Artificial Intelligence 174 (2010) 410–441

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Democratic instance selection: A linear complexity instance selection algorithm based on classifier ensemble concepts ☆

César García-Osorio [a], Aida de Haro-García [b], Nicolás García-Pedrajas [b,*]

[a] *Department of Civil Engineering of the University of Burgos, Spain*
[b] *Department of Computing and Numerical Analysis of the University of Córdoba, Spain*

ARTICLE INFO

Article history:
Received 4 November 2008
Received in revised form 18 January 2010
Accepted 21 January 2010
Available online 1 February 2010

Keywords:
Instance selection
Instance-based learning
Ensembles
Huge problems

ABSTRACT

Instance selection is becoming increasingly relevant due to the huge amount of data that is constantly being produced in many fields of research. Although current algorithms are useful for fairly large datasets, scaling problems are found when the number of instances is in the hundreds of thousands or millions. When we face huge problems, scalability becomes an issue, and most algorithms are not applicable.

Thus, paradoxically, instance selection algorithms are for the most part impracticable for the same problems that would benefit most from their use. This paper presents a way of avoiding this difficulty using several rounds of instance selection on subsets of the original dataset. These rounds are combined using a voting scheme to allow good performance in terms of testing error and storage reduction, while the execution time of the process is significantly reduced. The method is particularly efficient when we use instance selection algorithms that are high in computational cost. The proposed approach shares the philosophy underlying the construction of ensembles of classifiers. In an ensemble, several *weak* learners are combined to form a strong classifier; in our method several *weak* (in the sense that they are applied to subsets of the data) instance selection algorithms are combined to produce a strong and fast instance selection method.

An extensive comparison of 30 medium and large datasets from the UCI Machine Learning Repository using 3 different classifiers shows the usefulness of our method. Additionally, the method is applied to 5 huge datasets (from three hundred thousand to more than a million instances) with good results and fast execution time.

1. Introduction

The overwhelming amount of data that is available nowadays [1] in any field of research poses new problems for data mining and knowledge discovery methods. This huge amount of data makes most of the existing algorithms inapplicable to many real-world problems. Two approaches have been used to deal with this problem: scaling up data mining algorithms [2] and data reduction. However, scaling up a certain algorithm is not always feasible. On the other hand, data reduction consists of removing from the data missing, redundant and/or erroneous data to get a tractable amount of data. One common method for data reduction is instance selection.

☆ This work was supported in part by the Project TIN2008-03151 of the Spanish Ministry of Education and Science.

* Corresponding author.
E-mail addresses: cgosorio@ubu.es (C. García-Osorio), adeharo@uco.es (A. de Haro-García), npedrajas@uco.es (N. García-Pedrajas).
URL: http://cibrg.org (N. García-Pedrajas).

0004-3702/$ – see front matter
doi:10.1016/j.artint.2010.01.001

Instance selection [3] consists of choosing a subset of the total available data to achieve the original purpose of the data mining application as if the whole data were being used. Different variants of instance selection exist. We can distinguish two main models [4]: instance selection as a method for prototype selection for algorithms based on prototypes (such as *k*-nearest neighbors) and instance selection for obtaining the training set for a learning algorithm (such as decision trees or neural networks).

The problem of instance selection for instance based learning can be defined as "the isolation of the smallest set of instances that enable us to predict the class of a query instance with the same (or higher) accuracy than the original set" [5].

Many widely used instance selection algorithms are, at least, $O(n^2)$, n being the number of instances [6]. Although methods for scaling up these learning algorithms have been proposed [7], for many algorithms either these methods are not applicable or their application is troublesome. For huge problems, with hundreds of thousands or even millions of instances, most instance selection methods are not applicable. One natural way of scaling up a certain algorithm is dividing the original problem into several easier subproblems and applying the algorithm separately to each subproblem. In this way we might scale up instance selection by dividing the original dataset into several disjoint subsets and performing the instance selection process separately on each subset. However, this method does not work well, as the application of the algorithm to a subset suffers from the partial knowledge it has of the dataset. Instance selection algorithms must evaluate the relevance of each instance to decide whether to remove it. To evaluate the relevance of an instance, the algorithm needs to know the whole dataset, because that relevance depends on all the other instances. Thus, direct application of instance selection to subsets of the original dataset does not yield a good performance.

In this paper we propose a methodology for using this basic idea of applying the instance selection algorithm to subsets of the original dataset in a way that allows a performance close to the application of the algorithm to the whole dataset, while retaining the advantages of a smaller subset. The underlying idea is based upon the following premises:

1. As stated above, a promising way of scaling up instance selection algorithms is using smaller subsets. A simple way of doing that is partitioning the dataset into disjoint subsets and applying the instance selection algorithm to each subset separately.
2. The above solution does not perform well, as each subset is only a partial view of the original dataset. In this way, important instances may be removed and superfluous instances may be kept. In the same sense that we talk of "weak learners" in a classifier ensemble construction framework, we can consider an instance selection algorithm applied to a subset of the whole dataset as a "*weak* instance selection algorithm."
3. Following the philosophy of classifier ensembles we conduct several rounds of weak instance selection algorithms and combine them using a voting scheme. Therefore, our approach is called *democratic* instance selection, and can be considered a form of extending classifier ensemble philosophy to instance selection.

Democratic instance selection is thus based on repeating several rounds of a fast instance selection process. Each round on its own would not be able to achieve a good performance. However, the combination of several rounds using a voting scheme is able to match the performance of an instance selection algorithm applied to the whole dataset with a large reduction in the time of the algorithm. In a different setup from the case of ensembles of classifiers, we can consider our method a form of "ensembling" instance selection. In classification, several weak learners are combined into an ensemble which is able to improve the performance of any of the weak learners alone [8]. In our method, the instance selection algorithm applied to a partition into disjoint subsets of the original dataset can be considered a *weak instance selector*, as it has a partial view of the dataset. The combination of these weak selectors using a voting scheme is similar to the combination of different learners in an ensemble.

The main advantage of our method is that as the instance selection algorithm is applied only to small subsets, the time is reduced significantly. In fact, as the size of the subset is chosen by the researcher, we can apply the method to any problem regardless of the number of instances involved. As for the case of classifier ensembles, where the base learner is a parameter of the algorithm, in our method the instance selection method is a parameter, and any algorithm can be used.

This paper is organized as follows: Section 2 presents the proposed model for instance selection based on our approach; Section 3 reviews some related work; Section 4 describes the experimental setup; Section 5 shows the results of the experiments; and Section 6 states the conclusions of our work and directions for future research.

2. Democratic instance selection method

The democratic method for instance selection consists of performing r rounds of an instance selection algorithm which is applied to a number of disjoint subsets of the dataset that constitutes a partition of the available data. For each round, the process consists of dividing the original dataset into several disjoint subsets of approximately the same size. Then, the instance selection algorithm is applied to each subset separately. The instances that are selected by the algorithm to be removed receive a vote. Then, a new partition is performed and another round of votes is carried out. After the predefined number of rounds is performed, the instances that have received a number of votes above a certain threshold are removed. An outline of the method is shown in Algorithm 1. Each round can be considered to be similar to a classifier in an ensemble,

and the combination process by voting is similar to the combination of base learners in bagging or boosting [9]. Fig. 1 shows an example of the algorithm for 10 rounds of votes and a dataset of 50 instances.

Algorithm 1: Democratic instance selection (DEMOIS.) algorithm

Data : A training set $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, subset size s, and number of rounds r.
Result: The set of selected instances $S \subset T$.
for $k = 1$ **to** r **do**
1 Divide instances into n_s disjoint subsets t_i: $\bigcup_i t_i = T$ of size s
for $j = 1$ **to** n_s **do**
2 Apply instance selection algorithm to t_j
3 Store votes of removed instances from t_j
end
end
4 Obtain threshold of votes, v, to remove an instance
5 $S = T$
6 Remove from S all instances with a number of votes $\geqslant v$
7 **return** S

The most important advantage of our method is the large reduction in execution time. The reported experiments will show a large difference when using standard widely used instance selection algorithms. Additionally, the method is easy to implement in a parallel environment, because the execution of the instance selection algorithm over each subset is performed independently. Furthermore, as the size of the subsets is a parameter of the algorithm, we can choose the complexity of the execution in each of the processors.

However, as stated, the method still has two important issues to be addressed before we can obtain a useful algorithm. First, the partition method is not trivial, as a strictly random partition would not perform well. Second, the determination of the number of votes is problem-dependent. We carried out preliminary experiments using a fixed threshold for different problems with poor results. Depending on the problem, a certain threshold may be too low or too high. If we set a certain fixed threshold of votes to remove an instance for any problem, there are datasets for which that threshold means removing almost all the instances; on the other hand, there are other datasets for which that threshold results in keeping almost all instances. Thus a method must be developed for automatic determination of the number of votes needed to remove an instance from the training set. Automatic determination of this threshold has the additional advantage of relieving the researcher of the duty of setting a difficult parameter of the algorithm. These two issues are discussed in the following sections. We must also emphasize that our method is applicable to any instance selection algorithm, because the instance selection algorithm is a parameter of the method.

2.1. Partition of the dataset

An important step in our method is partitioning the training set into a number of disjoint subsets, t_i, which comprise the whole training set, $\bigcup_i t_i = T$. The size of the subsets is fixed by the user. The actual size has no relevant influence over the results provided it is small enough to avoid large execution time. Furthermore, the time spent by the algorithm depends on the size of the largest subset, so it is important that the partition algorithm produces subsets of approximately equal size.

We need a different partition of the dataset for each round of votes. Otherwise, the votes cast will be the same because most instance selection algorithms are deterministic. The simplest method would be a random partition, where each instance is randomly assigned to one of the subsets. Each round of votes will receive a different random partition. This was our first attempt at partition method inspired by [10] where bagging was used to get a sparse but not grandmother representation for Kernel Principal Component Analysis. However, this method has two problems: k-NN is a local learning algorithm, so because this partition does not keep, at least partially, the locality of the instances the performance of k-NN will be greatly affected. Thus, the first goal of our partition method is keeping, as much as possible, a certain locality in the partition. However, there is an additional important factor for the performance of the method that is subtler.

Each partition represents a different optimization problem, and thus a different error surface for the instance selection algorithm. If the partitions are very different, these error surfaces will also be very different. In such a case, the votes cast by the different rounds are almost randomly distributed, and the obtained performance is poor. Thus, to obtain a good performance the partitions of the different rounds of the algorithm must vary smoothly.[1]

These two requirements are met using the theory of Grand Tour [11]. The idea of the Grand Tour method, introduced by Asimov [11] and Buja and Asimov [12], is to generate a continuous sequence of low-dimensional projections of a high-dimensional dataset, based on the premise that to fully understand a subject item, one must examine it from all possible

[1] In fact, we performed experiments with random partitions with worse results. However, if the different random partitions are varied smoothly, for example, by performing and initial random partition and then exchanging a few instances between subsets at each round, the performance is clearly improved.

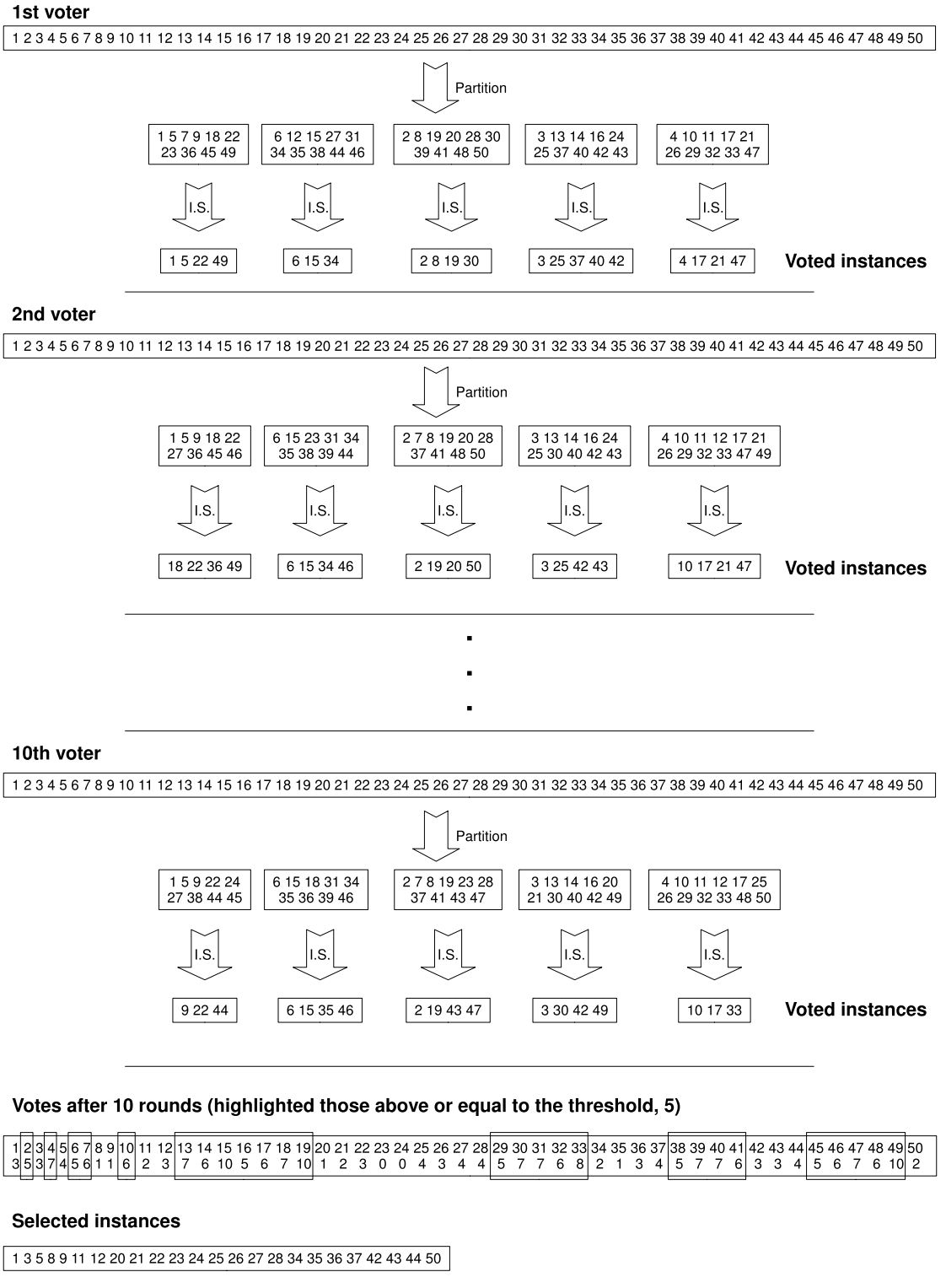

Fig. 1. Example of democratic instance selection for a dataset of 50 instances and subsets of 10 instances. The instance selection algorithm can be any one of the many available.

angles. The method rotates a plane in the high-dimensional space. The data are projected onto this plane for each of its orientations, and when the sequence of projections is visualized on a computer screen, an animation is obtained, which is useful for identifying structures in the data set, such as clusters and outliers. Grand Tour shares a common objective with exploratory projection pursuit techniques. In both cases the human ability for visual pattern recognition is exploited.

2.1.1. Algorithms

When Grand Tour is used for visualization the sequence of planes must hold two conditions:

a. The sequence should be dense in the set of all planes in the high-dimensional space.
b. The sequence should be smooth to give a visual impression of the data points moving in a continuous way.

The state-of-the-art algorithms for Grand Tour are "guided tours" and "manual tours" [13] and are based on the interpolation of a sequence of randomly generated planes. In the context of our algorithm it is enough to use a one-dimensional Grand Tour, thus, we project the data onto a rotating vector and then use this projection to divide the dataset into the subsets that we will pass to the underlying instance selection algorithm. In addition, we are more concerned with the simplicity of the algorithm. Thus, instead of an interpolation class algorithm, we chose a parametrization class algorithm, the torus method [11], based on obtaining a sequence of rotation matrices, which leaves us with the problem of how to obtain these matrices.

We want to obtain a generalized rotation matrix Q that we will use to rotate the vector onto the area where we are going to project the data. This is implemented by choosing Q as an element of the special orthogonal group, denoted by $SO(d)$, of orthogonal $d \times d$ matrices having determinant $+1$ (a matrix must have these two properties to be a rotation matrix). So, we need a continuous curve through $SO(d)$. In the torus method this is achieved by obtaining a continuous curve in a p-dimensional torus ($p = (d-1)d/2$, being d the dimension of the dataset) whose points give the angles to calculate Q. The idea is to get a varying vector of angles $\alpha(s) = (\theta_{1,2}, \theta_{1,3}, \ldots, \theta_{d-1,d})$ that we use to generate Q through the mapping $\beta : [0, 2\pi]^p \to SO(d)$ given by:

$$\beta(\theta_{1,2}, \theta_{1,3}, \ldots, \theta_{d-1,d}) = R_{1,2}(\theta_{1,2}) \times R_{1,3}(\theta_{1,3}) \times \cdots \times R_{d-1,d}(\theta_{d-1,d}) \tag{1}$$

The $R_{i,j}(\theta_{i,j})$ are elements of $SO(d)$ that rotate the $\mathbf{e}_i\mathbf{e}_j$ plane through an angle of $\theta_{i,j}$

$$R_{i,j}(\theta_{i,j}) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & \cos(\theta_{i,j}) & \cdots & -\sin(\theta_{i,j}) & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & \sin(\theta_{i,j}) & \cdots & \cos(\theta_{i,j}) & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

To summarize, the coordinates of a point of the p-torus give the angles of the rotation matrices $R_{i,j}$, which are combined to obtain the rotation matrix Q used for rotating the vector. There are different ways to get the curve through the p-torus [14]: the Asimov–Buja winding algorithm, the random curve algorithm or the fractal curve algorithm. In our experiments we use the random curve algorithm. First, we randomly use two points $\mathbf{s}_i, \mathbf{s}_j$ in $[0, 2\pi]^p$ to create a linear interpolant between the points going from $\mathbf{s}_i$ to $\mathbf{s}_j$, then, if needed, we take a third point $\mathbf{s}_k$ and join it with $\mathbf{s}_j$ and so on.

2.1.2. Some implementation details

Obtaining the curve strictly through the shortest path in the p-torus adds a burden of complexity that does not seem to give any extra advantage to our algorithm. So, instead we just interpolate the points through the hypercube $[0, 2\pi]^p$. If we have a point near the p-dimensional point $(2\pi, 2\pi, \ldots, 2\pi)$ and a point near the p-dimensional point $(0, 0, \ldots, 0)$, in an actual p-torus these two points are very close to each other and the shortest path should be through the walls of the hypercube $[0, 2\pi]^p$. However, in our current implementation, we interpolate the point only using the path strictly inside the hypercube (whose length is approximately $\sqrt{p(2\pi)^2}$). Furthermore, with the interpolation of the first two points in $[0, 2\pi]^p$ we usually obtain enough orientations to get all the partitions required by the algorithm.

