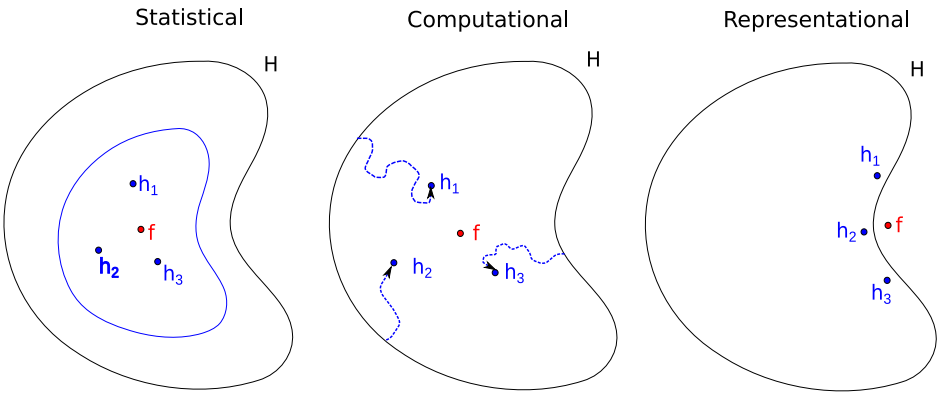

Figure 1: Graphical representation of the mechanisms by which ensembles can perform better than a single classifier. The outer curve represents the hypothesis space $\mathcal{H}$. The inner curve indicates the set of hypothesis with good accuracy on the training dataset. The f point is the actual function and h_t are the different hypothesis generated by a learning algorithm [2].

selecting a single classifier and running the risk of choosing a classifier that has poor performance, we can use an ensemble that combines the outputs of the different predictors. A graphical illustration is shown in Figure 1 (top left). The point labeled with an f is the target function to learn. Hypotheses $\{h_1, h_2, h_3\}$ are the classifiers generated by the learning algorithm. A solution that is closer to f can be obtained by averaging the hypotheses rather than by selecting a single classifier.

The second reason is computational. Many learning algorithms perform local searches that may get trapped in local optima. For example, neural networks are often trained by minimizing the training error with gradient descent. Decision trees perform successive splits on the training data based on the previous splits. This is a greedy search on the space of hypotheses based on selecting the splits that are locally optimal. Therefore, the solution obtained by the learning algorithm can be a local optimum instead the global one. If each classifier that is included in the ensemble is constructed from different starting points, the ensemble may provide a better approximation to the target function than any of the individual classifiers. This idea is depicted in Figure 1 (center).

The third reason is representational. The expressive capacity of the hypothesis that can be generated by the learning algorithm may be insufficient to accurately approximate the solution of the problem. For example, the XOR problem cannot be solved by linear classifiers, such as Fisher's discriminant or decision stumps. Even though a single linear classifier cannot solve a

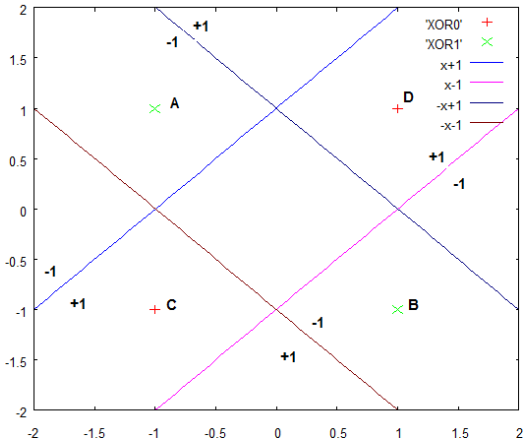

Figure 2: Ensemble solution for the XOR problem in bipolar encoding $(+1, -1)$. Straight lines trace the decision boundary of a linear classifier. The numbers represent the output that the classifier assigns to each side of the frontier. The class of the points A and B is -1 while the class of points C and D is $+1$. A and B receives three votes for class -1 and one for $+1$. C and D receives three votes for class $+1$ and one for -1. Using majority vote to select the final class, the ensemble formed by these four linear classifier solves the XOR problem.

non-linear problem, an ensemble composed of linear classifiers can, as shown in Figure 2.

These three arguments are heuristic. Therefore there is no guarantee that the ensemble will always perform better than a single classifier [16].

The key idea behind ensemble methods is the generation of complementary classifiers. Combining classifier whose outputs are similar cannot lead to improvements in performance. To obtain complementary classifiers, several methods exist [2]:

- Manipulating the training dataset:
  The same learning algorithm is executed using different training examples. This technique performs well with unstable algorithms. A learning algorithm is unstable when small changes in the training dataset can produce large changes in the output of the classifier. For example, neural networks and decision trees are generally unstable classifiers, while k-nearest neighbors is not. One of the more common ensemble methods is bagging, which builds different classifiers using different bootstrap samples of the original training dataset. Bootstrap samples are obtained by drawing instances randomly with replacement from

the original training data. Another well-known technique is boosting, which assigns weights to the instances and generates classifiers that minimize the weighted error using these weights. After each iteration, the weights are modified so that the next classifier focuses on the instances misclassified by the previous classifier. Bagging and boosting are described in detail in Section 3.

- Manipulating the attributes of the training instances:
  Instead of using all the available attributes, the base classifiers can be trained on different subsets of the attributes. The subsets of attributes can be generated at random, as in *randomized trees (extra-trees)* [20]. Extra-trees generates a list of attributes selected at random, makes a random split per attribute, and then selects the best split based on a target function.

- Manipulating the class of the training instances:
  This method consists in modifying the values of the class labels of the dataset used to train the classifiers. An example of ensemble construction algorithm that use this technique is *Error-correcting output coding (ECOC)* [21]. ECOC divides the classes into two subsets, A_1 and B_1, at random. It replaces the classes in A_1 by 0 and the classes of B_1 by 1. Using this modified dataset, the classifier h_1 is trained and the process is repeated L times to form the ensemble. To classify an instance $\mathbf{x}$, the predictions of the different classifiers are combined. If the output is 0, all the classes in A_1 receive a vote, otherwise all the classes in B_1 receive a vote. Finally, the most voted class is assigned to the instance $\mathbf{x}$. ECOC is an effective ensemble learning method when the number of classes is high [21].

- Injecting randomness in the learning algorithm:
  Introducing randomness in the learning algorithm is a simple method to obtain different classifiers. This randomness can be injected at some point of the learning algorithm, for example, selecting different initial random seeds in the search for optimal weights in a neural network. Examples of ensemble methods based on this technique are *Random forest* [22] and *class-switching* [23]. *Random forest* is a method that constructs decision trees using some form of randomization, such as the bootstrap sampling used in bagging, or the random selection of splits employed in extra-trees [18, 22]. Extra-trees are a particular form of *random forests* [20]. *Class-switching* modifies the class of instances selected at random with probability p and builds classifiers with train error 0 on the altered dataset. The error of this classifier on the original

training set is p.[23]. The outputs are combined by majority vote. If the number of classifiers aggregated is sufficient high, the errors that are uncorrelated (and in particular, those associated with the switched labels) average out and the final ensemble prediction is typically more accurate. Majority vote is studied in Section 3.1.

- Using different base learning algorithms:
  It is possible to use different base learning algorithms to generate the ensemble. For instance, neural networks, decision trees or k-nearest neighbour can be combined into a single consensus decision. Combining the outputs of classifiers of different nature is one of the most successful methods for classification. For example, ensembles constructed using this technique are one of the best classifiers submitted to the Netflix Prize [24, 25].

The use of ensembles has several drawbacks. Specifically, they are complex classification systems whose predictions are difficult to understand in terms of simple rules. For example, in medicine, a decision tree could be useful to understand how a diagnosis is made or which of the symptoms of a disease have the largest discriminant power. The classification rule given by the decision tree is a hierarchical sequence of Boolean tests on the attributes. Splits on attributes that are more relevant for classification tend to appear at shallower internal nodes in the tree. When the classifier is an ensemble of one hundred decision trees, it can be very difficult to identify the relevant variables, possibly with the exception of the nodes at low depth levels in the decision trees.

A further drawback is that ensembles have large memory requirements and slow speeds of training and classification. These aspects can be critical in online applications. These shortcomings can be alleviated by reducing the size of the ensemble without deteriorating its performance. Pruned subensembles can actually produce more accurate predictions than the complete ensemble, have lower memory requirement and increased speed of classification. Ensemble pruning techniques will be reviewed in Section 4.

3 Bagging and boosting

This section reviews in detail two of the most widely used methods to generate ensembles: bagging and boosting. The algorithms are described in detail and some of their variants are presented.

3.1 Bagging (bootstrap aggregating)

Bagging ensembles are composed of classifiers of the same type. Individual classifiers are generated by applying the same learning algorithm to different bootstrap samples obtained from the original training dataset $\mathcal{Z}_{train} = \{\mathbf{x}_i, y_i; i = 1, \ldots, N\}$. The final prediction of the ensemble is obtained by majority vote [2, 4, 16, 17, 26].

In bagging, *bootstrap samples* of size N are generated by random extractions with replacement from the original training dataset. Since the extractions are independent, on average, 63.2% of the examples in the original data appear in the bootstrap sample. The remaining 36.8% are repeated examples.

The proof is as follows: The probability of an example not being selected in one extraction is $\frac{N-1}{N}$. Since the bootstrap performs N extractions, the probability of not being selected in N independent trials is $(\frac{N-1}{N})^N$. This probability tends to e^{-1} as N tends to infinite.

$$\lim_{N\to\infty}\left(\frac{N-1}{N}\right)^N = \lim_{N\to\infty}\left(1-\frac{1}{N}\right)^N = e^{-1} \simeq 0.36788$$

Since the bootstrap sample has the same size as the original training set, this means that only $1 - e^{-1} \simeq 63.2\%$ different examples are included in a given bootstrap sample. The remaining 36.8% are repeated examples. Because of this property, each individual classifier in bagging is trained with a lower number of different examples than the original training dataset. The generalization error of these individual classifiers is typically larger than the error of a classifier trained with the complete dataset, because some information is discarded. However, aggregating the output of the base classifiers tends to improve the accuracy of the prediction by reducing the variance contribution to the error [26, 27]. This generally leads to ensembles that perform better than a single classifier built with the complete dataset $\mathcal{Z}_{train}$ [16, 28].

If the new bootstrap sample is obtained by sampling without replacement from the original training dataset (*subsampling*), then the method is called *subbagging* [29, 30].

Bagging generally performs better when the base classifiers are unstable. A learning algorithm is unstable when small changes in the training dataset can produce large changes in the output of the classifier. If the learning algorithm is stable, then the classifiers generated from the bootstrap samples will be rather similar. For example, neural networks and decision trees are unstable methods while k-nearest neighbour is not. k-nearest neighbors assigns the majority class in the k instances that are closest to the instance

to classify. Given that in many of the classification problems of interest, a given instance is usually surrounded by instances of the same class, removing a few of them by resampling will not generally modify the decision boundary. However, if the sampling is performed by subbagging with small dataset sizes (below the 50% of the original size) then k-nearest neighbors becomes unstable and can be used as a base classifier for bagging [30].

Algorithm 1 The bagging algorithm

Training

Input: T, the number of classifiers in the ensemble.
Output: The ensemble E_T

- $E_0 = \emptyset$.
- for $k = 1, \ldots, T$
  - Generate a bootstrap sample S_k from $\mathcal{Z}_{train}$.
  - Build a base classifier h_k using S_k as the training set.
  - Include the classifier in the ensemble, $E_k = E_{k-1} \cup h_k$

Classification

- Classify the input instance $\mathbf{x}$ using $\{h_i(\mathbf{x});\ i = 1, \ldots, T\}$.
- Assign the class with the maximum number of votes. Ties are resolved arbitrarily.

Majority vote Bagging combines the outputs of the classifiers by majority voting. That is, it assigns to the instance $\mathbf{x}$ the class label most frequently predicted by the classifiers in the ensemble. Using the indicator $\mathbb{I}(\cdot)$ ($\mathbb{I}(\text{true}) = 1$ and $\mathbb{I}(\text{false}) = 0$), the majority vote assigns the class c_k to the instance $\mathbf{x}$ if:

$$c_k = \underset{c_j}{\operatorname{argmax}} \sum_{i=1}^{T} \mathbb{I}(h_i(\mathbf{x}) = c_j) \tag{1}$$

Ties are resolved arbitrarily. In the case of two classes this rule coincides with the simple majority rule (50% of the votes + 1).

If we assume [31]:

- The number of classifiers, T, is odd to avoid ties in binary class problems.
- The success rate for each classifier is p for any instance $\mathbf{x}$.
- The outputs of the classifiers are independent.

then the probability of correct classification by the ensemble is:

$$P_{maj} = \sum_{m=\lfloor T/2+1 \rfloor}^{T} \binom{T}{m} p^m (1-p)^{T-m} \tag{2}$$

It can be shown that [31]:

- If $p > 0.5$, then P_{maj} is monotonically increasing and $P_{maj} \longrightarrow 1$ as $T \to \infty$.
- If $p < 0.5$, then P_{maj} is monotonically decreasing and $P_{maj} \longrightarrow 0$ as $T \to \infty$.
- If $p = 0.5$, then $P_{maj} = 0.5$ for any T.

That is, majority voting improves the accuracy of a single classifier if the base classifiers are complementary and accurate enough ($p > 0.5$). For example, suppose that three classifiers are built with an accuracy $p = 0.6$. The goal is to classify a dataset composed of ten instances. Figure 3 shows two possible scenarios: the rows correspond to the instances and the columns to the classifiers. Cell (i, j) is shadowed if the classifier h_j correctly predicts the class of instance $\mathbf{x}_i$, and is blank otherwise. The last column represents the correct predictions obtained by majority voting. The first ensemble obtains an accuracy of 0.6 while the second one obtains a higher accuracy of 0.9. The difference is that the second ensemble is composed by complementary classifiers. The most voted class always receives $\lfloor T/2 \rfloor + 1$ votes. Most of the instances in the first ensemble are assigned to the same class label by all the classifiers in the ensemble. In consequence, the errors of the individual classifiers cannot be compensated by the combination of the predictions.

A useful tool in machine learning is the decomposition of the classification error into bias and variance [1, 32, 33]. The bias is a measure of how close, on average over several different training sets, the classifier is to target function f. The variance is a measure of how stable the solution is over several different training sets. While this decomposition is well-defined in regression problems, there are several definitions in classification [1, 32, 33]. Typically, classifiers with a low bias have a larger variance and vice versa. In general,

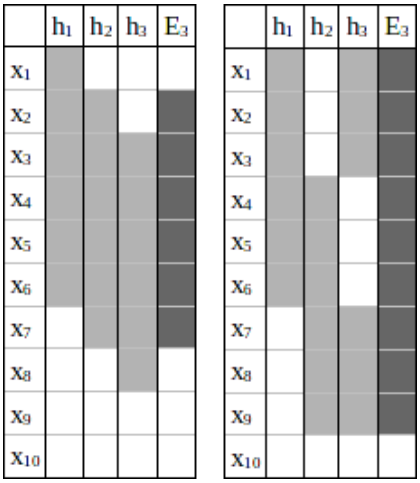

Figure 3: Depicts the global success rate of two possible ensembles formed by three classifier with accuracy $p = 0.6$ using the majority vote to combine the outputs.

bagging builds ensembles that perform better than the base classifiers [4, 18, 26]. These improvements can be explained in terms of a reduction of the variance term in the classification error [26]. Consider an individual classifier included in the ensemble. Its bias is B_0 and its variance is V_0. The bagging ensemble will have a different bias B_1 and a different variance V_1. Bagging builds classifiers using bootstrap samples of the training data. Because of the discarded data in the bootstrap sampling, the bias of the base classifiers is usually slightly larger than the bias of the original base algorithm. So that, the bias of the ensemble B_1 is slightly larger than $B0$. By contrast, bagging averages the outputs, so that the classification is more stable. The variance of the ensemble V_1 must be lower than the variance of the base classifier V_0. Therefore, bagging typically reduces the variance but not the bias [26].

Bagging is a very robust learning algorithm even in classification problems with high levels of noise in the class labels [18, 34].

3.2 Boosting

Boosting generates a sequence of classifiers that progressively focus on examples erroneously classified by previous classifiers in the ensemble. The shift in focus is obtained by modifying the weights of the examples according to the accuracy of the predictions of the most recently generated classifier: weights are increased for misclassified instances and decreased for correctly classified examples [5]. Boosting can transform a weak learning algorithm, which

performs just slightly better than random guessing, into a strong learning algorithm with a good generalization performance [5]. Because the way a classifier is built in boosting depends on the performance of the previously built ones, Breiman refers to this type of adaptative algorithms *arcing* (adaptively resample and combine) [28].

Boosting starts the training process with uniform weights, $\mathbf{w}^{(0)}$ on $\mathcal{Z}_{train}$ and follows this loop: Construct a classifier h_k using the weights $\mathbf{w}^{(k)}$. Compute the weighted error of h_k on $\mathcal{Z}_{train}$, ϵ_k, using the set of weights $\mathbf{w}^{(k)}$. If the error $\epsilon_k = 0$ the algorithm stops and assigns the longest possible weight to that particular classifier. If the error $\epsilon_k \geq 0.5$ the algorithm stops. Otherwise, new weights, $\mathbf{w}^{(k+1)}$ are computed so that h_k has an error equal to 0.5 on the training dataset using the weights $\mathbf{w}^{(k+1)}$. To classify an instance, boosting uses weighted majority vote, where the weight of each classifier is $log(\frac{1-\epsilon_k}{\epsilon_k})$. The process is described in detail in Algorithm 2.

The previous description corresponds to boosting with reweighting. Boosting with reweighting is used when the algorithm that builds the base classifiers can handle instances with different weights. Otherwise, an alternative version of boosting using resampling can be used. Boosting with resampling generates a bootstrap sample S_k using the weights $\mathbf{w}^{(k)}$. This sample is used to train the classifier h_k. In this version, if the error ϵ_k is equal to 0 or greater than 0.5, the algorithm is not stopped. Instead, the weights $\mathbf{w}^{(k+1)}$ are reset to uniform values and the process repeated. While boosting with reweighting is a completely deterministic process, boosting with resampling is a randomized algorithm because of the stochastic nature of the generation of bootstrap samples.

Boosting is one of the most successful ensemble methods [3, 26, 35] and outperforms bagging in many benchmark classification problems. However, the performance of boosting deteriorates when there is noise the class labels [18, 26, 35]. This is a consequence of the adaptative weight modification: in a few iterations the algorithm is focused on the more difficult examples of the training dataset, which can be outliers or erroneously labeled instances. This behaviour often leads to overfitting [3, 18, 26, 35].

In the experiments in Section 5, we use a modified version of boosting with reweighting. The goal is to generate ensembles of a fixed size. Boosting with reweighting algorithm stops when the error ϵ_k is 0 or greater than 0.5. Therefore, it is possible that the target ensemble size is not reached [6, 16]. The version of boosting used in this work does not stop generating classifiers when this condition is met. Instead, the weights are reset to uniform values and a bootstrap sample is generated so that more classifiers can be built. The algorithm finally halts when the ensemble has the specified size.

Boosting generally reduces both the bias and the variance terms in the

Algorithm 2 The Adaboost.M1 algorithm

Training

Input: T, the number of classifiers in the ensemble.
Output: The ensemble E_T, and the classifier weights $\{\beta_i; i = 1, \ldots, T\}$

- $E_0 = \emptyset$.
- Initialize the weights to uniform values: $\mathbf{w}^{(1)} = \frac{1}{N}$
- for $k = 1, \ldots, T$
  - Build a classifier h_k using the weights $\mathbf{w}^{(k)}$ as the training dataset.
  - Calculate the weighted ensemble error at step k:

$$\epsilon_k = \sum_{j=1}^{N} w_j^{(k)} l_k^j \tag{3}$$

    ($l_k^j = 1$ if $h_k(\mathbf{x}_j) \neq y_j$ and $l_k^j = 0$ if $h_k(\mathbf{x}_j) = y_j$)
  - If $\epsilon_k = 0$ or $\epsilon_k >= 0.5$, ignore h_k and halt.
  - Else, calculate:

$$\beta_k = \frac{\epsilon_k}{1 - \epsilon_k} \qquad 0 < \epsilon_k < 0.5 \Rightarrow 0 < \beta_k < 1 \tag{4}$$

  - Update the weights of the examples:

$$w_j^{(k+1)} = \frac{w_j^{(k)} \beta_k^{(1-l_k^j)}}{\sum_{i=1}^{N} w_i^{(k)} \beta_k^{1-l_k^j}} \qquad j = 1, \ldots, N. \tag{5}$$

  - Incorporate the classifier into the ensemble, $E_k = E_{k-1} \cup h_k$
- Return the ensemble E_T and the weights $\beta_i;\ i = 1, \ldots, T\}$

Classification

- Classify the input instance $\mathbf{x}$ using $\{h_i;\ i = 1, \ldots, T\}$.
- Calculate the support for class y_t:

$$\mu_t(\mathbf{x}) = \sum_{h_k(\mathbf{x}) = y_t} \ln\left(\frac{1}{\beta_k}\right) \tag{6}$$

- Assign the class with the maximum support, $c_t = \underset{c_t}{\operatorname{argmax}}\ \mu_t(\mathbf{x})$

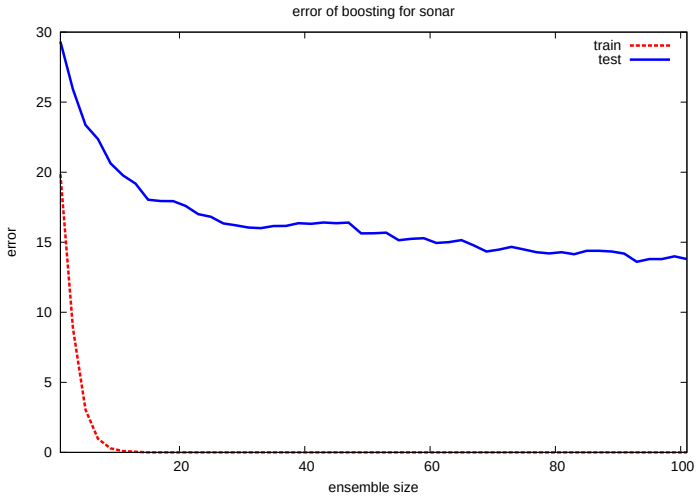

Figure 4: Dependence of the training error of boosting with the size of the ensemble for the sonar data set. Boosting drives the error on the training set to zero in the first few iterations.

classification error. Bias is reduced by the modification of weights, which is a mechanism to focus on difficult instances. Variance is reduced by the combination of the outputs of the base classifiers, as in bagging [26].

