Department of Computer Science and
Artificial Intelligence
University of Granada

Scaling data mining algorithms. Application to instance and feature selection

Ph.D. Thesis

Author: Aida de Haro García
Supervisor: Nicolás García Pedrajas
Granada, June 2011

Editor: Editorial de la Universidad de Granada
Autor: Aida de Haro García
D.L.: GR 965-2012
ISBN: 978-84-694-9409-7

Ph.D. Thesis

Scaling data mining algorithms. Application to instance and feature selection

Author: Aida de Haro García

Supervisor: Nicolás García Pedrajas

MSc: Soft Computing and Intelligent Systems

Department of Computer Science and Artificial Intelligence

University of Granada

Prof. Dr. D. Nicolás García Pedrajas, Titular de Universidad del Departamento de Informática y Análisis Numérico de la Escuela Politécnica Superior de la Universidad de Córdoba

Certifica

que Dña. Aida de Haro García, Licenciada en Ingeniería Informática, ha realizado en el Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada bajo su dirección el trabajo de investigación correspondiente a sus tesis doctoral titulado:

Scaling data mining algorithms.
Application to instance and feature selection

Revisado el mencionado trabajo, estima que puede ser presentado al tribunal que ha de juzgarlo y autoriza la defensa de esta Tesis Doctoral en la Universidad de Granada.

Granada, 14 de junio de 2011

Fdo.: Prof. Dr. D. Nicolás García Pedrajas
Titular de Universidad

Fdo. La doctorando:
Aida de Haro García

A mis abuelos, mis padres y mi hermana, siempre habéis creído en mí, mis pequeños logros son en definitiva vuestros.

A mis amigos, por ser apoyo y válvula de escape.

A Andrés, por estar siempre ahí, por la paciencia infinita y la ilusión diaria.

Agradecimientos

A Francisco Herrera, Manuel Lozano y las personas de la Universidad de Granada que tanto me han ayudado en este doctorado, por acortar las distancias entre Granada y Córdoba.

Merece una mención aparte el grupo IDEAL de Burgos, que me han demostrado todo este tiempo que sin duda son geniales en todos los ámbitos.

A Colin Fyfe, por su apoyo y por ser en definitiva una fantástica persona y amigo

También quiero expresar mi gran gratitud a Lucy Kuncheva, que tanto me enseñó y que me acogió con una calidez capaz de evaporar toda la lluvia de Bangor.

A los miembros de mi grupo de investigación, compañeros y por encima de todo amigos. Estoy en deuda con vosotros por todo lo que me habéis ayudado en este camino.

Estoy especialmente agradecida a mi director de tesis, Nicolás, su ayuda y el apoyo prestados a lo largo de estos años son innumerables. Por inculcarme la pasión investigadora y compartir sus conocimientos conmigo.

A todos y tantos otros que no caben en estas breves líneas, muchas gracias.

Acknowledgments

To Francisco Herrera, Manuel Lozano and everyone at the University of Granada who helped me so much along this Ph.D. They have been able to narrow the physical gap between Granada and Córdoba.

The IDEAL research group of Burgos deserves a special mention, they have shown they undoubtedly are great in all scopes.

To Colin Fyfe, thanks for your support and for being such a great person and friend.

I want to express my gratitude to Lucy Kuncheva, who has taught me so much during my short visit and that received me with a warmth capable to evaporate all the rain in Bangor.

To the members of my research group, partners and friends above all. I definitely owe you for the support you have always offered me.

I am specially grateful to my thesis director, Nicolás, his help and support over these years are uncountable. He has infused me his research passion and has always uninterestingly shared his knowledge with me.

To everyone and to so many others I left behind these short lines: thank you so much.

Contents

List of Algorithms

List of Tables

List of Figures

1 Introduction

In this chapter we are going to introduce the reader to the problem of scaling data mining algorithms applied to large datasets. We will also provide a taxonomy of the existing scaling algorithms so as to better explain the key features of the scaling methodology proposed in this thesis. Finally we will summarize the main objectives of this thesis and we will present the structure of the thesis.

1.1 Motivation

Traditionally, the bottleneck preventing the development of more intelligent systems via machine learning was the limited data available. However, in the last few years, the limiting factor is learners' inability to use all the data in the available time. This situation is the result of the application of machine learning techniques and more specifically data mining algorithms to new scientific problems, together with the ability to gather information on a massive scale, having the effect of increasing the sizes of the datasets where these methods are applied to hundreds of thousands or even millions of instances.

Most of the widely used algorithms in machine learning were developed when the typical dataset sizes were much smaller. Some of those learning algorithms can work with huge datasets, but many are not able to deal with such large amounts of information. Their application may be hindered by memory demands, impracticable running times or both. In all the fields of application of machine learning, with the growing size of the datasets, the need is also growing to scale up data mining algorithms.

Good examples of these new demanding fields are bioinformatics, text mining or security among others. The relatively new bioinformatics field has opened the doors to extremely large datasets such as the Protein Data Bank (Berman et al., 2000). Text mining problem is of great practical importance given the massive volume of online text available through the World Wide Web, emails and digital libraries, all of them needing an automatical assignment of predefined categories to be effectively processed. In a modern society in which computer systems play an increasingly vital role in modern society it is of crucial importance to protect them by means of analyzing huge amounts of audit data to obtain frequent activity patterns that guide the construction of an intrusion detection model.

Many useful algorithms can be inundated by the flood of data and become very slow in learning a model or classifier and consequently cannot be used

to address relevant problems due to scalability issues. Very large datasets present a challenge for both humans and machine learning algorithms. As Leavitt (2002) notes:

"The two most significant challenges driving changes in data mining are scalability and performance. Organizations want data mining to become more powerful so that they can analyze and compare multiple datasets, not just individual large datasets, as is traditionally the case."

Along with the large amount of data available, there is also a compelling need for producing results accurately and fast. Efficiency and scalability are, indeed, the key issues when designing data mining systems for very large datasets (Chawla et al., 2004). The ideal learner would effectively learn from infinite data in finite time.

The main objective of this thesis is the design of a general methodology to scale up effectively data mining algorithms without severe modifications. We think it is essential for the scaling methodology to be simple and generic. A very complex method will not be widely used by the data mining community and therefore will be less useful. Not only our method is simple but it also treats the data mining algorithms as black boxes so the researchers can use previously implemented versions of these data mining algorithms without adapting the original data mining algorithms or the scaling methodology to a specific problem. This feature of our scaling method makes it generalizable to any data mining problem as the algorithm to be scaled can be swapped to another data mining algorithm without changes, the input format expected by the scaling approach will not change from one problem to another and it will provide the same type of output no matter the problem being dealt with. The only step that varies from one scaled data mining problem to other is the last one that combines effectively the results from the different independent runs.

Due to the fact that we have not unlimited time and storage resources, we consider as a must that our scaling methodology succeeds in reducing execution time while maintaining good performance levels.

The efficiency of an algorithm can be measured in terms of execution time and needed resources. It should be clear that scaling up to very large datasets implies, in part, that fast learning algorithms must be developed, so the execution time is a key measure of the quality of a scaling method. Regarding the needed resources, almost all existing implementations of learning algorithms operate with the training set entirely in main memory and many algorithms achieve reduced runtime complexity with bookkeeping that increases the space used. However, no matter the runtime computational complexity of the algorithm, if exceeding the main memory limitation leads to virtual memory thrashing,

the algorithm will not scale well (Provost and Hennessy, 1996). We think that dealing with data progressively, by means of applying data mining algorithms to subsets of the original dataset, may be a solution to these problems.

Finally, the goal of the learning must be considered. Evaluating the performance of a scaling technique becomes complicated if a degradation in the quality of the learning is permitted. The vast majority of work on learning algorithms uses classification accuracy as the metric by which different algorithms are compared. In such cases, the most interesting methods are the ones that scale up without a substantial decrease in accuracy.

If we assumed that "we do not really need all the data", the first option would be to avoid scaling up methods and just select a representative subset of the original dataset. However, that solution has important problems. If the dataset is huge, subsampling a percentage can itself pose a challenge.

The most common reason for scaling up is that increasing the size of the training set often increases the accuracy of learned classification models (Provost and Kolluri, 1999). For example, a lot of business analysts want to identify interesting customer patterns in the datasets, so taking a subsample might not help in such a scenario. Moreover, in many cases the degradation in accuracy when learning from smaller samples stems from overfitting due to the need to allow the program to learn small disjuncts (Holte et al., 1989), elements of a class description that cover few data items. In some domains small disjuncts make up a large portion of the class description. In such domains, high accuracy depends on the ability to learn small disjuncts to account for these special cases. The existence of noise in the data, which is very common in genomic sequence because of sequencing errors, further complicates the problem, because with a small sample it is impossible to tell the difference between a special case and a spurious data point. Overfitting from small datasets may be also due to the existence of a large number of features describing the data. Large feature sets increase the size of the space of models. Searching through and evaluating more candidate models increases the likelihood that, by chance, the program will find a model that fits the data well, and thereby increases the need for larger example set.

Some data mining applications are not concerned with predictive modelling, but with the discovery of interesting knowledge from large databases. In such cases, increasing accuracy may not be a primary concern. However, scaling up may still be an issue. For example, the ability to learn small disjuncts well often is of interest to scientists, because small disjuncts often capture special cases that were unknown previously and that, in the case of gene recognition, may correspond to the less known structures which are very interesting for the researcher. As with classifier learning, in order not to be swamped with spurious small disjuncts, it is essential for a dataset to be large enough to

contain instances of each special case from which to generalize with confidence.