Because the density of the projections in the context of instance selection is not a critical factor (we do not need a long tour to get good results and ten to fifteen steps of Grand Tour are usually enough), we can use even simpler ways of obtaining the sequence of projections. In the case of uni-dimensional projections we could have applied only the pseudo Grand Tour obtained by using Andrews curves [15]. If we want a sequence of bi-dimensional projections we can use the orthogonal vectors given by Wegman curves [16].

One concern when Grand Tour is used for dynamic data visualization is that, in general, the mapped curve on $SO(d)$ cannot be uniformly distributed even when the curve on the p-torus is equally distributed. Here, we are also interested in uniformly rotating the projection vector, and we solve this by simply dynamically adapting the interpolation step used to obtain the curve on the p-hypercube whenever the angle changes more than 10% of the previous angle.

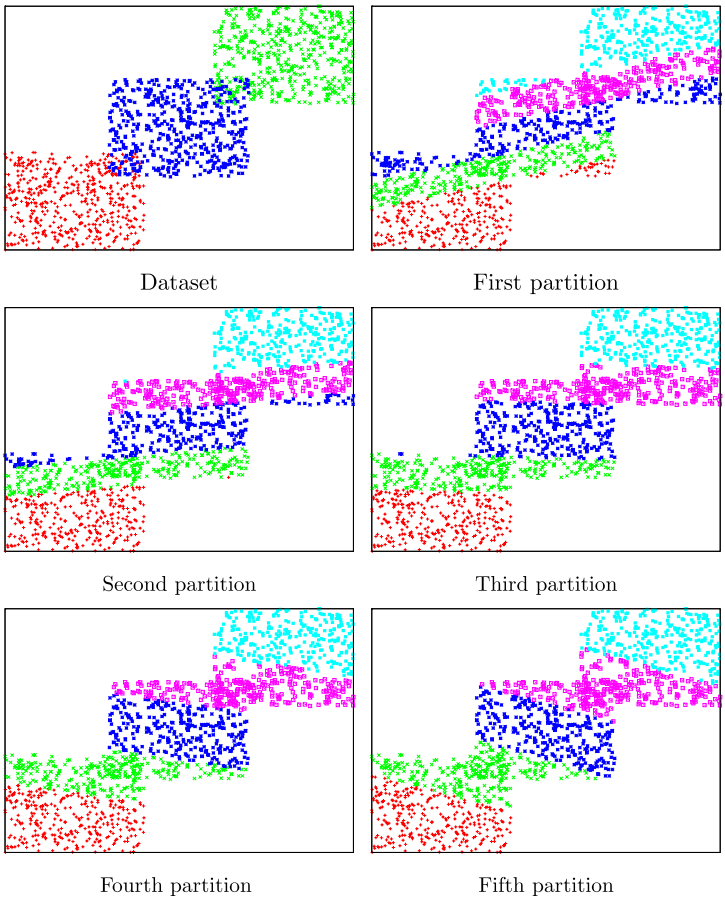

Fig. 2. Example of the method for partitioning the dataset with 1500 instances, three classes, and two features, with five rounds of votes. Four subsets are created each round.

2.2. Partition algorithm

Following this approach, we obtain the first partition by projecting our dataset onto a random vector and then dividing the projection into equal sized subsets. The next vector is obtained using the described procedure and a new partition is made. The procedure is repeated to get the subsequent partitions. Algorithm 2 shows the method for performing the partition based on this methodology.

Algorithm 2: Algorithm for partitioning the training set into disjoint subsets

Data : A training set $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, and subset size s.
Result: The partition into disjoint subsets t_i: $\bigcup_i t_i = T$.
1 Get next vector using Grand Tour method
2 Project all instances into vector
3 Divide the projected instances into subsets of size s using the linear ordering induced by the projection
4 Assign t_i to each subset
5 Return t_i: $\bigcup_i t_i = T$

An example of a partition performed in an artificial training set of 1500 instances, where the data are divided into four subsets, is depicted in Fig. 2. The figure shows the original dataset which contains three classes and the five partitions performed on five rounds of votes. The figure shows the smooth variation of the subsets as the rounds of votes are performed.

This partition is specifically designed for k-NN instance selection algorithms. If we apply our methodology to other classifiers a random partition of the dataset can be used as shown in Section 5.11.

2.3. Determining the threshold of votes

An important issue in our method is determining the threshold of votes to remove an instance from the training set. Preliminary experiments showed that this number depends on the specific dataset. Thus, it is not possible to set a general preestablished value usable in any dataset. On the contrary, we need a way of selecting this value directly from the dataset in run time. A first natural choice would be the use of a cross-validation procedure. However, this method is time consuming. A less costly method is estimating the best value for the number of votes from the effect on the training set. The choice of the number of votes must take into account two criteria: training error, ϵ_t, and storage, or memory, requirements m. Both values must be minimized. We define a criterion, $f(v)$, which is a combination of these two values:

$$f(v) = \alpha\epsilon_t(v) + (1-\alpha)m(v) \tag{2}$$

where m is measured as the percentage of instances retained, ϵ_t is the training error and α is a value in the interval $[0, 1]$ that measures the relative relevance of both values. Because the minimization of the error is usually more important than storage reduction, we have used a value of $\alpha = 0.75$. Different values can be used if the researcher is more interested in reduction than in error. Estimating the training error is time consuming if we have large datasets. To avoid this problem, the training error is estimated using only a small percentage of the whole dataset, which is 10% for medium and large datasets and 0.1% for huge datasets.

The process to obtain the threshold is the following: Once we have performed r rounds of the algorithm and stored the number of votes received by each instance, we must obtain the threshold of votes, v, to remove an instance. This value must be $v \in [1, r]$. We calculate the criterion $f(v)$ (Eq. (2)) for all the possible threshold values from 1 to r and assign v to the value that minimizes the criterion. After that, we remove the instances whose number of votes is above or equal to the obtained threshold v.

2.4. Complexity of our methodology

The aim of this work is to obtain an instance selection methodology that is able to scale up to large and even huge problems. Thus, an analysis of the complexity of the method is essential. In this section, we show how our algorithm is linear in the number of instances, n, of the dataset.

We divide the dataset into partitions of disjoint subsets of size s. Thus, the chosen instance selection algorithm is always applied to a subset of fixed size, s, which is independent of the actual size of the dataset. The complexity of this application of the algorithm depends on the base instance selection algorithm we are using, but will always be small because the size s is always small. Let K be the number of operations needed by the instance selection algorithm to perform its task in a dataset of size s. For a dataset of n instances we must perform this instance selection process once for each subset, that is, n/s times, spending a time proportional to $(n/s)K$. The total time needed by the algorithm to perform r rounds will be proportional to $r(n/s)K$, which is linear in the number of instances, because K is a constant value. Fig. 3 shows the computational cost, as a function of the number of instances, of a quadratic algorithm and our approach when that algorithm is used with subset sizes of $s = 100, 1000, 2500$ and 5000 instances and 10 rounds of votes. If the complexity of the instance selection algorithm is greater, the reduction of the execution will be even better.

The method has the additional advantage of allowing an easy parallel implementation. Because the application of the instance selection algorithm to each subset is independent of all the remaining subsets, all the subsets can be processed at the same time, even for different rounds of votes. Also, the communication between the nodes of the parallel execution is small.

As we have stated, two additional processes complete the method: the partition of the dataset and the determination of the number of votes. The determination of the number of votes can be accomplish in different ways. If we consider all the training instances, the cost of this step would be $O(n^2)$. However, to keep the complexity linear, we use a random subset of the training set to determine the number of votes, with a limit on the maximum size of this subset that is fixed for any dataset. In this way, in medium to large datasets we use 10% of the training set, for huge problems 0.1%, and the percentage is further reduced as the size of the dataset grows. In fact, we have experimentally verified that we can consider any reasonable bound[2] in the number of instances without damaging the performance of the algorithm. With this method the complexity of this step is $O(1)$ because the number of instances used is bounded regardless of the size of the dataset.

Finally, we consider the partition of the dataset apart from the algorithm because many different partition methods can be devised. The partition described in Section 2.1 can be implemented with a complexity $O(n\log(n))$, using a quicksort algorithm for sorting the values to make the subsets, or with a complexity $O(n)$ dividing the projection along the vector in equal width intervals. Both methods achieve the same performance as the obtained partition is similar. In our experiments we have used the latter method to keep the complexity of the whole procedure linear. However, this partition is specially designed for k-NN classifiers. When the method is used with other classifiers, other methods can be used, such as a random partition, which is also of complexity $O(n)$. In fact, in the experiments reported using decision trees and support vector machines, we have used a random partition of the dataset.

[2] This *reasonable* bound can be from a few hundred to a few thousand instances, even for huge datasets.

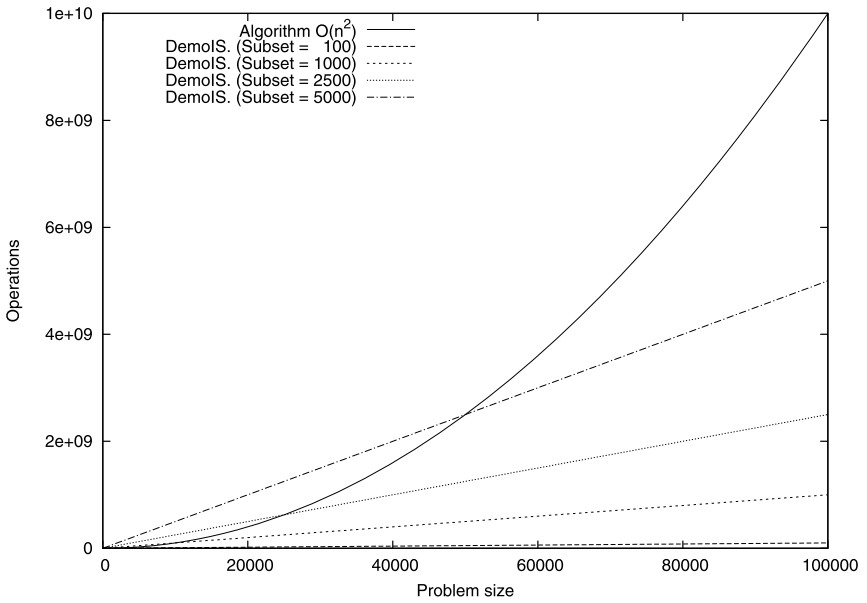

Fig. 3. Computational cost of our method and a base instance selection algorithm of $O(n^2)$.

3. Related work

The usefulness of applying instance selection to disjoint subsets is also shown in [17]. In this work a cooperative evolutionary algorithm is used. Several evolutionary algorithms are performed on disjoint subsets of instances and a global population is used to account for the global view. This method is scalable to medium to large problems but cannot be applied to huge problems.

There are not many previous studies that have dealt with instance selection for huge problems. Cano et al. [18] proposed an evolutionary stratified approach for large problems. Although the algorithm shows good performance, it is still too computationally expensive for huge datasets. Kim and Oommen [19] proposed a method based on a recursive application of instance selection to smaller datasets.

In a recent paper, De Haro-García and García-Pedrajas [20] showed that the application of a recursive divide-and-conquer approach is able to achieve a good performance while attaining a dramatic reduction in the execution time of the instance selection process.

Domingos and Hulten [21,22] developed a method for scaling up learning algorithms based on Hoeffding bounds [23]. The method can be applied to either choosing among a set of discrete models or estimating a continuous parameter. The method consists of three steps: first, it must derive an upper bound on the relative loss between using a subset of the available data and the whole dataset in each step of the learning algorithm. Then, it must derive an upper bound of the time complexity of the learning algorithm as a function of the number of samples used in each step. Finally, it must minimize the time bound, via the number of samples used in each step, subject to the target limits on the loss of performance of using a subset of the dataset. Although the method is able to achieve interesting results, the need to derive these bounds makes its application troublesome for many algorithms. On the other hand, the advantage or our method with respect to this approach, is that no modification of the original algorithm is needed. Furthermore, the experiments reported by the authors [22] show that the dataset size must be several million instances for the method to be worthwhile. The experiments reported show that our method is able to reduce the execution time of the tested algorithms from a problem size of a few thousands instances.

In a subsequent study [24], Domingos and Hulten developed a method for inductive algorithms based on discrete search. The complexity of the method is independent of the process of generating candidate solutions, but only if in this process the method does not need to access the data. In that way, it is applicable to randomized search processes. The general framework proposed [7] has been used for scaling up decision trees, Bayesian network learning, k-means clustering and the EM algorithm for mixtures of Gaussians. To the best of our knowledge, the approach has not been applied to instance selection.

There is a second advantage to our method. To apply the method of Domingos and Hulten we must derive an upper bound of the time complexity of the learning algorithm as a function of the number of samples used in each step. On the other hand, our proposal uses standard algorithms as black boxes, without any modification. In that way, it is applicable to any existing instance selection algorithm.

Table 1
Summary of datasets. The features of each dataset can be C (continuous), B (binary) or N (nominal).

	Data set	Cases	Features			Classes	Inputs	1-NN error
			C	B	N			
1	abalone	4177	7	–	1	29	10	0.8034
2	adult	48 842	6	1	7	2	105	0.2005
3	car	1728	–	–	6	4	16	0.1581
4	gene	3175	–	–	60	3	120	0.2767
5	german	1000	6	3	11	2	61	0.3120
6	hypothyroid	3772	7	20	2	4	29	0.0692
7	isolet	7797	617	–	–	26	617	0.1443
8	krkopt	28 056	6	–	–	18	6	0.4356
9	kr vs. kp	3196	–	34	2	2	38	0.0828
10	letter	20 000	16	–	–	26	16	0.0454
11	magic04	19 020	10	–	–	2	10	0.2084
12	mfeat-fac	2000	216	–	–	10	216	0.0350
13	mfeat-fou	2000	76	–	–	10	76	0.2080
14	mfeat-kar	2000	64	–	–	10	64	0.0435
15	mfeat-mor	2000	6	–	–	10	6	0.2925
16	mfeat-pix	2000	240	–	–	10	240	0.0270
17	mfeat-zer	2000	47	–	–	10	47	0.2140
18	nursery	12 960	–	1	7	5	23	0.2502
19	optdigits	5620	64	–	–	10	64	0.0256
20	page-blocks	5473	10	–	–	5	10	0.0369
21	pendigits	10 992	16	–	–	10	16	0.0066
22	phoneme	5404	5	–	–	2	5	0.0952
23	satimage	6435	36	–	–	6	36	0.0939
24	segment	2310	19	–	–	7	19	0.0398
25	shuttle	58 000	9	–	–	7	9	0.0010
26	sick	3772	7	20	2	2	33	0.0430
27	texture	5500	40	–	–	11	40	0.0105
28	waveform	5000	40	–	–	3	40	0.2860
29	yeast	1484	8	–	–	10	8	0.4879
30	zip	9298	256	–	–	10	256	0.0292

4. Experimental setup

To make a fair comparison between the standard algorithms and our proposal, we selected 30 problems from the UCI Machine Learning Repository [25]. We selected datasets with, at least, 1000 instances. For estimating the storage reduction and generalization error, we used a k-fold cross-validation (cv) method. In this method the available data is divided into k approximately equal subsets. Then the method is learned k times, using in turn each one of the k subsets as testing set, and the remaining $k-1$ subsets as training set. The estimated error is the average testing error of the k subsets. A fairly standard value for k is $k = 10$. A summary of these datasets is shown in Table 1. In some figures throughout the paper we use the number of order of each dataset to reduce the size needed by the graphs. The table shows the 10-fold cv generalization error of a 1-NN classifier without instance selection, which can be considered as a baseline measure of the error of each dataset. These datasets can be considered representative of medium to large problems.

As the main statistical test, we used the Wilcoxon test for comparing pairs of algorithms. We chose this test because it assumes limited commensurability and is safer than parametric tests, because it does not assume normal distributions or homogeneity of variance. Furthermore, empirical results [26] show that this test is also stronger than other tests.

The evaluation of an instance selection algorithm is not a trivial task. We can distinguish two basic approaches: direct and indirect evaluation [27]. Direct evaluation evaluates a certain algorithm based exclusively on the data. The objective is to measure to what extent the selected instances reflect the information present in the original data. Some proposed measures are entropy, moments, and histograms.

Indirect methods evaluate the effect of the instance selection algorithm on the learning task. If we are interested in classification, we evaluate the performance of the used classifier when using the reduced set obtained after instance selection as learning set.

Therefore, when evaluating instance selection algorithms for instance-based learning, the usual method for evaluation is estimating the performance of the algorithms on a set of benchmark problems. In those problems several criteria can be considered, such as [28] storage reduction, generalization accuracy, noise tolerance, and learning speed. Speed considerations are difficult to measure, because we are evaluating not only an algorithm but also a certain implementation. However, because the main aim of our work is scaling up instance selection algorithms, execution time is a basic issue. To allow a fair comparison, we performed all the experiments in the same machine, a bi-processor computer with two Intel Xeon QuadCore processors at 1.60 GHz. To perform sound experiments the algorithm used for the whole training set and the algorithm used in our method were exactly the same.

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

4.1. Instance selection algorithms

To obtain an accurate view of the usefulness of our method, we had to select some of the most widely used instance selection algorithms. We chose to test our model using several of the most successful state-of-the-art algorithms. Initially, we used the algorithms DROP3 [28], and ICF [5]. DROP3 (*Decremental Reduction Optimization Procedure 3*) is shown in Algorithm 3. This algorithm is an example of a new generation of algorithms that were designed taking into account the effect of the order of removal on the performance of the algorithm. ICF is designed to be insensitive to the order of presentation of the instances. It includes a noise-filtering step using a method similar to Wilson's *Edited Nearest-Neighbor* Rule [29]. Then, the instances are ordered by the distance to their nearest neighbor. The instances are removed beginning with the instances furthest from its nearest neighbor. This tends to remove the instances farthest from the boundaries first.