Updating rules for the weights In the bibliography, different but equivalent rules to update the weights of the training instances are found. In Algorithm 2 we are not using the original rule presented in [36] but a generalized version of it. The original description of the algorithm uses the following rule, which would replace the rules (4) and (5) in Algorithm 2:

- Compute

$$\alpha_k = \frac{1}{2}\ln\left(\frac{1-\epsilon_k}{\epsilon_k}\right) \qquad 0 < \alpha_k < 1 \tag{7}$$

- Update the weights of the examples:

$$\mathbf{w}_j^{(k+1)} = \frac{\mathbf{w}_j^{(k)} e^{-\alpha_k y_j h_k(\mathbf{x}_j)}}{\sum_{i=1}^{N} \mathbf{w}_i^{(k)} e^{-\alpha_k y_i h_k(\mathbf{x}_i)}} \qquad j = 1, \ldots, N. \tag{8}$$

where the class labels take values in $\{-1, +1\}$. That is, y_j has value -1 if instance j belongs to the first class and $+1$ if it belongs to the second class. $h_k(\mathbf{x}_j)$ is -1 is the classifier k predicts the first class to the instance $\mathbf{x}_j$, and $+1$ if predicts the second class. This rule is valid only for problems with two

classes. It can be generalized to an arbitrary number of classes using the rule in Algorithm 2. Both rules are equivalent in binary classification problems. The relationship between the weight modification in the instance correctly classified and the weight modification in the instance incorrectly classified is the same for both methods:

Proof.

$$\begin{aligned} \frac{\beta_k}{1} &= \frac{e^{-\alpha_k}}{e^{\alpha_k}} \\ e^{2\alpha_k} &= \frac{1}{\beta_k} \\ 2\alpha_k &= \log \frac{1}{\beta_k} \end{aligned} \qquad \Bigg| \qquad \begin{aligned} \log \frac{1-\epsilon_k}{\epsilon_k} &= \log \frac{1}{\beta_k} \\ \beta_k &= \frac{\epsilon_k}{1-\epsilon_k} \end{aligned}$$

□

As mentioned in the description of the boosting algorithm, the modification of the weights after each iteration is such that the weighted error of the classifier h_k is 0.5 using the weights $\mathbf{w}^{(k+1)}$. The proof is as follows:

Proof. If the examples are sorted so that the examples that classifier k misclassified, N_{inc}, appear first, then the error ϵ_k can be rewritten as:

$$\epsilon_k = \sum_{j=1}^{N_{inc}} w_j^{(k)} = 1 - \sum_{j=N_{inc}+1}^{N} w_j^{(k)} \tag{9}$$

The weights at step $k+1$, $\mathbf{w}^{(k+1)}$, can be rewritten in terms on the weights of the previous step, $\mathbf{w}^{(k)}$, as follows:

$$w_j^{(k+1)} = \begin{cases} K w_j^{(k)} & \text{if } j \leq N_{inc} \\ K w_j^{(k)} \frac{\epsilon_k}{1-\epsilon_k} & \text{if } j > N_{inc} \end{cases} \tag{10}$$

where K is a normalization constant so that the weights add up to 1. Using the weight update rule (9) and the definition of ϵ_k in terms of partial

sums of weights (10):

$$
\begin{aligned}
1 = \sum_{j=1}^{N} w_j^{(k+1)} &= \sum_{j=1}^{N_{inc}} w_j^{(k+1)} + \sum_{j=N_{inc}+1}^{N} w_j^{(k+1)} \\
&= K\Big(\sum_{j=1}^{N_{inc}} w_j^{(k)} + \sum_{j=N_{inc}+1}^{N} w_j^{(k)} \frac{\epsilon_k}{1-\epsilon_k} \Big) \\
&= K\Big(\epsilon_k + (1-\epsilon_k)\frac{\epsilon_k}{1-\epsilon_k} \Big) = 2K\epsilon_k
\end{aligned}
\tag{11}
$$

From (11), we obtain that the normalization constant K is:

$$
K = \frac{1}{2\epsilon_k} \tag{12}
$$

Finally, we can rewrite the error of the classifier h_k using the weights $\mathbf{w}^{(k+1)}$ and check that its value is equal to 0.5:

$$
\begin{aligned}
\sum_{j=1}^{N_{inc}} w_j^{(k+1)} &= K\Big(\sum_{j=1}^{N_{inc}} w_j^{(k)} \Big) = \frac{1}{2\epsilon_k}\Big(\sum_{j=1}^{N_{inc}} w_j^{(k)} \Big) \\
&= \frac{1}{2\epsilon_k}\epsilon_k = \frac{1}{2}
\end{aligned}
$$

□

Another weight updating rule is obtained from this proof by substituting Equation (12) into Equation (10) [26]. The rule multiplies by $\frac{1}{2\epsilon_k}$ the weights of the instance incorrectly classified and by $\frac{1}{1-2\epsilon_k}$ otherwise. This rule does not need normalization and is equivalent to the one specified earlier.

These modifications in the weights can have problems with underflows, as described in [26]. The weights of instances that are correctly classified in many iterations tend to 0. These weights are decreased in a factor of 2^k in k iterations of the algorithm. A possible solution is to set a minimum value of the weights for all the instances in the dataset [26]. Webb sets this minimum value to 10^{-8} [37].

Weighted majority vote Boosting uses weighted majority voting to combine the outputs of the base classifiers. Since the classifiers generated by boosting have different error rates on the training data, it is reasonable to assign different weights to their votes. Using the same definitions as in majority vote, weighted majority vote assigns the class c_k that maximizes:

$$c_k = \underset{c_j}{\operatorname{argmax}}\ g_j(\mathbf{x}) = \underset{c_j}{\operatorname{argmax}}\ \sum_{i=1}^{T} b_i\, \mathbb{I}(h_i(\mathbf{x}) = c_j) \tag{13}$$

where b_i is the weight assigned to classifier i.

In general, assuming an ensemble of T independent classifiers with probability of correct classification of $p_1, \ldots, p_T$ whose output is combined by weighted majority vote, the optimal weights for combining the decisions of the members of the ensemble are [16]:

$$b_i \propto \log \frac{p_i}{1 - p_i} \tag{14}$$

Proof. Let be $\mathbf{h} = [h_1(\mathbf{x}), \ldots, h_T(\mathbf{x})]$ the vector whose elements are the predicted class labels for each classifier of the ensemble to the instance $\mathbf{x}$. Based on Bayes' rule, the class posterior probability is:

$$P(c_k|\mathbf{h}) = \frac{P(c_k)P(\mathbf{h}|c_k)}{P(\mathbf{h})} \qquad k = 1, \ldots, L \tag{15}$$

Taking the log of (15):

$$g_k(\mathbf{x}) = \log P(c_k|\mathbf{h}) = \log \Big(P(c_k)P(\mathbf{h}|c_k)\Big) - \log \Big(P(\mathbf{h})\Big) \qquad k = 1, \ldots, L \tag{16}$$

Since the last term is equal for all the class labels, it can be ignored in the derivation of the decision rule. Assuming independence among the output of the classifiers:

$$\begin{aligned}
g_k(\mathbf{x}) &= \log \Big(P(c_k) \prod_{i=1}^{T} P(h_i|c_k) \Big) \\
&= \log P(c_k) + \log \prod_{i,h_i=c_k} P(h_i|c_k) + log \prod_{i,h_i \neq c_k} P(h_i|c_k) \\
&= \log P(c_k) + \log \prod_{i,h_i=c_k} p_i + \log \prod_{i,h_i \neq c_k} 1 - p_i \\
&= \log P(c_k) + \log \prod_{i,h_i=c_k} \frac{p_i(1-p_i)}{1-p_i} + \log \prod_{i,h_i \neq c_k} 1 - p_i \\
&= \log P(c_k) + \log \prod_{i,h_i=c_k} \frac{p_i}{1-p_i} + \log \prod_{i=1}^{T} 1 - p_i \\
&= \log P(c_k) + \sum_{i,h_i=c_k} \log \frac{p_i}{1-p_i} + \sum_{i=1}^{T} \log(1-p_i)
\end{aligned}$$

Since $\sum_{i=1}^{T} \log(1-p_i)$ is a constant which does not depend on the predicted class label c_k, we can reduce the equation to:

$$\tilde{g}_k(\mathbf{x}) = \log P(c_k) + \sum_{i, s_i = c_k} \log \frac{p_i}{1-p_i}$$

$$\tilde{g}_k(\mathbf{x}) = \log P(c_k) + \sum_{i=1}^{T} \mathbb{I}(h_i(\mathbf{x}) = c_k) \log \frac{p_i}{1-p_i}$$

□

To minimize the classification errors we would still need to take into account the prior probabilities.

For example, suppose we constructed an ensemble of five classifier with probability of correct classification (0.9 0.9 0.6 0.6 0.6). Using a majority vote scheme, we would obtain a probability of 0.877:

$$P_{maj} = 3 \cdot 0.9^2 \cdot 0.4 \cdot 0.6 + 0.6^3 + 6 \cdot 0.9 \cdot 0.1 \cdot 0.6^2 \cdot 0.4 \approx 0.877$$

while if we used a weighted majority votes with weights $(\frac{1}{3}\ \frac{1}{3}\ \frac{1}{9}\ \frac{1}{9}\ \frac{1}{9})$:

$$P^w_{maj} = 0.9^2 + 2 \cdot 3 \cdot 0.9 \cdot 0.1 \cdot 0.6^2 \cdot 0.4 + 2 \cdot 0.9 \cdot 0.1 \cdot 0.6^2 \approx 0.927$$

The details of how to derive these probabilities are given in Appendix A.

Boosting with shrinkage A modification of boosting that has been shown to improve the generalization performance in regression problems is shrinkage [38, 39, 40]. Shrinkage smooths the weights modification by multiplying the exponent in the weight modification factor in Equation (8) by a small positive constant $0 < \nu \leq 1$:

$$w_j^{(k+1)} = \frac{w_j^{(k)} e^{-\nu \alpha_k y_j h_j(\mathbf{x}_j)}}{\sum_{i=1}^{N} w_i^{(k)} e^{-\nu \alpha_k y_i h_i(\mathbf{x}_i)}} \qquad j = 1, \ldots, N. \tag{17}$$

Shrinkage reduces the speed of learning and can actually improve the accuracy of boosting in regression problems. However, the universal effectiveness of shrinkage has been recently questioned in [40].

Since boosting reduces the training error to 0 in the first few iterations (Figure 4), the remaining classifiers generated may not be complementary to the previous ones. Boosting with shrinkage slows down the learning process so that the training error tends to 0 more slowly than in the original algorithm. We investigate whether shrinkage improves the generalization performance of boosting ensembles in a set of preliminary experiments, which

are describe in Section 5.1.1. In those experiments, a ensemble is built using four variants of boosting: reweighting and resampling (no shrinkage), and reweighting and resampling with shrinkage ($\nu = 0.1$). The generated ensembles are pruned using the methods described in Section 4. The results indicates that the differences in performance among the different versions of boosting investigated are small. Since boosting with reweighting and without shrinkage is more standard in the literature, it will be used in the experiments presented on Section 5.

4 Ensemble pruning

As described in the previous sections, major drawbacks of ensemble learning are the difficulties in interpretation of the classification process, the large memory requirements and the slow speed of training and classification. A possible solution to the memory requirements and the classification speed is to select a subset of the classifiers from the original ensemble [3, 6]. Besides the reduction in complexity, a pruned ensemble of complementary classifiers can actually perform better than the complete ensemble.

A problem with this approach is that the selection of the subset of classifiers that has the best generalization properties on the basis of measure of performance estimated on the training data is a difficult task whose solution is computationally expensive. Even if a reliable estimate of the generalization performance is available, the problem of finding the optimal subensemble is a combinatorial search whose complexity grows exponentially with the size of the original ensemble. Particularly, for an ensemble of T classifiers, the set of non-empty subensembles that need to be evaluated has dimension $2^T - 1$. Furthermore, the solution reached in the search process strongly depends on the measure used to estimate the subensemble performance, e.g. the combined accuracy of the ensemble, the ensemble diversity or the complementarity among ensemble classifiers. Thus, the solution to this problem by exhaustive search is often unfeasible in practice. To overcome the practical difficulty of finding the optimal subensemble by exact methods, approximate optimization techniques can be used, such as genetic algorithms [11], semidefinite programming [7] and ordered aggregation [3, 6].

In summary, to solve the ensemble pruning problem it is important to solve two separate problems. The first one is to identify a quality measure whose estimate on the training set is a good predictor of the generalization performance of the ensembles considered. These measures are discussed in Section 4.1. The second is to design an efficient method to identify subensembles that optimize the selected measure. These methods are described in

Section 4.2.

4.1 Measures for ensemble pruning

This section describes the different measures that have been introduced in previous works to estimate the performance of the pruned subensembles based on the training dataset $\mathcal{Z}_{train}$. As discussed in the introduction, the goal is to identify subsets of complementary classifiers. Most of the measures introduced are based on estimates of accuracy and diversity on the training set. It is important to select an appropriate measure that can be used to predict the generalization performance on the basis of estimations on the ensemble on the training dataset. These measures are commonly defined in terms of accuracy or diversity of the classifiers or of combinations of both aspects.

4.1.1 Measures of accuracy

A simple method to prune ensembles is to choose classifiers that have the best individual performance on the training data. For this purpose, we would compute the error rate of every classifier on the training set $\mathcal{Z}_{train}$ and then select the classifiers with the lowest error rate. In practice, the training error is not a good estimation of the generalization error. Classifiers that are optimal with respect to training error tend to overfit to the training data, and often perform worse on instances that are not present in the training dataset [1]. That is, the error computed in an independent test dataset is usually higher than the training error. To solve this problem, we can split our dataset into a training dataset and a validation dataset, $\mathcal{Z}_{val}$, and then estimate the generalization error on this independent validation dataset [1, 15, 41].

Even if accurate estimates of the generalization error are available, selecting only the best classifiers does not allow the identification of subensembles with better generalization performance [3]. The classifiers selected on the basis on individual properties, such as the error on the training dataset, tend to be similar. In consequence, the performance of the ensemble is generally not improved. Figure 5 shows an example of an ensemble composed of six classifiers and two possible pruned subensembles of three classifiers. Rows correspond to instances and columns to classifiers. Cell (i, j) is shadowed if the classifier h_j classifies instance $\mathbf{x}_i$ correctly, and is blank otherwise. The last column represents the classification obtained by majority vote by each ensemble. The complete ensemble (left) has an accuracy of 0.8. The first subensemble (center) includes the most accurate classifiers from the ensemble and has the same accuracy as the complete ensemble. The second subensem-

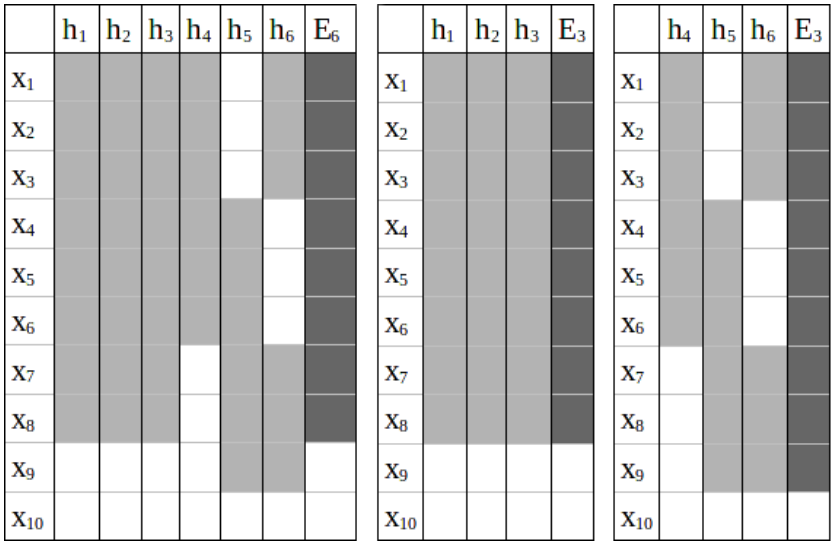

Figure 5: Example of an ensemble where pruning by selection of the most accurate classifiers leads to suboptimal results. The complete ensemble (left) is composed by six classifiers. The first subensemble (center) is composed by the three most accurate classifiers. The second subensemble (right) is composed by the remaining classifiers. The first subensemble does not improve the success rate of the original ensemble while the second subensemble does.

ble (right) is composed by the least accurate classifiers but they are more diverse. Its prediction accuracy improves to 0.9. Despite the fact that the first subensemble contains the best individual classifiers, their predictions coincide and therefore, their errors correspond to the same instances. On the other hand, the classifiers of the second subensemble give incorrect predictions for different instances, so that their errors are compensated. Therefore, accuracy of individual classifiers is by itself not sufficient to obtain good pruned subensembles. In the next section we introduce measures that take into account the complementary among classifiers as well.

4.1.2 Measures of diversity

Enhancing the diversity of the classifiers has also been proposed as a way to build ensembles with good generalization properties [2]. There are many different measures of diversity. We describe in this section some of the most relevant.

A simple measure of the diversity of the ensemble is the pairwise correlation between classifiers [16]. Correlation between the outputs of two classifiers h_1 and h_2 is defined as follows: Let $z_j^{(k)} = 1$ when classifier h_k

correctly classifies the instance $\mathbf{x}_j$ and $z_j^{(k)} = 0$ otherwise

$$z_j^{(k)} = \mathbb{I}(h_k(\mathbf{x}_j) = y_j)$$

Then, Spearman's correlation between $z^{(1)}$ and $z^{(2)}$ is computed:

$$\rho_{1,2} = \text{correlation}(z^{(1)}, z^{(2)}) = \frac{\text{covariance}(z^{(1)}, z^{(2)})}{\sigma_1 \sigma_2}$$

$$\text{covariance}(z^{(1)}, z^{(2)}) = \frac{1}{N}\sum_{j=1}^{N}(z_j^{(1)} - \mathbb{E}[z^{(1)}])(z_j^{(2)} - \mathbb{E}[z^{(2)}])$$

$$\sigma_k^2 = \text{variance}(z^{(k)}) = \frac{1}{N}\sum_{j=1}^{N}(z_j^{(k)} - \mathbb{E}[z^{(k)}])^2$$

$$\mathbb{E}[z^{(k)}] = \frac{1}{N}\sum_{j=1}^{N} z_j^{(k)}$$

The maximum correlation, $\rho_{1,2} = 1$, occurs when both classifiers agree in all training examples. The minimum correlation, $\rho_{1,2} = -1$, corresponds to the classifiers disagreeing in every example.

The κ statistic A relevant measure of diversity, which is used in several studies about ensemble pruning, is the κ statistic [6]. The κ statistic measures the level of agreement between two classifiers normalized to account for the degree of agreement expected by chance. In terms of the coincidence matrix $\mathbf{C}$, whose element C_{ij} counts the number of examples that the first classifier assigns to class i and the number of examples that the second classifier assigns to class j. The κ statistic is:

$$\kappa_{ij} = \frac{\theta_1 - \theta_2}{1 - \theta_2} \tag{18}$$

$$\theta_1 = \sum_{i=1}^{L}\frac{C_{ij}}{N} \qquad \theta_2 = \sum_{i=1}^{L}(\sum_{j=1}^{L}\frac{C_{ij}}{N}\sum_{k=1}^{L}\frac{C_{ki}}{N}) \tag{19}$$

where N is the number of instance in the dataset. θ_1 is a general measure of agreement, θ_2 is the probability that the two classifiers agree by chance. The value of κ is 0 when the agreement of the two classifiers is equal to that expected by chance (highest diversity). The value of κ is 1 when the predictions of the two classifiers agree on every example (lowest diversity). Kappa-pruning is based on this measure (Section 4.2.1). In Kappa-pruning, κ is computed for every pair of classifiers of the ensemble. Then, pairs of

classifiers are selected, starting with the pair that has the lowest k, and incorporated into the subensemble in order of increasing κ [6].

There are other measures of diversity, such as the Yule's statistic [42], the entropy measure, the generalized diversity or the coincident failure diversity [16]. Unfortunately, there is not a direct relationship between these diversity measures and the generalization error, as shown in [16, 33]. That is, obtaining the optimum of these measures often does not lead to the optimum generalization performance.

4.1.3 Measures that combine accuracy and diversity

Using measures of accuracy or diversity of the individual classifiers by themselves is not sufficient to select subensembles with improved generalization performance. To identify subsets of complementary classifiers, one needs to use combined measures that take into account both aspects.

Reduce-Error pruning (Section 4.2.1) uses estimates of the subensemble error to guide the pruning process. Specifically, it selects the subensemble that minimizes the error on a selection dataset [6], $\mathcal{Z}_{sel}$, which usually coincides with the training dataset, $\mathcal{Z}_{train}$.

As discussed in the previous sections, using accurate classifiers but with a large degree of similarity does not improve the accuracy of the ensemble. Therefore, directly quantifying the complementarity of the base classifiers is important. A measure of complementary between two classifiers h_i and h_j can be defined as the accuracy of h_j in the set of the examples that are incorrectly classifier by h_i [43]

$$\sum_{(\mathbf{x},y)\in\mathcal{Z}_{train}} \mathbb{I}(y = h_j(\mathbf{x}) \text{ and } h_i(\mathbf{x}) \neq y) \tag{20}$$

This measure is used in *Complementary pruning* which is described in detail in Section 4.2.1.

The $\tilde{g}$ accuracy-diversity measure: This measure takes into account the accuracy of the individual classifiers and the diversity of the ensemble [7].