In scaling up learning algorithms, the issue is not as much one of speeding up a slow algorithm as one of turning an impracticable algorithm into a practicable one. The crucial issue is seldom "how fast" we can run on a certain problem, but instead "how large" a problem we can (feasibly) deal with. From the point of view of complexity analyses, for most scaling problems the limiting factor of the dataset has been the number of examples. A large number of examples introduces potential problems with both time and space complexity. For time complexity, the appropriate algorithmic question is: what is the growth rate of the algorithm's run time as the number of examples increases? Also important are the number of attributes describing each example and the number of values for each attribute.

1.2 Current approaches

There are several methods for scaling up data mining algorithms, although none of them is a perfect solution. Provost and Kolluri (1999) identified three fundamental approaches for scaling up learning methods that are independent and can be applied simultaneously:

- Designing fast algorithms.
- Partitioning the data.
- Using a relational representation.

The fast algorithm approach includes a wide variety of algorithm design techniques for reducing the asymptotic complexity, for optimizing the search and representation, for finding approximate solutions instead of exact solutions, or for taking advantage of the inherent parallelism of the task. The major drawback of this approach is the need to modify the mining methods, a task that is often very difficult or even unfeasible.

The data partitioning approach involves breaking the dataset up into subsets, learning from one or more of these subsets, and possibly combining the results. Data partitioning is useful to avoid the thrashing by memory management systems that occurs when algorithms try to process huge datasets in main memory. In addition, if a learning algorithm's time complexity is worse than linear in the number of examples, processing small, fixed-size data subsets sequentially can make it linear, with the constant term dependent on the size of the subsets. In either case, it may be possible to use a system of distributed processors to mine the subsets concurrently. As an additional advantage, data-partitioning methods do not need to modify the mining algorithm. An orthogonal approach to the selection of example subsets is to select

subsets of relevant features upon which to focus attention.

The relational representation approach addresses data that cannot feasibly be treated as a flat file, including any large relational database, as well as other large relational structures such as those used for knowledge representation in artificial intelligence. In the literature, such techniques have been framed either as learning in first-order logic, or as learning from relational databases (without flattening them out), or as flat-file learning augmented with relational background knowledge. It requires as input, data with a high level of abstraction and therefore it cannot be applied to many data mining problems because many of them do not meet that requirement.

1.3 Objectives

The main aim of this thesis is the design of a methodology to scale up data mining algorithms and its practical application to instance selection and feature selection fields. More precisely we can divide this overall objective into the following specific objectives:

(1) Design a powerful scaling method focusing on the objective of achieving a good balance between efficiency and performance.
   (a) The efficiency of an algorithm can be measured in terms of execution time and needed resources. It should be clear that scaling up to very large datasets implies, in part, that fast learning algorithms must be developed, so the execution time is a key measure of the quality of a scaling method. Regarding the needed resources, almost all existing implementations of learning algorithms operate with the training set entirely in main memory and many algorithms achieve reduced runtime complexity with bookkeeping that increases the space used. However, no matter the runtime computational complexity of the algorithm, if exceeding the main memory limitation leads to virtual memory thrashing, the algorithm will not scale well (Provost and Hennessy, 1996). We think that dealing with data progressively, by means of applying data mining algorithms to subsets of the original dataset, may be a solution to these problems.
   (b) The goal of the learning must also be considered. Evaluating the performance of a scaling technique becomes complicated if a degradation in the quality of learning is permitted. The vast majority of work on learning algorithms uses classification accuracy as the metric by which different algorithms are compared. In such cases, the most interesting methods are the ones that scale up without a substantial decrease in accuracy. As in this thesis we are going to apply our scaling up methodology specifically to instance and feature selection

algorithms, another important measure is the storage reduction of the original dataset achieved by the scaled selection algorithms. Efficiency and performance are closely related in this field, as the storage reduction is directly linked to the memory requirements evaluated. This proves the great importance of achieving a good balance between efficiency and performance.

(2) It is important that our scaling method is generalizable to any data mining algorithm, so as to allow a fast and direct adaptation of the scaling methodology to any other field that needs to deal with large datasets. Otherwise a new custom scaling method would have to be designed and implemented each time we wanted to apply it to a new problem. Over the years multiple specific methods have been developed (Sonnenburg et al., 2007; Yang et al., 2008; de Oca et al., 2010).

(3) In order to be valuable, it is a must for a scaling method to be simple. On the contrary, a very complex algorithm will be neither implemented nor actively used and improved by the community.

(4) Application of the proposed methodology to scale up relevant data mining algorithms which have well known scaling problems when managing large datasets.
   (a) Scaling up some popular instance selection algorithms. This is a challenging problem because this type of algorithms have important scaling issues.
   (b) Scaling algorithms in a relevant domain of data mining as it is the feature selection problem.

1.4 Proposed methodology: democratization of algorithms

The methodology proposed in this thesis to scale up data mining algorithms fits within the group of data partitioning methods. Data partitioning is more general than the other two methods because it does not modify the original algorithm and simply divides the original input set into smaller and more manageable subsets. Considering the high complexity of many data mining algorithms, the fact that our method treats the basic method as a *black box* is very interesting, as no modification of the original method is needed.

This paradigm is based on a combination of the *divide-and-conquer* philosophy with the ensembles of classifiers approach. Following the *divide-and-conquer* philosophy, we apply a data mining algorithm to subsets of the whole training set. A simple approach is to partition the dataset into disjoint subsets and apply the data mining algorithm to each subset separately. An example of this methodology is the use of stratified random sampling in instance selection (Liu et al., 2002; Cano et al., 2005; Derrac et al., 2010). We can consider the

execution of each data mining algorithm applied to a subset of the whole dataset as a "*weak* data mining algorithm". Extending classifier ensemble concepts (Rokach, 2009) we can carry out several rounds of weak data mining algorithms applied to disjoint subsets of the original dataset, these rounds would not achieve good results on their own but their results can be improved using a combination scheme that produces a final result. Furthermore, the divide-and-conquer methodology has the additional advantage that we can adapt the size of the subproblems to the available resources.

To sum up, we propose a scaling method that for each step i divides the dataset S into several small disjoint datasets $S_{i,j}$. A data mining algorithm is applied with no modifications to each one of these subsets, and a result $C_{i,j}$ is produced from each subset of the data. After all the n_s rounds are applied, (which can be done in parallel, as all of them are independent from each other), the combination method constructs the final result C as the output of the data mining process.

The key difference introduced by our approach is modifying the view we have of data partitioning. In previous methods, the application of the learning algorithm to a subset was considered as a valid result and a concept was learned from the subset. However, learning from just a subset of the original dataset suffers from locality and makes the algorithm more vulnerable to noise. Consequently, we consider that new concepts can only be learned from the combination of the application of the learning algorithm to the whole partition of the dataset into many subsets. This difference, though subtle, is the cornerstone of the success achieved by our method. It also differs from other approaches in repeating the data partitioning step many times, using the basis of the ensembles of classifiers, which combine several weak classifiers to produce an efficient result. In our case repeating many times the process assures we have gathered enough information to produce relevant knowledge.

The main advantage of our method is that it maintains good accuracy values while the time is reduced significantly, because the selection algorithm is applied only to small subsets whose results are afterwards strategically combined.

In fact, as the size of the subset is chosen by the researcher, we can apply the method to any problem regardless of its size. The size of the working dataset is strongly linked to the memory requirements of the method. On the one hand, some algorithms need the entire training set to be in memory during the execution, but keeping all data in memory is not always feasible and usually leads to the inefficiency of memory swapping. On the other hand, our method requires each subset to be in memory only while it is processed but it is not needed during the processing of the remaining subsets. In this way, our method is scalable both in time and storage requirements.

It is worth noting that unlike previous scaling methods, we are able to achieve this time and storage reduction while avoiding an unacceptable performance decrease. When scaling learning algorithms the decrease in performance is a very common side effect because we are trying to generalize from partial views of data. The key to a small decrease in the performance is a good design of the step that combines the progressively obtained results.

Moreover, our proposal is naturally parallelizable, as the application of the data mining algorithm to each small subset of instances or features is independent of all the remaining subsets and can therefore be processed at the same time.

Finally it is generalizable, as in the case of ensembles of classifiers where the base learner is a parameter of the algorithm. In our methodology the data mining algorithm is also a parameter, and therefore it can be used to scale up any algorithm without modifications.

1.5 Contribution of the thesis

In this thesis, we present a new method for scaling up data mining algorithms, which consists of performing several rounds of weak learners on independent subsets of the original dataset and combine their results into a final result. The proposed methodology is successfully applied to scale up various data mining tasks, including instance selection and feature selection. It is worth noting that our scaling approach is applicable to any data mining method without any modifications.

Our main objective is to design a method that is able to successfully scale up data mining algorithms, meaning that the running time as well as the accuracy will be considerably reduced and that the storage reduction will not drop to inadmissible values. The experiments will show if our method is able to shorten the execution time significantly compared to the original algorithms, while preserving a similar performance. In terms of storage requirements and testing error, the main aim of our approach is to be able to match or improve if it is possible the standard results applied to the non-partitioned datasets. These results will demonstrate if our approach is able to scale up large datasets successfully, where other methods are inapplicable.

Moreover, we will apply of our *democratization* methodology to scale up state of the art instance and feature selection algorithms, testing our methodology behavior on real-world datasets and huge problems.

The research performed during the development of this thesis had fruitful results in the form of presentations at international congresses (de Haro-García

and Pedrajas, 2010; de Haro-García et al., 2010) and the publication of several articles in outstanding journals in the area (de Haro-García and Pedrajas, 2009; García-Osorio et al., 2010; Pedrajas et al., 2011).