Algorithm 3: DROP3 algorithm

```
   Data  : A training set T = {(x_1, y_1), ..., (x_n, y_n)}, a selector S = ∅
   Result: The set of selected instances S ⊂ T.
 1 Noise filtering: Remove any instance in T misclassified by its k neighbors
 2 S = T
 3 Sort instances in S by distance to their nearest enemy
   foreach Instance P ∈ S do
 4 |   Find P.N_{1..k+1}, the k+1 nearest neighbors of P in S
 5 |   Add P to each of its neighbors' list of associates
   end
   foreach Instance P ∈ S do
 6 |   Let with = # of associates of P classified correctly with P as a neighbor
 7 |   Let without = # of associates of P classified correctly without P
   |   if without ⩾ with then
 8 |   |   Remove P from S
   |   |   foreach Associate A of P do
 9 |   |   |   Remove P from A's list of nearest neighbors
10 |   |   |   Find a new nearest neighbor for A
11 |   |   |   Add A to its new neighbor's list of associated
   |   |   end
   |   end
   end
12 return S
```

ICF (Iterative Case Filtering) is shown in Algorithm 4. For ICF algorithm *coverage* and *reachability* are defined as follows:

$$Coverage(c) = \left\{c' \in T\colon\ c \in LocalSet\left(c'\right)\right\} \tag{3}$$

$$Reachable(c) = \left\{c' \in T\colon\ c' \in LocalSet(c)\right\} \tag{4}$$

The local set of a case c is defined as "the set of cases contained in the largest hypersphere centered on c such that only cases in the same class as c are contained in the hypersphere" [5] so that the hypersphere is bounded by the first instance of a different class. The coverage set of an instance includes the instances that have it as one of their neighbors, and the reachable set is formed by the instances that are its neighbors. The algorithm is based on repeatedly applying a deleting rule to the set of retained instances until no more instances fulfill the deleting rule.

In addition to these two methods, it is worth mentioning the Reduced Nearest Neighbor (RNN) rule [30]. This method is extremely simple, but it also shows a good performance in terms of storage reduction. However, RNN has a serious drawback: its computational complexity. Among the standard methods used, RNN shows the worst scalability, taking several hundreds of hours for the largest problems. Therefore, RNN is the perfect target for our methodology, an instance selection method that is highly efficient but with a serious scalability problem. So, we also tested our approach using RNN, shown in Algorithm 5, as base instance selection method.

We also used one of the most recent algorithms for instance selection, the Modified Selective Subset (MSS) method [31]. The procedure is shown in Algorithm 6. We chose this algorithm as an example of a fast algorithm. With MSS we want to test whether our method is also able to improve the execution time of algorithms that are not as time-demanding as the previous ones.

As an alternative to these standard methods, we also applied genetic algorithms to instance selection, considering this task to be a search problem. The application is easy and straightforward. Each individual is a binary vector that codes a certain sample of the training set. The evaluation is usually made considering both data reduction and classification accuracy. Examples of applications of genetic algorithms to instance selection can be found in [32,33] and [34]. Cano et al. [4] performed a comprehensive comparison of the performance of different evolutionary algorithms for instance selection.

Algorithm 4: ICF algorithm

```
   Data : A training set T = {(x_1, y_1), ..., (x_n, y_n)},
   Result: The set of selected instances S ⊂ T.
1  Noise filtering: Remove any instance in T misclassified by its k neighbors
   repeat
       forall x ∈ T do
2          Compute reachable(x)
3          Compute coverage(x)
       end
4      progress = false
       forall x ∈ T do
           if |reachable(x)| > |coverage(x)| then
5              Flag x for removal
6              progress = true
           end
       end
       forall x ∈ T do
7          if x flagged for removal then T = T − {x}
       end
   until not progress
8  return T
```

Algorithm 5: RNN algorithm

```
   Data : A training set T = {(x_1, y_1), ..., (x_n, y_n)}, a selector S = ∅
   Result: The set of selected instances S ⊂ T.
   /* Obtain Condensed Nearest Neighbor set                                  */
1  S = {x_1}
   foreach Instance P ∈ T do
       if P is misclassified using S then
2          Add P to S
3          Restart
       end
   end
   /* Obtain Reduced Nearest Neighbor set                                    */
   foreach Instance P ∈ S do
4      Remove P from S
       if any instance of T is misclassified using S then
5          Add P to S
       end
   end
6  return S
```

Algorithm 6: MSS algorithm

```
   Data : A training set T = {(x_1, y_1), ..., (x_n, y_n)}, a selector S = ∅
   Result: The set of selected instances S ⊂ T.
1  S = ∅
2  Sort instances x_i ∈ T by distance, D_i, to their nearest enemy
   for i = 1 to n do
3      add = false
       for j = i to n do
           if x_j ∈ T ∧ d(x_i, x_j) < D_j then
4              T = T − {x_j}
5              add = true
           end
       end
6      if add then S = S ∪ {x_i}
7      if T = ∅ then return S
   end
8  return S
```

They compared a generational genetic algorithm [35], a steady-state genetic algorithm [36], a CHC genetic algorithm [37], and a population based incremental learning algorithm [38]. They found that evolutionary-based methods were able to outperform classical algorithms in both classification accuracy and data reduction. Among the evolutionary algorithms, CHC was able to achieve the best overall performance.

Nevertheless, the most critical problem addressed when applying genetic algorithms to instance selection is the scaling of the algorithm. As the number of instances grows, the time needed for the genetic algorithm to reach a good solution

increases exponentially, making it totally useless for large problems. Because we are concerned with this problem, we used as a fifth instance selection method a genetic algorithm using CHC methodology. The execution time of CHC is clearly longer than the time spent by ICF, DROP3 or MSS, so it gives us a good benchmark to test our methodology on an algorithm that, like RNN, has a scalability problem.

5. Experimental results

The same parameters were used for the standard version of every algorithm and its application within our methodology. None of the standard methods have relevant parameters. The only value we had to set was k, the number of nearest neighbors. For DROP3, ICF, we used $k = 3$ neighbors, and $k = 1$ for RNN and MSS. For CHC we performed 100 generations of a population with 100 individuals and $k = 1$. Mutation was applied with a 10% probability. These are fairly standard values [4]. Our method has two parameters: subset size, s, and number of rounds, r. For subset size, we have to use a value large enough to allow for a meaningful application of the instance selection algorithm on the subset, and small enough to allow fast execution because the time used by our method grows with s. As a compromise value, we chose $s = 1000$, and a minimum of two subsets if the dataset has 1000 or fewer instances. For the number of rounds, we chose a small value to allow for fast execution, $r = 10$. In Section 5.9 we carry out a study of the effect of these two parameters on the performance of the algorithm. The application of our method with a certain instance selection algorithm X will be named DEMOIS.x. A summary of the results using the five algorithms is shown in Tables 2 and 3, for standard and DEMOIS.x methods.

5.1. DROP3 vs. DEMOIS.drop3

The results using a standard DROP3 algorithm and our method with DROP3 as base algorithm are plotted in Fig. 4. The figure shows results for testing error, storage requirements and execution time. Throughout the paper, we will use a graphic representation based on the kappa-error relative movement diagrams [39]. However, here, instead of the kappa difference value, we will use the storage difference. The idea of these diagrams is to represent with an arrow the results of two methods applied to the same dataset. The arrow starts at the coordinate origin and the coordinates of the tip of the arrow are given by the difference between the errors and storages of our method and the standard instance selection algorithm. The numbers indicate the dataset according to Table 1. These graphs are a convenient way of summarizing the results. For example, arrows pointing down-left represent datasets for which our method outperformed the standard algorithm in both error and storage, arrows pointing up-left indicate that our algorithm improved the storage but had a worse testing error, and so on.

Numerical results are shown in Tables 2 and 3. In terms of error, our method is able to match the results of the original DROP3 algorithm, the differences between them being small. In fact, DEMOIS.drop3 is even able to improve the performance of DROP3 in some datasets, such as car, mfeat-zer and nursery. In terms of storage reduction, DEMOIS.drop3 performs better than DROP3. Although it achieves worse results than DROP3 in a few problems, DEMOIS.drop3 is able to obtain a large reduction over the results of DROP3 in abalone, gene, german, isolet, krkopt, waveform and yeast datasets. In terms of execution time, the advantage of DEMOIS.drop3 is significant. For small problems, there is a small overload due to the 10 rounds of votes performed; however, as the problem grows in size our approach shows a large reduction in the time needed to perform the instance selection process. In this way, for the most time-consuming problem, adult dataset, DEMOIS.drop3 needs only 5% of the time spent by the original DROP3 to achieve a similar error and better storage reduction.

5.2. ICF vs. DEMOIS.icf

Results for ICF and DEMOIS.icf are plotted in Fig. 5, and the numerical results are shown in Tables 2 and 3. In terms of testing error, DEMOIS.icf is able to improve, or at least match, the results of ICF in all the datasets, with the only exceptions being gene, krkopt, kr vs. kp and nursery problems. Furthermore, for some problems, such as isolet, letter, mfeat-kar, page-blocks and zip, the test error is clearly better than the error achieved by ICF. In terms of storage reduction, the average performance of both algorithms is similar, with a remarkably good performance of DEMOIS.icf for nursery dataset. In terms of execution time the behavior is similar to the case for DROP3. For complex problems, the advantage of DEMOIS.icf over ICF is clear.

5.3. MSS vs. DEMOIS.mss

Results for MSS and DEMOIS.mss are plotted in Fig. 6, and the numerical results are shown in Tables 2 and 3. In terms of both testing error and storage reduction, the performances of DEMOIS.mss and MSS are similar. The relevance of this experiment, as we have stated, was to test whether our approach is still able to reduce the execution time of a simpler algorithm, as was the case with more complex ones, such as DROP3 and ICF. The results show that for large datasets, such as adult, krkopt, letter and shuttle, improvement in the execution time is significant.

Table 2
Testing error, storage requirements and execution time (in seconds) for standard instance selection algorithms.

Dataset	DROP3			ICF			MSS			RNN			CHC		
	Storage	Error	Time (s)	Storage	Error	Time (s)	Storage	Error	Time (s)	Storage	Error	Time (s)	Storage	Error	Time (s)
abalone	0.3069	0.7782	1.8	0.2510	0.8082	1.7	0.6435	0.8053	1.6	0.0079	0.7935	7111.7	0.3818	0.7998	7722.2
adult	0.1248	0.1714	22 853.9	0.1082	0.2194	9170.8	0.2950	0.2281	2990.8	0.0333	0.1951	1 896 540.3	0.1988	0.2257	91 096.0
car	0.2668	0.2378	1.9	0.3813	0.2709	1.1	0.3424	0.2424	0.3	0.0984	0.2471	12.4	0.4192	0.2639	5483.6
gene	0.3877	0.2776	35.6	0.2508	0.3527	26.1	0.4442	0.3274	6.9	0.0402	0.3997	1633.2	0.3004	0.2968	7236.6
german	0.3073	0.2870	0.9	0.1485	0.3260	0.4	0.4309	0.3550	0.2	0.0296	0.2950	28.2	0.3483	0.3290	440.1
hypothyroid	0.0514	0.0610	11.8	0.0398	0.1156	3.7	0.1675	0.0995	0.7	0.0313	0.0655	168.4	0.2613	0.0775	7183.9
isolet	0.2852	0.1770	208.9	0.1713	0.2648	103.1	0.3414	0.1871	39.5	0.0447	0.2665	5950.8	0.2993	0.2026	10 885.4
krkopt	0.4431	0.4803	1533.0	0.5290	0.4032	1109.8	0.6565	0.4323	356.7	0.0425	0.5678	1 057 715.5	0.5237	0.4711	137 015.0
kr vs. kp	0.2229	0.1016	10.1	0.2707	0.1267	5.3	0.3192	0.0843	1.3	0.0558	0.1423	128.9	0.2712	0.1276	7107.5
letter	0.1744	0.1037	1849.8	0.1362	0.2018	760.3	0.2265	0.0749	266.7	0.0581	0.1420	21 394.0	0.2952	0.0905	51 024.0
magic04	0.1789	0.1978	199.8	0.1160	0.2395	138	0.3204	0.2440	23.4	0.0293	0.1805	50 817.2	0.2952	0.1225	29 327.0
mfeat-fac	0.1208	0.0600	39.3	0.0896	0.0905	17.5	0.1672	0.0555	6.2	0.0387	0.0925	47.7	0.3933	0.0455	6518.4
mfeat-fou	0.2473	0.2320	5.6	0.1395	0.3280	2.4	0.3453	0.2515	1.0	0.0444	0.3135	146.2	0.3384	0.2280	7026.9
mfeat-kar	0.1655	0.0835	6.5	0.1035	0.1725	2.5	0.2159	0.0715	0.8	0.0544	0.1265	31.5	0.3838	0.0650	7010.8
mfeat-mor	0.2062	0.2885	1.4	0.2008	0.3685	0.6	0.3253	0.3170	0.4	0.0239	0.3135	64.9	0.3573	0.3265	7054.2
mfeat-pix	0.1095	0.0480	76.5	0.0864	0.1000	27	0.1661	0.0420	8.5	0.0413	0.0810	39.4	0.3759	0.0440	6441.9
mfeat-zer	0.2231	0.2375	4.0	0.1503	0.2715	1.7	0.3488	0.2475	0.8	0.0351	0.3010	102.7	0.3439	0.2205	7076.3
nursery	0.2934	0.3327	337.4	0.8752	0.2414	287.2	0.4160	0.2249	57.2	0.0579	0.2802	5959.7	0.2941	0.2427	22 397.3
optdigits	0.0911	0.0420	161.0	0.0606	0.1103	82.8	0.1663	0.0425	21.0	0.0309	0.0881	281.4	0.2755	0.0404	7846.4
page-blocks	0.0430	0.0437	15.7	0.0307	0.2185	5.6	0.0991	0.0432	0.8	0.0143	0.0559	99.8	0.2786	0.0408	7662.8
pendigits	0.0451	0.0168	175.0	0.0348	0.0651	70.6	0.0900	0.0135	17.8	0.0188	0.0276	289.9	0.2903	0.0121	11 636.6
phoneme	0.1852	0.1383	11.5	0.1392	0.1941	4.6	0.2433	0.1291	1.1	0.0472	0.1778	485.8	0.2846	0.1457	7772.9
satimage	0.1366	0.1101	57.7	0.0713	0.1677	25.4	0.2032	0.1212	8.2	0.0254	0.1345	976.3	0.2825	0.1157	8491.8
segment	0.1219	0.0784	4.1	0.1077	0.1394	1.6	0.1628	0.0541	0.3	0.0428	0.0866	17.8	0.3030	0.0649	7062.4
shuttle	0.0028	0.0016	7543.4	0.0229	0.0473	2640.0	0.0078	0.0012	584.7	0.0014	0.0018	1339.4	0.2638	0.0055	77 089.0
sick	0.0625	0.0509	14.8	0.0452	0.0912	4.7	0.1240	0.0608	0.8	0.0207	0.0594	65.8	0.2578	0.0514	7179.8
texture	0.0878	0.0329	97.0	0.0725	0.0973	46.8	0.1335	0.0213	13.9	0.0329	0.0518	131.8	0.2825	0.0249	7876.1
waveform	0.2961	0.2276	28.8	0.1211	0.2840	15	0.3435	0.3052	5.3	0.0130	0.3198	2132.2	0.2911	0.2878	7926.8
yeast	0.3193	0.4500	0.6	0.2137	0.5095	0.3	0.5339	0.5412	0.1	0.0266	0.5230	49.1	0.3711	0.5014	2981.1
zip	0.1040	0.0497	601.7	0.0497	0.2549	219.8	0.2283	0.0440	71.7	0.0348	0.0884	1420.2	0.2871	0.0510	9748.2

Table 3
Testing error, storage requirements and execution time (in seconds) for democratic instance selection algorithms.

Dataset	DEMOIS.drop3			DEMOIS.icf			DEMOIS.mss			DEMOIS.rnn			DEMOIS.chc		
	Storage	Error	Time (s)	Storage	Error	Time (s)	Storage	Error	Time (s)	Storage	Error	Time (s)	Storage	Error	Time (s)
abalone	0.0822	0.7782	5.1	0.0802	0.7837	4.0	0.5030	0.7945	4.0	0.0167	0.7873	963.4	0.0425	0.8084	1231.5
adult	0.0899	0.1848	1204.5	0.0890	0.1942	621.6	0.3448	0.2076	1309.0	0.0238	0.1746	12 144.9	0.0203	0.2114	8152.1
car	0.3278	0.2163	8.0	0.3775	0.2448	4.9	0.3634	0.2593	1.0	0.0844	0.2797	41.0	0.2893	0.2826	272.4
gene	0.1872	0.2738	43.7	0.2012	0.3628	23.1	0.3547	0.3290	9.1	0.1161	0.3309	1432.9	0.1001	0.3804	558.3
german	0.2012	0.2870	6.8	0.0807	0.3040	3.0	0.4309	0.3280	1.2	0.0296	0.2860	249.1	0.0804	0.3390	141.5
hypothyroid	0.0631	0.0690	26.4	0.0294	0.0804	8.3	0.1782	0.0751	1.6	0.0258	0.0705	102.6	0.0306	0.0822	480.0
isolet	0.1635	0.1840	50.7	0.1722	0.2060	24.6	0.2789	0.1959	12.4	0.1335	0.2109	1122.7	0.1112	0.2381	1609.2
krkopt	0.2679	0.4916	70.4	0.3370	0.4721	56.2	0.5443	0.4114	162.1	0.2889	0.4634	5168.2	0.2678	0.4691	7627.5
kr vs. kp	0.2347	0.1031	28.3	0.2221	0.1279	13.3	0.3128	0.0837	3.9	0.1471	0.1201	182.6	0.1538	0.1684	498.9
letter	0.2236	0.1203	140.1	0.2408	0.1267	81.0	0.2830	0.0885	58.6	0.1775	0.1152	1877.2	0.2244	0.0958	4585.8
magic04	0.1130	0.2048	95.5	0.0967	0.2154	37.1	0.3168	0.2269	19.3	0.0612	0.2469	1645.0	0.0614	0.2620	2965.8
mfeat-fac	0.1436	0.0680	88.8	0.1216	0.0715	32.6	0.1842	0.0550	12.7	0.0804	0.0955	141.8	0.1273	0.0680	258.6
mfeat-fou	0.2210	0.2415	21.7	0.1569	0.2785	9.9	0.3363	0.2445	4.7	0.1184	0.2860	409.5	0.1962	0.2500	322.7
mfeat-kar	0.1966	0.0855	25.4	0.1402	0.1200	10.6	0.2314	0.0725	3.5	0.1157	0.1025	106.6	0.1902	0.0855	291.9
mfeat-mor	0.1494	0.2865	6.0	0.1585	0.3385	2.8	0.3619	0.3315	1.3	0.0392	0.3775	165.5	0.1323	0.3330	269.8
mfeat-pix	0.1776	0.0410	87.3	0.1201	0.0635	40.8	0.1604	0.0560	18.2	0.1093	0.0600	130.4	0.1253	0.0660	267.1
mfeat-zer	0.1710	0.2200	15.5	0.1395	0.2680	7.3	0.3567	0.2255	3.3	0.0931	0.2640	279.9	0.1491	0.2545	301.7
nursery	0.2299	0.2300	87.0	0.2137	0.2449	59.0	0.4105	0.2393	22.6	0.1440	0.2417	750.6	0.2017	0.2439	2425.0
optdigits	0.1093	0.0459	69.6	0.1110	0.0617	28.3	0.1242	0.0591	10.4	0.0861	0.0550	164.2	0.0810	0.0619	827.5
page-blocks	0.0530	0.0448	33.6	0.0392	0.0583	10.5	0.0884	0.0428	1.7	0.0215	0.0596	53.3	0.0609	0.0594	695.3
pendigits	0.0822	0.0218	83.1	0.0790	0.0293	31.7	0.0900	0.0205	10.3	0.0490	0.0197	70.6	0.0656	0.0246	1549.5
phoneme	0.1792	0.1439	21.1	0.1735	0.1646	8.1	0.2943	0.1491	2.8	0.0827	0.1681	213.8	0.1462	0.1683	789.4
satimage	0.1260	0.1173	57.4	0.1110	0.1356	21.4	0.1942	0.1163	8.1	0.0697	0.1236	360.4	0.0789	0.1330	907.2
segment	0.1561	0.0731	12.1	0.1462	0.1117	4.8	0.1788	0.0888	1.6	0.0796	0.1139	24.2	0.1612	0.0926	297.0
shuttle	0.0164	0.0034	337.1	0.0588	0.0126	225.6	0.0275	0.0063	19.6	0.0138	0.0058	19.6	0.0130	0.0048	7286.7
sick	0.0814	0.0565	29.5	0.0480	0.0682	9.8	0.1530	0.0488	1.5	0.0204	0.0610	47.8	0.0107	0.0674	472.5
texture	0.1260	0.0400	59.6	0.1293	0.0460	25.6	0.1553	0.0302	8.1	0.0970	0.0371	90.3	0.1023	0.0587	782.9
waveform	0.1120	0.2354	31.4	0.0742	0.2706	14.5	0.2386	0.2758	6.4	0.0381	0.2690	943.9	0.0592	0.3012	840.5
yeast	0.1460	0.4561	2.5	0.1094	0.4865	1.4	0.5137	0.5014	1.0	0.0567	0.4804	125.6	0.1124	0.5284	245.5
zip	0.1180	0.0646	106.2	0.1644	0.0723	42.7	0.1456	0.0653	22.0	0.0773	0.0639	476.8	0.0802	0.0715	1628.0