Consider an ensemble composed of T classifiers and a training dataset $\mathcal{Z}_{train}$ of labeled instances. Define a matrix $\mathbf{G}$, whose element G_{ij} is the number of common errors between classifier i and classifier j, where $i, j = 1, 2, \ldots, T$. The value of the diagonal term G_{ii} is the number of errors made by classifier i. The matrix is normalized so that its elements are in the same scale (values between 0 and 1).

$$\tilde{G}_{ii} = \frac{G_{ii}}{N}, \quad \tilde{G}_{ij,i\neq j} = \frac{1}{2}\left(\frac{G_{ij}}{G_{ii}} + \frac{G_{ji}}{G_{jj}}\right) \tag{21}$$

The sum of the diagonal terms, $\sum_i \tilde{G}_{ii}$, measures the average error rate of the individual classifiers in the dataset $\mathcal{Z}_{train}$. The sum of the out-of-diagonal terms, $\sum_{ij,i\neq j} \tilde{G}_{ij}$, measures the diversity of the ensemble. From the matrix $\mathbf{G}$, we define

$$\tilde{g} = \sum_{ij} \tilde{G}_{ij} \tag{22}$$

which is a quality measure that combines measures of accuracy and diversity of the ensemble. Note that even when the aggregated prediction of the ensemble has zero error on the training set, the value of $\tilde{g}$ need not be equal to zero.

This measure is used in [7] to rewrite the ensemble pruning problem as a quadratic integer programming problem. In general cases, this problem cannot be solved exactly in polynomial time. However, an exact solution of a relaxed version of the problem can be found using semidefinite programming (SDP). This method is described in Section 4.2.2.

The methods we propose to prune boosting ensembles in this work are also based on this measure. Boosting drives the training error of the subensemble to 0 quickly, so that the pruning methods based on the training error of the subensemble may be unable to distinguish between different subensembles. The methods based on the $\mathbf{G}$ matrix should not suffer from this problem, because the measure $\tilde{g}$ can be different to 0 even when the training error of the subensemble is 0.

4.2 Ensemble pruning methods

In this section we review some of the pruning methods introduced in the literature. Most of the ensemble pruning techniques described employ either the combined subensemble error estimated on the training set or some of the measures of diversity described in the previous section to discriminate among subsets of classifiers [3, 6, 7, 11, 43, 44]. This procedure is not effective when the classifiers have been generated by boosting, because the training error of the ensemble is typically driven to zero very quickly by this algorithm [3, 5]. Therefore, measures based on the training error are unable to distinguish between different subensembles.

To avoid using the ensemble training error to direct the pruning process, we introduce pruning methods that use the measure of accuracy-diversity $\tilde{g}$ described in Section 4.1.3. Since $\tilde{g}$ takes into account both the errors of

the individual classifiers and the correlation between the errors of pairs of classifiers, its value on the training dataset for subensembles generated by boosting need not be 0, even when the subensembles have error 0 [7]. Using this measure, the subensemble selection problem of size k can be formulated as a quadratic integer programming problem

$$\underset{\xi}{\operatorname{argmin}}\ \xi^{\mathbf{T}} \cdot \tilde{\mathbf{G}} \cdot \xi, \quad s.t. \sum_{i=1}^{T} \xi_i = k, \quad \xi_i \in \{0, 1\} \tag{23}$$

The binary variable ξ_i indicates whether classifier i is included in the pruned subensemble ($\xi_i = 1$) or not ($\xi_i = 0$). The size of the pruned ensemble, k, is specified beforehand. This is a standard binary optimization problem, which is NP-hard. In [7], the solution is approximated in polynomial time by applying semi-definite programming (SDP) to a convex relaxation of the original problem.

The solution of Equation (23) can be approximated applying general optimization techniques. Specifically, we propose using a greedy method based on ordered aggregation, genetic algorithms and simulated annealing.

4.2.1 Pruning by early stopping in ordered aggregation

This technique is based on modifying the order in which classifiers are aggregated in the ensemble [3, 43] and stopping when the number of classifiers in the subensemble is sufficiently large. Bagging generates independent classifiers. The order of generation and aggregation of these classifiers is left unspecified by the algorithm. In boosting the order is prescribed: it is a sequential algorithm in which the weights of the instances used to train a classifier are determined in terms of the errors of the previous classifier on those instances.

In ordered aggregation, the order of the ensemble is modified so that the classifiers that are expected to perform better when combined are aggregated first. Starting with an optimal subensemble of size $u - 1$, a near-optimal subensemble of size u is obtained by incorporating the classifier from the original initial pool of classifiers that is expected to improve the performance of the subensemble the most. Note that this is a greedy procedure because it uses only local information. In consequence, the solution that is reached can be a local optimum instead of the global one.

When the order of aggregation is left unspecified by the algorithm (bagging, random forest, ...), the error of ensemble usually exhibits a monotonic decrease as a function of the number of elements in the ensemble (upper left curve in Figure 6). In ordered bagging ensembles, these error curves typically exhibit a minimum at intermediate ensemble sizes (lower left curve in Figure

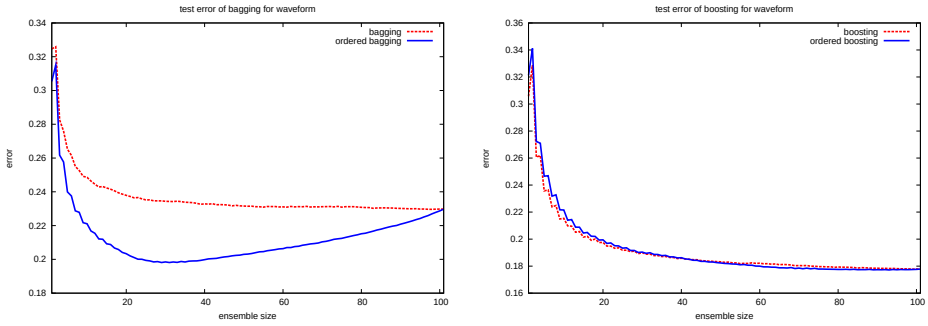

Figure 6: Test error curves of bagging (left) and boosting (right) as a function of the ensemble size (upper curve), and error curves for the ordered ensembles.

6), which is below the error of the complete ensemble. Depending on the desired amount of pruning, the first k classifiers of the ordered ensemble are selected. k should correspond to the minimum the test dataset. As the minimum can occur at different sizes in the train and test datasets, determining its precise location is a difficult problem. Nevertheless, the error curve tends to have a rather broad minimum, which means that subensembles with a large range of sizes have better performance than the complete ensemble. An independent dataset could be used to estimate the location of the minimum, if sufficient amount of data are available for training. If training data are scarce, a heuristic that performs well in bagging ensembles is to selected the first $20-40\%$ of the classifiers of the ordered ensemble.

While ordered aggregation is shown to perform well in bagging [3], it is not expected to work as well in boosting, where the order of the classifiers is specified by the algorithm. Figure 6 (right) shows the error curve of a boosting ensemble and of an ordered boosting ensemble. The curve of the ordered boosting does not exhibit the same behaviour as the ordered bagging. The minimum error corresponds to larger subensembles and sometimes is not present. In those cases, the lowest error corresponds to the complete ensemble.

We now review some of the rules used to guide the ordering procedure. Following definitions are used: $\mathcal{Z}_{train}$ is the training set. $\mathcal{Z}_{sel}$ is the selection set that is used by the pruning algorithm. It can coincide with $\mathcal{Z}_{train}$. $\mathbb{I}(\cdot)$ is an indication function ($\mathbb{I}(\text{true}) = 1$ and $\mathbb{I}(\text{false}) = 0$). S_u denotes a subensemble of size u and s_u the classifier incorporated by S_u at step u. $H_S(\mathbf{x})$ denotes the prediction given by the subensemble S. E_T is the original ensemble.

Reduce-Error pruning: The first classifier in the ensemble is the one with the lowest classification error estimated in the selection dataset $\mathcal{Z}_{sel}$. The remaining classifiers are incorporated sequentially so that the error of subensemble estimated in $\mathcal{Z}_{sel}$ is minimum [3, 6].

$$s_u = \underset{k}{\operatorname{argmin}} \sum_{(\mathbf{x},y)\in\mathcal{Z}_{sel}} \mathbb{I}(H_{S_{u-1}\cup h_k}(\mathbf{x}) \neq y), \qquad k \in E_T, k \notin S_{u-1} \tag{24}$$

Kappa pruning: This method attempts to select the most diverse subensemble using the κ statistic described in Section 4.1.2.

Kappa pruning introduced originally in [6] computes the κ statistic for every pair of classifiers. The ensemble is built by progressively incorporating pairs of classifiers with the lowest values of κ (highest diversity). Pruned ensembles identified by this method generally perform worse than the complete ensemble, because the κ statistic measures only the pairwise diversity of the classifiers and not the diversity of the unselected classifiers with respect to the selected subensemble, which is the relevant quantity. An improved version of kappa pruning computes the κ statistics for the subensemble and the unselected classifiers, and then incorporates the classifier with the lowest value of this measure [3]

$$s_u = \underset{k}{\operatorname{argmin}}\ \kappa_{\mathcal{Z}_{sel}}(h_k, H_{S_{u-1}}), \qquad k \in E_T, k \notin S_{u-1} \tag{25}$$

Complementarity measure: The first classifier selected is the one with lowest error on $\mathcal{Z}_{sel}$. The algorithm then iteratively incorporates the classifier whose output is the most complementary to that of the selected subensemble, according to the measure of complementarity (Equation (20)) [3]:

$$s_u = \underset{k}{\operatorname{argmax}} \sum_{(\mathbf{x},y)\in\mathcal{Z}_{sel}} \mathbb{I}(h_k(\mathbf{x}) = y \text{ and } H_{S_{u-1}}(\mathbf{x}) \neq y), \qquad k \in E_T, k \notin S_{u-1} \tag{26}$$

Margin Distance Minimization: This method uses a signature vector $\mathbf{c}^{(t)}$ for each classifier. For the t-th classifier, $\mathbf{c}^{(t)}$ is a N_{sel}-dimensional vector where N_{sel} is the number of instances in the selection set, $\mathcal{Z}_{sel}$. The i-th component of the vector is 1 if the classifier t correctly classifies the i example in $\mathcal{Z}_{sel}$ and -1 otherwise [3].

$$c_i^{(t)} = 2\mathbb{I}(h_t(\mathbf{x}_i) = y_i) - 1, \qquad (\mathbf{x}_i, y_i) \in \mathcal{Z}_{sel} \tag{27}$$

The signature vector for an ensemble of T classifiers is defined as

$$\langle \mathbf{c} \rangle = \frac{1}{T} \sum_{t=1}^{T} \mathbf{c}^{(t)} \tag{28}$$

In a binary classifier problem, the i-th component of $\langle \mathbf{c} \rangle$ is the margin of the i-th example in $\mathcal{Z}_{sel}$, normalized in $[-1, 1]$. Therefore, example i is correctly classified by the ensemble if $\langle \mathbf{c}_i \rangle > 0$. A subensemble whose signature vector $\langle \mathbf{c} \rangle$ is in the first quadrant of the N_{sel}-dimensional hyperspace, correctly classifies all the examples in $\mathcal{Z}_{sel}$. The objective of this method is to select a subensemble whose signature vector is as close as possible to a reference vector, $\mathbf{o}$, placed in the first quadrant:

$$o_i = p \qquad \text{with } i = 1, \ldots, N_{sel} \text{ and } 0 < p < 1 \tag{29}$$

The algorithm incorporates into the ensemble the classifiers that minimizes the distance between the reference vector and the signature vector of the ensemble.

$$s_u = \underset{k}{\operatorname{argmin}} \quad d\left(\mathbf{o}, \frac{1}{T}\left(\mathbf{c}^{(k)} + \sum_{t=1}^{u-1} \mathbf{c}^{(t)}\right)\right), \qquad k \in E_T, k \notin S_{u-1} \tag{30}$$

where $d(\mathbf{v}_1, \mathbf{v}_2)$ is the Euclidean distance between points $\mathbf{v}_1$ and $\mathbf{v}_2$.

The value of p should be small (for example, $p \sim 0.075$) so that the examples that are correctly classified by most of the base classifiers achieve a value close to p in the first iterations of the algorithm. Since these examples are already close to p, the algorithm focuses on examples that are more difficult to classify in the following iterations. If a value of p close to 1 were used, then most examples would achieve a value close to p only in the last iterations, so that the algorithm would not distinguish between easy and difficult examples.

Boosting-based Ordering: This method iteratively incorporates the classifier that minimizes the weighted error in $\mathcal{Z}_{sel}$. The weights used to compute the error are the same as in Adaboost.M1 (Algorithm 2, Equation (3)) [3]. The algorithm is similar to boosting, with the difference that the classifiers are not built at each iteration but are selected from the initial pool of classifiers generated by bagging. The final decision of the ensemble is performed by majority vote.

Minimization of $\tilde{g}$: The previous rules are not expected to perform well in boosting ensembles because the generated subensembles quickly reach errors close to zero. Instead, we propose approximating the solution of Equation (23) by a greedy search. This greedy search results in an ordered aggregation process: Starting with an empty subensemble, the method iteratively incorporates the classifier that minimizes the objective function of Equation (23). The Algorithm 3 presents the pseudocode of this greedy search.

The pruning techniques described, except for the minimization of $\tilde{g}$, were analyzed in [3] using bagging ensembles of 101 CART trees [45]. Pruned subensembles of 20 trees outperform the complete bagging ensemble irrespective of the pruning technique. The errors rates of the pruned subensembles exhibit a dependence on ensemble size similar to the error curves in Figure 6. The best overall results correspond to margin distance minimization, followed by orientation ordering and boosting-based pruning. Kappa pruning, which is based only on diversity, has the poorest overall performance on the datasets investigated. These methods were also compared to boosting. In problems with low levels of noise, boosting often outperforms bagging and also pruned bagging ensembles. This indicates that pruning boosting ensembles in these problems is a harder task than pruning bagging ensembles. By contrast, the performance of boosting in problems with noise in the class labels is worse than bagging and pruned bagging [3].

4.2.2 SDP pruning

In SDP pruning [7] approximate solutions to Equation (23) are obtained by transforming the original problem into the *max cut with size k problem* (MC-K) [7]. MC-k searches for partition of a weighted-graph in two sets, one of size k, that maximizes the sum of the weights of edges crossing the partition. MC-k is known to have a good approximate solution algorithm based on semi-definite programming (SDP) [8]. MC-k can be formulated as

$$\underset{\gamma}{\operatorname{argmin}}\ \gamma^{\mathbf{T}} \cdot \mathbf{W} \cdot \gamma \qquad \text{s. t.} \qquad \begin{array}{l} \sum_{i=1}^{T} \gamma_i = N_v - 2k \\ \gamma_i \in \{-1, 1\} \end{array} \tag{31}$$

where N_v is the number of vertices, the elements W_{ij} are the weights between i and j vertices ($W_{ii} = 0$). γ_z has value -1 when vertex z is assigned to the first subset, and +1 when it is assigned to the second one.

SDP pruning applies the approximate algorithm used to obtain solutions of MC-k to the ensemble pruning problem. To apply that method, the Equation (23) is rewritten into a formulation similar to MC-k (Equation (31)).

Algorithm 3 Greedy search for ensemble pruning

Input: The $\tilde{\mathbf{G}}$ matrix.
Output: An ordered vector of the classifier indexes, $\mathbf{s}$.

- $\mathbf{s} \leftarrow$ *empty vector*
- for (u from 1 to T)
  - $minimum \leftarrow \infty$
  - for ($k \in \{1, \ldots, T\}, k \notin \{s_1, \ldots, s_{u-1}\}$)
    - $value \leftarrow \frac{\sum_{i=1}^{u-1}\sum_{j=1}^{u-1}\tilde{G}_{s_i,s_j}+2\sum_{i=1}^{u-1}\tilde{G}_{s_i,k}+\tilde{G}_{kk}}{u^2}$
    - if ($value < minimum$) {
      - $s_u \leftarrow k$
      - $minimum \leftarrow value$
    - }
- return $\mathbf{s}$

Introducing the variables

$$\xi_i = \frac{v_i + 1}{2} \qquad v_i \in \{-1, 1\} \quad i = 1, 2, \ldots, T \tag{32}$$

the pruning problem defined in Equation (23) becomes

$$\underset{\mathbf{v}}{\operatorname{argmin}} \frac{1}{4}(\mathbf{v}+\mathbf{e})^T\tilde{\mathbf{G}}(\mathbf{v}+\mathbf{e}), \qquad \text{s.t.} \qquad (\mathbf{v}+\mathbf{e})^T\mathbf{I}(\mathbf{v}+\mathbf{e}) = 4k \tag{33}$$

where $\mathbf{e}$ is a vector of T ones, and $\mathbf{I}$ the $(T \times T)$ identity matrix.
Using the definitions

$$\tilde{\mathbf{v}}^T = \begin{pmatrix} 1 & \mathbf{v}^T \end{pmatrix} \tag{34}$$

$$\mathbf{H} = \begin{pmatrix} \mathbf{e}^T\tilde{\mathbf{G}}\mathbf{e} & \mathbf{e}^T\tilde{\mathbf{G}} \\ \tilde{\mathbf{G}}\mathbf{e} & \tilde{\mathbf{G}} \end{pmatrix} \tag{35}$$

$$\mathbf{D} = \begin{pmatrix} T & \mathbf{e}^T \\ \mathbf{e} & \mathbf{I} \end{pmatrix} \tag{36}$$

The target function in Equation (33) can be rewritten as

$$\tilde{\mathbf{v}}^T\mathbf{H}\tilde{\mathbf{v}} = \begin{pmatrix} 1 & \mathbf{v}^T \end{pmatrix} \begin{pmatrix} \mathbf{e}^T\tilde{\mathbf{G}}\mathbf{e} & \mathbf{e}^T\tilde{\mathbf{G}} \\ \tilde{\mathbf{G}}\mathbf{e} & \tilde{\mathbf{G}} \end{pmatrix} \begin{pmatrix} 1 \\ \mathbf{v} \end{pmatrix} \tag{37}$$

The constraint in Equation (33) becomes

$$\tilde{\mathbf{v}}^T \mathbf{D} \tilde{\mathbf{v}} = \begin{pmatrix} 1 & \mathbf{v}^T \end{pmatrix} \begin{pmatrix} T & \mathbf{e}^T \\ \mathbf{e} & \mathbf{I} \end{pmatrix} \begin{pmatrix} 1 \\ \mathbf{v} \end{pmatrix} \tag{38}$$

Using this simplified notation, the problem is

$$\underset{\tilde{\mathbf{v}}}{\operatorname{argmin}} \, \tilde{\mathbf{v}}^T \cdot \tilde{\mathbf{H}} \cdot \tilde{\mathbf{v}} \quad \text{s. t.} \quad \begin{array}{l} \tilde{\mathbf{v}}^T \mathbf{D} \tilde{\mathbf{v}} = 4k \\ \tilde{v}_0 = 1 \\ \tilde{v}_i \in \{-1, 1\} \; \forall i \neq 0 \end{array} \tag{39}$$

That is equivalent to the original problem:

Proof.

$$\begin{pmatrix} 1 & \mathbf{v}^T \end{pmatrix} \begin{pmatrix} \mathbf{e}^T \tilde{\mathbf{G}} \mathbf{e} & \mathbf{e}^T \tilde{\mathbf{G}} \\ \tilde{\mathbf{G}} \mathbf{e} & \tilde{\mathbf{G}} \end{pmatrix} \begin{pmatrix} 1 \\ \mathbf{v} \end{pmatrix} = \begin{pmatrix} \mathbf{e}^T \tilde{\mathbf{G}} \mathbf{e} + \mathbf{v}^T \tilde{\mathbf{G}} \mathbf{e} & \mathbf{e}^T \tilde{\mathbf{G}} + \mathbf{v}^T \tilde{\mathbf{G}} \end{pmatrix} \begin{pmatrix} 1 \\ \mathbf{v} \end{pmatrix} =$$
$$\mathbf{e}^T \tilde{\mathbf{G}} \mathbf{e} + \mathbf{v}^T \tilde{\mathbf{G}} \mathbf{e} + \mathbf{e}^T \tilde{\mathbf{G}} \mathbf{v} + \mathbf{v}^T \tilde{\mathbf{G}} \mathbf{v} = (\mathbf{v} + \mathbf{e})^T \tilde{\mathbf{G}} (\mathbf{v} + \mathbf{e}) \tag{40}$$

$$\begin{pmatrix} 1 & \mathbf{v}^T \end{pmatrix} \begin{pmatrix} T & \mathbf{e}^T \\ \mathbf{e} & \mathbf{I} \end{pmatrix} \begin{pmatrix} 1 \\ \mathbf{v} \end{pmatrix} = \begin{pmatrix} T + \mathbf{v}^T \mathbf{e} & \mathbf{e}^T + \mathbf{v}^T \mathbf{I} \end{pmatrix} \begin{pmatrix} 1 \\ \mathbf{v} \end{pmatrix} =$$
$$T + \mathbf{v}^T \mathbf{e} + \mathbf{e}^T \mathbf{v} + \mathbf{v}^T \mathbf{I} \mathbf{v} = (\mathbf{v} + \mathbf{e})^T \mathbf{I} (\mathbf{v} + \mathbf{e}) \tag{41}$$

□

The constraint $\tilde{v}_0 = 1$ can be relaxed to $\tilde{v}_0 \in \{-1, 1\}$ without loss of generality, because $-\tilde{\mathbf{v}}$ fulfills the remaining constraints if and only if $\tilde{\mathbf{v}}$ does:

$$\underset{\tilde{\mathbf{v}}}{\operatorname{argmin}} \, \mathbf{H} : \tilde{\mathbf{v}} \tilde{\mathbf{v}}^{\mathbf{T}} \qquad \text{s. t.} \qquad \begin{array}{l} \mathbf{D} : \tilde{\mathbf{v}} \tilde{\mathbf{v}}^T = 4k \\ \operatorname{diag}(\tilde{\mathbf{v}} \tilde{\mathbf{v}}^T) = e \end{array} \tag{42}$$

where $\mathbf{A} : \mathbf{B} \equiv \sum_{ij} \mathbf{A}_{ij} \mathbf{B}_{ij}$.