1.6 Thesis organization

This memory is organized as follows. Chapter 2 provides a detailed insight into the current situation of the scaling up problem and an analysis of the advantages and weaknesses found in some of the related works; Chapter 3 presents in depth of the proposed model to scale up data mining methods and includes a description of how to apply our methodology to different data mining problems; Chapter 4 and Chapter 5 detail the application of the democratic approach to scale up instance selection and feature selection algorithms, respectively. These chapters provide a discussion on the most relevant aspects in their implementation details as well as the experimental setup and the study of the results. Finally, Chapter 6 states the final conclusions of our work, the possible applications of the designed methodology and the future research lines.

2 Background

The scaling up problem appears in any algorithm when the data size increases beyond the capacity of the traditional data mining algorithms, harming their performance and efficiency. The scaling problem produces excessive storage requirements, increases time complexity and affects generalization accuracy, introducing noise and overfitting. These drawbacks are increased by the size of the dataset, more specifically:

- Efficiency: Most of data mining algorithms present an efficiency of at least $O(n^2)$. Consequently, when the size grows, the time needed by the algorithm also increases.
- Resources: Most of the algorithms assessed need to have the complete dataset stored in memory to carry out their execution. If the size of the dataset is too big, the computer would need to use the disk as swap memory. This loss of resources has an adverse effect on efficiency due to the increased access to the disk.
- Generalization: Algorithms are affected in their generalization capabilities due to the noise and overfitting effect introduced by larger size datasets.
- Convergence: If a data mining algorithm is applied to a large dataset it will usually result in a huge search space and subsequent convergence problems. Furthermore the search space will have many local minima where the data mining algorithm can be trapped.

Consequently, many data mining algorithms cannot even be applied to large datasets due to their inefficiency. These algorithms applied directly on full-sized datasets are low-performing and inefficient. On that account scaling methods are needed. Provost and Kolluri (1999) identified three fundamental an independent approaches for scaling up learning methods:

- Designing fast algorithms: includes a wide variety of algorithm design techniques for reducing the asymptotic complexity, for optimizing the search and representation, for finding approximate solutions instead of exact solutions, or for taking advantage of the inherent parallelism of the task. The major drawback of this approach is the need to modify the mining methods.
- Partitioning the data: involves breaking the dataset up into subsets, learning from one or more of these subsets, and possibly combining the results. Data partitioning is useful to avoid the thrashing by memory management systems and is able to make a learning algorithm's time complexity linear by processing small, fixed-size data subsets sequentially.
- Using a relational representation: addresses data that cannot feasibly be treated as a flat file, including any large relational database, as well as other large relational structures such as those used for knowledge representation in artificial intelligence. This approach requires as input, data with a high

level of abstraction and therefore it cannot be applied to many data mining problems because many of them do not meet that requirement.

It is worth noting that in this section we are going to describe only those previous works that proposed a generic methodology to scale up algorithms and are applicable to any data mining algorithm without severely modifying the original scaling method. We will not make a survey on the multiple specific scaling method that have been developed because the main aim of this thesis is the scaling of data mining algorithms in a generic way. Examples of interesting scaling algorithms that have been specifically designed for a single problem are (Lazarevic and Obradovic, 2002; Sonnenburg et al., 2007; Yang et al., 2008; de Oca et al., 2010).

Classification is a very popular research field and its scalability problems have been frequently analysed by the data mining community. An interesting proposal is the one by Lazarevic and Obradovic (2002) of a parallel boosting algorithm, specifically designed for tightly coupled shared memory systems with a small number of processors. Their parallel boosting algorithm is proposed primarily for learning from several disjoint data sites when the data cannot be merged together, although it can also be used for parallel learning where a massive dataset is partitioned into several disjoint subsets for a more efficient analysis. At each boosting round, the proposed method combines classifiers from all sites and creates a classifier ensemble on each site. The final classifier is constructed as an ensemble of all classifiers, that are combined according to the confidence of their prediction and built on disjoint datasets. Results from the experiments on one synthetic dataset and four datasets from the UCI machine learning repository (Hettich et al., 1998) indicate that distributed boosting has comparable or slightly improved classification accuracy over standard boosting, while requiring much less memory and computational time since it uses smaller datasets.

Sonnenburg et al. (2007) presented another specific scaling approach which enhanced performance of large scale learning with string kernels, specifically to the string kernels in the context of biological sequence analysis. Their algorithm, the *linadd algorithm*, iteratively selects a set of Q variables based on the quality of the current solution and then solves the reduced problem with respect to the working set of variables. This strategy sped up stand alone SVM training and it also reduced training times for multiple kernel learning. Their proposal was favourably evaluated on human splice and web spam datasets, they compared their linadd algorithm and its parallel version (parallelizing the most expensive part: evaluating g(x) function, based on multiple threads and gaining reasonable speed ups) to the original weighted degree kernel algorithm and spectrum kernel formulation. It is interesting to note that the linadd algorithm requires several transmissions of information on each iteration between distributed CPU's and also to the master node.

Due to their usually high complexities, evolutionary algorithms have serious scaling issues when dealing with datasets of large size. Over the years several specific scaling solutions have been designed to tackle this problem. As an example we can cite the work developed by Yang et al. (2008) proposing a cooperative coevolution framework to scale up a differential evolution algorithm applied to optimize separable and non-separable functions, the key to their cooperative coevolution method is a grouping and adaptative weighting strategy that produces good results when tested on classical benchmark functions and the functions provided by CEC2005 Special Session (Suganthan et al., 2005). Their experimental setup reports favourably to their cooperative coevolution method when compared to other cooperative coevolution optimization algorithms as well as to conventional evolutionary and self-adaptative algorithms.

Other interesting approach to scale up specifically evolutionary algorithms is the work of de Oca et al. (2010) in which they made extensive usage of automatic algorithm configuration methods to redesign and specialize a generic particle swarm based optimization algorithm, for tackling large-scale continuous optimization problems. They described the whole redesign process of an incremental particle swam optimizer with local search algorithm (IPSOLS) in six stages that among others include adding a conjugate directions method and a mechanism to deal with bound constraints. They presented a study of the scalability behavior of their tuned algorithm with datasets of 50, 100, 200, 500 and 1000 dimensions. They compared their tuned and non-tuned IPSOLS algorithm with the differential evolution algorithm (DE), the CHC algorithm and the G-CMA-ES algorithm. In particular, DE and G-CMA-ES are considered state of the art algorithms for continuous optimization problems. The IPSOLS algorithm was found to clearly outperform CHC and G-CMA-ES in its default and its tuned version. The time execution scalability of the tuned IPSOLS algorithm is quite good, probably as a consequence of the fact that Powell's conjugate directions method exploits the separability of the functions. However, there were no significant differences when compared to the differential evolution algorithm, but the authors claimed that a more refined restart criterion would solve those convergence problems.

The truth is that not many scaling methods that are truly generalizable to any data mining problem have been developed over the last few years. In this Related Work section we are going to provide an analysis of the most relevant ones that fulfil this generalization requirement.

Breiman (1999) proposed pasting votes to build many classifiers from small training sets or "bites" of data. He presented two strategies of pasting votes: Ivote and Rvote. In Ivote, the small training set (bite) of each subsequent classifier relies on the combined hypothesis of the previous classifiers, and the sampling is done with replacement. The sampling probabilities rely on the out-of-bag error, that is, a classifier is only tested on the instances not belonging

to its training set. This out-of-bag estimation gives good estimates of the generalization error and is used to determine the number of iterations in the pasting votes procedure. Ivote is, thus, very similar to boosting, but the "bites" are much smaller in size than the original dataset. Thus, Ivote sequentially generates training sets (and thus classifiers) by importance sampling. Rvote creates many random bites, and is a fast and simple approach.

Breiman found that Rvote was not competitive in accuracy to Ivote or Adaboost. Moreover, sampling from the pool of training data can entail multiple random disk accesses, which could swamp the CPU times. So Breiman proposed an alternate scheme: a sequential pass through the dataset. In this scheme, an instance is read and checked to see if it will make the training set for the next classifier in the aggregate. However, the sequential pass through the dataset approach led to a degradation in accuracy for a majority of the datasets. Breiman also pointed out that this approach of sequentially reading instances from the disk will not work for highly skewed datasets.

To deal with these problems, Chawla et al. (2004) distributed Breiman's algorithms, dividing the original dataset into T disjoint subsets and assigning each disjoint subset to a different processor. On each of the disjoint partitions sampled randomly, they followed Breiman's approach of pasting small votes. Chawla et al. combined the predictions of all the classifiers by majority vote. Again, using the above framework of memory requirement, if we break up the dataset into T disjoint subsets, the memory requirement will decrease by a factor of 1/T , which is substantial. So, DIvote can be more scalable than Ivote in memory. One can essentially divide a dataset into subsets easily managed by the computer's main memory.

Pasting DRvotes follows a procedure similar to DIvotes. The only difference is that each bite is a bootstrap replicate of size N. Each instance through all iterations has the same probability of being selected. DRvote is faster than DIvote as the intermediate steps: 4 and 5 in the above algorithm are not required. However, DRvote also provide worse accuracies than DIvote. This agrees with Breiman's observations on Rvote and Ivote.

Their proposal is evaluated on six small to moderate sized datasets, just one of them can be considered large. These are the publicly available datasets used by Breiman (1999); Lazarevic and Obradovic (2002) to facilitate direct comparisons. They did not use a fixed subset size but claimed that it can be dynamically adjusted to fit the available computational resources.