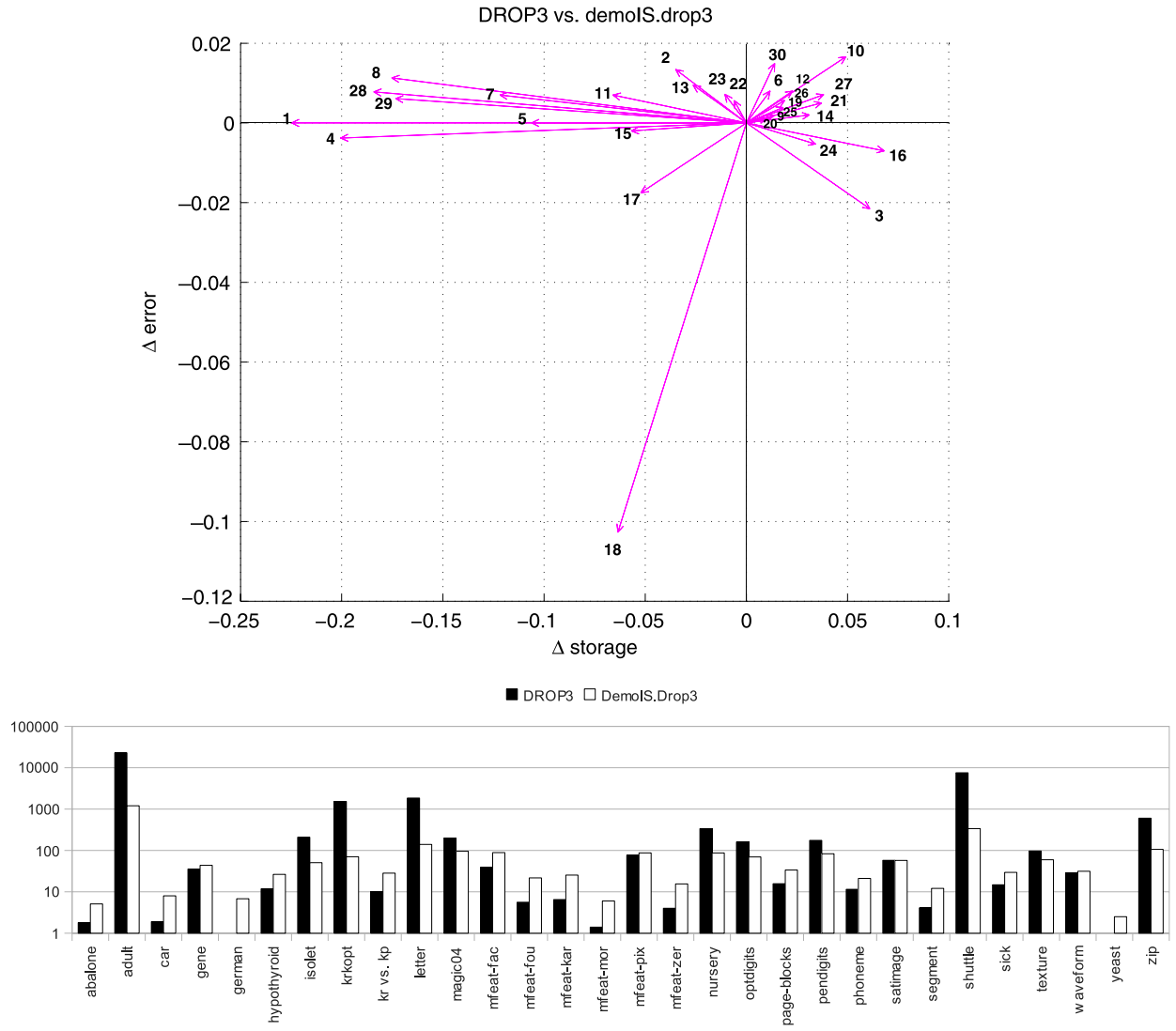

Fig. 4. Storage requirements/testing error (*top*) and execution time in seconds (*bottom* – using a logarithmic scale) for standard DROP3 algorithm and our approach.

5.4. *RNN vs.* DEMOIS.*rnn*

The next experiment was conducted using RNN as the base instance selection algorithm. The results are plotted in Fig. 7 with numerical values shown in Tables 2 and 3. As we stated in the previous section, this is a perfect example of the potentialities of our approach. In our experiments RNN showed the best performance in terms of storage reduction. However, the algorithm has a serious problem of scalability. As an extreme example, for the adult problem RNN took more than 500 hours per experiment. This scalability problem prevents its application in those problems where it would be most useful. The figure shows how DEMOIS.rnn is able to solve the scalability problem of RNN. In terms of testing error, DEMOIS.rnn is also able to improve the performance of RNN, with a better performance in 21 of the 30 datasets. In terms of storage reduction our algorithm performs worse than RNN. However, the performance of DEMOIS.rnn is still good, in fact, better than any other of the previous algorithms. So, our approach is able to scale RNN to complex problems, improving its results in terms of testing error, but with worse results in terms of storage reduction. Execution time results are remarkable, and the reduction of the time spent by the selection process is large. The extreme example of these results is the two most time-consuming datasets, adult and krkopt, where the speed-up is more than a hundred times.

5.5. *CHC vs.* DEMOIS.*chc*

Fig. 8 plots the results of the CHC algorithm, with numerical values shown in Tables 2 and 3. Due to the high computational cost of CHC we chose for this algorithm a smaller subset size of $s = 250$. A first interesting result is the problem of scalability of the CHC algorithm, which is more marked for this algorithm than for the previous ones. In other studies [4,17],

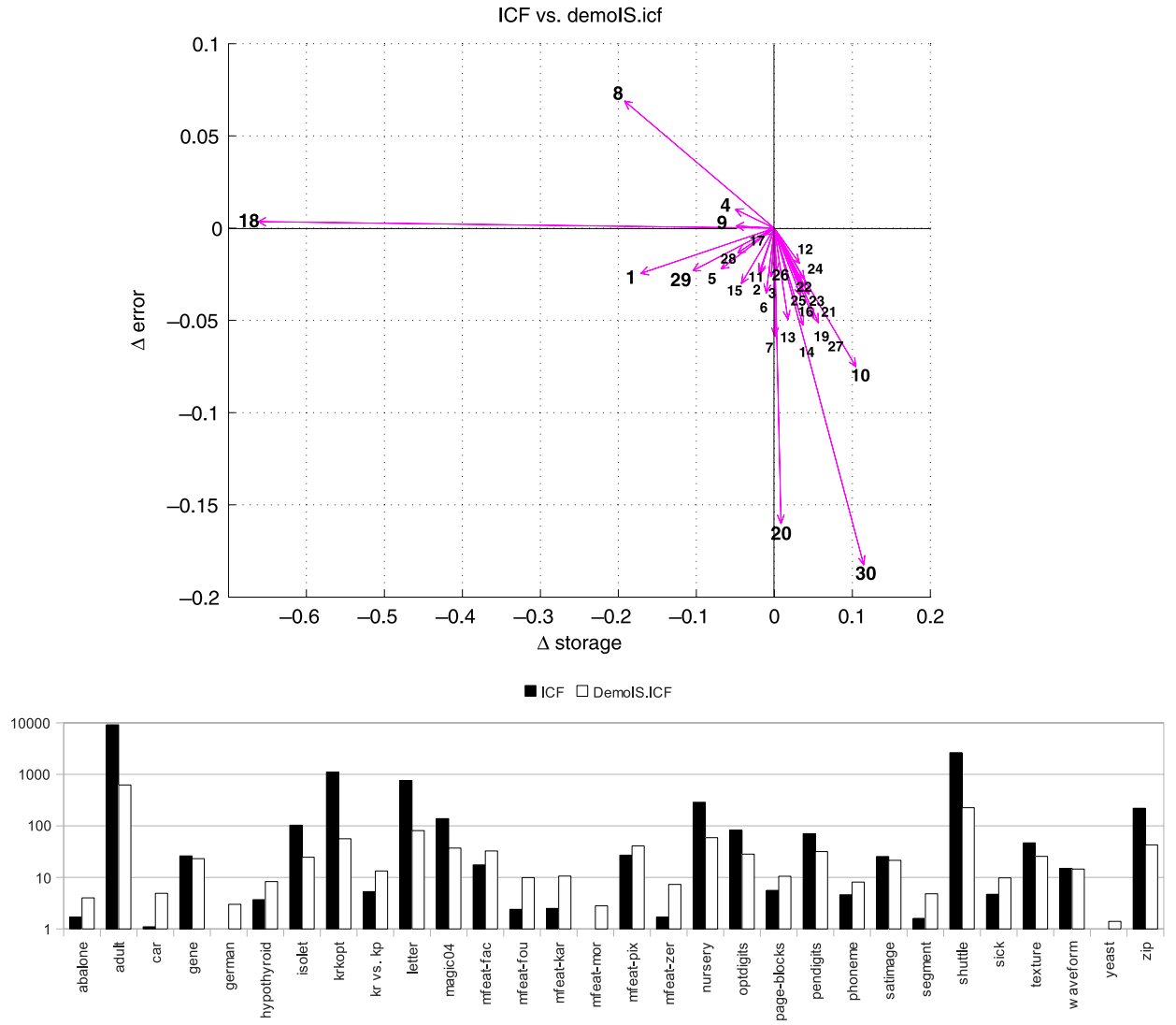

Fig. 5. Storage requirements/testing error (*top*) and execution time in seconds (*bottom* – using a logarithmic scale) for standard ICF algorithm and our approach.

the CHC algorithm was compared with standard methods in small to medium problems. For those problems, the performance of CHC was better than the performance of other methods. However, because the datasets are larger, the scalability problem of CHC becomes relevant. In our set of problems, CHC clearly performs worse than DROP3, ICF, MSS or RNN in terms of storage reduction. We must take into account that for CHC we need one bit in the chromosome for each instance in the dataset. This means that for large problems, such as adult, krkopt, letter, magic or shuttle, the chromosome has more than 10 000 bits, making the convergence of the algorithm problematic. Thus, CHC is, together with RNN, an excellent example of the applicability of our approach. For this method, the scaling up of CHC provided by DEMOIS.chc is evident not only in terms of running time, with a large reduction in all 30 datasets, but also in terms of storage reduction. DEMOIS.chc is able to improve the reduction of CHC in all 30 datasets, with an average improvement of more than 20%, from an average storage of CHC of 31.83% to an average storage of 11.58%. The negative side effect is a worse testing error, which is, however, compensated by the improvement in running time and storage reduction.

5.6. Control experiments

The previous experiments showed that our method is able to, at least, match the performance of standard methods with a significant reduction in the execution time. However, it may be argued that this reduction with respect to standard methods is significant only if the standard methods are useful themselves. In this way, if a simple random sampling is no worse than standard methods, the usefulness of our approach would be partly compromised. In any case, we must not

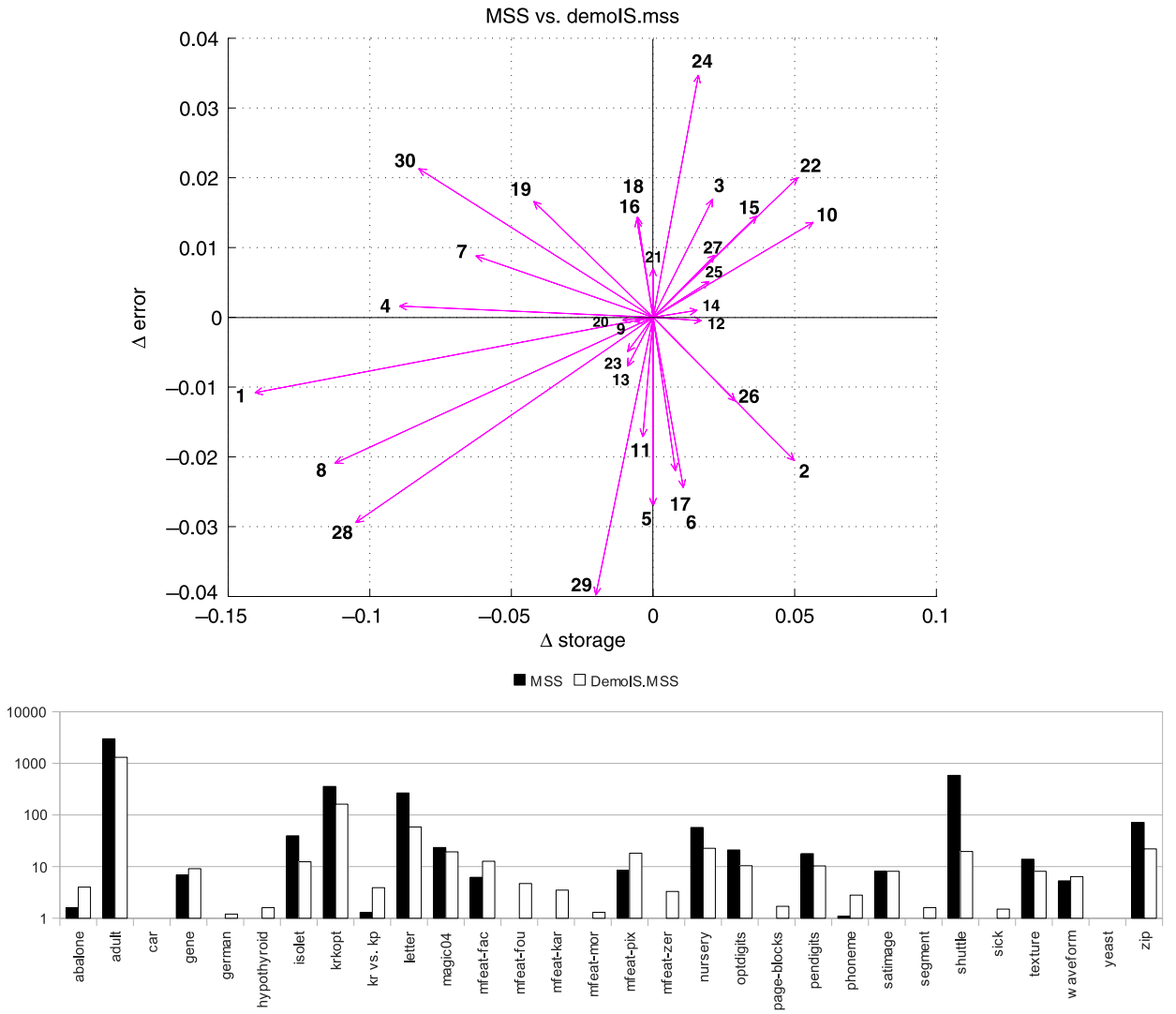

Fig. 6. Storage requirements/testing error (*top*) and execution time in seconds (*bottom* – using a logarithmic scale) for standard MSS algorithm and our approach.

forget that, because random sampling does not determine the number of instances to retain in the subset, it solves only part of the problem [28], even in such cases when the random sampling achieves good results.

In this section we show the results of a control experiment designed to test whether the simple random approach is competitive with respect to standard instance selection methods. For each problem we performed a random sampling with a sampling rate equal to the storage obtained by each algorithm and compared the testing error of each standard method and random sampling. Table 4 shows the comparison for all methods.

The experiment shows interesting results. First, we can see that the most widely used algorithms, Drop3, ICF and RNN, are able to improve the performance of random sampling in a consistent way. All of these algorithms are significantly better than random sampling. The same conclusion is valid for their *democratic* counterparts. This control experiment validates the usefulness of these algorithms. However, the experiment also shows that new algorithms must be compared with random sampling to assure their viability, because MSS and CHC do not show a significantly better behavior than random sampling.

Nevertheless, we must not rule out the use of these algorithms, because the comparison is made using the value obtained by the corresponding instance selection algorithm as the random sampling rate. If we consider random sampling alone, we will not be able to determine the percentage of instances to sample. Thus, if instance selection algorithms are not able to improve the results of random sampling, they are still useful to obtain the sampling rate.