Consider the matrix $\tilde{\mathbf{V}} = \tilde{\mathbf{v}} \tilde{\mathbf{v}}^T$. $\tilde{\mathbf{V}}$ is a semidefinite positive matrix ($\tilde{\mathbf{V}} \succeq 0$) with ones in the diagonal and $\operatorname{rank}(\tilde{\mathbf{V}}) = 1$. The rank of this matrix is 1 because it is defined as the product of a vector, so its columns are linearly dependent. In terms of $\tilde{\mathbf{V}}$, the problem is

$$\underset{\tilde{\mathbf{v}}}{\operatorname{argmin}} \; \mathbf{H} : \tilde{\mathbf{V}} \qquad \text{s. t.} \qquad \begin{array}{l} \mathbf{D} : \tilde{\mathbf{V}} = 4k \\ \operatorname{diag}(\tilde{\mathbf{V}}) = \mathbf{e} \\ \tilde{\mathbf{V}} \succeq 0 \\ \operatorname{rank}(\tilde{\mathbf{V}}) = 1 \end{array} \tag{43}$$

Finally, we obtain the SDP relaxation by dropping the rank constraint:

$$\underset{\tilde{\mathbf{V}}}{\operatorname{argmin}}\ \mathbf{H} : \tilde{\mathbf{V}} \qquad \text{s. t.} \qquad \begin{array}{l} \mathbf{D} : \tilde{\mathbf{V}} = 4k \\ \operatorname{diag}(\tilde{\mathbf{V}}) = \mathbf{e} \\ \tilde{\mathbf{V}} \succeq 0 \end{array} \tag{44}$$

Dropping the rank constraint, means that the components of $\mathbf{v}$ are now real values, instead of being restricted to the integers $\{-1, 1\}$, as in Equation (39). Therefore, after obtaining a solution for $\mathbf{v}$, additional manipulations are required to recover integer values in $\{-1, 1\}$. These are described in [46].

It is important to remark that the optimal solutions of the original problem coincide with the optimal solutions of the MC-k problem, but their optimal values do not. Since approximation guaranties of SDP can only be proven for the MC-k problem (Equation (31)), they do not apply for the relaxed problem (Equation (33)). Therefore, the solutions obtained by SDP on the relaxed version of ensemble pruning need not be close to the global optimum.

This method is compared to ordered aggregation techniques in [3], using bagging ensembles of 101 CART trees [45]. SDP pruning obtains comparable results to margin distance minimization (MSDQ). However, SDP pruning is much slower than MSDQ.

The performance of SDP pruning is also analyzed in [7] using boosting ensembles for 100 C4.5 trees [47], and pruning them to 25 classifiers. Sixteen UCI repository data sets were used [14] in that study. Problems with more than two classed were binarized. The conclusion of this investigation is that SDP pruning outperforms kappa pruning, as in [3], but not boosting. The complete boosting ensemble performs better than SDP pruning in 50% of the datasets. We were unable to reproduce the results with boosting and C4.5. In fact, boosting ensembles generated in this work achieve lower generalization errors than the ensembles used in [7] under similar experimental conditions.

4.2.3 Genetic algorithms

Genetic algorithms (GA) are a class of optimization methods that mimic the process of natural evolution [9, 48]. Optimization is achieved by selection from a population that exhibits some random variability. This variability is introduced using crossover and mutation operators, which depend on the encoding of individuals of the population. The pseudocode of a general genetic algorithm is shown in Algorithm 4.

In ensemble pruning, two different encodings can be used: binary and floating point representations [3, 11, 49]. In binary encoding, individuals are characterized by a bitstring whose size is equal to the number of classifiers

Algorithm 4 General outline of a Genetic Algorithm [13]

- Generate an initial population $\mathcal{P}_0$ with P individuals.
- For each individual $I_j \in \mathcal{P}_0$, calculate fitness $\Phi(I_j)$.
- Initialize the generation counter $t \leftarrow 0$.
- While convergence criteria are not met:
  - Increase the generation counter $t \leftarrow t + 1$.
  - Select a parent set $\Pi_t \subset \mathcal{P}_t$ composed of n_P individuals from the population.
  - While $\Pi_t \neq \emptyset$:
    - Extract two individuals I_1 and I_2 from Π_t.
    - Apply the crossover operator $\Theta(I_1, I_2)$ and generate n_C children (with probability p_C).
    - Apply the mutation operator to the n_C children (with probability p_M).
  - Calculate the fitness value of the new individuals.
  - Add the new individuals to the population.
  - Select P individuals that make up $\mathcal{P}_{t+1}$, the population for generation $t + 1$.

in the ensemble, T. The j-th position of the individual I_k indicates whether classifier D_j is selected in the subensemble ($I_{k,j} = 1$) or not ($I_{k,j} = 0$). A common mutation operator is the exchange of two bits per individual with probability p_m. The usual crossover operator is uniform crossover. Uniform crossover exchanges the genes of parents I_1 and I_2 at random.

In floating point encoding, individuals are characterized by a string of doubles of size equal to the number of classifiers T. The position j represents the weight of the classifier in the ensemble. Usually, classifiers whose weight is below a threshold are eliminated from the ensemble at the end of the process [49]. A common mutation operator used in conjunction with this representation consists in adding some random value obtained from a probability distribution (e.g. a uniform distribution between $[-1, 1]$).

Another encoding that is widely used in cardinality constrained problems is set encoding [12]. In set encoding, the individuals are identified by a set of labels (for example, integers) that identify the classifiers that are selected in the subensemble. The size of the sets is equal to the target subensemble size, k. The use of this representation simplifies the design of crossover and mutation operators that preserve the cardinality of the candidate solutions. An example of a mutation operator that respects the cardinality of the solution is to exchange one of the selected classifiers with one of the unselected classifiers with probability p_m. Crossover operators on sets were introduced in [12]. They are defined by taking into account the properties of *respect* and *assortment* [50]. Respect ensures that the offspring inherit the common genetic material of the parents. Assortment guarantees that every combination of the alleles of the two parents is possible in the child, provided that these alleles are compatible. When cardinality constraints are considered, it is not possible to design crossover operators that guarantee both respect and assortment. An operator that provides a good balance between these properties is Random Assorting Recombination (RAR). The RAR crossover operation is described in Algorithm 5. In this pseudocode, the integer parameter w determines the amount of common information from the parents that is retained by the offspring. Using set representation with RAR-crossover is known to be superior to other representations and crossover operators in a variety of cardinality constraint problems [13, 51].

The usual fitness function used for ensemble pruning is minus the error of the subensemble on the training data [3, 11]. These methods obtain similar generalization error to SDP pruning and the ordered aggregation method when pruning bagging ensembles [3, 11], usually at a greater computational cost.

Algorithm 5 Random Assortment Recombination algorithm

Input: Two parents I_1 and I_2, and a fixed cardinality k.
Output: A child chromosome Θ.

- Create auxiliary sets A, B, C, D, E:
  - $A =$ classifiers present in both parents.
  - $B =$ classifiers not present in any of the parents.
  - $C \equiv D$ classifiers present in only one parent.
  - $E = \emptyset$.
- Build set $G = \{w$ copies of elements from A and B, and 1 copy of elements in C and $D\}$
- While $|\Theta| < k$ and $G \neq \emptyset$:
  - Extract $g \in G$ without replacement.
  - If $g \in A$ or $g \in C$, and $g \notin E$, $\Theta = \Theta \cup \{g\}$.
  - If $g \in B$ or $g \in D$, $E = E \cup \{g\}$.
- If $|\Theta| < k$, add classifiers chosen at random from the set of classifiers from the ensemble not selected yet, $\mathcal{D} - \Theta$, until chromosome is complete.

4.2.4 Simulated Annealing

Simulated annealing (SA) is an optimization technique inspired in the field of thermodynamics [10]. The main idea is to mimic the physical process of melting a solid and then cooling it to allow the formation of a regular crystalline structure that attains a minimum of the system's free energy.

In simulated annealing the function to be minimized $F(\mathbf{z})$ (objective or cost function) takes the role of the free energy in the physical system. The physical configuration space is replaced by the space of candidate solutions, which are connected by transitions defined by a neighborhood operator. The stochastic search proceeds by considering possible transitions from the current state $\mathbf{z}^{(cur)}$ to a neighboring configuration $\mathbf{z}_l \in \mathcal{N}(\mathbf{z}^{(cur)})$ generated at random. The proposed transition is accepted if it lowers the value of the objective function. Otherwise, if the candidate configuration is of higher cost, the transition is accepted only with a certain probability, which is expressed as a Boltzmann factor

$$P_{\text{accept}}(\mathbf{z}_l, \mathbf{z}^{(cur)}; T_k) = \exp\left(-\frac{F(\mathbf{z}_l) - F(\mathbf{z}^{(cur)})}{T_k}\right), \tag{45}$$

where the parameter T_k plays the role of a temperature. A general version of the algorithm is given in Algorithm 6.

We propose to use SA to obtain approximate solutions to Equation (23). Specifically, the candidate solutions can be encoded as sets of specified cardinality k. The components of the binary vector $\mathbf{z}$ are then interpreted as indicating membership to the set: if $z_i = 1$, the ith element is included in the solution. Otherwise, if $z_i = 0$ it is excluded from the selection. It is also necessary to design a neighborhood operator that preserves the cardinality constraints, so that no penalty or repair mechanisms are needed. A simple design is to exchange an element included in the current candidate solution with an element excluded from it. This mechanism for the generation of neighboring solutions is the same as the mutation operator used in GA. This is the version of SA that will be used in the experiments in the following section.

5 Experiments

In this section we present the results of the experiments on pruning boosting ensembles based on the accuracy-diversity measure $\tilde{g}$. The final goal is to determine whether the size of the boosting ensembles can be reduced to decrease the memory requirements and to increase the classification speed

Algorithm 6 Simulated annealing

- Generate initial configuration $\mathbf{z}^{(\mathbf{0})}$ and initial temperature T_0
- $\mathbf{z}^{(cur)} \leftarrow \mathbf{z}^{(0)}$
- $k \leftarrow 0$
- While convergence criteria are not met [Annealing loop]
  - $k \leftarrow k + 1$
  - Fix temperature for epoch k
    $T_k = \text{annealingSchedule}(T_{k-1})$
  - For $l = 1, \ldots, L_k$ [Epoch loop]
    1. Select randomly an element $\mathbf{z}_l \in \mathcal{N}(\mathbf{z}^{(cur)})$.
    2. If $F(\mathbf{z}_l) < F(\mathbf{z}^{(cur)})$, then $\mathbf{z}^{(cur)} \leftarrow \mathbf{z}_l$
    3. Else, generate $u \sim U[0, 1]$
       If $u < P_{\text{accept}}(\mathbf{z}_l, \mathbf{z}^{(cur)}; T_k)$, then $\mathbf{z}^{(cur)} \leftarrow \mathbf{z}_l$
- Return the best value found.

without deterioration of their performance. The methods that are compared are:

- a simple strategy that consists in generating a smaller boosting ensemble (early stopping)
- early stopping in ordered aggregation by greedy minimization of $\tilde{g}$ (Greedy)
- solution of a relaxed version of the pruning problem by semidefinite programming pruning (SDP)
- minimization of $\tilde{g}$ by means of a genetic algorithm with set representation and RAR crossover operator (GA-RAR)
- minimization of $\tilde{g}$ using simulated annealing (SA)

The first batch of experiments compares these methods to an exact solution obtained by exhaustive search in ensembles of medium sizes. The objective is to determine how close to the optimal solution are the approximate solutions obtained by the different pruning methods. The second group of experiments

compares the performance of the different pruning heuristics with larger ensembles. Finally, the time complexity of the pruning techniques is analyzed.

These experiments are preceded by exploratory tests. The goal is to determine which implementation of boosting is better to generate ensembles for pruning. The implementations tested are boosting with reweighting, boosting resampling without and with shrinkage. These experiments show that the differences between the methods are small. In consequence, the standard version of boosting with reweighting is used in the main experiments.

There are some previous investigations on methods to prune boosting ensembles. In [6] kappa-pruning and reduced-error pruning (Section 4) are compared to early stopping and to the standard boosting without pruning. None of the pruned subensembles identified by these techniques perform consistently better than the complete ensemble. Early stopping performs worse than the pruning strategies. In [7], SDP pruning is compared to kappa-pruning. Ensembles generated with SDP pruning perform better than those built with kappa-pruning. Their performance is comparable to complete boosting in approximately a half of the tested datasets. The conclusion of this work is that pruning boosting does not improve the performance of the ensemble, but that the size of the ensemble can be reduced without a significant deterioration of performance. Nevertheless, the version of boosting implemented in [7] builds complete ensembles whose performance is lower than the complete ensembles built with our implementation of boosting.

SDP pruning is based on the measure of accuracy-diversity $\tilde{g}$, which seems to be a good heuristic to prune ensembles. Indeed, SDP pruning is very effective in bagging ensembles [3]. On the other hand, the pruning techniques based in ordered aggregation described in [3] do not perform well in ensembles when the training error is near 0. Under these conditions, the metrics used to guide ordered aggregation cannot discriminate among the different subensembles. Therefore, using methods based on $\tilde{g}$ seems to be a reasonable approach to prune boosting.

We also investigate whether the minimization of $\tilde{g}$ with other optimization methods different from SDP is an effective method for ensemble pruning: early stopping based on minimization of $\tilde{g}$ (Greedy), a genetic algorithm with set representation an RAR crossover operator (GA-RAR) and simulated annealing (SA). SDP pruning obtains exact solutions for a relaxed version of the problem (see Section 4.2.2). The original and the relaxed problem have the same optima. However, the global optimum of the relaxed problem need not be the global optimum of the original problem. Therefore the solution need not be the global optimum. Greedy search obtains approximate

Table 1: Characteristics of the datasets and testing methods

Dataset	Instances	Test	Attrib.	Classes	Dataset	Instances	Test	Attrib.	Classes
Audio	226	10-fold-cv	69	24	Ringnorm	300	5000 cases	20	2
Australian	690	10-fold-cv	14	2	Satellite	6435	10-fold-cv	36	2
Breast W.	699	10-fold-cv	9	2	Segment	2310	10-fold-cv	19	7
Diabetes	768	10-fold-cv	8	2	Soybean	683	10-fold-cv	35	19
Ecoli	336	10-fold-cv	7	8	Sonar	208	10-fold-cv	60	2
German	1000	10-fold-cv	20	2	Spam	4601	10-fold-cv	57	2
Glass	214	10-fold-cv	9	6	Tic-tac-toe	958	10-fold-cv	9	2
Heart	270	10-fold-cv	13	2	Twonorm	300	5000 cases	20	2
Horse-Colic	368	10-fold-cv	21	2	Vehicle	846	10-fold-cv	18	4
Ionosphere	351	10-fold-cv	34	2	Votes	435	10-fold-cv	16	2
Labor	57	10-fold-cv	16	2	Vowel	990	10-fold-cv	10	11
Liver	345	10-fold-cv	6	2	Waveform	300	5000 cases	21	3
New-thyroid	215	10-fold-cv	5	3	Wine	178	10-fold-cv	13	3
Pendigits	10992	10-fold-cv	16	10					

solutions that are generally suboptimal local minima. It has the advantage of being a very fast algorithm. SA guarantees convergence to the global optimum if the search space is connected and if a sufficient slow annealing schedule is used [10]. There are no equivalent theorems of convergence for GA. However, extensive empirical evidence shows that GA is effective in identifying near-optimal solutions, provided that an adequate representation and crossover and mutation operators are used. It also is important to recall that $\tilde{g}$ is used because minimizing $\tilde{g}$ in the training dataset generally leads to good generalization performance. However, there is not guarantee that minimizing $\tilde{g}$ on the training set leads to optimal generalization performance.

The parameters for the pruning methods are determined in exploratory experiments using the results of SDP pruning as a gauge. For the GA-RAR, populations with 100 individuals are evolved using a steady state generational substitution scheme. The crossover probability is set to 1 and the mutation probability is 10^{-3} for GAs with set representation. The parameter w of the crossover operator is set to 1. A geometric annealing schedules with $\gamma = 0.9$ is used in SA. In these experiments, the best solution in 10 independent executions of the SA algorithm is chosen.

5.1 Preliminary experiments

This section presents the results of preliminary experiments. Their goal is to determine in which implementation of boosting is best as a starting point for pruning.

5.1.1 Variants of boosting

Section 3.2 describes a modified version of boosting that incorporates shrinkage. Shrinkage makes a smooth modification of the weights at each step. This entails longer training times than standard versions of boosting. Nevertheless, it is generally believed that shrinkage improves the generalization performance and robustness of boosting. The goal of these experiments is to determine whether boosting with shrinkage obtains a initial ensemble whose base classifiers are less correlated. Boosting can be implemented with reweighting, if the base learning algorithm can handle training instances with different weights. Otherwise, resampling can be used. In boosting with resampling, the classifiers are built using bootstrap samples in which the sampling probability in proportional to the weights of the instances. Since shrinkage can be used in combination with both versions of boosting, we include in this experiment the four variants of boosting: boosting with reweighting, boosting with resampling, boosting with reweighting and shrinkage, and boosting with resampling and shrinkage.

For these experiments, 5 representative classification problems from the UCI repository are selected [14]: diabetes, heart, sonar, waveform and wine. For each of the datasets, ensembles of 101 CART trees [45] are built using boosting with resampling and with reweighting without shrinkage. Ensembles of 201 CART trees are built using boosting with resampling and with reweighting with shrinkage ($\nu = 0.1$). More classifiers are generated in boosting with shrinkage because learning is slower and typically larger ensembles are needed to obtain a good generalization performance. The ensembles are pruned to a 20% of the original size using greedy pruning and SDP pruning. GA-RAR and SA are not employed because of their large computational cost. Nevertheless, the results obtained for greedy and SDP pruning are similar to what would be obtained with these methods. The accuracy-diversity measure $\tilde{g}$ on the training set and the test error are averaged over 10 ten-fold cross-validations for diabetes, heart, sonar and wine and over 100 independent realizations of the training and test data for the synthetic problem waveform.

Table 2 displays the test error rates for the different strategies. None of the methods achieve error rates lower than the complete ensemble. The pruning methods exhibit better performance than the early stopping strategy in sonar, waveform and wine using boosting with reweighting, but not in the other variants of boosting. The complete ensembles built with boosting with shrinkage obtain a lower generalization error than the versions without shrinkage, because the size of the ensemble is greater. Boosting with reweighting and shrinkage has worse performance than boosting with resam-

pling and shrinkage. The reason for this behavior could be that ensembles built with boosting with reweighting reach zero error in the first iterations, which means that these type of ensemble could be more prone to overfitting. By contrast, the version of boosting with resampling does not reach zero error, because of the variability of bootstrap samples that are used to construct each base classifier.

Therefore, using boosting resampling and shrinkage do not seem to improve by a significant amount the performance of the pruning methods in the datasets investigated. In consequence, we will use boosting with reweighting in the remaining experiments because it is the more standard version of boosting.

Table 2: Comparison of the different implementations of boosting. The test error (%) is shown for the different ensemble methods and the different pruning strategies. Pruning rate = 20%.

Problem	Ensemble method	Complete Ensemble	Early Stopping	Greedy	SDP
diabetes	reweighting	26.76(±3.86)	27.56(±4.71)	27.61(±4.47)	27.53(±4.61)
	resampling	27.54(±4.46)	27.71(±4.44)	28.50(±4.69)	28.70(±4.66)
	reweighting shrinkage	25.44(±4.38)	25.19(±4.39)	26.18(±4.54)	26.18(±4.36)
	resampling shrinkage	**24.96**(±**4.29**)	**24.10**(±**3.86**)	**26.03**(±**4.13**)	**26.12**(±**4.10**)
heart	reweighting	20.85(±7.57)	21.52(±7.70)	22.56(±7.61)	22.22(±7.61)
	resampling	20.41(±7.36)	20.63(±7.26)	21.37(±7.12)	21.33(±7.33)
	reweighting shrinkage	20.07(±7.67)	19.48(±7.59)	**20.56**(±**7.62**)	20.96(±7.49)
	resampling shrinkage	**18.93**(±**7.37**)	**17.67**(±**6.18**)	20.78(±7.73)	**20.63**(±**7.56**)
sonar	reweighting	13.32(±7.98)	17.69(±8.29)	17.07(±8.30)	16.74(±8.31)
	resampling	14.43(±7.56)	19.72(±8.77)	18.71(±8.06)	18.52(±8.09)
	reweighting shrinkage	13.90(±7.60)	**15.91**(±**7.87**)	**14.55**(±**8.16**)	14.70(±8.23)
	resampling shrinkage	**12.83**(±**7.77**)	18.17(±9.72)	14.86(±7.83)	**14.48**(±**7.60**)
waveform	reweighting	17.95(±0.79)	20.11(±0.97)	19.68(±0.86)	19.69(±0.86)
	resampling	17.77(±0.81)	19.94(±0.91)	19.90(±0.90)	19.91(±0.91)
	reweighting shrinkage	19.80(±1.10)	20.73(±1.23)	20.44(±1.14)	20.48(±1.10)
	resampling shrinkage	**17.60**(±**0.83**)	**18.83**(±**0.94**)	**18.55**(±**0.80**)	**18.56**(±**0.80**)
wine	reweighting	3.36(±4.21)	3.31(±4.08)	3.59(±4.33)	3.25(±4.14)
	resampling	**3.20**(±**4.24**)	3.37(±4.16)	**3.25**(±**3.83**)	**3.19**(±**3.92**)
	reweighting shrinkage	3.37(±4.15)	**3.20**(±**3.92**)	3.53(±4.10)	3.64(±4.16)
	resampling shrinkage	4.48(±4.64)	3.47(±4.12)	3.70(±4.65)	3.70(±4.65)

5.2 Exact solution by exhaustive search

In this section the accuracy of the approximate solution obtained by the methods described in Section 4 is evaluated. For this purpose, the performance of the different algorithms is compared with the exact solution obtained by exhaustive search. Exhaustive search generates all the possible subsets of a given cardinality and selects the one that minimizes $\tilde{g}$. Since all the possible subsets are considered, this solution is the optimal one. The cost of exhaustive search grows exponentially with the size of the ensemble. For example, for an ensemble of 100 classifiers which is a size that is commonly used in the literature, the algorithm needs to evaluate $\mathcal{O}(2^{100})$ possible subensembles. For this reason, we restrict the experiment to ensembles of 31 classifiers only. Nevertheless, we expect that the conclusion remain valid for larger ensembles.