DIvotes and RVotes are compared with the original Ivote and Rvote algorithms, as well as with a single cascade correlation neural network (CC) and Lazarevic's distributed boosting. As expected DIvote and IVote are significantly better than a single CC and they also outperform DRvote and RVote.

DIvote achieved classification accuracies similar to the distributed boosting approach but required no inter-processor communication. Consequently there is no time lost in communication among processors, as trees are built independently on each processor.

Another interesting proposal is presented by Domingos and Hulten (2001a) whose approach to scaling up learning algorithms is based on Hoeffding bounds (Howffding, 1963). The method can be applied either to choose among a set of discrete models or to estimate 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. Moreover, 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. Finally, the experiments reported by Domingos and Hulten (2001b) showed that the dataset size must be several million instances for the method to be worthwhile.

The general framework proposed has been used for scaling up decision trees, Bayesian network learning, k-means clustering and the EM algorithm for mixtures of Gaussians. In a subsequent study Hulten and Domingos (2002) developed a method for inductive algorithms based on discrete search. To the best of our knowledge, the approach has not been applied to instance or feature selection yet.

One of the ways of solving complex problems is decomposing them into several simpler tasks that can be dealt with separately. An interesting proposal that follows this philosophy is the *stratified strategy*, originally applied to prototype selection methods in the context of data reduction. Stratification divides the initial dataset into disjoint strata with equal class distribution, as the prototypes are independent of each other, the distribution of the data into strata do not degrade their representation capabilities.

The number of strata chosen will determine their size, depending on the size of the dataset. Using the proper number of strata, we can significantly reduce the training set and we could avoid the drawbacks mentioned above. Following the stratified strategy, initial dataset D is divided into t disjoint sets D_j , strata of

equal size, $D_1, D_2, \ldots, D_t$ maintaining class distribution within each subset. Then, PS algorithms will be applied to each D_j obtaining a selected subset DS_j .

The test set TS will be the TR complementary one in D.

$TS = D/TR$

PS algorithms (classical or evolutionary ones) are applied to each D_j obtaining a subset selected DS_j. The prototype selected set is obtained using DS_j and it is called Stratified Prototype Subset Selected (SPSS).

$SPSS = \cup_(j \in J) DS_j, J \subset 1, 2, \ldots, t$

The last stage, where the DS_j are being reunited, is not time-consuming, as it does not present any kind of additional processing. The time needed for the stratified execution is the one associated to the instance selection algorithms execution in each strata.

We can use a preliminary approach of stratification (Cano et al., 2003) dividing the dataset into several disjoint subsets, and then apply an evolutionary algorithm model in which they have taken into account a specific instance selection perspectives to each subset. The best results came from the CHC algorithm but took a considerable amount of processing time. Moreover, in such a method, the solution obtained by each algorithm is contaminated by the partial view of the dataset.

In stratified random sampling within evolutionary algorithms a set of n instances is divided into k non-overlapping subsets of sizes $n_1, n_2, \ldots, n_k$, where $\sum_i n_i = n$. Each subset is called a *stratum*. Then a random sample is extracted from each stratum. The set of instances is stratified and each stratum is assigned to evolve following a genetic algorithm. In this way, the initial populations are constructed by stratified random sampling. The evolution of the individuals in each population optimizes the classification and storage requirements within the stratum.

An example of this type of stratification is (García-Pedrajas et al., 2004), sharing some of the ideas underlying *stratified random sampling* (Liu and Motoda, 2002). This method used cooperative coevolution to promote collaboration among the strata. In this way, another population is created that keeps track of the best combination of individuals so far and enforces cooperation among the individuals that evolve using instances of each stratum. The model allows interaction among the simpler searches that are carried out in each stratum, instead of performing a large search in the whole set. This is a way of achieving at least a partial problem decomposition. Several subpopulations evolve taking only into account the instances of a certain subset. The use of another

population that combines the results of each subpopulation is able to give the model the necessary global view to accomplish its task.

We consider stratification a general scaling method as it has been successfully applied to very diverse evolutionary algorithms in different instance selection problems over the years, without adapting the original scaling methodology to each problem.

Cano et al. (2005) presented an evolutionary stratified approach to solve the drawbacks introduced by the evaluation of large size datasets using evolutionary prototype selection algorithms. The stratification reduces the dataset size for algorithm runs, while evolutionary algorithms select the best local training subset.

This study included a comparison between their proposal (stratified CHC algorithm with a HUX recombination operator that performs re-seeding of the population) and other non-evolutionary prototype selection algorithms (Cnn, Drop1, Drop2, Drop3, Ib2, Ib3) combined with the stratified strategy.

In their experimental setup they selected six datasets from the UCI Repository, ranging from medium to huge size datasets. Although non-evolutionary algorithms were the most efficient than evolutionary ones, their result are worse. On the other hand, stratified CHC presents the best reduction rates while offering an accuracy rate similar to the 1-NN algorithm applied over the whole dataset. Although the algorithm shows very good performance, it is still too computationally expensive for huge datasets.

When the size of the databases increases, evolutionary algorithms suffer from the scaling problems previously described as well as its behavior deteriorates considerably because of a lack of convergence. Due to the fact that memetic algorithms combine a population-based algorithm with a local search, Garcia et al. (2008) proposed a model of memetic algorithm that incorporates an ad hoc local search specifically designed for optimizing the properties of prototype selection problem with the aim of tackling the scaling up problem. Their experimental study was carried out to establish a comparison between their stratified memetic algorithm version (SSMA) and previous evolutionary algorithms (CHC, GGA, IGA, PBIL, SSGA) as well as several non-evolutionary approaches (1-NN, Allknn, Cpruner, Drop3, Enn, Explore, Ib3, Pop, Rmhc, Rng, Rnn) studied in the literature.

SSMA outperforms the classical prototype selection algorithms, irrespectively of the scale of dataset. Those algorithms that could be competitive with it in classification accuracy are not so when the reduction rate is considered. Furthermore, it usually outperforms in test accuracy other methods that obtain good rates of reduction. Only CHC presents the same behaviour in small datasets, but when the problem scales up, SSMA again outperforms CHC.

Derrac et al. (2010) tested the combination of stratification with their previously published steady-state memetic algorithm for prototype selection in various problems, ranging from 50,000 to more than 1 million instances.

Their proposal, the SSMA-PS algorithm, interweaves global and local search phases that influences each other, choosing good starting points at the same time as aiming to an accurate representation.

They tested the performance of SSMA-PS on seven large datasets from the UCI Machine Learning Repository. They employed two well-known prototype selection methods for comparison: the classical Drop3 and one of the fastest prototype selection methods "Fast Condensed Nearest Neighbor" (FCNN), all of them were used by themselves and along with the stratification procedure in all the problems where the computational cost was not too high, to have a reference of how the use of stratification modifies their behavior.

They tried three representative sizes of strata: 1,000, 5,000 and 10,000 instances. The strata size of 10,000 instances offered the best results in accuracy (i.e. the best possible reduced subsets from the training sets), keeping a reasonable runtime in the training.

Both modes of SSMA-PS (with and without stratification) were able to converge, but the employment of stratification allowed SSMA-PS to converge faster, resulting in more stable behavior when compared to the execution when stratification is not employed. SSMA-PS has offered the best behavior over the large problems in the experimental study, allowing to reduce the size of the training sets without severely harming their inherent accuracy. The main drawback found for SSMA-PS is their large time consumption in the PS phase. However, authors claimed that this is compensated by its ability to generate smaller training sets than the rest of proposals, leading it to obtain lower execution times in classification phases.

To recapitulate, there are different scaling approaches, most of them have been specifically designed to solve a particular problem. We are mainly interested in generalizable approaches, as this is main aim of our thesis. Specific approaches require complex customizations to be applied to a problem different from the originally intended one. Unfortunately, there are not many truly generalizable scaling methods. It is noteworthy the stratification approach (Cano et al., 2003) and the one proposed by Hulten and Domingos (2002). The latter approach is inconveniently complex, presenting a troublesome application.

3 Scaling methodology: *democratization*

In the current chapter we are going to describe the key aspects of our methodology to scale up data mining methods, its basic steps and the basis of its application to some of the most used data mining methods: instance and feature selection. Finally we will describe the novelty aspects of our approach and the advantages of using it.

As previously indicated, this thesis proposes a general data scaling method that specifically can be classified into the data partitioning scaling methods. Following the *divide-and-conquer* philosophy, it applies separately the data mining algorithm to disjoint subsets of the original dataset. A key aspect of the partition of our method is that the sum of all the disjoint subsets result in the whole dataset. The reason for using this type of partition is considering that new concepts can only be learned from the totality of a dataset not just from a subset, so as to avoid locality and noise vulnerability.

Moreover, our scaling methodology is based on the ensembles of classifiers approach, which combines several weak classifiers to produce an efficient result. In our approach we do not have weak classifiers but weak data mining algorithms, each one applied to an independent and small-sized subset of the original dataset.

So as to fully understand the similarities between the ensembles of classifiers and our scaling methodology, in the following lines we provide an overview of the ensembles of classifiers.

An ensemble of classifiers consists of a combination of different classifiers, homogeneous or heterogeneous, to jointly perform a classification task. Ensemble construction is one of the fields of Artificial Intelligence that is receiving most research attention, mainly due to the significant performance improvements over single classifiers that have been reported with ensemble methods (Breiman, 1996c; Kohavi and Kunz, 1997; Bauer and Kohavi, 1999; Webb, 2000; García-Pedrajas et al., 2005).