5.7. Study of execution time

In the previous sections we showed that our method's computational cost is linear in the number of instances. To illustrate this property, we show the behavior of the standard algorithms and our approach in terms of execution time

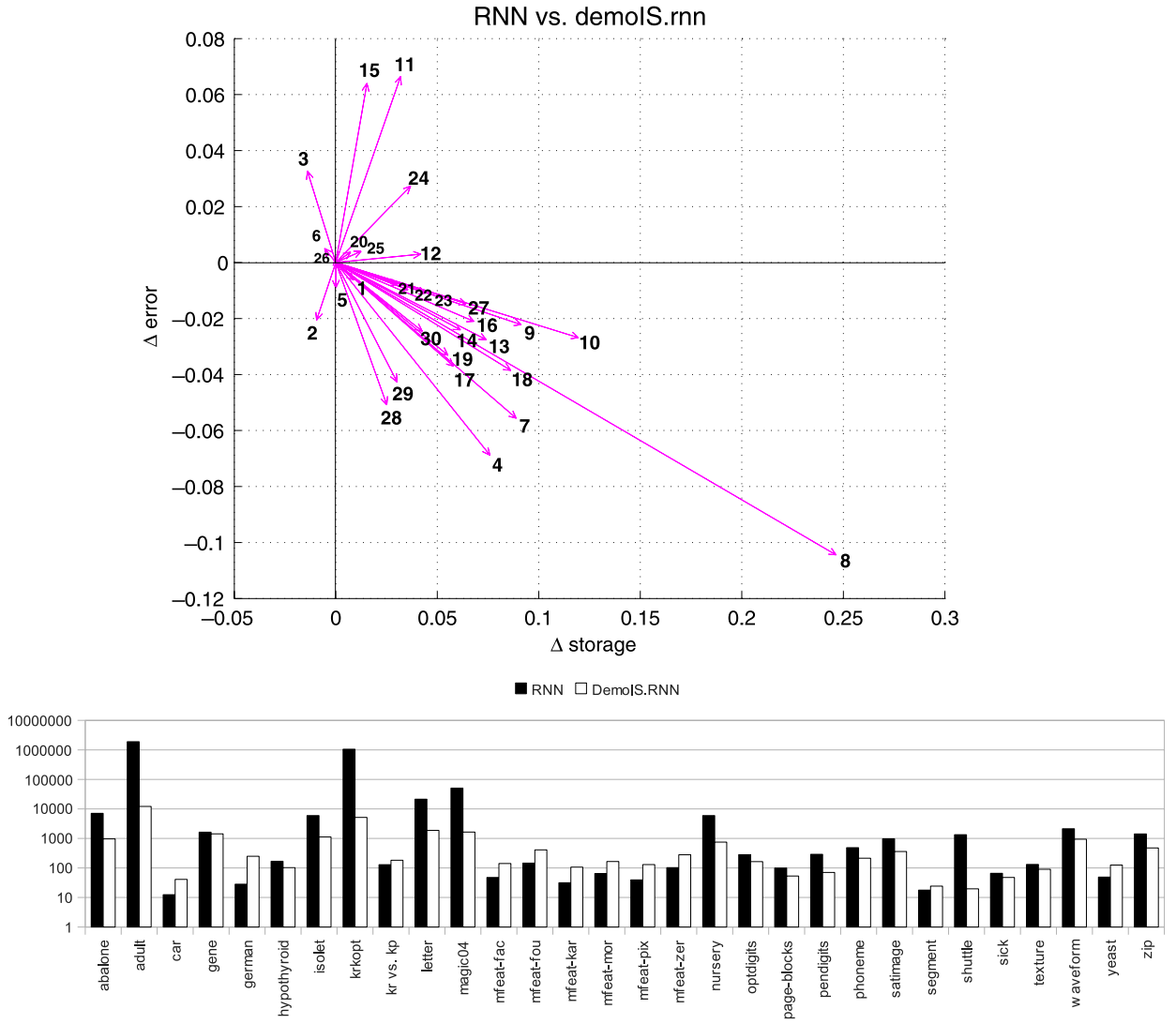

Fig. 7. Storage requirements/testing error (*top*) and execution time in seconds (*bottom* – using a logarithmic scale) for standard RNN algorithm and our approach.

in function of the number of instances in Fig. 9. We plot the time spent by the algorithms as the number of instances increases. A Bezier line is drawn using those points to create a clearer plot. The figure shows that the MSS, ICF and DROP3 methods have an execution time that is approximately quadratic with respect to number of instances. RNN and CHC show a worse behavior, with a far longer execution time. Our proposal is approximately linear, allowing the use of the methods even with hundreds of thousands of instances. This corroborates our theoretical arguments in Section 2.4.

5.8. Summary of results

As a summary of the previous experiments, Table 5 shows a comparison of our approach when using the five tested instance selection algorithms averaged over all the datasets shown in previous tables. The table shows the advantage of using our approach. In terms of testing error, DEMOIS is no worse than the standard algorithms in all of the methods with the exception of CHC. However, although for CHC there is a small increment in testing error, it is coupled with a large decrement in the storage reduction. In terms of storage reduction, DEMOIS is no worse in all of the cases with the exception of RNN. However, for RNN the reduction in terms of execution time is remarkable, and the storage reduction achieved by DEMOIS.rnn is worse than RNN but still better than the other algorithms. In terms of execution time, as showed in Fig. 9, the behavior is excellent for the five algorithms.

In Section 3 we discussed a previous method based on a recursive divide-and-conquer approach [20] that was able to obtain good results in terms of execution time and storage reduction. However, the main drawback of that method is in

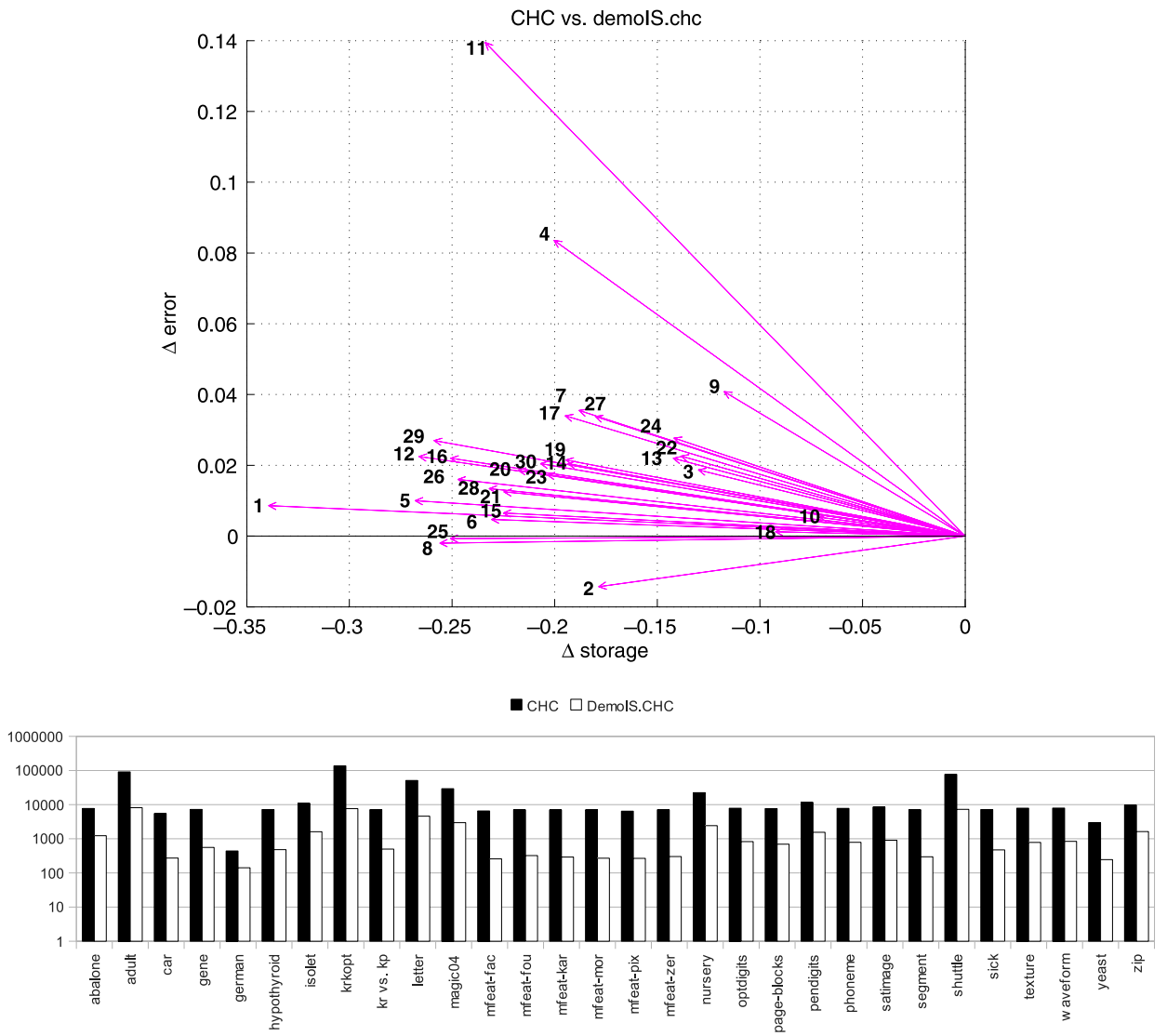

Fig. 8. Storage requirements/testing error (*top*) and execution time in seconds (*bottom* – using a logarithmic scale) for standard CHC algorithm and our approach.

Table 4
Summary of the performance of instance selection methods in terms of testing error against a random sample with the same sampling ratio. The table shows the win/draw/loss record of each algorithm against the random sampling. The row labeled p_s is the result of a two-tailed sign test on the win/loss record and the row labeled p_w shows the result of a Wilcoxon test. Significant differences at a confidence level of 95% using a Wilcoxon test are indicated with a ✓.

	Standard methods				
	Drop3	ICF	MSS	CHC	RNN
Win/draw/loss	24/0/6	22/0/8	18/0/12	16/0/14	27/0/3
p_s	0.0014	0.0161	0.3616	0.8555	0.0000
p_w	0.0039✓	0.0012✓	0.2289	0.7971	0.0000✓
	Democratic methods				
	Drop3	ICF	MSS	CHC	RNN
Win/draw/loss	25/0/5	21/0/9	18/1/11	19/0/11	24/1/5
p_s	0.0003	0.0428	0.2649	0.2005	0.0005
p_w	0.0010✓	0.1020	0.3820	0.2134	0.0009✓

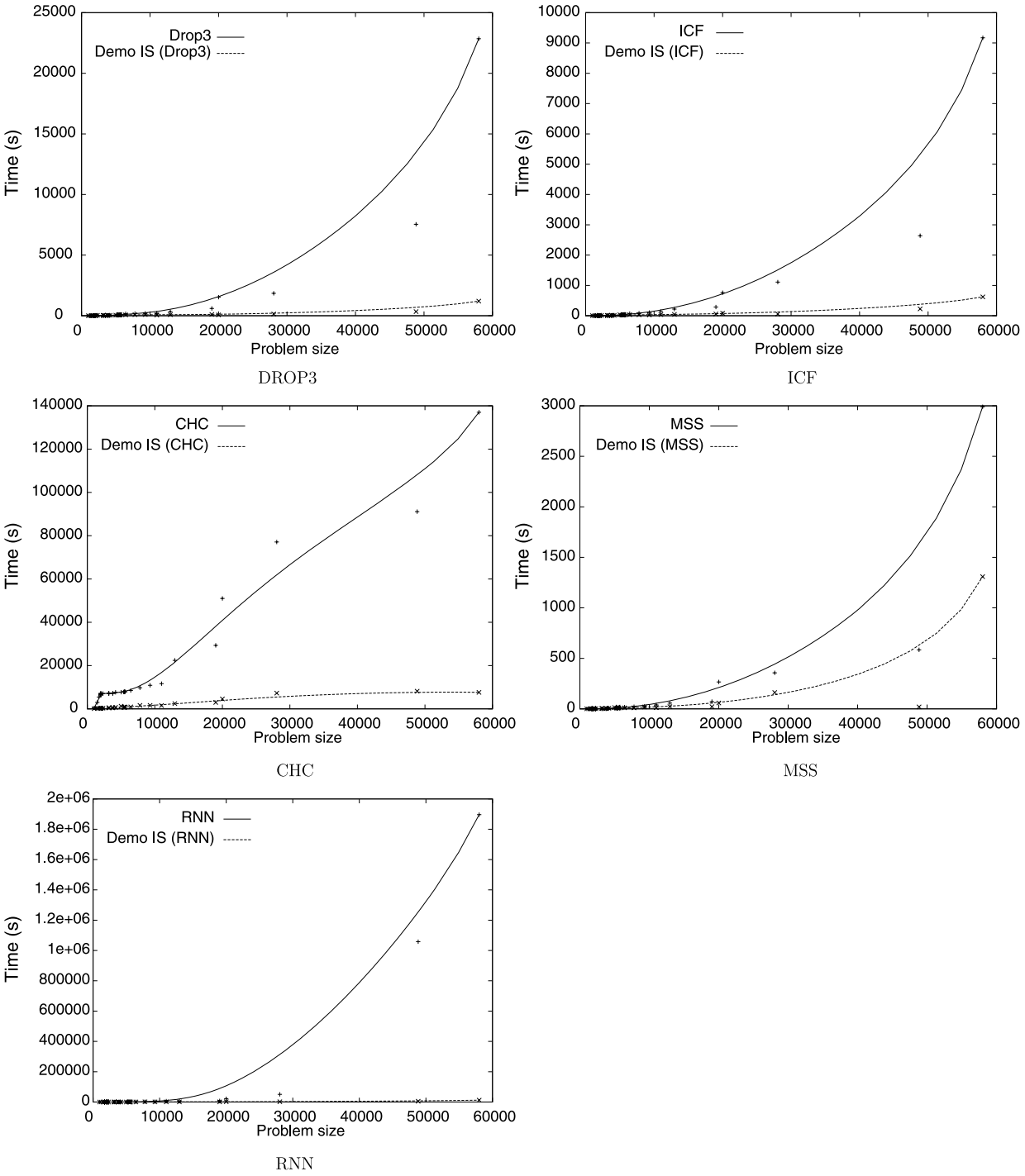

Fig. 9. Execution time (in seconds) for standard methods and our approach as a function of the number of instances.

the testing error, which is worse than that obtained if we apply the original method alone. Fig. 10 shows a comparison of DEMOIS. and this method in terms of testing error for DROP3 and ICF as base methods. The figure shows how DEMOIS. improves the testing error of our previous recursive approach in almost all of the problems. A pairwise comparison of both algorithms, for DROP3 and ICF methods separately, shows significant differences using Wilcoxon test at a confidence level of 99%.

Table 5
Summary of the performance of our methodology against standard methods in terms of testing error, storage requirements and execution time. Significant differences, for testing error and storage reduction, at a confidence level of 95% using Wilcoxon test are indicated by a ✓.

Method	Democratic instance selection		
	Error	Storage	Time
DROP3	Equal	Better	Better
ICF	Better ✓	Better	Better
MSS	Equal	Equal	Better
CHC	Worse ✓	Better ✓	Better
RNN	Better ✓	Worse	Better

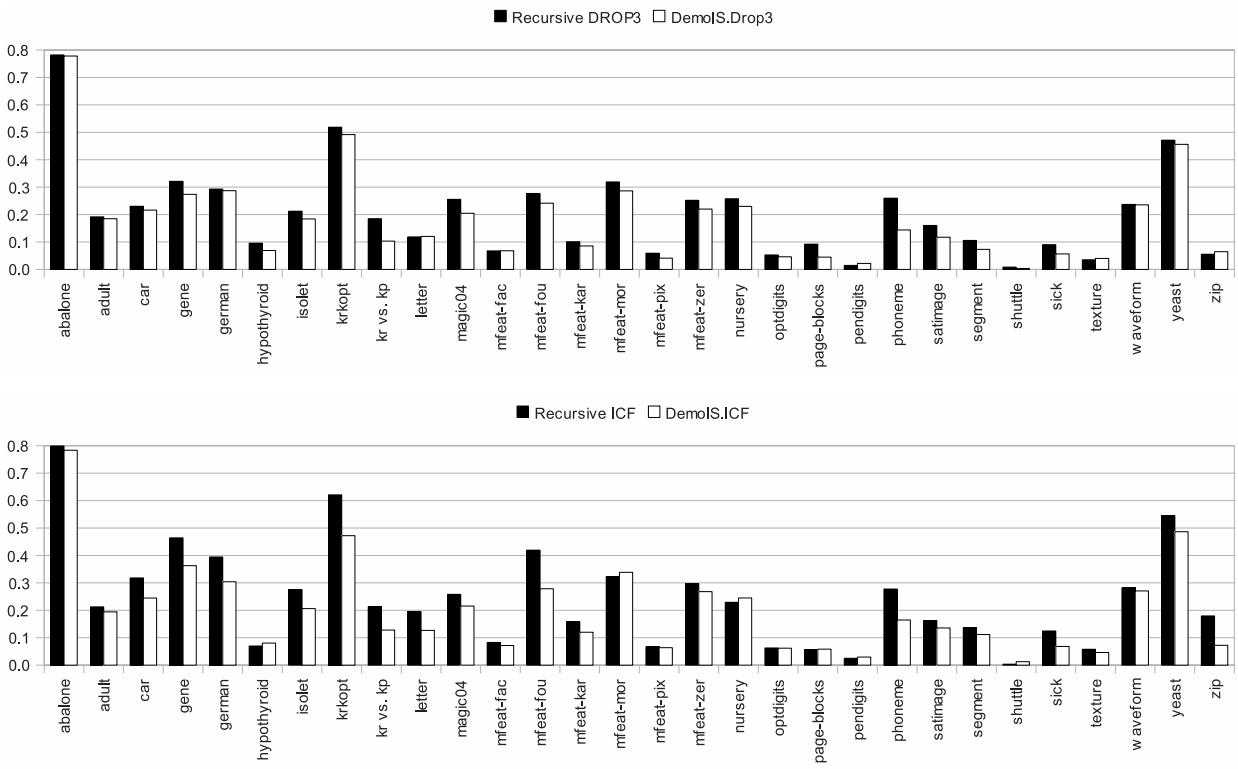

Fig. 10. Testing error for recursive and democratic instance selection using DROP3 (*top*) and ICF (*bottom*) as base methods.