The experiments are performed on 27 classification problems obtained from the UCI repository [14]. Each experiment consists in 100 executions for each dataset. For the synthetic problems (Ringnorm, Twonorm and Waveform) random training and testing samples are generated. In the remaining datasets, 10x10-fold cross-validation is used. Each execution involves the following steps:

- Generate the training and testing datasets, $\mathcal{Z}_{train}$ and $\mathcal{Z}_{test}$, by repeating 10 times 10-fold-cv for real-world classification problems and by random sampling for synthetic problems (see Table 5). Each 10-fold-cv is generated by splitting the data into 10 disjoint folds. For each of these different folds, 9 are used as training set $\mathcal{Z}_{train}$ and the remaining as test dataset.

- Build a boosting ensemble of 31 standard (pruned) CART trees [45]. For the synthetic problems, 100 realizations of the training and test sets are generated.

- Prune the ensemble using exhaustive search and the heuristics described in Section 4: Greedy search (Greedy), Semidefinite-programming (SDP), Genetic Algorithms (GA-RAR) and Simulated annealing (SA). The selection dataset $\mathcal{Z}_{sel}$ used is the training dataset $\mathcal{Z}_{train}$. The ensembles are pruned to odd sizes between 1 and 31, in order to avoid ties in the majority vote for binary classification problems and to reduce the computation time.

Figure 7 displays the curves that trace the dependence of the average value of the objective function $\tilde{g}$ (left) and the test error (right) with the size of the

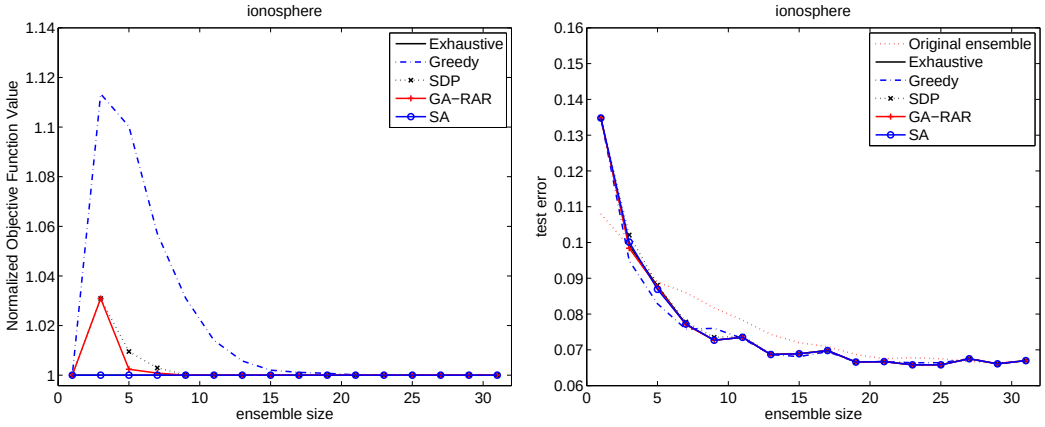

Figure 7: (left) Average value of the objective function defined in Equation (23) that is obtained by each pruning strategy, for each different subensemble. The value of this function is normalized by the optimal value, i.e. the value obtained by the exhaustive search procedure. (right) Average test error. The test error of the original ensemble is also displayed as a reference.

subensemble for the different pruning strategies in the problem *ionosphere*. The curves for the remaining datasets are included in the appendix and are similar to the ones in Figure 7.

The value of the objective function is normalized by the optimal value, i.e. the value that is obtained by exhaustive search. These results confirm that in most cases the pruning methods select subensembles that are close to the optimal ones. The best optimization procedure is SA. This technique obtains values that are at worst only 3% above the optimal ones. The GA with the RAR operator and SDP pruning obtain similar results. Finally, the greedy algorithm provides subensembles whose associated objective function is at worst 10% above the optimal value.

Figure 7 (right) displays the average test error obtained by the different subensembles obtained with the different pruning strategies. The test error of the original ensemble is also displayed as a reference. This figure shows that the errors of all the pruning strategies are very similar. For subensembles of small size ($k < 5$) all the strategies select subensembles whose generalization performance is worse than the subensembles obtained by early stopping. However, for subensembles of intermediate size ($k > 5$), the opposite effect is observed. The test errors of the subensembles selected by the different pruning strategies show a steeper descent. In consequence, the error rates of these subensembles is lower than what would be obtained by early stopping.

5.3 Classification performance of pruned ensembles

A series of experiments are carried out to analyze and compare the performance of the different pruning heuristics in a range of classification problems from different domains. Boosting ensembles of 101 pruned CART trees are generated. Then, subensembles of sizes of 21, 31 and 51 are selected using the pruning heuristics described in Section 4. Only odd sizes are considered to avoid ties in binary classifier problems. Different sizes are considered to determine whether the performance of the method depends on the pruning rate.

The protocol followed in the experiments is:

- Generate the training and testing datasets, $\mathcal{Z}_{train}$ and $\mathcal{Z}_{test}$, by either 10-fold-cv (real-word data) or by random sampling (synthetic data). Table 5 summarizes the characteristics of the datasets. The partitions are the same as in the previous experiments.

- Build a boosting ensemble of 101 standard (pruned) CART trees [45].

- Prune the ensemble using the approximate methods described in Section 4 to minimize $\tilde{g}$: greedy search (Greedy), semidefinite-programming (SDP), genetic algorithms (GA-RAR) and simulated annealing (SA). The selection dataset $\mathcal{Z}_{sel}$ used is the training dataset $\mathcal{Z}_{train}$. The ensembles are pruned to sizes that are equal to $\approx 20\%$, $\approx 30\%$ and $\approx 50\%$ of the original ensemble size (that is, $k = 21$, $k = 31$ and $k = 51$ classifiers). The values of $\tilde{g}$ in the training set and the test error are calculated for each algorithm and subensemble size. We also compute these both measures for the full ensemble and for the subensemble that results from early stopping.

Table 5 displays the average generalization error of the full ensemble and for the different pruning strategies for $k = 21$. Table 4 displays the value of the objective function. The best error rate among the different pruning strategies is highlighted in boldface. The second best is underlined. The table shows that the errors of pruned ensembles with $k = 21$ are typically larger than the error of the complete ensemble. However, in many problems these errors are smaller than the error of the subensemble of the same size obtained by early stopping. We compare the overall performance of the different pruning strategies and the early stopping procedure following the methodology introduced in [52]. A Friedman test is first performed. The null-hypothesis of this test states that all the algorithms are equivalent. This hypothesis is rejected for all tested values of k ($k = 21$, $k = 31$, $k = 51$) with

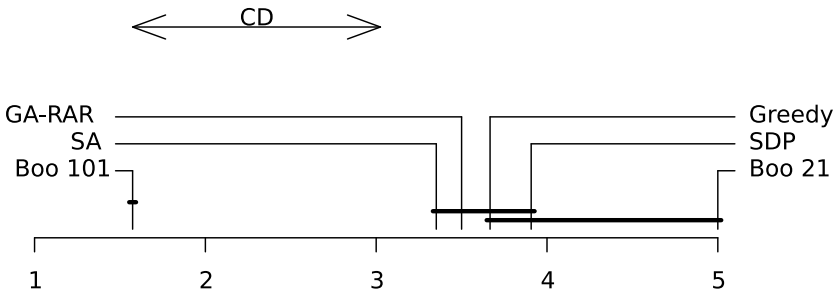

Figure 8: Results of a Nemenyi test on the average ranks of each pruning strategy, early stopping and the complete ensemble, $k = 21$. The critical value (CD) is displayed at the top of the picture. This value indicates the differences in average rank that are statistically significant at the α-value = 5%.

an α-value of 5%. Then, a Nemenyi test is performed to determine whether the differences in average rank among these methods are statistically significant. Figure 8 displays the result of this test. Methods whose differences in average rank are not statistically significant with an α-value of 5% are linked with a solid black segment. Thus, this figure shows that Greedy, GA-RAR and SA outperform early stopping in the set of classification problems investigated. These results also show that there is not enough statistical evidence to discriminate among the different pruning strategies and early stopping. The differences among early stopping and all the pruning methods are very small.

The generalization error of the pruning strategies is expected to decrease when the number of selected classifiers is increased. Therefore, the results of subensembles selected with lower pruning rates are also reported. Tables 7 and 9 display the average generalization error of the full ensemble and each different pruning strategy for $k = 31$ and $k = 51$, respectively. Tables 6 and 8 display the average value of $\tilde{g}$. In both cases, the test error of the pruning strategies and early stopping decreases as a result of considering larger subensembles. Furthermore, some of the pruning strategies obtain a generalization error lower than the complete ensemble. To determine whether the differences in performance are statistically significant, we perform a Nemenyi test including the complete ensemble (Figures 9 and 10). These figures show that pruning the boosting ensemble to these larger sizes (30%-50% of the original size) does not significant deteriorate the overall generalization performance of the ensemble. Early stopping has the lowest average rank.

The conclusion from this empirical study is that pruning the complete boosting ensemble is generally better than early stopping in boosting, al-

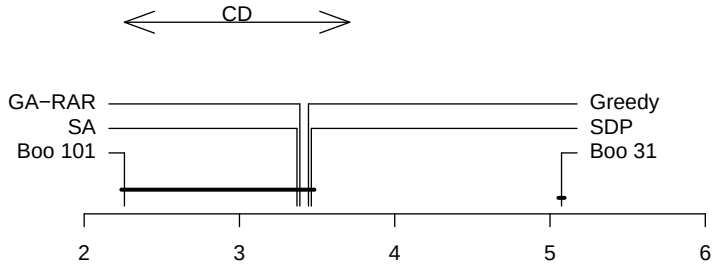

Figure 9: Results of a Nemenyi test on the average ranks of each pruning strategy, early stopping and the complete ensemble, $k = 31$. The critical value (CD) is displayed at the top of the picture. This value indicates the differences in average rank that are statistically significant at the α-value = 5%.

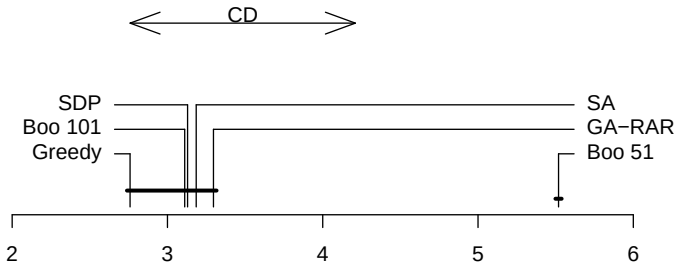

Figure 10: Results of a Nemenyi test on the average ranks of each pruning strategy, early stopping and the complete ensemble, $k = 51$. The critical value (CD) is displayed at the top of the picture. This value indicates the differences in average rank that are statistically significant at the α-value = 5%.

though the differences in the generalization performance are small. When the number of selected classifiers in the pruned subensemble grows ($k \geq 31$ in our tests), then the performance of the pruned ensembles is indistinguishable from the complete ensemble. Therefore, boosting ensembles can be pruned to reduce memory and to reduce the costs of classifying new instances without a significant deterioration of the performance.

The tables with the values of the training errors of the pruned ensembles are not included because boosting reaches zero error in most of the classification problems.

5.4 Efficiency analysis

Since the generalization performance of the different strategies based on the minimization of $\tilde{g}$ is similar, the choice of a specific pruning method should be guided by its computational efficiency. For this purpose, we perform a complexity analysis to determine the cost of the different strategies. Table 3 displays the theoretical complexity of the algorithms investigated in terms of the ensemble size, T. For the GA-RAR, the complexity is also given in terms of the initial population, P, and the number epochs, E. For the SA, the maximum number of allowed iterations, M, is taken into account too. Another parameter controls the maximum number of iterations that SA is allowed to run without an improvement of the solution. This parameter is typically smaller than M. The total cost includes the cost of pruning and the cost of computing the matrix $\mathbf{G}$, which is $\mathcal{O}(T^2 \cdot N)$, where N is the number of training instances.

The cost of the greedy algorithm and the GA-RAR strategy is quadratic in T. The cost of SDP pruning approach is cubic in T. The complexity of GA-RAR and SA strongly depends on the values of the parameters of the algorithm. The empirical dependence on T of the execution time of the different pruning strategies is investigated in a series of experiments on the *diabetes* dataset. Table 3 displays the average execution time of each algorithm for initial ensembles of 51, 101, 201, 401, 801 and 1601 classifiers. The size of the pruned subensembles is $\approx 20\%$ of the original size, that is, $k = 11$, $k = 21$, $k = 41$, $k = 81$, $k = 161$ and $k = 321$ respectively. The times reported are averages over 10 realizations on a QuadCore Q6660 processor. The results displayed in this table show that the best results correspond to the greedy algorithm, as expected. However, SDP pruning is faster than the GA-RAR algorithm for $D < 401$. The slower method is SA, because of the algorithm selects the best of 10 runs, where run takes a long time to converge. The run with $T = 1601$ required to increment the maximum number of iterations without an improvement on the candidate solution. The

Table 3: Theoretical time complexity and average execution time in seconds for the different pruning strategies for the *diabetes* dataset as a function of T, the number of classifiers in the ensemble.

	Theoretical Complexity	Average execution time for T=					
Pruning Method		51	101	201	401	801	1601
Greedy	$\mathcal{O}(T^2)$	0.022	0.076	0.252	0.933	3.671	15.070
SDP	$\mathcal{O}(T^3)$	0.113	0.7036	3.567	27.48	277.2	2840.04
GA-RAR	$\mathcal{O}(E \cdot P \cdot T^2)$	0.349	1.099	3.76	13.86	53.31	210.96
SA	$\mathcal{O}(M \cdot T^2)$	8.586	25.313	132.869	790.837	4937.82	>7200

algorithm did not converged due to the huge search space.

The results of the experiments presented in Section 5.3 show that the different pruning strategies have a similar generalization performance. Therefore, the greedy strategy should be preferred because of its lower computational cost. This conclusion is consistent with those obtained in [3]. In that research, subensembles selected with ordered aggregation were shown to achieve similar error rates as the ones obtained by SDP pruning at a lower computational cost.

6 Conclusions and future work

This work introduces different methods to prune boosting ensembles and analyze their effectiveness and complexity. In contrast to bagging, where the order of aggregation is left unspecified, boosting is a sequential ensemble learning algorithm, in which classifiers are incorporated in the ensemble in a prescribed order. While modifying the order in which classifiers are aggregated in bagging has been shown to be effective [3], it is not clear whether this should be a successful strategy in boosting. Furthermore, boosting often drives the training error to zero in the first few iterations. Common measures of performance used in ensemble pruning, which are based on the accuracy of the ensemble on the training dataset, are unable to distinguish among different subensembles. Therefore, measures based on the accuracy and diversity of the ensemble should be used. Specifically, this work proposes to use the measure $\tilde{g}$ developed in [7]. $\tilde{g}$ combines terms that favor the selection of individual classifiers that are accurate, with terms that favor classifiers whose performance is complementary. In [7], subensembles that optimize this measure are obtained using an relaxed version of the original problem which can be solved exactly by semidefinite-programming (SDP) [8].

In this work, three pruning methods based on optimizing $\tilde{g}$ are developed. The first one is ordered aggregation. In ordered aggregation, the ensemble grows by incorporating from the pool generated by boosting the classifier

that minimizes the value of $\tilde{g}$ in the enlarged subensemble (Greedy). Ordered aggregation is a very efficient method for pruning. It selects subensembles that are suboptimal with respect to the quality measure $\tilde{g}$ evaluated in the training set, but which have good generalization performance. The second one is a genetic algorithm (GA) with set encoding and RAR crossover. RAR crossover is specially adapted to solve this type of problem because it respects the cardinality of the candidate solutions. Set encoding with RAR crossover has been shown to perform better than other implementations of GAs in a variety of cardinality constrained problems [13]. Finally, simulated annealing (SA) is used to obtain pruned subensembles based on minimizing $\tilde{g}$. SA is a well-known optimization method [10].

A series of experiments are carried out to analyze and compare the performance of the different pruning heuristics: Greedy, SDP, GA-RAR and SA. A method consisting on early stopping of the boosting process is also analyzed. The experiments consist in pruning boosting ensembles of 101 CART trees to sizes of $k = 21$ classifiers ($\approx 20\%$), $k = 31$ ($\approx 30\%$) and $k = 51$ ($\approx 50\%$). Boosting with reweighting is used to build the original ensembles which serve as a starting point for pruning. For $k = 21$, all the pruning methods investigated outperform early stopping. However, none of the pruned subensembles improves the performance of the complete boosting ensemble. For $k = 31$ and $k = 51$, the generalization performance of the pruned subensembles improves and is similar to the complete boosting ensemble. In conclusion, boosting can be pruned by a large amount ($\approx 20\%$) with a small but significant deterioration of performance. In contrast, boosting can be pruned to sizes around 30% and 50% of the original size without a significant deterioration of its performance. In fact, the performance is slightly improved in some problems.

The results obtained also show that the different pruning algorithms have similar performance irrespective of the optimization method employed. Therefore, the choice of pruning method should be made on the basis of computational efficiency. As expected, the greedy method is the fastest one. The slowest method is SA. These conclusions are consistent with previous work in bagging ensembles [3], in which ordered aggregation strategies show similar performance to another more complex methods at a lower computational cost.

Some questions remain unanswered in this work. The noise tolerance of the pruned subensembles has not been tested. Boosting is known to be sensitive to noise in the class labels. In the presence of noise, boosting tends to overfit the noisy instances. This effect often leads to a worse generalization performance. Pruning the ensemble probably reduces the amount of overfitting. This could be one of the reason why pruning is beneficial in boosting

ensembles. The error of the pruned subensemble may improve the accuracy of the complete ensemble.

The dependence of the pruning methods on the complexity of the base classifiers should be analyzed as well. Previous work [3] shows that pruning bagging ensembles of unpruned CART trees does not improve and sometimes even deteriorates the generalization performance of the ensemble.

It would also be desirable to automatically determine the optimal pruning rate. The generalization error curves for pruned subensembles generated by bagging typically have a broad minimum at $\approx 20\%$ of the original size [3]. We used this heuristic as a guide, and also tested for subensembles of size $\approx 30\%$ and $\approx 50\%$ of the original size. The experiments show that the minimum is generally found at sizes larger than 20% of the original boosting ensemble size. Subensembles of size 20% have worse performance than the complete ensemble, but subensembles of size $\approx 30\%$ and $\approx 50\%$ outperform it in some datasets. To determine where the minimum is actually located, the generalization error for each subensemble size between 1 and the original size should be computed.

Another interesting question is to analyze the influence of the size the initial pool of classifiers on the effectiveness of the pruning methods. In bagging, the curves that trace the dependence of the error with the size of the subensembles in ordered aggregation for initial bagging ensembles of lower size are nested with in the error curves of ensembles with larger sizes. Besides, increasing the size of the initial pool of classifiers generally leads to an improvement of the generalization error of the bagging ensembles, although beyond a certain size, the rate of error reduction is small [3]. Boosting ensembles may also exhibit this behavior. In fact, the experiments on boosting with shrinkage using that larger boosting ensembles leads to lower generalization errors.