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

The task is to find a definition for the unknown function, $f(\mathbf{x})$, given the

set of training instances. In a classifier ensemble framework we have a set of classifiers $\mathbb{C} = \{C_1, C_2, \ldots, C_m\}$, each classifier performing a mapping of an instance vector $\mathbf{x} \in \mathbb{R}^D$ onto the set of labels $Y = \{1, \ldots, K\}$. The design of classifier ensembles must face two main tasks: constructing the individuals classifiers, C_i, and developing a combination rule that finds a class label for $\mathbf{x}$ based on the outputs of the classifiers $\{C_1(\mathbf{x}), C_2(\mathbf{x}), \ldots, C_m(\mathbf{x})\}$.

For more detailed descriptions of ensembles the reader is referred to other reviews: Dietterich (2000b), Webb (2000), Dzeroski and Zenko (2004), Merz (1999), or Fern and Givan (2003).

Techniques using multiple models usually consist of two independent phases: model generation and model combination (Merz, 1999). Most techniques are focused on obtaining a group of classifiers which are as accurate as possible but which disagree as much as possible. These two objectives are somewhat conflicting, since if the classifiers are more accurate, it is obvious that they must agree more frequently. Many methods have been developed to enforce diversity on the classifiers that form the ensemble (Dietterich, 2000b). Kuncheva (2001) identifies four fundamental approaches: (i) using different combination schemes, (ii) using different classifier models, (iii) using different feature subsets, and (iv) using different training sets. Perhaps the last one is the most commonly used. The algorithms in this last approach can be divided into two groups: algorithms that adaptively change the distribution of the training set based on the performance of the previous classifiers, and algorithms that do not adapt the distribution. Boosting methods are the most representative methods of the first group. The most widely used boosting methods are AdaBoost (Freund and Schapire, 1996) and its numerous variants, and Arc-x4 (Breiman, 1998). They are based on adaptively increasing the probability of sampling the instances that are not classified correctly by the previous classifiers.

Bagging (Breiman, 1996a) is the most representative algorithm of the second group. Bagging (after *B*ootstrap *agg*regat*ing*) just generates different bootstrap samples from the training set. Several empirical studies have shown that AdaBoost is able to reduce both bias and variance components of the error (Breiman, 1996b; Schapire et al., 1998; Bauer and Kohavi, 1999). On the other hand, bagging seems to be more efficient in reducing bias than AdaBoost (Bauer and Kohavi, 1999). Although these techniques are focused on obtaining as diverse classifiers as possible, without deteriorating the accuracy of each classifier, Kuncheva and Whitaker (2003) failed to establish a clear relationship between diversity and ensemble performance.

Boosting methods are the most popular techniques for constructing ensembles of classifiers. Its popularity is mainly due to the success of AdaBoost. However, AdaBoost tends to perform very well for some problems but can

also perform very poorly on other problems. One of the sources of the bad behaviour of AdaBoost is that although it is always able to construct diverse ensembles, in some problems the individual classifiers tend to have large training errors (Dietterich, 2000a). Moreover, AdaBoost usually performs poorly on noisy problems (Bauer and Kohavi, 1999; Dietterich, 2000a).

In conclusion, ensembles of classifiers perform a learning task by means of combining weak but fast classifiers into a strong one. One of the key aspects to produce an efficient result is the combination strategy of the weak classifiers results. In our approach we do not have weak classifiers but weak data mining algorithms, each one applied to an independent and small-sized subset of the original dataset. The weak steps of data mining algorithms in our approach provide an efficient and accurate final result when their outputs are suitably combined.

The main difference introduced by our scaling methodology is that it uses the whole dataset for the data mining algorithms to learn, not restricting the learning process to a subset of the original dataset. Using just a subset of the whole dataset, as many scaling algorithms do, leads to less accurate results as not all the available information is used in the learning stage.

Furthermore, our methodology is able to scale up data mining algorithms without performing changes in the original scaling approach as it treats the data mining algorithms as black boxes, making our scaling proposal easier to apply and expand.

Moreover, not only do we succeed in reducing execution time but we also keep the decrease of performance under acceptable boundaries. When scaling algorithms we must not forget the original learning purpose because a scaled data mining algorithm with poor performance is useless as it does not achieve its goals

3.1 The Democratization methodology steps

Our scaling methodology can be summarized in the three following stages:

(1) Partition of the datasets.
(2) Application of the data mining algorithm to the subsets.
(3) Combination of the results.

For each of the iterations of our scaling methodology, i, which we call "rounds"‘ our approach divides the dataset S into several small disjoint datasets $S_{i,j}$. The data mining algorithm is applied with no modifications to each one of these subsets, and a result $C_{i,j}$ is produced from each subset of the data. After

all the n_s rounds are applied, (which can be done in parallel, as all of them are independent from each other), the combination method constructs the final result C as the output of the data mining process.

$\bigcup S_{ij} = TR$

$S_{ij} \cap S_{ik} = \emptyset, j \neq k$

$TS = S \setminus TR$

Two out of the three steps of our methodology are common when scaling any data mining algorithm. Partitioning the dataset is a generic process to any data mining algorithm (being the simplest option to perform a random partition and leaving the door open to the application of a more suitable partition method if the problem requires it) and so it is the application as a black box of a data mining algorithm to the created subsets, taking as inputs the subsets and providing as output the results in the same format as the original data mining algorithm provides them. The only stage specifically designed for the data mining algorithm is the last one: the combination of results, because each data mining algorithm provides its results in a specific format (eg. weights associated to instances, ranking of features)

It is worth highlighting that the addition of all the subsets in a round results in the whole dataset. Consequently our approach not only does not scale using just a subset of the original dataset but it actually learns from the full dataset several times, as many times as the value assigned to the number of rounds.

It is noteworthy that our scaling methodology only has two parameters to set: the subset size M and the number of iterations, which we call "rounds" r. We analysed the effect of both parameters on the performance of the different applications of *democratization* to data mining algorithms and we observed the following behavior.

The number of rounds should not be set to a very small figure, because there will be no chance for the data mining algorithm to learn, but after the first few rounds are added the testing error is not affected. As more rounds are added, the reduction in storage decreases because the threshold for removing an instance is higher and more rounds must agree to remove it. What is more, when many rounds are used, the execution time increases and a high portion of the votes are redundant so there is no advantage in having so many rounds, regarding testing error. This behavior is similar to the case of classifier ensembles, where little gain is obtained after the first few classifiers are added (García-Pedrajas et al., 2007)

A similar test was performed with the other parameter of our scaling approach, the size of the subsets. We have stated that the size of the subset is not relevant

provided it is kept small to obtain a significant reduction of execution time and large enough to allow a meaningful learning process. With a large subset size the reduction is greater but with no significant differences, being the processing time of each subset more important than the achieved performance. Fixing the subset size to a meaningful minimum makes the performance reach a good value, which is not incremented if we increase the subset size but that will result in a large increment in execution time. So the goal is to obtain a good compromise among the sizes that favor performance and execution time.

In the following lines, we are going to generically describe the steps that our scaling approach is comprised of, to subsequently explain its application to scale different data mining algorithms.

3.1.1 Partition of the datasets

The partition process consists of dividing the original dataset into several disjoint subsets of approximately the same size that cover the full dataset.

The first choice among partitioning methods is to use the simplest method available, i.e. strictly a random partition, where each feature or instance is randomly assigned to one of the subsets. Moreover, a random partition naturally keeps the class distribution of the original dataset over the different created subsets. However a more complex partitioning method can be chosen if the problem requires it. For example, in Section 4.4.1 we describe how to use the Grand Tour method to partition the original dataset when scaling up instance selection algorithms. It is remarkable that we need a different partition of the dataset for each round of the algorithm; otherwise, the results will be the same because the subsets will be identical.

It is worth noting that we do not consider the application of the data mining algorithm to a subset as a solution to the problem; in that way, the combination is not made at the round level. We chose random partition without replacement to avoid overlapping subsets and assure that every instance or feature will be evaluated just once per round. Equivalently, a standard sampling approach was also discarded because it may never pick some features or instances to be in any subset (depending on which factor is the basis of the partition). Our method takes into account all features or instances in each round. Therefore, we can state that each element of the entire dataset is present the same number of times in the learning/testing process: once per round.

These rounds will not select the best subset on their own because they only have partial knowledge of the original dataset. They can be seen as weak executions of a data mining algorithm that provide a fast approximate solution which suffers from locality, and moreover, they are more sensitive to noise.

Nevertheless, the key to the success of our method comes from the combination of all these rounds based on the quality of the subsets. This approach achieves a good performance globally while keeping the complexity and execution time manageable. The combination of rounds applied will be detailed further in this section.

The partition of the datasets can be carried out in two ways: we can choose to split datasets by instances or by features [1] . This decision should be closely related to the complexity of the data mining algorithm chosen to be the core of our method. Depending on the design, the complexity of a certain data mining algorithm may depend on the number of features, or, on the other hand, it may come from the number of instances in the dataset. That is the reason why it would be highly advisable to divide datasets by one or the other, depending on which factor determines the complexity of the algorithm involved. It is remarkable that the complexity of many of the most used data mining algorithms depends on the number of instances.

Preliminary experiments have shown that the size of the subsets has no significant impact on the results of the method, provided that it is small enough to avoid large execution times and large enough to allow a meaningful application of the data mining process. 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. The number of rounds experience a similar behavior, it is recommended to iterate for several rounds so as to allow the learning process, but not too many because it would lead to redundant knowledge.