5.9. Study of subset size and number of rounds effect

We have stated that the size of the subset is not relevant provided it is kept small, that is, about a few hundreds or thousands instances. Thus, we chose a subset size of 1000 instances as a good compromise between a subset small enough to obtain a significant reduction of execution time and large enough to allow a meaningful instance selection process. In this section we study the effect of subset size on the behavior of our method. We performed experiments using DROP3 and ICF and subset sizes of 100, 250, 500, 1000, 2500 and 5000 instances and 10 rounds of votes. Figs. 11 and 12 show the results for testing error, storage requirements and execution time with the six different sizes using DROP3 and ICF respectively. For DROP3 the reduction is kept similar regardless of the subset size. With a larger subset size the reduction is somewhat smaller, but the differences are not significant. In terms of testing error, the method needs a subset size large enough to form meaningful subsets. In this way, subsets smaller than 1000 instances obtain worse results, but once the minimum size of 1000 instances is achieved, there is no longer a decrement in testing error. In fact, the results for subset sizes of 1000, 2500 and 5000 instances are almost equal. In terms of execution time we observe a large increment, for example, with DROP3 as the base method, the time grows approximately quadratically as the subset size grows.

The behavior for ICF is similar. In this case, there is a more significant reduction in storage requirements as the subset size becomes larger. This reduction has the side effect of worsening the testing error for subset sizes of 2500 and 5000 instances. The behavior of execution time is the same as for DROP3. As the subset size grows, the execution time grows. As the size becomes larger, the $O(n^2)$ of the algorithm begins to be relevant, and the processing of each subset is more

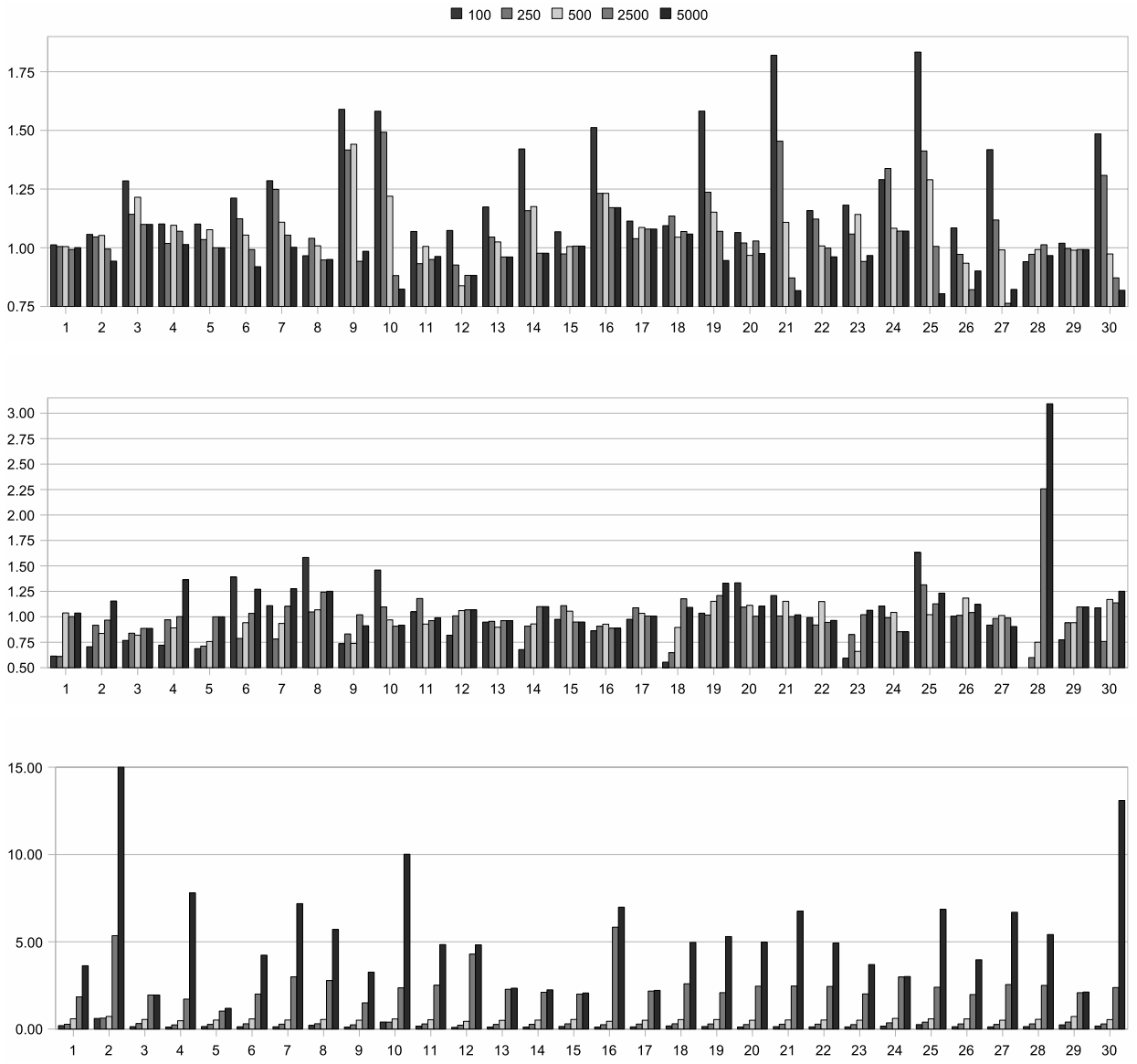

Fig. 11. Average testing error (*top*), storage requirements (*middle*) and execution time (*bottom*) as a function of subset size for DROP3 algorithm. The plots show relative values with respect to the results using a subset of 1000 instances.

important than the reduction of the number of subsets processed. The results corroborate that 1000 instances is a good compromise among the sizes that favor storage reduction, testing error and execution time.

A similar test was performed to determine the effect of the number of rounds on the performance of the method. We ran the method using 5, 10, 25 and 100 rounds. The results for DROP3 and ICF are shown in Figs. 13 and 14, respectively. Again, similar behavior is observed in both algorithms. As more rounds are added, the reduction in storage decreases. This effect is due to the fact that the threshold for removing an instance is higher and so more rounds must agree to remove it. The testing error is not affected after the first few rounds are added. Inspecting the results, we observed that when many rounds are used, 25 or more, many of the votes cast are redundant and there is no advantage in having so many rounds. In this way, a value measured at around 10 rounds is enough. The effect of adding more voters in the testing error is marginal, and each new round increases execution time. This behavior is similar to the case of classifier ensembles, where little gain is obtained after the first few classifiers are added [8].

5.10. Huge problems

In the previous experiments we have shown the performance of our methodology in problems that can be considered medium to large. In this section we consider huge problems, with a few hundreds thousands to more than a million

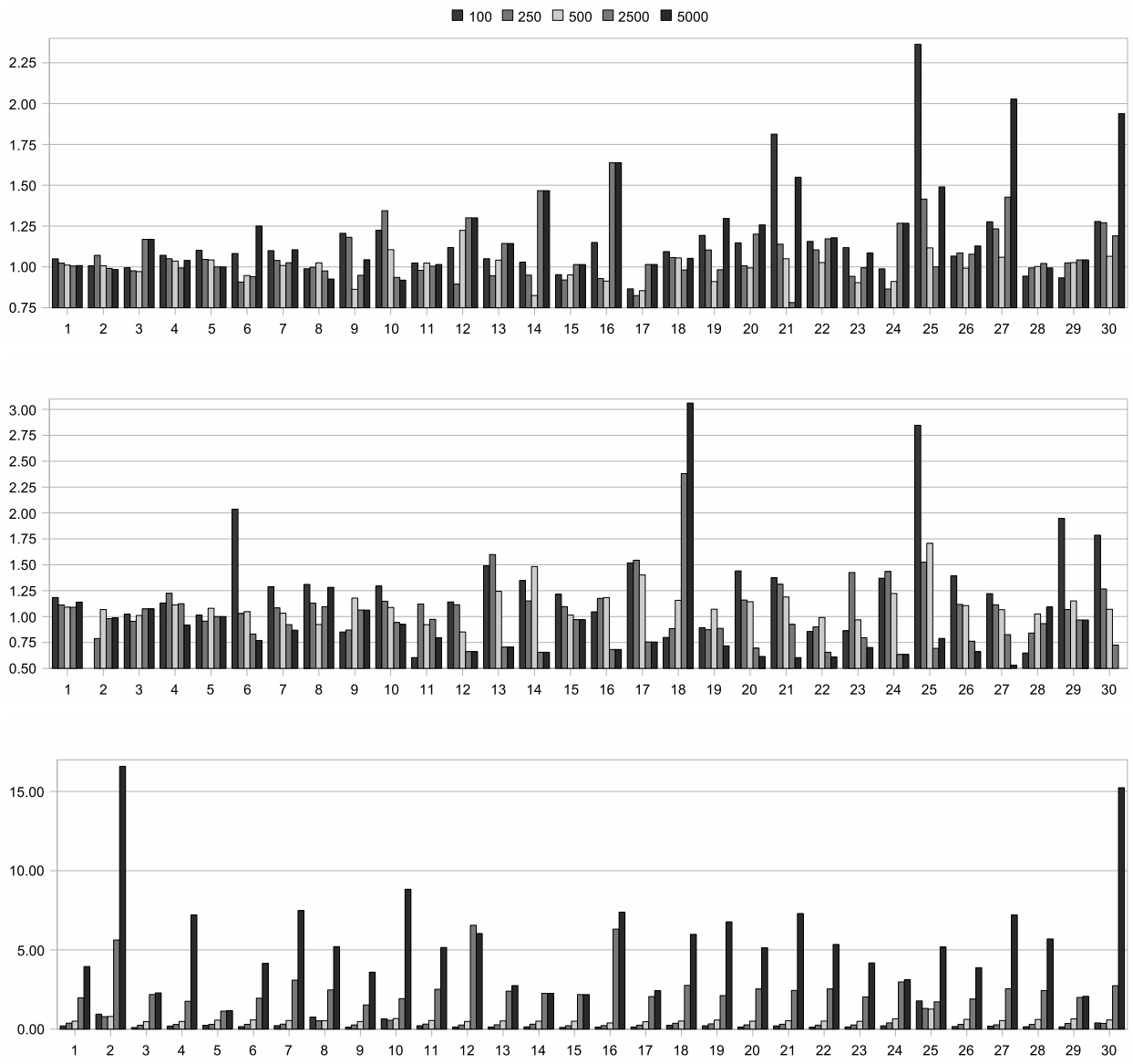

Fig. 12. Average testing error (*top*), storage requirements (*middle*) and execution time (*bottom*) as a function of subset size for the ICF algorithm. The plots show relative values with respect to the results obtained using a subset of 1000 instances.

instances, that are shown in Table 6. These datasets will show whether our methodology allows scaling up standard algorithms to huge problems. As in the previous experiments, testing error and storage reduction are obtained using 10-fold cross-validation. The size of the datasets prevents the execution of the standard algorithms within a reasonable time, so the validity of our approach was tested using the 1-NN 10-fold cv testing error shown in the table. For these problems we used DEMOIS.drop3, DEMOIS.icf and DEMOIS.rnn.

Results are shown in Table 7. The first remarkable result is that our method is able to scale up even to huge problems. In fact, our algorithm makes it possible to do instance selection with datasets whose execution time was prohibitive. In the worst case, in the DEMOIS.rnn for poker dataset, our approach took 93 hours. This value is good if we take into account that the standard RNN took more than 500 hours in adult dataset, a problem with 48 842 instances whereas poker dataset has 1 025 010 instances. Regarding the effectiveness of the scalability, the results are good. The achieved testing error is close to the 1-NN error for all problems and methods, with the only exception being the covtype problem with the DEMOIS.icf method. This testing error comes together with a remarkable reduction in storage size, which for DEMOIS.rnn is less than 1% of the original dataset for census, kddcup99, kddcup991M and poker. Similarly, DEMOIS.drop3 and DEMOIS.icf achieve large reductions for these problems.

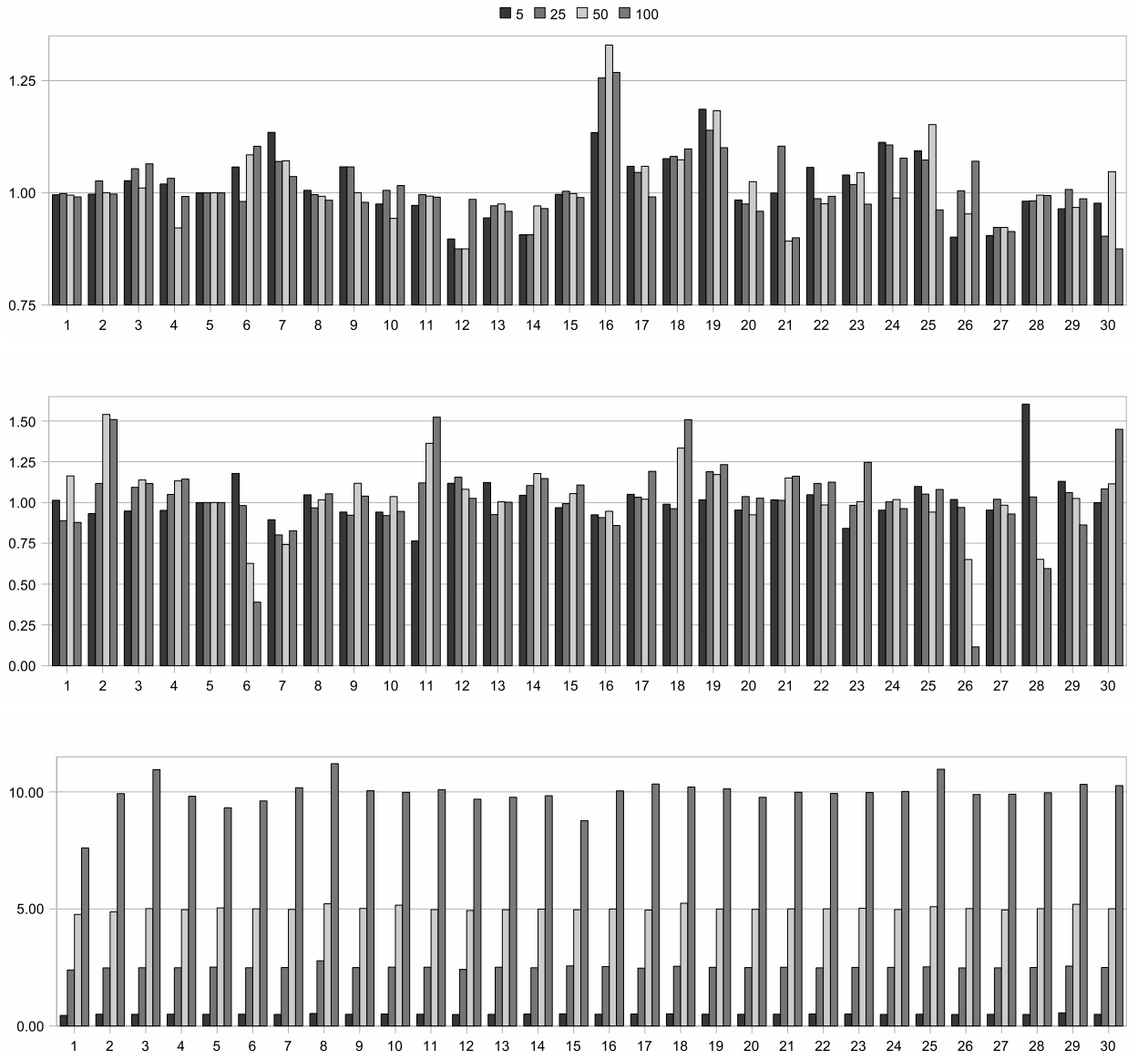

Fig. 13. Average testing error (*top*), storage requirements (*middle*) and execution time (*bottom*) as a function of number of rounds for the DROP3 algorithm. The plots show relative values with respect to the results obtained using 10 rounds.

5.11. Application to other methods of classification

We have stated that our approach can be applied to other classifiers as well. Other learners can benefit from the democratization of the instance selection algorithm, as it provides a way to scale up any instance selection algorithm. In this way, classifiers whose complexity is related to the size of the training set, such as decision trees and support vector machines (SVM), can benefit from instance selection because the constructed classifier would be simpler [6,40]. When the instances selected are used as a training set for an instance-based learner, such as an SVM or a decision tree, the term prototype selection is used more often than instance selection. We will use instance selection for k-NN oriented methods, and prototype selection for methods developed for selecting training instances for an instance-based learner.

Our method can be used without any significant modification with any of these classifiers. We just need a prototype selection algorithm suitable for the used classifier, then we can apply the procedure described in Algorithm 1. As in the described instance selection methods, prototype selection algorithms for decision trees, neural networks or SVMs suffer a problem of scalability. Thus, our method can contribute to scaling up these algorithms as has been shown for k-NN-based instance selection.

In this section we present experiments showing the applicability of the *democratic* algorithm when using decision trees and SVMs as classifiers. We chose these two classifiers because their complexity depends on the quality of the training

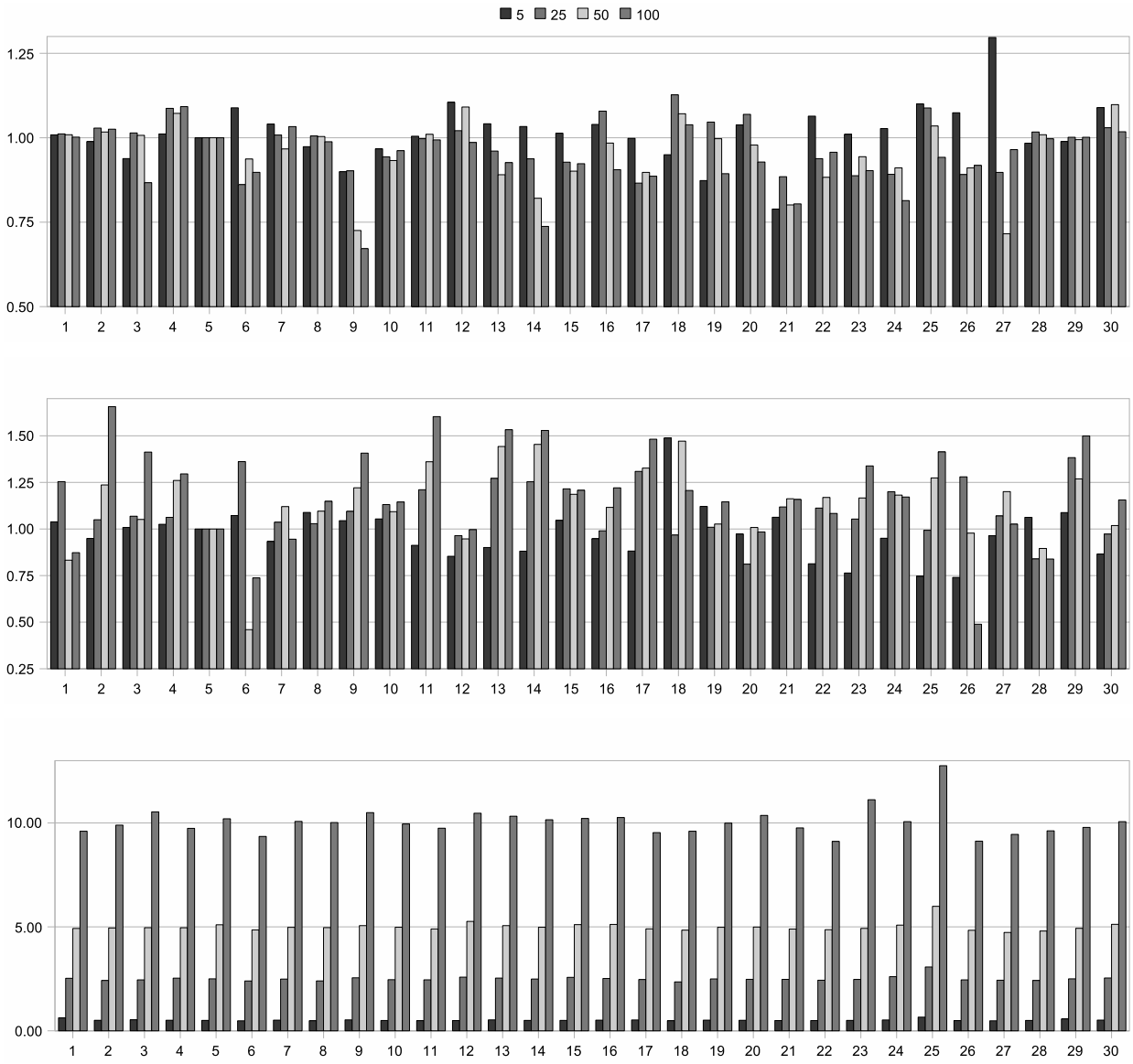

Fig. 14. Average testing error (*top*), storage requirements (*middle*) and execution time (*bottom*) as a function of number of rounds for the ICF algorithm. The plots show relative values with respect to the results obtained using 10 rounds.