References

1. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern classification. John Wiley & Sons (2001)

2. Dietterich, T.G.: Ensemble methods in machine learning multiple classifier systems. Lectures Notes in Computer Science **1857** (2001) 1–15

3. Martínez-Muñoz, G., Hernández-Lobato, D., Suárez, A.: An analysis of ensemble pruining techniques based on ordered aggregation. IEEE Transactions on Pattern Analysis and Machine Intelligence **31** (2009) 245–259

4. Breiman, L.: Bagging predictors. In: Machine Learning. Volume 24. (1996) 123–140

5. Schapire, R.E., Freund, Y., Bartlett, P., Lee, W.S.: Boosting the margin: A new explanation for the effectiveness of voting methods. The Annals of Statistics **12**(5) (1998) 1651–1686

6. Margineantu, D.D., Dietterich, T.G.: Pruning adaptive boosting. Proceedings of the Fourteenth International Conference on Machine Learning (1997) 211–218

7. Yi Zang, S.B., Street, W.N.: Ensemble pruning via semi-definite programming. Journal of Machine Learning **7** (2006) 1315–1338

8. Vandenberghe, L., Boyd, S.: Semidefinite programming. SIAM Review **38**(1) (1996) 49–95

9. Mitchell, M.: An introduction to genetic algorithms. The MIT Press (1996)

10. Kirkpatrick, S., Gelatt, C.D., M. P. Vecchi, J.: Optimization by simulated annealing. Science **4598** (1983) 671–679

11. Hernández-Lobato, D., Hernández-Lobato, J.M., Ruiz-Torrubiano, R., Valle, A.: Pruning adaptive boosting ensembles by means of a genetic algorithm. In 7th Intelligent Data Engineering and Automated Learning (4224) (2006) 995–1002

12. Radcliffe, N.J.: Genetic set recombination. In Foundations of Genetic Algorithms **11** (1993)

13. Ruiz-Torrubiano, R., García-Moratilla, S., Suárez, A.: Optimization problems with cardinality constraints. Computational Intelligence in Optimization-Applications and Implementations (accepted for publication) (2009)

14. Asuncion, A., Newman, D.: UCI machine learning repository (2007)

15. Demsar, J.: Statistical comparison of classifiers over multiple data sets. Journal of Machine Learning Research 7 (2006) 1–30

16. Kuncheva, L.I.: Combining Pattern Classifiers. Methods and algorithms. John Wiley & Sons (2004)

17. Optiz, D., Maclin, R.: Popular ensemble methods: an empirical study. Journal Of Artificial Intelligence Research **11** (1999) 169–198

18. Dietterich, T.G.: An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. Machine Learning **40**(2) (2000) 139–157

19. Geurts, P., Ernst, D., Wehenkel, L.: Extremely randomized trees. Machine Learning **63** (2006) 3–42

20. Geurts, P., , Ernst, D., Wehenkel, L.: Extremely randomized trees. Machine Learning **36**(1) (2006) 3–42

21. Dietterich, T.G., Bakiri, G.: Solving multiclass learning problems via error-correcting output codes. Journal of Artificial Intelligence Research, **2** (1995) 263–286

22. Breiman, L.: Random forest. Machine Learning **45** (2000) 5–32

23. Martínez-Muñoz, G., Suárez, A.: Switching class labels to generate classification ensembles. Pattern Recognition **38**(10) (2005) 1483–1494

24. Bennett, J., Lanning, S.: The Netflix Prize. KDD Cup and Workshop in conjunction with KDD (2007)

25. Bell, R.M., Koren, Y.: Lessons from the Netflix Prize challenge. SIGKDD Explorations **9**(2) (2007) 75–79

26. Bauer, E., Kohavi, R.: An empirical comparison of voting classiffication algorithms: Bagging, boosting, and variants. Machine Learning **36**(1-2) (1999) 105–139

27. Breiman, L.: Bias, variance, and arcing classifiers. Technical report (1996)

28. Breiman, L.: Arcing classifiers. The Annals of Statistics **26**(3) (1998) 801–849

29. Bühlmann, P.: Bagging, subagging and bragging for improving some prediction algorithms. Recent Advances and Trends in Nonparametric Statistics (2003)

30. Martínez-Muñoz, G., Suárez, A.: Out-of-bag estimation of the optimal sample size in bagging. Pattern Recognition **43** (2010)

31. Hansen, L.K., Salamon, P.: Neural network ensembles. IEEE Transaction on Pattern Analysis and Machine Intelligence **12**(10) (1999)

32. Geman, S., Bienenstock, E., René: Neural networks and the bias/variance dilemma. Neural Compututation **4**(1) (1992) 1–58

33. Brown, G.: Diversity in Neural Network Ensembles. PhD thesis, The University of Birmingham (2004)

34. Quinlan, J.R.: Bagging, boosting, and c4.5. in Proceedings of the 13th National Conference on Artificial Intelligence. AAAI (1996) 725–730

35. Bauer, E., Kohavi, R.: Popular ensemble methods: An empirical study. Journal of Artificial Intelligence Research (11) (1999) 169–198

36. Freund, Y., Schapire, R.E.: A short introduction to boosting. Japonese Society for Artificial Intelligence **14**(5) (1999) 771–780

37. Webb, G.I.: Multiboosting: A technique for combining boosting and wagging. Machine Learning (40) (2000)

38. Friedman, J.H., Hastie, T., Tibshirani, R.: Additive logistic regression: a statistical view of boosting. Annals of Statistics **28** (1998) 2000

39. Friedman, J.H.: Stochastic gradient boosting. Computational Statistics & Data Analysis **38**(4) (2002) 367–378

40. Mease, D., Wyner, A.: Evidence contrary to the statistical view of boosting. Journal of Machine Learning Research **9** (2007) 131–156

41. Dietterich, T.G.: Approximate statistical tests for comparing supervised classification learning algorithms. Neural Computation **10** (1998) 1895–1924

42. Yule, G.U.: On the association of attributes in statistics. Philosophical Transactions of the Royal Society of London (1900) 257–319

43. Martínez-Muñoz, G., Suárez, A.: Pruning in ordered bagging ensembles. In ACM International Conference Proceeding Series **148** (2006) 609–616

44. Hernández-Lobato, D., Martínez-Muñoz, G., Suárez, A.: Pruning in ordered regression bagging ensembles. Proceedings of the IEEE World Congress on Computational Intelligence (2006) 1266–1273

45. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. New York: Chapman & Hall (1984)

46. Han, Q., Ye, Y., Zhang, J.: An improved rounding method and semidefinite programming relaxation for graph partition. Mathematical Programming (2002) 509–535

47. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufman, San Manteo, CA (1993)

48. Goldberg, D.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Weasley, Reading, MA (1989)

49. Zhou, Z.H., Wu, J.X., Tang, W.: Ensembling neural networks: Many could be better than all. Artificial Intelligence **137**(1-2) (2002) 239–263

50. Radcliffe, N.J.: Equivalence class analysis of genetic algorithms. Complex Systems **5**(2) (1991) 183–205

51. Moral-Escudero, R., Hernández-Lobato, D., Suárez, A.: Selection of optimal investment portfolios with cardinality constraints. IEEE Congress on Evolutionary Computation (2006)

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

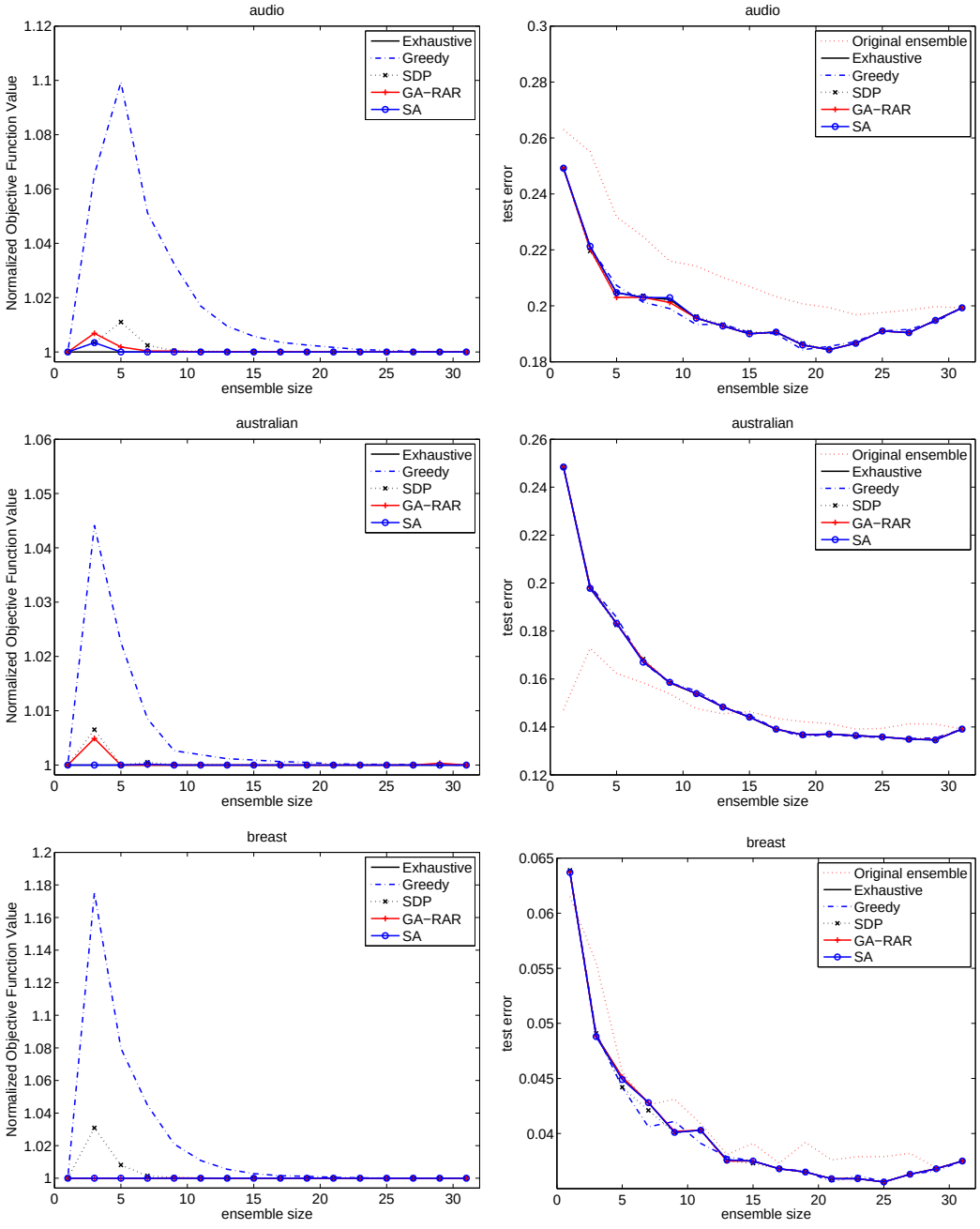

Figure 11: (left) Average value of the objective function defined in Equation (23) that is obtained by each pruning strategy, for each different subensemble. The value of this function is normalized by the optimal value, i.e. the value obtained by the exhaustive search procedure. (right) Average test error of each subensemble obtained by each different pruning strategy. The test error of the original ensemble is also displayed as a reference.

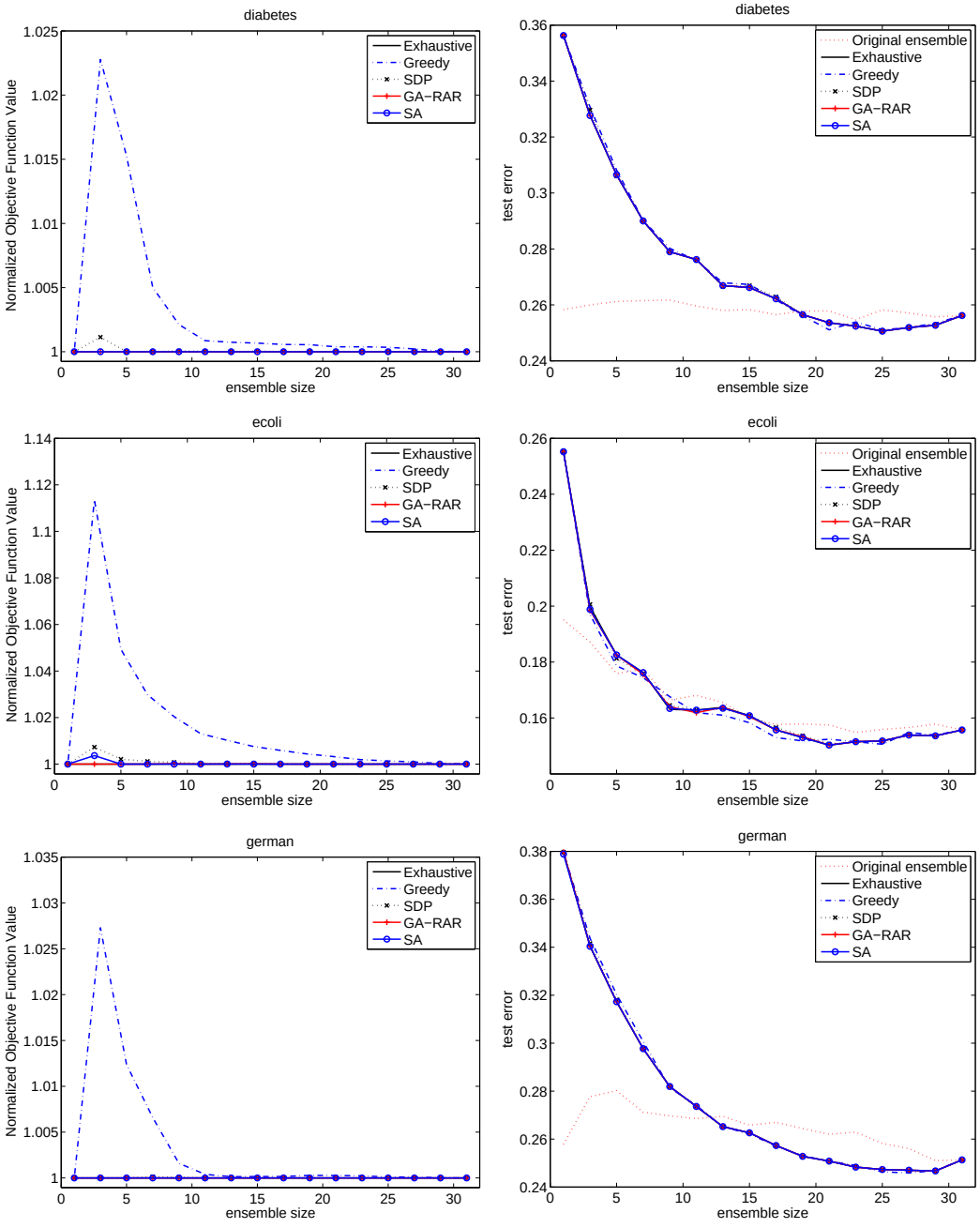

Figure 12: (left) Average value of the objective function defined in Equation (23) that is obtained by each pruning strategy, for each different subensemble. The value of this function is normalized by the optimal value, i.e. the value obtained by the exhaustive search procedure. (right) Average test error of each subensemble obtained by each different pruning strategy. The test error of the original ensemble is also displayed as a reference.

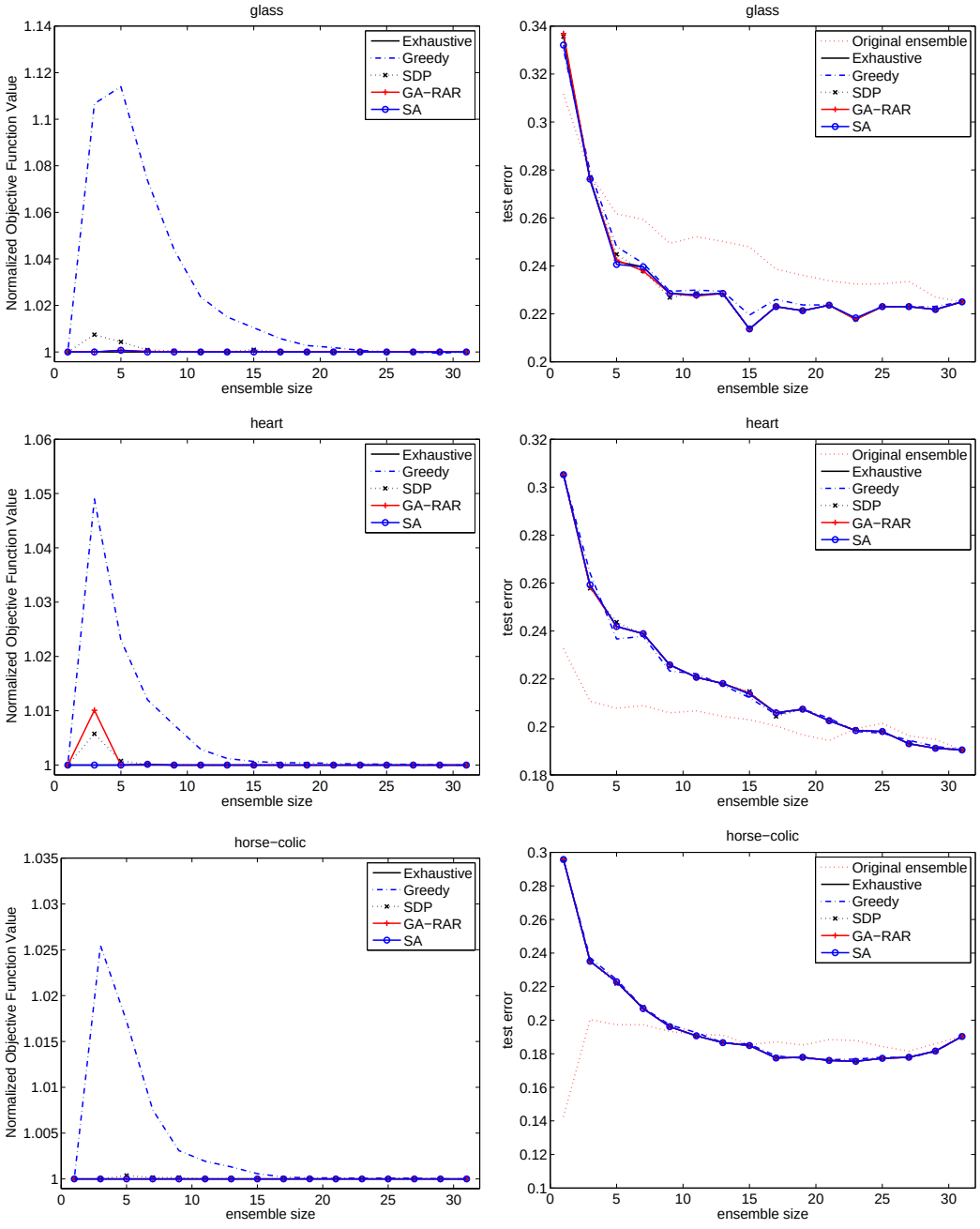

Figure 13: (left) Average value of the objective function defined in Equation (23) that is obtained by each pruning strategy, for each different subensemble. The value of this function is normalized by the optimal value, i.e. the value obtained by the exhaustive search procedure. (right) Average test error of each subensemble obtained by each different pruning strategy. The test error of the original ensemble is also displayed as a reference.

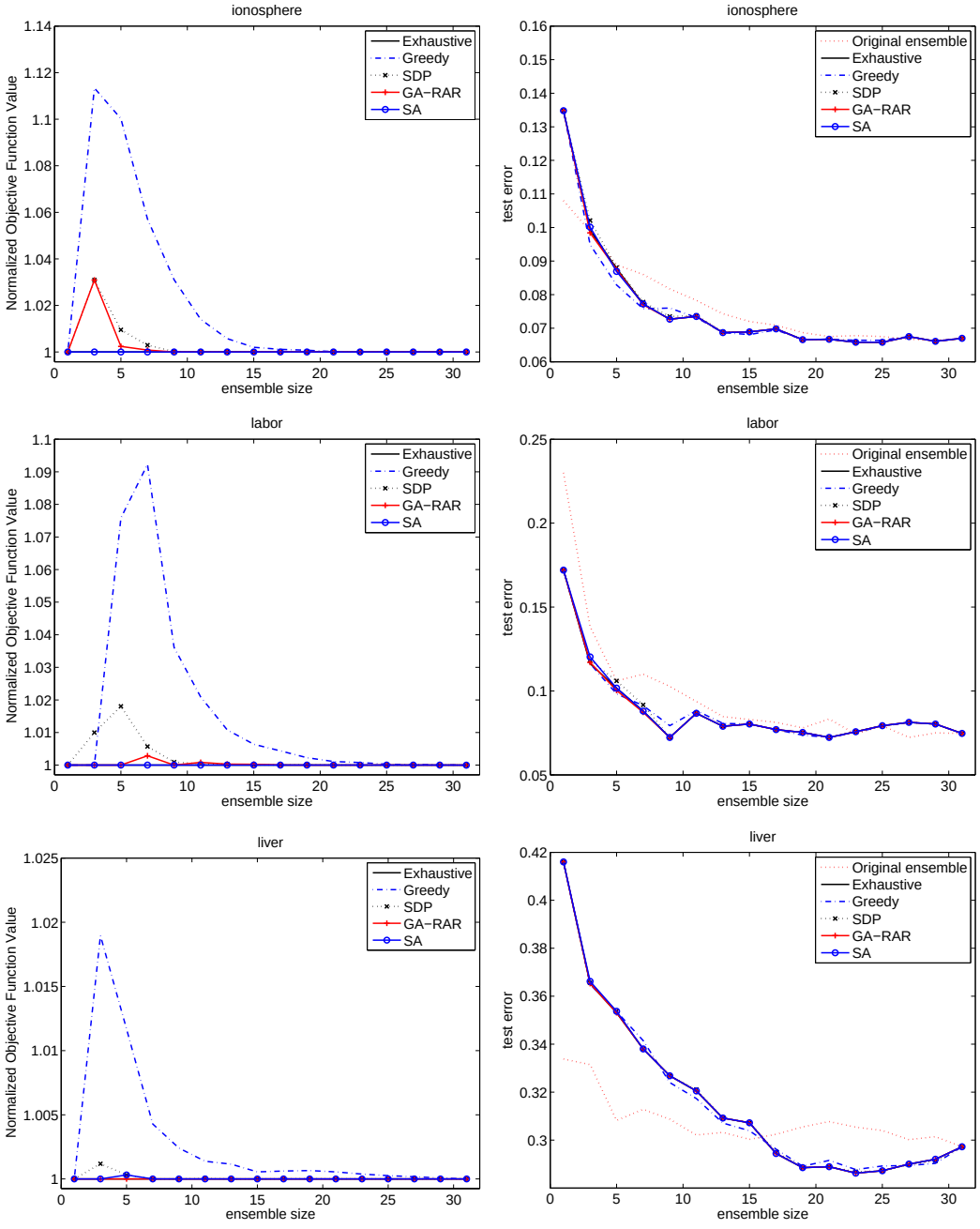

Figure 14: (left) Average value of the objective function defined in Equation (23) that is obtained by each pruning strategy, for each different subensemble. The value of this function is normalized by the optimal value, i.e. the value obtained by the exhaustive search procedure. (right) Average test error of each subensemble obtained by each different pruning strategy. The test error of the original ensemble is also displayed as a reference.