3.1.2 Application of the data mining algorithm to the subsets

The data mining algorithm is applied to all the datasets in several iterations, which we call "rounds". This repetition ensures that we have gathered enough information for the combination step to be useful. The advantage of our method is not needing any modification of the learning algorithm, thus saving the time spent in adapting known algorithms. Although the same data mining algorithm will be applied to different subsets, it will produce different results that can be advantageously combined.

In classification, several weak learners are combined into an ensemble which is able to improve the performance of any of the weak learners isolated (García-Pedrajas et al., 2007). In the same sense as we talk of "weak learners" in a classifier ensemble construction framework, we can consider a data mining

[1] It is also possible to partition the datasets by features and instances together. In such a case, each subset will be formed by a subset of instances and a subset of features. This possibility has yet to be explored experimentally.

algorithm applied to a subset of the whole dataset as a "*weak*" data mining algorithm. Following a similar philosophy to the construction of ensembles of classifiers (Rokach, 2009) we can carry out several rounds of weak data mining algorithms applied to disjoint subsets of the original dataset, these rounds would not achieve good results on their own but their results can be improved using a combination scheme that produces a final result. Therefore, our approach is called *democratization*, and can be considered a form of extending classifier ensemble concepts to data mining algorithms. Moreover, *democratization* selection deals with the data mining algorithms as black boxes, so it is possible to include any learning algorithm with no modifications. Thus, the output of our method is dynamic, depending on which type of output was originally provided by the selection algorithm. Our methodology will provide the output in the same format as the original algorithm.

3.1.3 Combination of the results

The combination of results is a key step in our method. In the same sense as weak classifiers are combined in an ensemble, we combine simple executions of "weak" data mining algorithms. Generically, ensembles of classifiers combine the output of their weak classifiers to get the final result in three ways:

- Summing up the outputs of each classifier and posteriorly setting a threshold to obtain the final result.
- Each output of each weak classifier receives a vote of equal value and the instance is classified into the class that receives the majority of votes.
- Each output of each weak classifier receives a vote, but not all the votes have the same value but they are weighted by the quality of their classification. Each instance will be classified into the class that receives more votes.

Following the experience of classifier ensembles where complex methods have not achieved consistently better performance than simple ones (Kuncheva, 2001), we have opted for simple combination methods that would be further described in the subsequent sections. The first choice to combine "weak" stages of an ensemble-based methodology is a voting method, but more complex methods can be used if the problem requires it.

3.2 Democratization applied to different data mining problems

In this subsection we are going to discuss the peculiarities and possible implementations of the generic steps of our methodology when applied to scale up particular data mining problems. A detailed description and the results of

the application of the *democratization* methodology to instance selection and feature selection are provided in chapters 4 and 5, respectively.

3.2.1 Scaling instance selection algorithms

(1) Partition of the datasets.

The first thing that we must take into account is the fact that we need several different partitions of the dataset, as each round needs a different partition. Otherwise, the results will be the same as most instance selection algorithms are deterministic. Since we use k-NN evaluator, among other evaluators, its performance is directly related to the locality of instances within each partition. Moreover, in order to avoid a random assignation of weights to instances in each round, it is recommended to vary partitions smoothly between subsequent rounds. These requirements are achieved by the theory of Grand Tour (Asimov, 1985) that generates a continuous sequence of low dimensional projections of a high dimensional dataset. When using the Grand Tour method we force the class distribution of the resulting subsets to be the same as the one of the original dataset.

As far as the subset size to scale up instance selection algorithms is concerned, we have stated that the size is not relevant provided it is kept small, that is, of about a few hundreds or thousands of instances. With a larger subset size the reduction is somewhat smaller, but the differences are not significant.

(2) Application of the data mining algorithm to the subsets.

In each round we apply an instance selection algorithm to every subset of instances created in the previous step. To obtain an accurate view of the usefulness of our scaling method we recommend using successful state-of-the-art instance selection algorithms, for example DROP3 (Wilson and Martinez, 2000), ICF (Brighton and Mellish, 2002) algorithms, evolutionary algorithms such as CHC (Eshelman, 1990) or RNN (Gates, 1972; Cano et al., 2003).

The chosen instance selection algorithms are applied as blackboxes, so to begin with, any instance selection algorithm can be used. Any picked instance selection algorithm receives the same input, i.e. an independent subset of instances, and provides as a result a subset of the most relevant instances. Our approach applies the instance selection algorithm to each subset separately. The instances selected by the algorithm to be removed receive a vote. Then a new partition is performed and another round of votes is carried out.

(3) Combination of the results.

As previously stated, the votes for an instance to be removed will be accumulated over the different rounds. After the predefined number of rounds is made, we will have a number of votes for each instance and the

instances that have received a number of votes above a certain threshold are removed. On that account, determining the number of votes needed to remove an instance is an important issue in our method. As this threshold is closely linked to a specific dataset, it is not feasible to set a general pre-established value usable in any dataset. Consequently a good alternative is to automatically estimate the best value for the number of votes from the effect on a percentage of the training set. Moreover, automatically setting this parameter over the training set relieves the user from manually setting a threshold value for each dataset. The best value will be the one that minimizes the training error and the memory requirements to the possible extent.

3.2.2 Scaling feature selection algorithms

(1) Partition of the datasets.

To scale feature selection algorithms by our approach we need to partition the original dataset in disjoint subsets of approximately the same size that cover the full dataset. The partition of the datasets can be done splitting by instances or by features, depending on the complexity of the feature selection algorithm being used. We recommend to divide datasets by instances or by features depending on which factor determines the complexity of the chosen feature selection algorithm involved.

In this problem a good alternative is to use the simplest partition method available, ie. strictly a random partition, where each feature or instance is randomly assigned to one of the subsets. A standard sampling approach was discarded for the partition because it may never pick some features or instances (depending on which factor is the basis of the partition). Our method takes into account all features or instances in each round. Furthermore, we need a different partition of the dataset for each round of the algorithm; otherwise, the results will be the same because the subsets will be identical. These rounds can be seen as weak feature selectors that provide a fast approximate solution that suffer from locality but the key to the success of our method comes from the combination of all these rounds based on the quality of the selected features.

Previous experiments have proven that the size of the subsets do not determine the results of the method, as long as it is of moderate size allowing a fast but meaningful application of the feature selection process. An appropriate value when partitioning by instances is a few hundreds or thousands of instances and less than ten features when scaling by features.

(2) Application of the data mining algorithm to the subsets.

In each round we apply a feature selection algorithm to every subset of instances or features created in the previous step. To prove the effectiveness of our scaling methodology we must use some of the most used

and successful state-of-the-art feature selection algorithms, such as Focus (Fountain et al., 1991), Relief (Kira and Rendell, 1992), LVF (Liu and Setiono, 1996), SVM-RFE (Guyon et al., 2002) or a genetic algorithm selector (Yang and Honavar, 1998; Li et al., 2001).

Our scaling methodology applies the feature selection algorithm as a blackbox so we can pick any type of feature selection algorithm, no matter its output. All feature selection algorithms receive as input an independent subset of instances or features but may provide their output in different formats: a subset of relevant features or a ranking weighting the relevance of each of the features composing the dataset. During the rounds of this stage we store all the output values assigned to each feature, no matter the format, in order to compute their final value in the next step. In the following lines we specify the application of *democratization* to both types of feature selection algorithms.

(a) Feature subset selection: several rounds are performed and each of them creates a new partition and a subset of features are selected to be removed, receiving a vote. After a round is completed, the votes of each feature are accumulated and a new round begins.

(b) Ranking of features: if we use a method that outputs a ranking of variables as the base feature selector, the ranking assigned to each variable on every subset and round will be recorded in order to use it to compute the final ranks in the combination step.

(3) Combination of the results. The combination step approximation is different depending on the output format of the feature selection algorithm previously used. We are going to summarize how to combine the outputs of each round for feature subset selection algorithms and ranking of features.

(a) Feature subset selection: in a similar way to the combination step of the scaled instance selection, we would have an accumulated number of votes to each feature after the stipulated number rounds is performed. This number of votes represents the number of times that particular feature has been selected in a round to be removed. The features that have received a number of votes above a certain threshold are removed.

Setting that parameter is a determinant issue in our method, it is not possible to use a general value and apply that threshold in any situation as it is closely linked to the characteristics of a specific dataset. We automatically set this parameter to the value in the training data that minimizes the training error and the memory requirements to the extent possible, relieving the user from the complexity of fixing it manually.

(b) Ranking of features: to combine the different rankings of each round into a final ranking we simply compute the average ranking value for each feature and then sort them into a new rank. This new averaged rank is the final ranking achieved by the scaling methodology.

3.3 Complexity of the method

As the objective of our methodology is to scale up large to huge problems of data mining, the complexity of the method is a must. In this section we show how our algorithm is linear.

We divide the dataset into partitions of disjoint subsets of size s. Thus, the chosen data mining algorithm is always applied to a subset of fixed size, s, which is independent from the actual size of the dataset. The complexity of this application of the algorithm depends on the data mining algorithm we are using, but will always be small, as the size s is always small. Let K be the number of operations needed by the data mining algorithm to perform its task in a dataset of size s. For a dataset of n instances and a data mining algorithm whose complexity depends on the number of instances, we will perform this data mining process once for each subset that is n/s times, spending a time proportional to $(n/s)K$. It will equally occur if the complexity relies on the number of features, being this time n the number of features, and we split the original dataset by features. In either cases, the total time needed by the algorithm to perform r rounds will be proportional to $r(n/s)K$, which is linear, as K is a constant value.