Table 6
Summary of datasets. The features of each dataset can be C (continuous), B (binary) or N (nominal). The Inputs column shows the number of input variables.

Data set	Cases	Features			Classes	Inputs	1-NN error
		C	B	N			
census	299 285	7	–	30	2	409	0.0743
covtype	581 012	54	–	–	7	54	0.3024
kddcup99	494 021	33	4	3	23	118	0.0006
kddcup991M	1 000 000	33	4	3	21	119	0.0002
poker	1 025 010	5	–	5	10	25	0.4975

set [40] and also because they are among the most widely used in any machine learning application. As we have stated, the partition method described in Section 2.1 is specially designed for the *k*-NN method. Thus, for the experiments with decision trees and SVMs we used a simple random partition of the training set into disjoint subsets of approximately the same size.

Table 7
Testing error, storage requirements and execution time (in seconds) for our approach for huge problems.

Dataset	Storage	Error	Time
	DEMOIS.drop3		
census	0.0289	0.0771	20 894.3
covtype	0.1627	0.3333	7352.0
kddcup99	0.0123	0.0066	44 198.0
kddcup991M	0.0114	0.0019	89 547.0
poker	0.0247	0.5009	6660.0
	DEMOIS.icf		
census	0.0296	0.0818	6548.0
covtype	0.2250	0.4003	3891.3
kddcup99	0.0266	0.0112	4924.1
kddcup991M	0.0097	0.0072	15 120.0
poker	0.0483	0.5099	5265.3
	DEMOIS.rnn		
census	0.0006	0.0623	75 181.0
covtype	0.2653	0.2955	190 903.0
kddcup99	0.0063	0.0036	112 947.0
kddcup991M	0.0026	0.0037	229 273.0
poker	0.0001	0.4990	335 141.7

As the prototype selection algorithm, we can use any of those previously described. However, because these algorithms are specially designed for k-NN classifiers, their results on other classifiers are rather poor. Thus, we use a method designed for any type of classifier. This method [41] is a filter approach based on using a set of different classifiers as noise filters. These classifiers should detect the noisy, outliers or mislabeled instances and remove them from the training set. The procedure is shown in Algorithm 7. Brodley and Friedl proposed two versions of the method, consensus filter and majority vote. In consensus filter, a set of classifiers $D = \{d_1, d_2, \ldots, d_k\}$ is available, and each classifier d_i is trained on the original training set. After that, instances that are misclassified by *all* classifiers in D are discarded. Then the classifier of our choice is trained on the remaining instances. In majority vote, the procedure is the same, but instances are discarded if *a majority* of the learners misclassify them. In our experiments we used the latter approach because the former resulted in removal of very few instances. We will use the term Majority Vote Filter (MVF) algorithm to refer to this method.

Algorithm 7: Majority vote filter algorithm

```
   Data  : A training set T = {(x_1, y_1), ..., (x_n, y_n)} and a set of learners D
   Result: The subset of selected instances S
   foreach d_i ∈ D do
1  |  Train d_i on T
   end
2  S = T
   foreach x_i ∈ S do
   |  if x_i is misclassified by a majority in D then
3  |  |  Remove x_i from S
   |  end
   end
```

As classifiers in D we chose a 1-NN classifier, a k-NN classifier where k is obtained by cross-validation, a C4.5 decision tree [42], an SVM with a linear kernel, and an SVM with a Gaussian kernel. Decision trees and SVMs are sensitive to parameters, so we performed our experiments using cross-validation for setting the values of the parameters. For each of the classifiers used, we obtained the best parameters from a set of different values. For SVM with a linear kernel we tried $C \in \{0.1, 1, 10\}$, and for an SVM with a Gaussian kernel we tried $C \in \{0.1, 1, 10\}$ and $\gamma \in \{0.0001, 0.001, 0.01, 0.1, 1, 10\}$, testing all 18 possible combinations. For C4.5, we tested 1 and 10 trials and softening of thresholds trying all 4 possible combinations. Although this method does not assure an optimum set of parameters, at least a good set of parameters is obtained in a reasonable time. The SVM learning algorithm was programmed using functions from the LIBSVM library [43].

The experiments were performed with the same experimental setup used previously. There is only one change that must be made in the *democratic* algorithm: the threshold of votes (see Section 2.3) is evaluated using the classifier we are going to learn with the prototypes selected by the algorithm.

The experiments were performed using the standard classifiers, both C4.5 and SVM, on the whole dataset. Then, we applied the MVF algorithm and trained C4.5 and SVM on the dataset selected by MVF. Finally we performed the same experiment using the *democratic* version of MVF, DEMOIS.MVF. Results for C4.5 classifier are shown in Table 8 and for SVM are shown in Table 9. These results are plotted in Figs. 15 and 16, respectively.

Table 8
Testing error, tree size (number of nodes) and execution time (in seconds) for a standard C.45 algorithm, majority vote filtering (MVF) and DEMOIS.MVF.

Dataset	C4.5		MVF + C4.5			DEMOIS.MVF + C4.5		
	Error	Size	Error	Size	Time (s)	Error	Size	Time (s)
abalone	0.7914	2310.2	0.7568	187.2	62.3	0.7391	252.2	30.1
adult	0.1470	1988.56	0.1735	257.6	387 406.1	0.1427	223.0	520.8
car	0.1331	130.2	0.1582	102.0	8.3	0.1657	88.0	6.4
gene	0.0946	274.4	0.0729	163.6	188.7	0.0757	183.6	52.8
german	0.3170	288.2	0.2720	127.6	3811.3	0.2770	137.4	7.1
hypothyroid	0.0056	27.2	0.0114	20.4	216.1	0.0202	18.6	15.8
isolet	0.2997	1368.2	0.2988	1259.4	1128.0	0.2987	1253.0	171.0
krkopt	0.1933	7527.4	0.2739	3692.4	3708.7	0.2554	4320.8	169.6
kr vs. kp	0.0063	70.2	0.0081	56.8	40.6	0.0088	53.6	17.9
letter	0.1221	2553	0.1219	2208.6	2089.6	0.1245	2124.0	141.5
magic04	0.1661	719	0.1763	457.6	18 577.2	0.1699	449.6	103.7
mfeat-fac	0.1150	148.8	0.1100	138.8	265.7	0.1210	128.8	69.6
mfeat-fou	0.2415	272.2	0.2405	166.8	141.7	0.2365	216.6	57.9
mfeat-kar	0.1825	232.2	0.1645	221.2	147.8	0.1740	213.8	49.7
mfeat-mor	0.2955	219	0.2933	97.0	2029.3	0.2770	77.4	10.1
mfeat-pix	0.1190	174.2	0.1130	164.6	206.8	0.1050	157.2	61.0
mfeat-zer	0.3088	290.5	0.3200	267.8	114.8	0.3095	244.8	36.8
nursery	0.1678	367	0.1683	310.8	660.9	0.1739	292.8	67.3
optdigits	0.1082	439.8	0.0998	383.4	380.3	0.0929	374.6	47.5
page-blocks	0.0331	121	0.0269	58.4	2756.4	0.0285	46.6	20.3
pendigits	0.0348	402.4	0.0359	342.0	1059.5	0.0332	336.0	50.5
phoneme	0.1326	254.8	0.1411	196.6	56.0	0.1326	227.6	22.1
satimage	0.1446	654.4	0.1271	420.2	747.6	0.1299	402.6	44.1
segment	0.0307	89.2	0.0329	77.0	47.5	0.0407	71.2	10.3
shuttle	0.0002	58.6	0.0007	47.8	298 567.8	0.0006	43.6	193.1
sick	0.0098	54.2	0.0196	29.6	132.6	0.0167	31.6	16.1
texture	0.0666	308.4	0.0645	258.6	269.2	0.0640	257.2	68.5
waveform	0.2474	599.8	0.2288	460.2	224.2	0.2384	544.2	51.8
yeast	0.4527	453.6	0.4209	99.2	6.5	0.4068	101.4	8.2
zip	0.1340	795.4	0.1324	729.8	916.4	0.1273	721.0	140.5

Table 9
Testing error, size (number of support vectors) and execution time (in seconds) for an SVM, majority vote filtering (MVF) and DEMOIS.MVF.

Dataset	SVM		MVF + SVM			DEMOIS.MVF + SVM		
	Error	Size	Error	Size	Time (s)	Error	Size	Time (s)
abalone	0.7372	3753.2	0.7511	595.2	62.3	0.7413	609.1	31.6
adult	0.1546	15 175.3	0.1572	1415.5	387 406.1	0.1528	2937.5	2095.0
car	0.1430	539.6	0.1064	343.5	8.3	0.1442	581.3	11.0
gene	0.0735	1720.0	0.0773	1459.4	188.7	0.0729	1741.1	119.4
german	0.2460	549.5	0.2630	297.5	3811.3	0.2640	335	9.7
hypothyroid	0.0239	261.3	0.0316	127.7	216.1	0.0515	214.4	28.4
isolet	0.0568	3362.8	0.0605	3046.1	1128.0	0.0587	3477.8	292.6
krkopt	0.1644	18 108.7	0.2598	17 658.3	3708.7	0.2404	13 472.7	317.5
kr vs. kp	0.0081	513.0	0.0085	547.8	40.6	0.0094	372.2	41.1
letter	0.0160	7370.0	0.0303	10 112.7	2089.6	0.0305	6952.7	664.5
magic04	0.3190	4822.7	0.1964	4517.5	18 577.2	0.1589	2399.1	259.2
mfeat-fac	0.0195	535.1	0.0250	328.3	265.7	0.0250	612.8	105.1
mfeat-fou	0.1690	1190.0	0.1530	734.6	141.7	0.1690	955.5	61.3
mfeat-kar	0.0305	870.0	0.0250	700.9	147.8	0.0280	901.3	52.5
mfeat-mor	0.2645	993.9	0.3233	334.7	2029.3	0.2675	234.8	8.4
mfeat-pix	0.0195	823.4	0.0195	678.9	206.8	0.0235	805.9	119.4
mfeat-zer	0.1695	972.1	0.1940	402.0	114.8	0.1780	755.3	39.8
nursery	0.1279	2846.6	0.1044	1177.5	660.9	0.1471	2975.2	903.2
optdigits	0.0171	1244.1	0.0107	1281.5	380.3	0.0153	1161.2	197.5
page-blocks	0.0364	532.4	0.0287	157.7	2756.4	0.0380	294.8	28.5
pendigits	0.0046	1125.5	0.0040	1062.1	1059.5	0.0064	1085.9	127.0
phoneme	0.1065	2489.9	0.1189	1714.6	56.0	0.1264	1293.25	27.8
satimage	0.0748	1746.9	0.0877	1927.9	747.6	0.0823	1295.5	124.4
segment	0.0424	450.0	0.0403	593.8	47.5	0.0467	395.1	12.6
shuttle	0.0014	563.6	0.0019	627.1	298 567.8	0.0018	767.2	380.4
sick	0.0311	505.9	0.0485	149.6	132.6	0.0361	365.7	24.5
texture	0.0016	664.0	0.0049	817.6	269.2	0.0038	737.8	96.5
waveform	0.1410	2767.6	0.1374	1508.0	224.2	0.1370	1828.1	119.7
yeast	0.4149	1083.6	0.4236	341.0	6.5	0.4000	424	8.4
zip	0.0102	1964.6	0.0120	1146.5	916.4	0.0126	2085.1	547.3

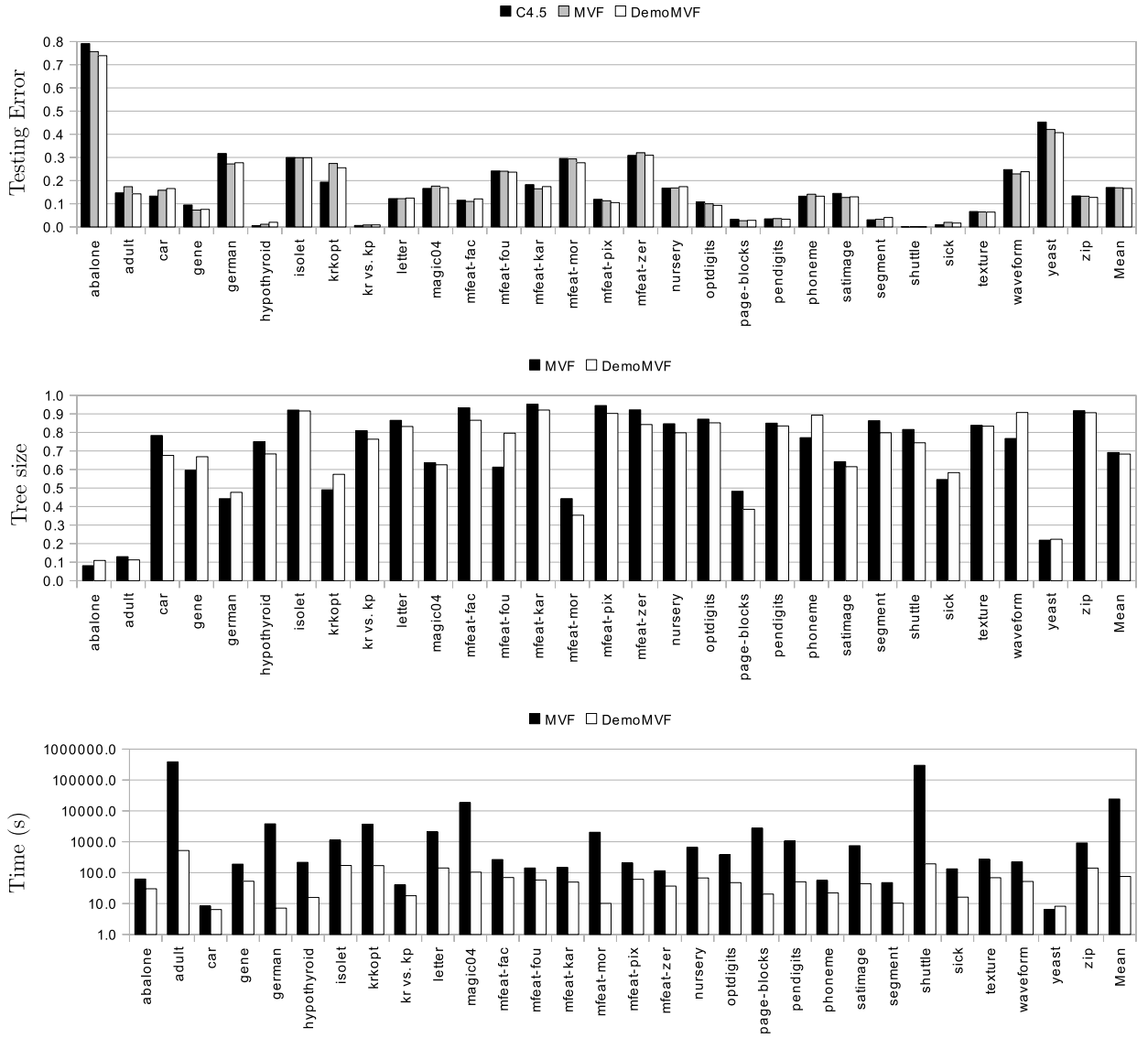

Fig. 15. Testing error, relative tree size, measured as the ratio of the number of nodes with respect to C4.5 applied to the whole dataset, and execution time in seconds (using a logarithmic scale) for standard MVF algorithm and our approach, compared with the C4.5 algorithm applied to the whole dataset.

Both the tables and the results show the usefulness of our approach. MVF is able to obtain classifiers, in both cases, that are simpler than those obtained using all the instances in the training set, and match their testing error. However, as in the previous methods, MVF has a scalability problem for large datasets. This problem is especially noticeable for adult and shuttle datasets. DEMOIS.MVF is able to keep the performance of MVF but with a significant reduction in the execution time. As it was the case for the experiments using *k*-NN, the reduction is more significant when the problem is larger, supporting our claim that the proposed method is able to scale up prototype selection algorithms as well as instance selection methods efficiently.

These results show that our methodology can be applied to different kinds of classifiers provided there is a prototype selection method for them. Other methods that have reported good results, such as PSRCG [44] and SiS [45], can be used as well.