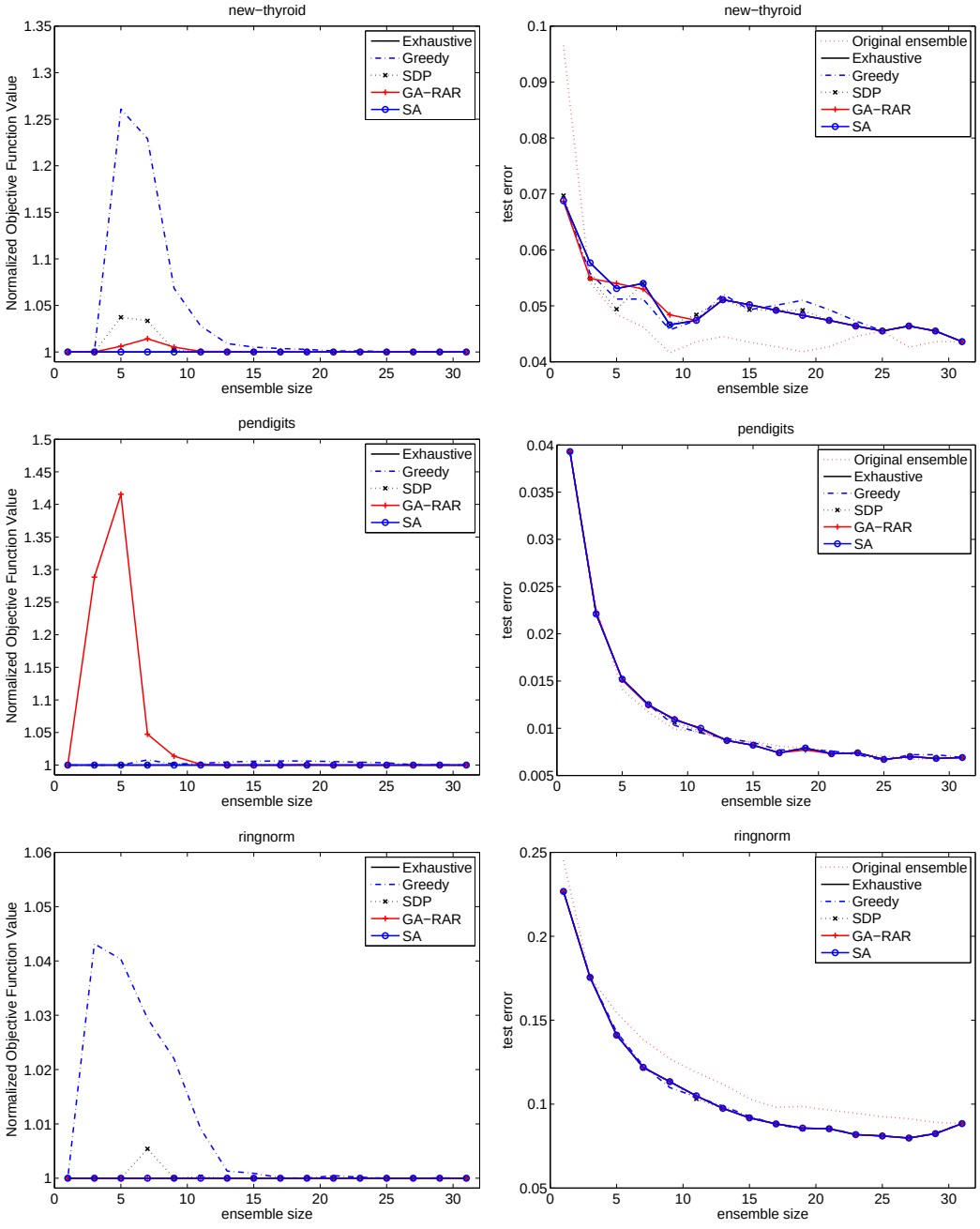

Figure 15: (left) Average value of the objective function defined in Equation (23) that is obtained by each pruning strategy, for each different subensemble. The value of this function is normalized by the optimal value, i.e. the value obtained by the exhaustive search procedure. (right) Average test error of each subensemble obtained by each different pruning strategy. The test error of the original ensemble is also displayed as a reference.

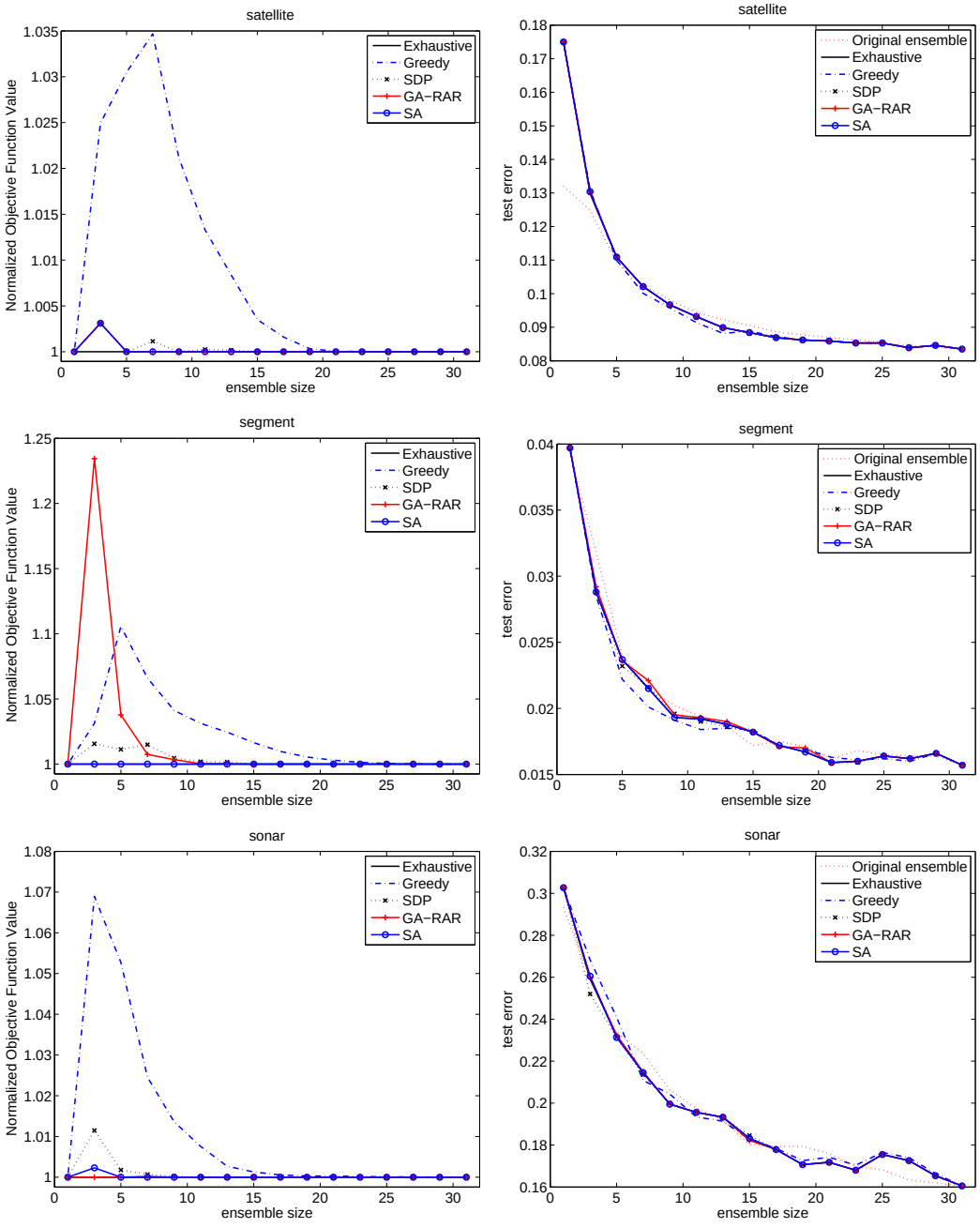

Figure 16: (left) Average value of the objective function defined in Equation (23) that is obtained by each pruning strategy, for each different subensemble. The value of this function is normalized by the optimal value, i.e. the value obtained by the exhaustive search procedure. (right) Average test error of each subensemble obtained by each different pruning strategy. The test error of the original ensemble is also displayed as a reference.

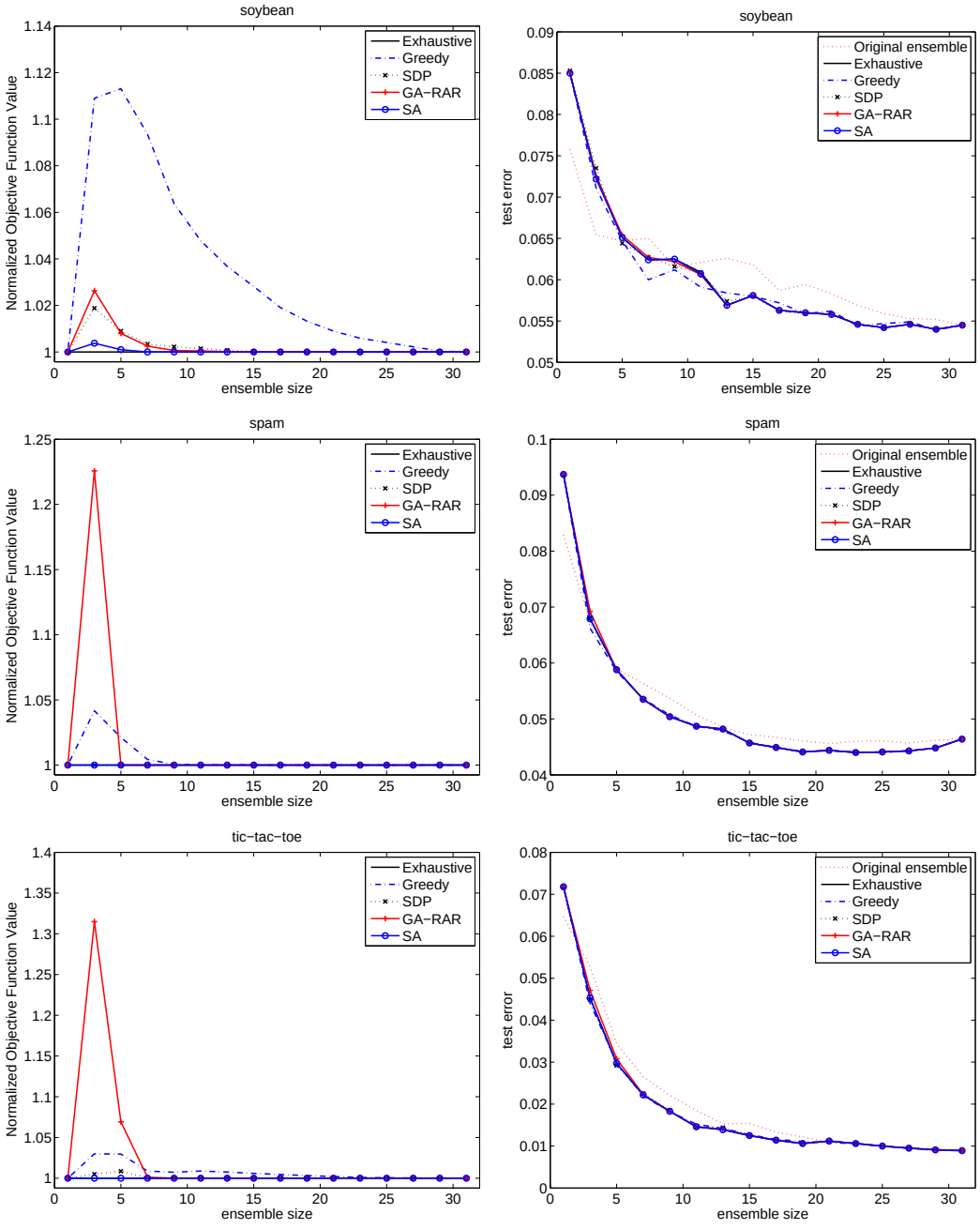

Figure 17: (left) Average value of the objective function defined in Equation (23) that is obtained by each pruning strategy, for each different subensemble. The value of this function is normalized by the optimal value, i.e. the value obtained by the exhaustive search procedure. (right) Average test error of each subensemble obtained by each different pruning strategy. The test error of the original ensemble is also displayed as a reference.

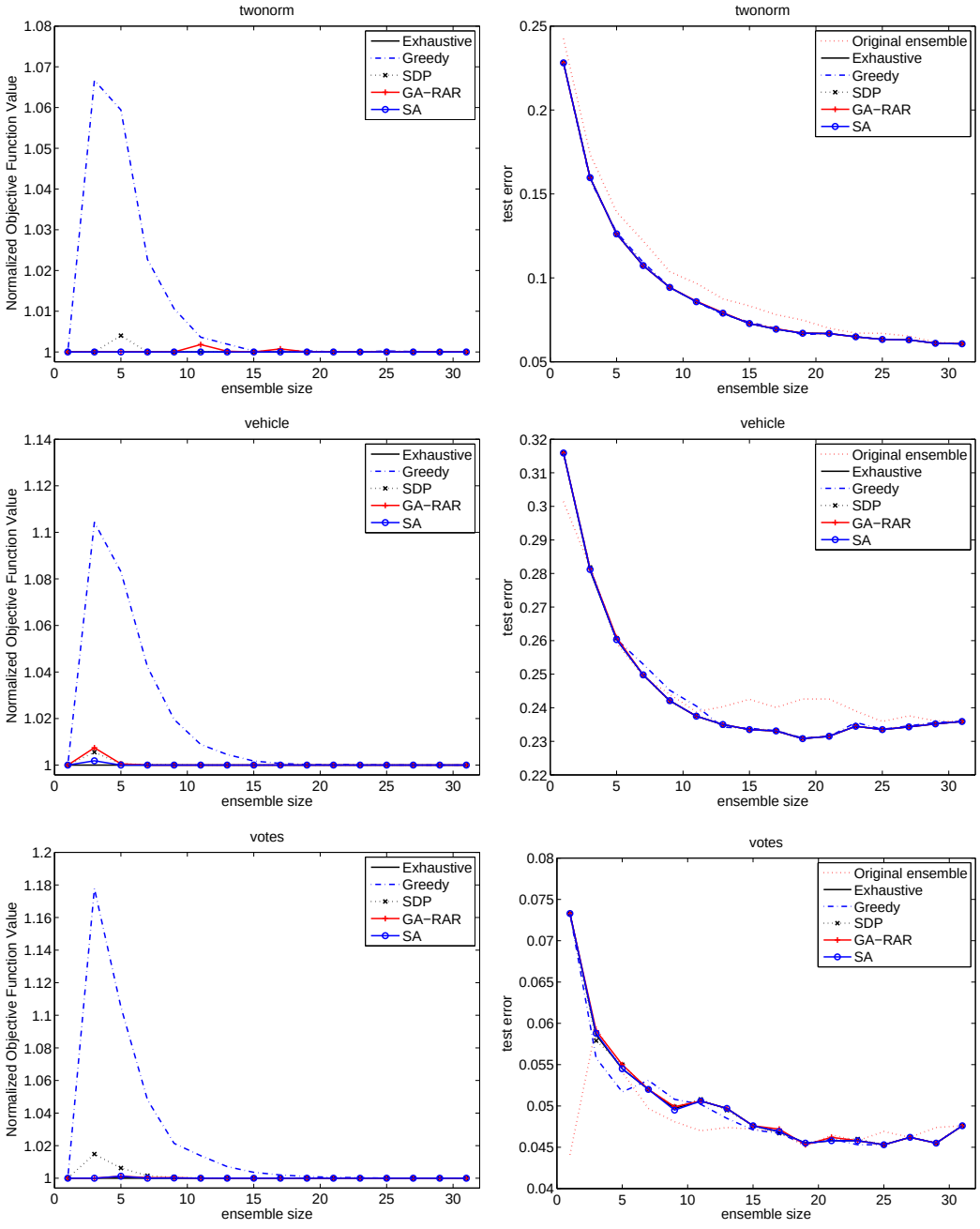

Figure 18: (left) Average value of the objective function defined in Equation (23) that is obtained by each pruning strategy, for each different subensemble. The value of this function is normalized by the optimal value, i.e. the value obtained by the exhaustive search procedure. (right) Average test error of each subensemble obtained by each different pruning strategy. The test error of the original ensemble is also displayed as a reference.

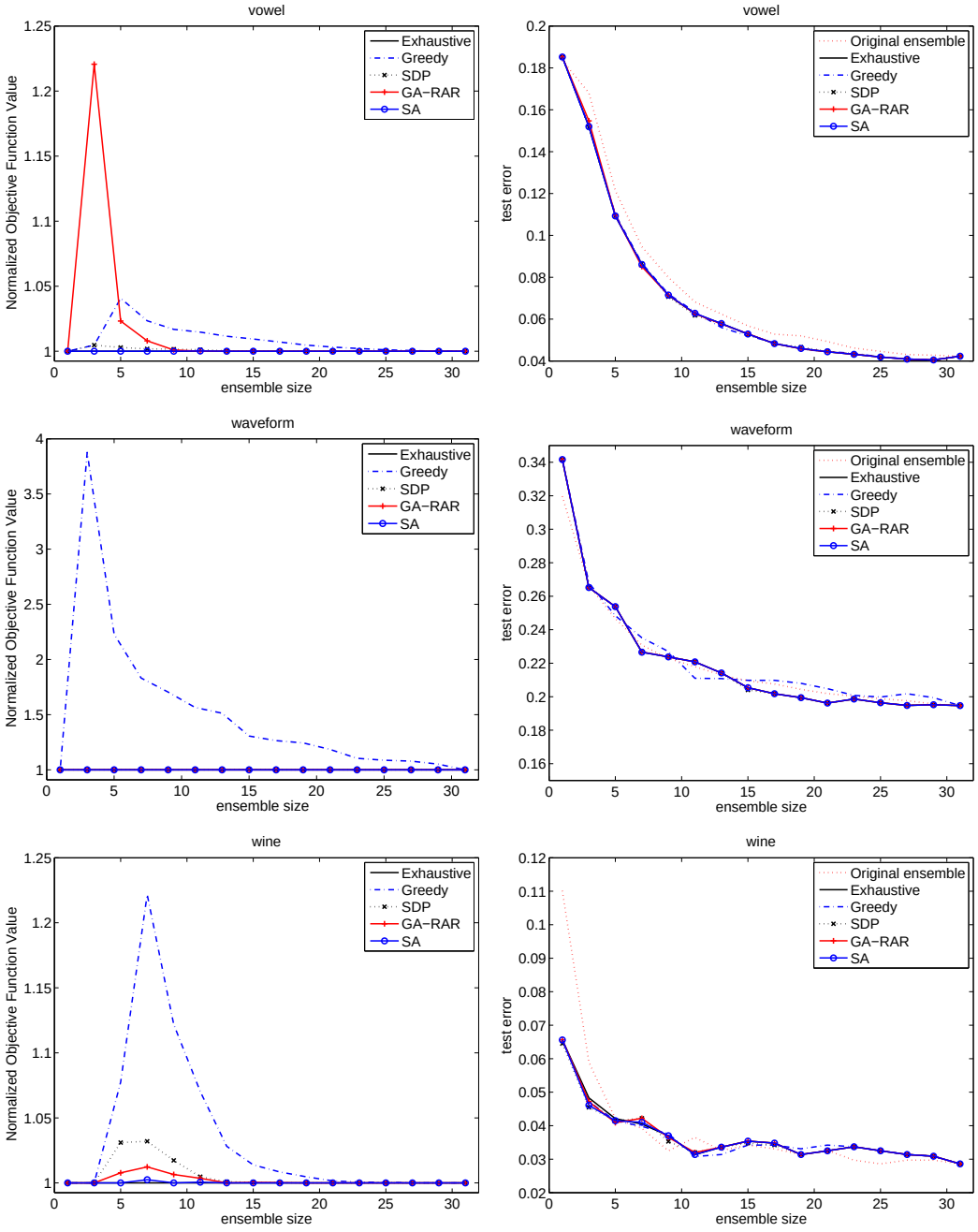

Figure 19: (left) Average value of the objective function defined in Equation (23) that is obtained by each pruning strategy, for each different subensemble. The value of this function is normalized by the optimal value, i.e. the value obtained by the exhaustive search procedure. (right) Average test error of each subensemble obtained by each different pruning strategy. The test error of the original ensemble is also displayed as a reference.

Table 4: Training G value for ensembles of standard pruned CART trees, using $k = 21$. Values for Boosting 101 are scaled dividing by $(101^2/21^2)$ for comparison reasons.