Thus, the gaining in execution time would be greater as the size of the datasets is larger. If the complexity of the data mining algorithm is greater, the reduction of the execution will be even better. The method has the additional advantage of allowing an easy parallel implementation. As the application of the data mining algorithm to each subset is independent from all the remaining subsets, all the subsets can be processed at the same time, even for different rounds. Also, the communication between the nodes of the parallel execution is small.

As we have stated, two additional processes complete our methodology: the partition of the dataset and the combination of the results. Applying a random partition requires a complexity of $O(n)$, thereby keeping the complexity of the whole process linear. However, in the following sections of application of our scaling methodology to instance and feature selection we will detail the complexity of the combination step and also the complexity of any other partition method used.

Figure 1 shows an example of 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 $r = 10$ rounds of votes.

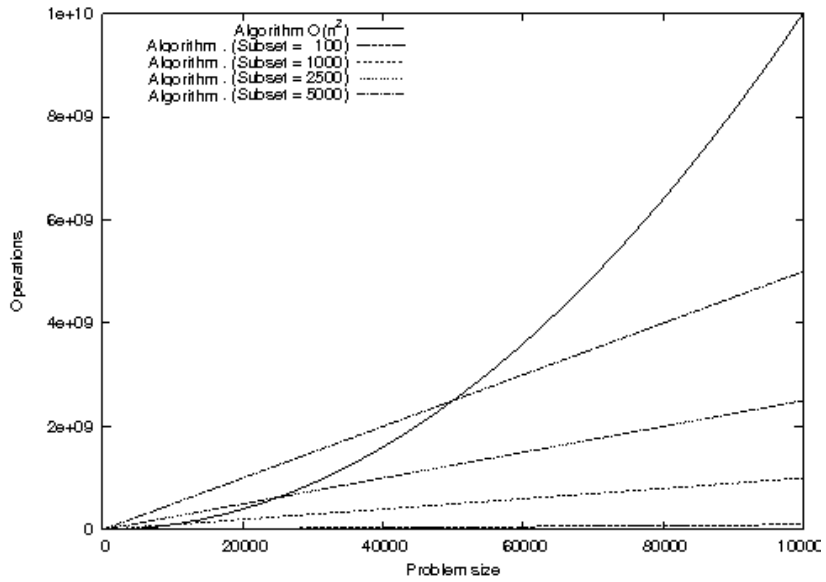


Fig. 1. Computational cost of our method dividing by instances and a base data mining algorithm of $O(n^2)$.

3.4 Contribution of the method

The main advantage of our method is that it maintains good accuracy values while the time is reduced significantly, because the data mining algorithm is applied only to small subsets whose results are afterwards strategically combined.

The key difference introduced by our approach is modifying the view we have of data partitioning. In previous methods, the application of the learning algorithm to a subset was considered as a valid result and a concept is learned from the subset. However, learning from just a subset of the original dataset suffers from locality and makes the algorithm more vulnerable to noise. Consequently, we consider that new concepts can only be learned from the combination of the application of the learning algorithm to the whole partition of the dataset into many subsets. This difference, though subtle, is the cornerstone of the success achieved by our method. It also differs from other approaches in repeating the data partitioning step many times, using the basis of the ensembles of classifiers, which combine several weak classifiers to produce an efficient result. In our case repeating many times the process assures we have gathered enough information to produce relevant knowledge.

It is worth noting that unlike previous scaling methods, we are able to achieve this time an storage reduction while avoiding an unacceptable performance decrease. When scaling learning algorithms, the decrease in performance is a

very common side effect because we are trying to generalize from partial views of data. The key to a small decrease in the performance is a good design of the step that combines the progressively obtained results.

In fact, as the size of the subset is chosen by the researcher, we can apply the method to any problem regardless of its size. The size of the working dataset is strongly linked to the memory requirements of the method. On the one hand, some algorithms need the entire training set to be in memory during the execution, but keeping all data in memory is not always feasible and usually leads to the inefficiency of memory swapping. On the other hand, our method requires each subset to be in memory only while it is processed but it is not needed during the processing of the remaining subsets. In this way, our method is scalable both in time and storage requirements.

Moreover, our proposal is naturally parallelizable, as the application of the data mining algorithm to each small subset of instances or features is independent of all the remaining subsets and can therefore be processed at the same time.

Finally it is generalizable, as in the case of ensembles of classifiers where the base learner is a parameter of the algorithm. In our methodology the data mining algorithm is also a parameter, and therefore it can be used to scale up any algorithm without modifications.

4 Scaling up instance selection algorithms

In this section we are going to thoroughly describe the application of the generic scaling methodology exposed in Section 3 to instance selection algorithms. First of all we will describe a recursive approach which is the precursor idea of our main methodology to scale up instance selection algorithms and we will end up giving some brief notes on its parallel version.

Instance selection is becoming more and more relevant due to the huge amount of data that is being constantly produced. However, although current algorithms are useful for fairly large datasets, scaling problems are found when the number of instances is of hundreds of thousands or millions, 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. In this section we present a way of avoiding this difficulty applying instance selection algorithms to subsets of the original dataset.

Our first approach to scale up instance selection algorithms was the design of the recursive method. The *recursive divide-and-conquer method* (de Haro-García and Pedrajas, 2009) obtained worse performance values compared with the standard algorithms. Based on the obtained results we improved the partition step and the philosophy of our scaling methodology, resulting in the DEMOIS. algorithm.

In the following lines we provide a brief description of the recursive method and its improvement, the DEMOIS. algorithm. The recursive method divides the original training set into small subsets where the instance selection algorithm is applied until a certain criterion is met. Then the selected instances are rejoined in a new training set and the same procedure, partitioning and application of an instance selection algorithm, is repeated. The proposed approach is able to match, and even improve the results concerning storage reduction of well-known standard algorithms, with a very significant reduction of execution time of the instance selection process. However, the main drawback of that method is in the testing error, which is worse than that obtained if we apply the original method alone. Moreover, its generalization to other mining algorithms is troublesome.

In order to improve the performance results we designed a new scaling method to scale up instance selection algorithms called DEMOIS. (García-Osorio et al., 2010) and based on the previous recursive approach. DEMOIS. also applies the chosen instance selection algorithm to disjoint subsets of instances of manageable size. However, instead of making just one step of partitioning at the beginning and progressively reducing the set of selected instances, DEMOIS.

makes several independent iterations of instance selection over disjoint subsets. The key difference between the two methods is that each independent iteration of DEMOIS. deals with the whole dataset. Consequently at the end of a DEMOIS. execution, each instance has been processed by the instance selection algorithm several times (as many as the number of iterations) in different contexts and partitions. DEMOIS. repeats the selection process more times than the previous recursive approach, having the chance to gather more information about the quality of each instance to the group. The outputs produced in the different iterations are suitably weighted by the accuracy quality and storage reduction to produce a reliable final selection of instances.

Accordingly to our theoretical studies, comparative experiments proved that DEMOIS. improves the testing error of our previous recursive approach in almost all of the problems. A pairwise comparison of both algorithms will be detailed in section 4.4.8.1. DEMOIS. is particularly efficient when we use instance selection algorithms that are high in computational cost. As previously stated, the proposed approach shares the philosophy underlying the construction of ensembles of classifiers, where several *weak* learners are combined to form a strong classifier. In a similar way our method uses several *weak* (in the sense that they are applied to subsets of the data) instance selection algorithms that are combined to produce a strong and fast instance selection method.

4.1 Preliminary concepts on Instance Selection Problems

The overwhelming amount of data that is available nowadays in any field of research poses new problems for data mining and knowledge discovery methods.

Selection seems a necessity in the world surrounding us. It stems from the sheer fact of limited resources. No exception for data mining. Many factors give rise to data selection. First, data is not purely collected for data mining or for one particular application. Second, there are missing data, redundant data, and errors during collecting and recording. Third, data can be too overwhelming to handle.

Instance selection, as another topic for data reduction, is recently getting more and more attention from researchers and practitioners. There are many reasons for this new trend: first, instance selection concerns some aspects of data reduction that feature selection cannot blanket; second, it is possible to attempt it now with advanced statistics and accumulated experience; and third, doing so can result in many advantages in data mining applications.

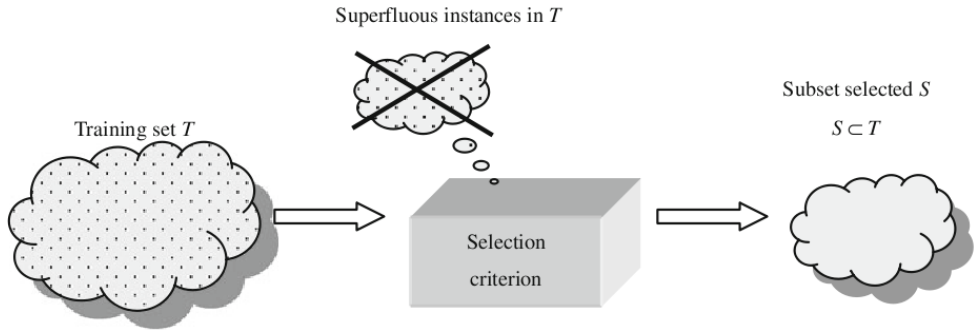


Fig. 2. Instance selection process

Instance selection is one avenue to the empire of data selection. Data is stored in a flat file and described by terms called attributes or features. Each line in the file consists of attribute-values and forms an instance, also named as a record, tuple, or a data point in a multi-dimensional space defined by the attributes.