5.12. Noise tolerance

Instance selection algorithms, as any other learning algorithm [46], have degraded performance in the presence of noise. In the field of ensembles of classifiers, Dietterich [47] tested the effect of noise on learning algorithms by introducing artificial noise in the class labels of different datasets. Real-world problems do have noise, thus, it is relevant to study the

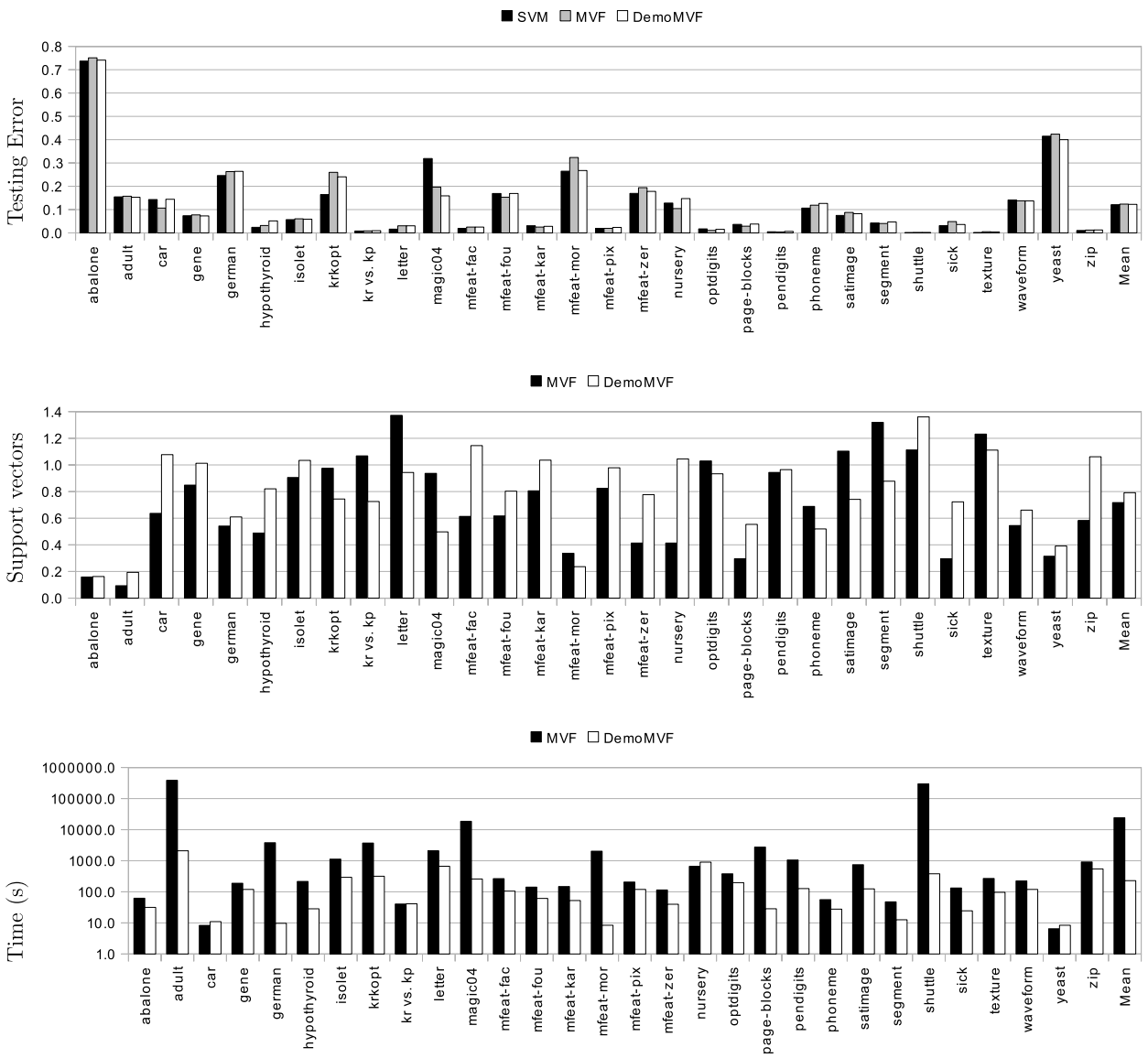

Fig. 16. Testing error, relative tree size, measured as the ratio of the number of support vectors with respect to SVM applied to the whole dataset, and execution time in seconds (using a logarithmic scale) for standard MVF algorithm and our approach, compared with SVM applied to the whole dataset.

behavior of any learning algorithm in the presence of noise. In this section, we study the sensitivity of our method to noise and compare it with standard algorithms.

To add noise to the class labels, we follow the method of Dietterich [47]. To add classification noise at a rate ρ, we chose a fraction ρ of the instances and changed their class labels to be incorrect, choosing uniformly from the set of incorrect labels. We chose all the datasets and three rates of noise, 5%, 10%, and 20%. With these three levels of noise we performed the experiments using DROP3 and ICF, and their democratic counterparts DEMOIS.drop3 and DEMOIS.icf. Fig. 17 shows the results for the four methods at noise levels of 5%, 10% and 20%, and using DROP3 and ICF algorithms. The figure demonstrates the robustness of our method. The degradation of performance is smooth as class label noise is added. It is important to note that our method is able to maintain a good performance in the presence of noise because it uses partial views of the dataset that might be more sensitive to noise. The figures show that our method is able to keep its relative behavior with respect to the original algorithms as noise is added.

6. Conclusions and future work

In this paper we have presented a new method for scaling up instance selection algorithms that is applicable to any instance selection method without any modification. The method consists of performing several rounds of applying instance

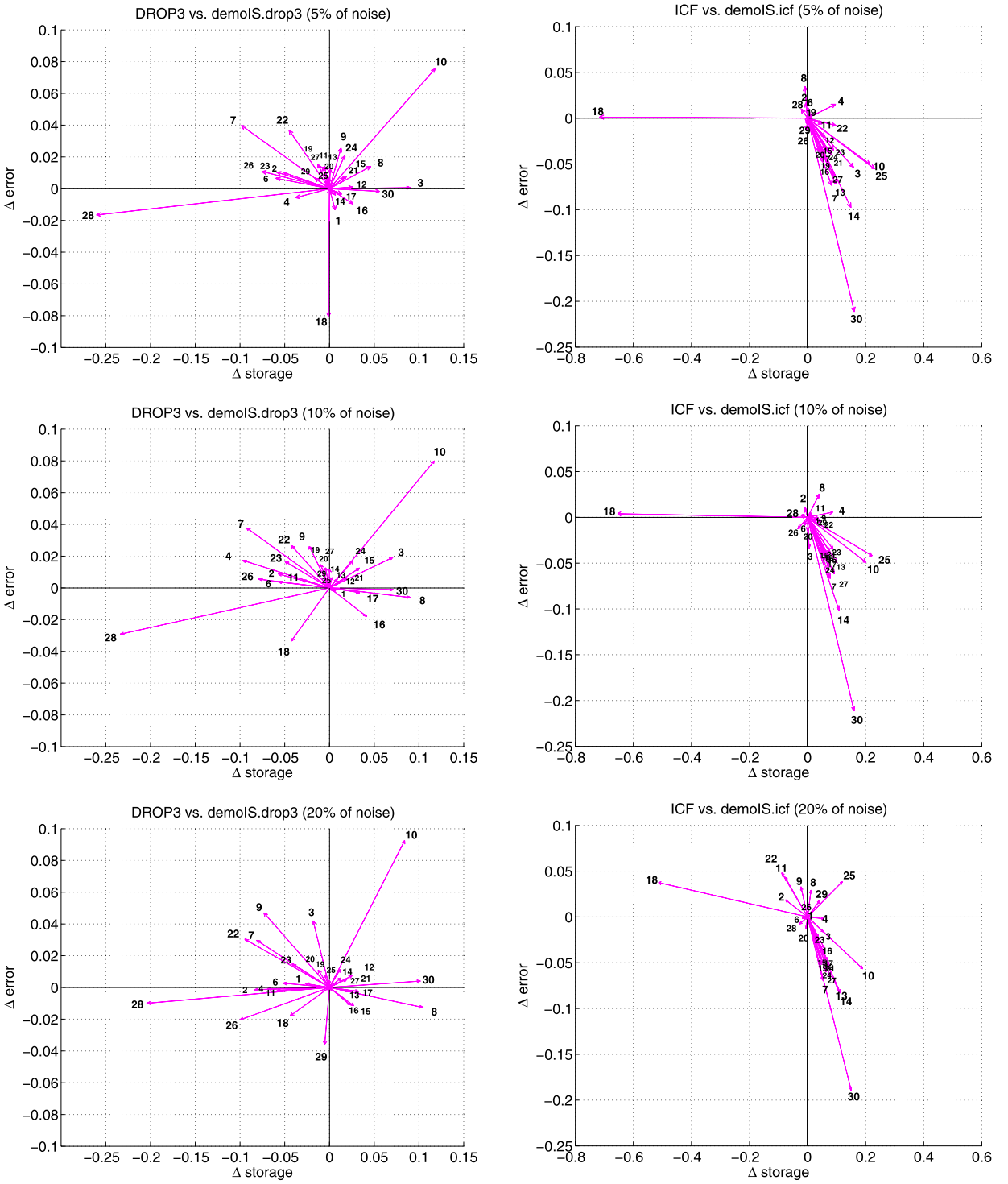

Fig. 17. Comparison between DROP3 and DEMOIS.drop3 and between ICF and DEMOIS.icf in presence of noise.

selection on disjoint subsets of the original dataset and combining them by means of a voting method. Using five well-known instance selection algorithms, DROP3, ICF, RNN, MSS and a CHC genetic algorithm, we have shown that our method is able to match the performance of the original algorithms with a considerable reduction in execution time. In terms of reduction of storage requirements and testing error, our approach is even better than the use of the original instance

selection algorithm over the whole dataset for some of the methods. Our method is straightforwardly paralellizable without modifications.

Additionally, our method has also been tested using prototype selection algorithms and two additional classifiers, decision trees and support vector machines. In both cases it has shown its ability to scale prototype selection algorithms as well as instance selection algorithms.

We have also shown that our approach is able to scale up to huge problems with hundreds of thousands of instances. Using five of those huge datasets our method is able to execute rapidly, achieving a significant reduction of storage while keeping the testing error similar to the 1-NN error using whole datasets. We think that the proposed method might be a breakthrough in instance selection algorithms design, because it allows the development of more complex methods for instance selection. This is due to the relaxation of the constraints on the complexity of the base method through the possibility of using democratic instance selection.

As a principal research approach, we are working on the development of better methods of partitioning the original dataset that we believe may have a relevant influence on the performance of the method. Additionally, in a recent paper [48] it was shown that instance selection can be used as a mechanism for constructing ensembles of classifiers. The method presented in this paper provides a promising way to extend this method to larger datasets.

References

[1] M. Craven, D. DiPasquoa, D. Freitagb, A. McCalluma, T. Mitchella, K. Nigama, S. Slatterya, Learning to construct knowledge bases from the World Wide Web, Artificial Intelligence 118 (1–2) (2000) 69–113.
[2] F.J. Provost, V. Kolluri, A survey of methods for scaling up inductive learning algorithms, Data Mining and Knowledge Discovery 2 (1999) 131–169.
[3] H. Liu, H. Motada, L. Yu, A selective sampling approach to active feature selection, Artificial Intelligence 159 (1–2) (2004) 49–74.
[4] J.R. Cano, F. Herrera, M. Lozano, Using evolutionary algorithms as instance selection for data reduction in KDD: An experimental study, IEEE Transactions on Evolutionary Computation 7 (6) (2003) 561–575.
[5] H. Brighton, C. Mellish, Advances in instance selection for instance-based learning algorithms, Data Mining and Knowledge Discovery 6 (2002) 153–172.
[6] J.R. Cano, F. Herrera, M. Lozano, Evolutionary stratified training set selection for extracting classification rules with trade off precision-interpretability, Data & Knowledge Engineering 60 (1) (2007) 90–108.
[7] P. Domingos, G. Hulten, A general framework for mining massive data streams, Journal of Computational and Graphical Statistics 12 (4) (2003) 945–949.
[8] N. García-Pedrajas, C. García-Osorio, C. Fyfe, Nonlinear boosting projections for ensemble construction, Journal of Machine Learning Research 8 (2007) 1–33.
[9] R.E. Schapire, Y. Freund, P.L. Bartlett, W.S. Lee, Boosting the margin: A new explanation for the effectiveness of voting methods, Annals of Statistics 26 (5) (1998) 1651–1686.
[10] C. García-Osorio, C. Fyfe, Regaining sparsity in kernel principal components, Neurocomputing 67 (2005) 398–402.
[11] D. Asimov, The grand tour: A tool for viewing multidimensional data, SIAM Journal on Scientific and Statistical Computing 6 (1) (1985) 128–143.
[12] A. Buja, D. Asimov, Grand Tour methods: An outline, in: D. Allen (Ed.), Computer Science and Statistics: Proceedings of the Seventeenth Symposium on the Interface, Elsevier Science Publisher B.V., North Holland, Amsterdam, 1986, pp. 63–67.
[13] A. Buja, D. Cook, D. Asimov, C. Hurley, Computational Methods for High-Dimensional Rotations in Data Visualization, North-Holland Publishing Co., 2005, Ch. 14, pp. 391–414.
[14] E.J. Wegman, J.L. Solka, On some mathematics for visualising high dimensional data, Indian Journal of Statistics (Series A Pt. 2) 64 (2002) 429–452.
[15] D.F. Andrews, Plots of high dimensional data, Biometrics 28 (1972) 125–136.
[16] E.J. Wegman, J. Shen, Three-dimensional Andrews plots and the grand tour, Computing Science and Statistics 25 (1993) 284–288.
[17] N. García-Pedrajas, J.A. Romero del Castillo, D. Ortiz-Boyer, A cooperative coevolutionary algorithm for instance selection for instance-based learning, Machine Learning 78 (3) (2010) 381–420.
[18] J.R. Cano, F. Herrera, M. Lozano, Stratification for scaling up evolutionary prototype selection, Pattern Recognition Letters 26 (7) (2005) 953–963.
[19] S.-W. Kim, B.J. Oommen, Enhancing prototype reduction schemes with recursion: A method applicable for "large" data sets, IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics 34 (3) (2004) 1384–1397.
[20] A. de Haro-García, N.G. Pedrajas, A divide-and-conquer recursive approach for scaling up instance selection algorithms, Data Mining and Knowledge Discovery 18 (3) (2009) 392–418.
[21] P. Domingos, G. Hulten, A general method for scaling up machine learning algorithms and its application to clustering, in: Proceedings of the Eighteenth International Conference on Machine Learning, Morgan Kaufmann, 2001, pp. 106–113.
[22] P. Domingos, G. Hulten, Learning from infinite data in finite time, in: Proceedings of Advances in Neural Information Systems, vol. 14, Vancouver, Canada, 2001, pp. 673–680.
[23] W. Howffding, Probability inequalities for sums of bounded random variables, Journal of the American Statistical Association 58 (1963) 13–30.
[24] G. Hulten, P. Domingos, Mining complex models from arbitrarily large databases in constant time, in: Proceedings of the International Conference on Knowledge Discovery and Data Mining, Edmonton, Canada, 2002, pp. 525–531.
[25] S. Hettich, C. Blake, C. Merz, UCI Repository of machine learning databases, http://www.ics.uci.edu/~mlearn/MLRepository.html, 1998.
[26] J. Demšar, Statistical comparisons of classifiers over multiple data sets, Journal of Machine Learning Research 7 (2006) 1–30.
[27] H. Liu, H. Motoda, On issues of instance selection, Data Mining and Knowledge Discovery 6 (2002) 115–130.
[28] D.R. Wilson, T.R. Martinez, Reduction techniques for instance-based learning algorithms, Machine Learning 38 (2000) 257–286.
[29] D.L. Wilson, Asymptotic properties of nearest neighbor rules using edited data, IEEE Transactions on Systems, Man, and Cybernetics 2 (3) (1972) 408–421.
[30] G.W. Gates, The reduced nearest neighbor rule, IEEE Transactions on Information Theory 18 (3) (1972) 431–433.
[31] R. Barandela, F.J. Ferri, J.S. Sánchez, Decision boundary preserving prototype selection for nearest neighbor classification, International Journal of Pattern Recognition and Artificial Intelligence 19 (6) (2005) 787–806.
[32] L. Kuncheva, Editing for the *k*-nearest neighbors rule by a genetic algorithm, Pattern Recognition Letters 16 (1995) 809–814.
[33] H. Ishibuchi, T. Nakashima, Pattern and feature selection by genetic algorithms in nearest neighbor classification, Journal of Advanced Computational Intelligence and Intelligent Informatics 4 (2) (2000) 138–145.
[34] C.R. Reeves, D.R. Bush, Using genetic algorithms for training data selection in RBF networks, in: H. Liu, H. Motoda (Eds.), Instances Selection and Construction for Data Mining, Kluwer, Norwell, Massachusetts, USA, 2001, pp. 339–356.
[35] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison–Wesley, Reading, MA, 1989.

[36] D. Whitley, The GENITOR algorithm and selective pressure, in: M.K. Publishers (Ed.), Proc. 3rd International Conf. on Genetic Algorithms, Los Altos, CA, 1989, pp. 116–121.
[37] L.J. Eshelman, The CHC Adaptive Search Algorithm: How to Have Safe Search when Engaging in Nontraditional Genetic Recombination, Morgan Kaufman, San Mateo, CA, 1990.
[38] S. Baluja, Population-based incremental learning, Tech. Rep. CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, 1994.
[39] J. Maudes-Raedo, J.J. Rodríguez-Díez, C. García-Osorio, Disturbing neighbors diversity for decision forest, in: G. Valentini, O. Okun (Eds.), Workshop on Supervised and Unsupervised Ensemble Methods and Their Applications (SUEMA 2008), Patras, Greece, 2008, pp. 67–71.
[40] M. Sebban, R. Nock, J.H. Chauchat, R. Rakotomalala, Impact of learning set quality and size on decision tree performances, International Journal of Computers, Systems and Signals 1 (1) (2000) 85–105.
[41] C.E. Brodley, M.A. Friedl, Identifying mislabeled training data, Journal of Artificial Intelligence Research 11 (1999) 131–167.
[42] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, San Mateo, 1993.
[43] C.-C. Chang, C.-J. Lin, LIBSVM: A library for support vector machines, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm, 2001.
[44] M. Sebban, R. Nock, Identifying and eliminating irrelevant instances using information theory, in: H. Hamilton, Q. Yang (Eds.), 13th Biennial Conference of the Canadian Society for Computational Studies of Intelligence, AI 2000, Montreal, in: Lecture Notes in Artificial Intelligence, vol. 1822, Springer, 2000, pp. 90–101.
[45] S.S. Sane, A.A. Ghatol, A novel Supervised Instance Selection algorithm, International Journal on Business Intelligence and Data Mining 2 (4) (2007) 471–495.
[46] E. Bauer, R. Kohavi, An empirical comparison of voting classification algorithms: Bagging, boosting, and variants, Machine Learning 36 (1/2) (1999) 105–142.
[47] T.G. Dietterich, An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization, Machine Learning 40 (2000) 139–157.
[48] N. García-Pedrajas, Constructing ensembles of classifiers by means of weighted instance selection, IEEE Transactions on Neural Networks 20 (2) (2008) 258–277.