Dataset	Boosting 101	Boosting 21	Greedy	SDP	GA-RAR	SA
audio	86.39	84.14	52.63	30.62	30.62	**30.59**
australian	136.69	122.72	95.42	**68.04**	68.07	**68.04**
breast	92.43	80.22	56.35	32.15	**32.14**	**32.14**
diabetes	163.54	148.67	117.14	**85.57**	**85.57**	**85.57**
ecoli	118.26	115.91	88.35	60.31	60.32	**60.29**
german	154.44	141.73	111.54	**81.30**	**81.30**	**81.30**
glass	89.75	84.68	61.22	37.48	37.46	**37.45**
heart	143.37	128.18	100.79	**73.30**	**73.30**	**73.30**
horse-colic	156.65	128.98	111.78	94.48	94.48	**94.47**
ionosphere	100.02	79.98	57.14	33.99	33.98	**33.97**
labor	92.37	76.15	50.58	24.65	24.68	**24.64**
liver	164.31	148.86	117.73	**86.55**	**86.55**	**86.55**
new-thyroid	84.82	71.62	45.77	19.47	19.47	**19.42**
pendigits	19.06	14.91	13.59	12.19	12.18	**12.16**
ringnorm	109.78	90.16	54.25	53.97	53.96	**53.95**
satellite	57.11	48.70	32.11	32.00	**31.99**	**31.99**
segment	32.75	26.65	13.25	12.83	12.84	**12.76**
sonar	116.99	102.83	54.11	53.81	**53.80**	**53.80**
soybean	55.15	50.21	26.09	24.89	24.86	**24.82**
spam	128.04	85.14	64.04	**64.01**	**64.01**	**64.01**
tic-tac-toe	81.95	61.52	41.99	41.42	41.41	**41.40**
twonorm	97.48	79.24	46.60	46.28	46.28	**46.27**
vehicle	91.93	88.53	44.77	44.34	**44.33**	**44.33**
votes	108.31	94.96	36.96	36.71	36.73	**36.69**
vowel	33.02	28.88	20.87	20.55	20.51	**20.50**
waveform	114.40	103.97	54.11	**53.99**	**53.99**	**53.99**
wine	61.97	51.70	12.90	12.37	12.43	**12.29**

Table 5: Test error rates (average ± standard deviation) for ensembles of standard pruned CART trees, using $k = 21$

Dataset	Boosting 101	Boosting 21	Greedy	SDP	GA-RAR	SA
audio	**17.83**(±**7.87**)	18.50(±7.89)	18.98(±7.47)	19.22(±6.89)	19.34(±7.06)	19.15(±7.06)
australian	**13.97**(±**3.95**)	15.39(±4.12)	14.32(±4.08)	14.55(±4.18)	14.43(±4.23)	14.51(±4.17)
breast	**3.35**(±**1.93**)	3.81(±2.29)	3.50(±2.01)	3.48(±2.06)	3.46(±2.09)	3.45(±2.06)
diabetes	**26.76**(±**3.86**)	27.56(±4.71)	27.61(±4.47)	27.53(±4.61)	27.53(±4.61)	27.53(±4.61)
ecoli	**14.14**(±**4.97**)	15.77(±5.55)	15.95(±5.27)	15.95(±5.41)	15.86(±5.44)	15.92(±5.44)
german	**25.01**(±**3.21**)	27.47(±3.84)	26.52(±3.55)	26.45(±3.52)	26.45(±3.52)	26.45(±3.52)
glass	**21.58**(±**8.54**)	22.92(±8.60)	22.32(±9.48)	22.12(±9.07)	21.85(±8.91)	22.03(±8.99)
heart	**20.85**(±**7.57**)	21.52(±7.70)	22.56(±7.61)	22.22(±7.61)	22.26(±7.56)	22.19(±7.56)
horse-colic	18.65(±6.14)	19.63(±6.46)	**18.08**(±**6.00**)	18.35(±6.01)	18.19(±6.01)	18.24(±5.95)
ionosphere	6.41(±3.75)	6.73(±4.30)	6.41(±3.82)	6.46(±3.71)	6.46(±3.69)	**6.38**(±**3.69**)
labor	**6.87**(±**10.29**)	8.87(±10.81)	**6.87**(±**9.89**)	7.57(±10.55)	7.37(±10.50)	7.57(±10.55)
liver	**28.51**(±**6.90**)	32.05(±7.05)	31.41(±7.84)	31.81(±8.04)	31.81(±8.04)	31.81(±8.04)
new-thyroid	4.74(±4.22)	**4.61**(±**4.00**)	4.74(±4.22)	5.07(±4.29)	4.98(±4.13)	5.12(±4.20)
pendigits	**0.57**(±**0.23**)	0.76(±0.24)	0.72(±0.27)	0.74(±0.25)	0.72(±0.25)	0.72(±0.25)
ringnorm	**5.87**(±**0.70**)	8.95(±1.05)	8.25(±1.08)	8.23(±1.10)	8.23(±1.12)	8.22(±1.09)
satellite	**7.59**(±**1.01**)	8.71(±1.07)	8.24(±0.40)	8.02(±0.51)	8.04(±0.51)	8.03(±0.52)
segment	**1.49**(±**0.77**)	1.66(±0.76)	1.61(±0.72)	1.65(±0.80)	1.61(±0.77)	1.60(±0.76)
sonar	**13.32**(±**7.98**)	17.69(±8.29)	17.07(±8.30)	16.74(±8.31)	16.75(±8.54)	16.74(±8.50)
soybean	5.62(±2.57)	5.75(±2.62)	**5.53**(±**2.63**)	5.80(±2.44)	5.59(±2.49)	5.59(±2.47)
spam	4.61(±1.12)	4.81(±1.15)	**4.48**(±**1.18**)	4.46(±1.15)	4.46(±1.16)	4.46(±1.16)
tic-tac-toe	**0.73**(±**0.94**)	1.18(±1.14)	0.93(±0.97)	0.98(±0.97)	0.96(±0.93)	0.97(±0.95)
twonorm	**4.46**(±**0.42**)	7.12(±0.80)	6.53(±0.56)	6.53(±0.57)	6.55(±0.56)	6.55(±0.57)
vehicle	**22.40**(±**3.50**)	23.57(±4.25)	22.78(±3.95)	23.10(±3.75)	23.12(±3.81)	23.18(±3.80)
votes	**4.53**(±**3.00**)	4.74(±3.07)	4.87(±3.13)	4.85(±2.99)	4.90(±3.02)	4.87(±3.01)
vowel	**3.30**(±**1.93**)	5.00(±2.11)	4.47(±2.29)	4.29(±2.15)	4.37(±2.17)	4.34(±2.10)
wine	3.36(±4.21)	3.31(±4.08)	3.59(±4.33)	**3.25**(±**4.14**)	3.36(±4.13)	3.36(±4.13)
waveform	**17.95**(±**0.79**)	20.11(±0.97)	19.68(±0.86)	19.69(±0.86)	19.69(±0.86)	19.69(±0.86)

Table 6: Training G value for ensembles of standard pruned CART trees, using $k = 31$. Values for Boosting 101 are scaled dividing by $(101^2/31^2)$.

Dataset	Boosting 101	Boosting 31	Greedy	SDP	GA-RAR	SA
audio	188.26	186.38	81.38	80.97	80.97	**80.95**
australian	297.87	275.60	172.58	**172.55**	172.58	**172.55**
breast	201.43	182.93	87.32	87.11	87.11	**87.10**
diabetes	356.38	333.81	222.66	**222.61**	**222.61**	**222.61**
ecoli	257.70	253.59	154.13	153.58	153.59	**153.56**
german	336.54	317.45	204.37	**204.36**	**204.36**	**204.36**
glass	195.58	189.51	97.48	97.19	**97.18**	**97.18**
heart	312.42	289.08	187.30	**187.24**	187.25	**187.24**
horse-colic	341.36	298.89	232.91	**232.84**	232.87	**232.84**
ionosphere	217.96	188.08	90.30	**90.15**	**90.15**	**90.15**
labor	201.28	178.67	80.72	80.43	80.43	**80.42**
liver	358.06	335.56	224.44	**224.42**	**224.42**	**224.42**
new-thyroid	184.83	167.35	66.18	65.83	65.82	**65.81**
pendigits	41.54	35.48	29.51	29.29	29.28	**29.26**
ringnorm	239.23	212.40	134.23	**134.04**	**134.04**	**134.04**
satellite	124.44	114.27	76.80	76.64	**76.63**	**76.63**
segment	71.37	63.54	34.02	33.51	33.49	**33.44**
sonar	254.93	234.79	136.92	136.81	**136.80**	**136.80**
soybean	120.19	113.51	64.44	62.95	62.94	**62.89**
spam	279.02	217.14	167.01	**166.99**	**166.99**	**166.99**
tic-tac-toe	178.58	148.75	103.14	102.53	102.53	**102.52**
twonorm	212.42	186.36	116.43	116.22	116.22	**116.21**
vehicle	200.32	195.34	109.97	**109.69**	**109.69**	**109.69**
votes	236.02	214.60	103.79	**103.59**	103.63	**103.59**
vowel	71.95	66.35	49.80	49.33	49.30	**49.28**
waveform	135.04	119.66	43.47	43.08	43.12	**43.02**
wine	249.29	233.95	138.15	**138.12**	138.18	**138.12**

Table 7: Test error rates (average ± standard deviation) for ensembles of standard pruned CART trees, using $k = 31$

Dataset	Boosting 101	Boosting 31	Greedy	SDP	GA-RAR	SA
audio	**17.83(±7.87)**	18.50(±7.48)	18.93(±8.04)	18.66(±8.11)	18.75(±8.00)	18.71(±7.97)
australian	13.97(±3.95)	14.86(±4.09)	**13.65(±3.75)**	13.78(±3.69)	13.68(±3.74)	13.72(±3.69)
breast	**3.35(±1.93)**	3.82(±2.18)	3.40(±1.90)	3.48(±1.93)	3.49(±1.94)	3.48(±1.94)
diabetes	26.76(±3.86)	27.24(±4.90)	26.49(±4.02)	26.48(±3.96)	**26.42(±4.01)**	26.43(±4.00)
ecoli	**14.14(±4.97)**	15.21(±5.28)	15.41(±5.36)	15.54(±5.34)	15.48(±5.48)	15.60(±5.38)
german	**25.01(±3.21)**	26.47(±3.75)	25.41(±2.99)	25.44(±3.01)	25.43(±3.00)	25.43(±3.00)
glass	21.58(±8.54)	21.84(±8.65)	21.62(±9.13)	**21.48(±8.94)**	21.53(±8.86)	21.49(±8.87)
heart	**20.85(±7.57)**	22.07(±8.07)	21.41(±7.66)	21.44(±7.71)	21.44(±7.74)	21.44(±7.71)
horse-colic	18.65(±6.14)	19.23(±6.49)	**17.70(±5.61)**	17.72(±5.57)	17.78(±5.52)	17.72(±5.57)
ionosphere	6.41(±3.75)	6.47(±3.87)	6.15(±3.87)	**6.09(±3.76)**	6.12(±3.77)	6.12(±3.77)
labor	6.87(±10.29)	7.57(±10.55)	6.67(±9.53)	7.03(±9.91)	**6.50(±9.50)**	6.67(±9.53)
liver	**28.51(±6.90)**	30.95(±7.21)	30.34(±6.95)	30.22(±7.09)	30.25(±7.04)	30.22(±7.09)
new-thyroid	4.74(±4.22)	**4.71(±4.06)**	4.89(±4.24)	4.84(±4.16)	4.88(±4.19)	4.88(±4.19)
pendigits	**0.57(±0.23)**	0.69(±0.26)	0.66(±0.24)	0.66(±0.24)	0.66(±0.25)	0.66(±0.26)
ringnorm	**5.87(±0.70)**	7.84(±0.94)	7.35(±0.93)	7.33(±0.95)	7.34(±0.95)	7.33(±0.96)
satellite	**7.59(±1.01)**	8.29(±1.11)	8.03(±0.93)	8.08(±0.93)	8.07(±0.94)	8.07(±0.93)
segment	**1.49(±0.77)**	1.55(±0.78)	1.55(±0.69)	1.56(±0.74)	1.57(±0.75)	1.56(±0.72)
sonar	**13.32(±7.98)**	16.50(±7.91)	15.15(±8.12)	15.05(±8.42)	15.00(±8.31)	15.00(±8.31)
soybean	5.62(±2.57)	5.55(±2.61)	**5.48(±2.49)**	5.64(±2.46)	5.55(±2.43)	5.56(±2.43)
spam	4.61(±1.12)	4.75(±1.14)	4.44(±1.15)	4.44(±1.14)	**4.43(±1.13)**	4.44(±1.14)
tic-tac-toe	**0.73(±0.94)**	0.91(±1.00)	0.89(±0.97)	0.87(±0.93)	0.87(±0.93)	0.87(±0.93)
twonorm	**4.46(±0.42)**	6.07(±0.62)	5.70(±0.54)	5.69(±0.54)	5.68(±0.55)	5.68(±0.55)
vehicle	**22.40(±3.50)**	23.02(±3.84)	22.46(±3.63)	22.48(±3.70)	22.52(±3.73)	22.52(±3.73)
votes	**4.53(±3.00)**	4.99(±3.22)	4.71(±3.00)	4.64(±3.04)	4.66(±3.03)	4.64(±3.04)
vowel	**3.30(±1.93)**	4.14(±1.88)	3.80(±1.96)	3.91(±2.06)	3.93(±2.16)	3.93(±2.17)
wine	3.36(±4.21)	**3.31(±4.16)**	3.37(±4.15)	**3.31(±4.08)**	**3.31(±4.08)**	3.37(±4.07)
waveform	**17.95(±0.79)**	19.31(±0.84)	18.95(±0.72)	18.97(±0.75)	18.97(±0.75)	18.97(±0.75)

Table 8: Training G value for ensembles of standard pruned CART trees, using $k = 51$. Values for Boosting 101 are scaled dividing by $(101^2/51^2)$.

Dataset	Boosting 101	Boosting 51	Greedy	SDP	GA-RAR	SA
audio	509.53	505.32	293.08	**292.88**	**292.88**	**292.88**
australian	806.21	772.38	581.65	**581.62**	581.66	**581.62**
breast	545.17	515.58	315.16	**315.07**	**315.07**	**315.07**
diabetes	964.56	931.98	747.39	**747.30**	**747.30**	**747.30**
ecoli	697.49	690.33	510.38	509.69	509.70	**509.68**
german	910.87	885.97	682.69	**682.65**	**682.65**	**682.65**
glass	529.34	523.99	336.88	**336.78**	**336.78**	**336.78**
heart	845.58	814.57	627.93	627.88	627.88	**627.87**
horse-colic	923.90	861.72	727.00	**726.97**	726.98	**726.97**
ionosphere	589.91	553.20	322.80	**322.79**	**322.79**	**322.79**
labor	544.79	518.79	325.66	325.55	325.58	**325.54**
liver	969.11	941.29	757.94	**757.89**	**757.89**	**757.89**
new-thyroid	500.24	479.67	280.05	**279.92**	279.93	**279.92**
pendigits	112.43	104.62	88.87	88.52	**88.47**	**88.47**
ringnorm	647.49	612.28	435.78	**435.75**	**435.75**	**435.75**
satellite	336.81	321.18	235.80	**235.75**	235.76	**235.75**
segment	193.16	183.11	113.02	112.57	112.57	**112.56**
sonar	689.98	668.98	464.45	**464.40**	**464.40**	**464.40**
soybean	325.29	316.34	210.25	208.74	208.74	**208.73**
spam	755.18	679.58	547.86	**547.84**	**547.84**	**547.84**
tic-tac-toe	483.34	444.79	327.07	**326.61**	**326.61**	**326.61**
twonorm	574.94	538.94	385.25	**385.21**	**385.21**	385.36
vehicle	542.17	537.27	365.54	**365.44**	**365.44**	**365.44**
votes	638.79	608.15	392.22	**392.16**	392.22	**392.16**
vowel	194.74	188.10	150.95	150.53	150.52	**150.51**
waveform	365.50	345.07	179.03	178.86	178.87	**178.84**
wine	674.71	654.96	475.69	475.66	475.66	**475.21**

Table 9: Test error rates (average ± standard deviation) for ensembles of standard pruned CART trees, using $k = 51$

Dataset	Boosting 101	Boosting 51	Greedy	SDP	GA-RAR	SA
australian	13.97(±3.95)	14.32(±4.17)	**13.23(±4.00)**	13.28(±3.99)	13.30(±3.89)	13.28(±3.98)
breast	3.35(±1.93)	3.45(±2.08)	3.32(±1.94)	**3.30(±1.95)**	**3.30(±1.95)**	3.32(±1.94)
diabetes	26.76(±3.86)	27.29(±5.08)	**25.44(±3.96)**	25.48(±3.89)	25.53(±3.90)	25.50(±3.87)
ecoli	**14.14(±4.97)**	14.73(±4.91)	14.47(±5.05)	14.50(±5.18)	14.44(±5.19)	14.53(±5.22)
german	25.01(±3.21)	26.13(±3.69)	**24.21(±3.18)**	**24.21(±3.26)**	**24.21(±3.26)**	**24.21(±3.26)**
glass	**21.58(±8.54)**	21.94(±8.96)	22.18(±8.93)	22.27(±9.03)	22.31(±9.08)	22.27(±9.03)
heart	20.85(±7.57)	21.15(±7.88)	**19.81(±7.51)**	20.07(±7.50)	20.00(±7.44)	20.07(±7.50)
horse-colic	18.65(±6.14)	18.95(±6.18)	**17.26(±5.66)**	17.29(±5.80)	17.29(±5.80)	17.29(±5.80)
ionosphere	6.41(±3.75)	6.58(±4.07)	**6.24(±3.71)**	6.27(±3.72)	6.27(±3.72)	6.27(±3.72)
labor	6.87(±10.29)	6.90(±10.60)	5.93(±9.30)	5.73(±9.21)	6.13(±9.39)	**5.57(±9.16)**
liver	28.51(±6.90)	29.97(±7.15)	27.56(±6.66)	27.59(±6.59)	27.51(±6.60)	**27.50(±6.62)**
new-thyroid	**4.74(±4.22)**	5.02(±4.56)	4.79(±4.00)	4.84(±4.02)	4.89(±4.11)	4.84(±4.02)
pendigits	**0.57(±0.23)**	0.64(±0.24)	0.60(±0.24)	0.59(±0.24)	0.59(±0.24)	0.59(±0.24)
ringnorm	**5.87(±0.70)**	6.77(±0.76)	6.45(±0.82)	6.44(±0.83)	6.44(±0.83)	6.44(±0.83)
satellite	**7.59(±1.01)**	7.96(±1.02)	7.73(±0.96)	7.73(±0.93)	7.73(±0.93)	7.73(±0.93)
segment	**1.49(±0.77)**	1.50(±0.74)	1.51(±0.71)	1.51(±0.72)	1.50(±0.72)	1.50(±0.72)
sonar	**13.32(±7.98)**	15.64(±8.40)	14.67(±8.11)	14.81(±8.31)	14.81(±8.31)	14.81(±8.31)
soybean	5.62(±2.57)	5.49(±2.51)	**5.37(±2.43)**	5.43(±2.50)	5.45(±2.50)	5.46(±2.51)
spam	4.61(±1.12)	4.65(±1.12)	4.57(±1.09)	**4.56(±1.08)**	4.57(±1.08)	4.57(±1.08)
tic-tac-toe	**0.73(±0.94)**	0.83(±0.98)	0.78(±0.91)	0.78(±0.91)	0.78(±0.91)	0.78(±0.91)
twonorm	**4.46(±0.42)**	5.14(±0.46)	4.97(±0.46)	4.97(±0.46)	4.97(±0.46)	4.96(±0.45)
vehicle	22.40(±3.50)	23.02(±3.80)	**22.06(±3.80)**	22.08(±3.77)	22.08(±3.77)	22.08(±3.77)
votes	4.53(±3.00)	4.76(±2.83)	4.37(±2.88)	**4.35(±2.88)**	**4.35(±2.88)**	**4.35(±2.88)**
vowel	**3.30(±1.93)**	3.72(±1.92)	3.46(±1.94)	3.51(±1.93)	3.49(±1.99)	3.47(±1.96)
wine	3.36(±4.21)	3.47(±4.29)	3.19(±3.93)	**3.08(±3.86)**	3.14(±4.01)	3.14(±3.93)
waveform	**17.95(±0.79)**	18.63(±0.81)	18.20(±0.81)	18.20(±0.81)	18.20(±0.81)	18.22(±0.80)

A Details of weighted majority vote examples

This section provides the derivation of the probabilities of the examples given in the weighted majority vote section (Section 3.2).

The first example shows the probability of correct classification of an ensemble composed by 5 base classifier whose probabilities are (0.9 0.9 0.6 0.6 0.6). Assuming binary classification problems, the ensemble assigns the class to the instance voted by at least 3 base classifiers. Let be $P(i)$ the probability of exactly i classifiers being correct:

$$\begin{aligned} P_{maj} = & P(\text{exactly 3 classifiers are correct}) + \\ & P(\text{exactly 4 classifiers are correct}) + \\ & P(\text{exactly 5 classifiers are correct}) \\ & P(3) + P(4) + P(5) \end{aligned}$$

$$\begin{aligned} P(3) = & 3 \cdot 0.9^2 \cdot 0.4^2 \cdot 0.6 + \\ & 0.6^3 \cdot 0.1^2 + \\ & 6 \cdot 0.6^2 \cdot 0.4 \cdot 0.9 \cdot 0.1 \end{aligned}$$

$$\begin{aligned} P(4) = & 2 \cdot 0.9^2 \cdot 0.1 \cdot 0.6^3 + \\ & 3 \cdot 0.9^2 \cdot 0.6^2 \cdot 0.4 \end{aligned}$$

$$P(5) = 0.9^2 \cdot 0.6^3$$

Grouping terms:

$$\begin{aligned} P_{maj} = & 3 \cdot 0.9^2 \cdot 0.4 \cdot 0.6 \cdot (0.4 + 0.6) + \\ & 0.6^3 \cdot (0.1^2 + 2 \cdot 0.9 \cdot 0.1 + 0.9^2) + \\ & 6 \cdot 0.6^2 \cdot 0.4 \cdot 0.9 \cdot 0.1 \end{aligned}$$

And simplifying:

$$P_{maj} = 3 \cdot 0.9^2 \cdot 0.4 \cdot 0.6 + 0.6^3 + 6 \cdot 0.9 \cdot 0.1 \cdot 0.6^2 \cdot 0.4 \approx 0.877$$

The second example shows that assigning weights to the votes of the classifiers improves the probability of correct classification. If weights are assigned to the base classifiers, the rule to assign a class label to a instance changes. In binary classification problems, a class c_k is assigned when the sum

of the weights of the base classifiers that vote class c_k overpasses 0.5. Assume an ensemble whose base classifiers have a probability of correct classification of $(0.9\,0.9\,0.6\,0.6\,0.6)$ and weights $(\frac{1}{3}\ \frac{1}{3}\ \frac{1}{9}\ \frac{1}{9}\ \frac{1}{9})$. Then, class c_k is assigned when the two classifiers with probability of correct classification $p = 0.9$ agree $(\frac{1}{3} + \frac{1}{3} > 0.5)$, or when one classifier with $p = 0.9$ and two with $p = 0.6$ agree $(\frac{1}{3} + \frac{1}{6} + \frac{1}{9} > 0.5)$.

$$
\begin{aligned}
P^w_{maj} = & P(\text{2 classifiers with p = 0.9 are correct}) + \\
& P(\text{1 classifier with p = 0.9 and 2 with p = 0.6 are correct})
\end{aligned}
$$

$$
\begin{aligned}
= & 0.9^2 \cdot 0.6^3 + \\
& 0.9^2 \cdot 0.4^3 + \\
& 3 \cdot 0.9^2 \cdot 0.4 \cdot 0.6^2 + \\
& 3 \cdot 0.9^2 \cdot 0.4^2 \cdot 0.6 + \\
& \\
& 2 \cdot 0.9 \cdot 0.1 \cdot 3 \cdot 0.6^2 \cdot 0.4 + \\
& 2 \cdot 0.9 \cdot 0.1 \cdot 0.6^3
\end{aligned}
$$

$$
\begin{aligned}
= & 0.9^2 \cdot (0.6^3 + 0.4^3 + 3 \cdot 0.4 \cdot 0.6^2 + 3 \cdot 0.4^2 \cdot 0.6) + \\
& 2 \cdot 3 \cdot 0.9 \cdot 0.1 \cdot 0.6^2 \cdot 0.4 + \\
& 2 \cdot 0.9 \cdot 0.1 \cdot 0.6^3
\end{aligned}
$$

$$
P^w_{maj} = 0.9^2 + 2 \cdot 3 \cdot 0.9 \cdot 0.1 \cdot 0.6^2 \cdot 0.4 + 2 \cdot 0.9 \cdot 0.1 \cdot 0.6^2 \approx 0.927
$$