As it can be seen in Figure 2, instance selection (Liu and Motoda, 2002) 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 used. Different variants of instance selection exist. Many of the approaches are based on some form of sampling (Cochran, 1977) (Kivinen and Mannila, 1994). There are other more modern methods that are based on different principles, such as, Modified Selective Subset (MSS) (Barandela et al., 2005), entropy-based instance selection (Son and Kim, 2006), Intelligent Multiobjective Evolutionary Algorithm (IMOEA) (Chen et al., 2005), and LVQPRU method (Li et al., 2005).

More often than not, we need to perform instance selection in order to obtain meaningful results. Instance selection has the following prominent functions (Liu and Motoda, 2002):

(1) Enabling: Instance selection offers many advantages dealing with large datasets. As we know, every data mining algorithm is somehow limited by its capability in handling data in terms of sizes, types, formats. When a dataset is too huge, it may not be possible to run a data mining algorithm or the data mining task cannot be effectively carried out without data reduction. Instance selection reduces data and enables a data mining algorithm to function and work effectively with huge data.
(2) Focusing: The data includes almost everything in a domain (recall that data is not solely collected for data mining), but one application is normally only about one aspect of the domain. It is natural and sensible to focus on the relevant part of the data for the application so that search is more focused and the mining is more efficient.

(3) Cleaning: The GIGO (garbage-in-garbage-out) principle applies to almost all, if not all, data mining algorithms. It is therefore paramount to clean data, if possible, before mining. By selecting relevant instances, we can usually remove irrelevant ones as well as noise and /or redundant data. The high quality data will lead to high quality results and reduced costs for data mining.

The above three main functions of instance selection may intertwine. For example, cleaning can sometimes be a by-product of the first two. Focusing can also serve a function of enabling under certain circumstances.

The problem of instance selection for instance based learning can be defined as (Brighton and Mellish, 2002) "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".

Applying instance selection is useful for reducing the runtime in the training process, particularly in the classification process for instance-based classifiers since to classify just one instance, these classifiers use the whole training set. Like in feature selection, according to the strategy used for selecting instances, we can divide the instance selection methods in two groups (Olvera-López et al., 2010):

- Wrapper: The selection criterion is based on the accuracy obtained by a classifier (commonly, those instances that do not contribute with the classification accuracy are discarded from the training set).
- Filter: The selection criterion uses a selection function which is not based on a classifier

4.1.1 Wrapper instance selection algorithms

Most of the wrapper methods have been proposed based on the k-NN classifier (Cover and Hart, 1967).

- Methods Based on NN Rules: One of the earliest methods is the Condensed Nearest Neighbor (CNN) (Hart, 1968). It tries to find a consistent subset, which correctly classifies all of the remaining points in the sample set. However, this algorithm will not find a minimal consistent subset and also can retain noisy instances.

  Another early instance selection method is the Edited Nearest Neighbor (Wilson, 1972) which is focused on discarding noisy instances in a training set. This method discards an instance in T when its class is different from the majority class of its k nearest neighbors. It leaves smoother decision boundaries but it also retains all internal points, which keeps it from reduc-

ing the storage requirements as much as most other reduction algorithms.

An extension of ENN is the RENN (Repeated ENN) (Wilson, 1972) method which repeatedly applies ENN until all instances in S have the same class that the majority of their k Nearest Neighbors.

The reduced NN rule (RNN) (Gates, 1972) searches in Cnn's consistent set, the minimal subset which correctly classifies all the learning instances. However, this approach is efficient if and only if Cnn's consistent set contains the minimal consistent set of the learning set, which is not always the case.

In Aha et al. (1991) the IB2 and IB3 (Instance Based) methods were proposed; they are incremental methods, IB2 is similar to Cnn but using a different selection strategy, it selects the instances misclassified by 1-NN (as CNN); IB3 is an extension of IB2 where a classification record is used in order to determine the instances to be retained (instances such that their deletion does not impact the classification accuracy).

ICF (Brighton and Mellish, 2002) tries to select the instances which classify more prototypes correctly. Icf uses coverage and reachable concepts to carry out the selection which are the associate and neighbor sets respectively. ICF discards p if $|Reachable(p)|>|Coverage(p)|$ which means that some instances in T can classify instances similar to p without considering it in the training set.

- Other wrapper methods are based on an ordered removal:

  Wilson and Martinez (1997) proposed five methods: DROP1, DROP2, DROP3, DROP4, DROP5 (Decremental Reduction Optimization Procedure); these methods are based on the concept of associate. The associates of an instance p are those instances such that p is one of their k nearest neighbors. DROP1 discards an instance p from T if the associates of p in S are correctly classified without p; through this rule, DROP1 discards noisy instances since the associates of a noisy instance can be correctly classified without it but in DROP1, when the neighbors of a noisy instance are first removed, then the noisy instance will not be discarded. In order to solve this problem, DROP2 is similar to DROP1 but the associates of an instance are searched in the whole training set, that is, p is deleted only if its associates in T are classified correctly without p. DROP3 and DROP4 first discard noisy instances using a filter similar to ENN and then they apply DROP2. DROP5 is based on DROP2 but it starts discarding the nearest enemies (nearest instances with different class) in order to smooth the decision boundaries.

  Evolutionary algorithms (EA) have been used to solve the IS problem, with promising results (Kuncheva, 1995), (Kuncheva and Bezdek, 1998), (Bezdek and Kuncheva, 2000), (Cano et al., 2003) EA are based on the natural evolution. Their main idea is as follows: given an initial chromosome population (set of solutions commonly represented by a binary-coded array, in our context, a set of instances) and according to a fitness function (reflecting the objective function value with respect to a particular objective function to be optimized,in the instance selection context the fit-

ness function is based on a classifier) the individuals from the population are evaluated, the best chromosomes are selected (which maximize the fitness function) in order to be muted and combined (crossover) for generating new chromosomes. The algorithm is repeated a specific number of iterations (generations) specified by the user and the best chromosome from the last generation is selected. The mutation operator introduces innovation into the population by generating variations of individuals and the recombination operator typically performs an information exchange between different individuals from a population. The selection operator imposes a driving force on the process of evolution by preferring better individuals to survive and reproduce when the members of the next generation are selected.

Examples of important EA to select instances are SGA and CHC. The Steady-State Genetic Algorithm (SGA): In SGAs (Whitley and Kauth, 1988), usually only one or two offspring are produced in each generation. Parents are selected to produce offspring and then a replacement strategy defines which member of the population will be replaced by the new offspring. A widely used combination is to replace the worst individual only if the new individual is better. CHC is one of the most used EA's due to its high performance. Heterogenious recombination and cataclysmic mutation (CHC), is a classical model that introduces different features to obtain a tradeoff between exploration and exploitation. CHC Adaptive Search Algorithm: During each generation the CHC (Eshelman, 1990) develops the following steps: it uses a parent population of size to generate an intermediate population of individuals, which are randomly paired and used to generate potential offspring and then, a survival competition is held where the best chromosomes from the parent and offspring populations are selected to form the next generation.

4.1.2 Filter instance selection algorithms

This section describes filter methods proposed in the literature; unlike wrapper methods they are not based on a classifier to determine the instances to be discarded from the training set. In a training set, an instance can be either a border instance or an interior instance. The border instances of a class provide useful information for preserving the class discrimination regions (Wilson and Martinez, 2000), (Brighton and Mellish, 2002) therefore some filter methods are focused on selecting border instances.

Riquelme et al. (2003) proposed the POP (Pattern by Ordered Projections) method that discards interior instances and selects some border instances. This approach is based on the weakness(p) concept which is defined as the numbers of times that p is not a border in a class (with respect to its attribute values). The selection rule discards irrelevant instances that according to this method, are those instances such that weakness(p) = m, where m is the total

number of features describing p. The weakness of an instance is computed by increasing weakness for each attribute where the instance is not near to another instance with different class.

A filter method based on kd trees (Friedman et al., 1977) was proposed in Narayan et al. (2006), where a binary tree is constructed. In the root node all the instances are included; for constructing each child node, a pivot is selected which is the feature with the maximum difference (Maxdiff) between consecutive ordered values; the left child contains those instances whose values for the corresponding attribute are lower than Maxdiff and the remaining instances are contained in the right node. The splitting process is repeated until the nodes cannot be split (the variance of the data along the features is greater than a threshold given by the user). Finally, instances located in the leaves are selected.

Some authors (Bezdek and Kuncheva, 2000), (Liu and Motoda, 2002), (Spillmann et al., 2006) have stated the idea of using clustering for instance selection; this idea consists of: after splitting T in n clusters, the selected instances will be the centers of the clusters.

In the literature, some filter methods consist in assigning a weight to the instances and selecting those with an acceptable weights range (according to a threshold). The WP (Weighting Prototypes) (Paredes and Vidal, 2000) method uses gradient descent for computing weights for each instance in terms of both nearest neighbors an nearest enemies; then the instances having weights larger than a certain threshold are removed.

4.2 Related Work on Large-Scale Instance Selection Problems

The main problem with existing instance selection algorithms is their excessive computational complexity, presenting in the best case an efficiency of $O(n^2)$, n being the number of instances. For huge problems, with hundreds of thousands or even millions of instances, these methods are not applicable. Trying to develop algorithms with a lower efficiency order is likely to be a fruitless search. Obtaining the nearest neighbor of a given instance is $O(n)$. To test whether removing an instance affects the accuracy of the nearest neighbor rule, we must measure the effect on the other instances of the absence of the removed one. Measuring this effect involves recalculating, directly or indirectly, the nearest neighbors of the instances. The result is a process of $O(n^2)$. In this way, the attempt to develop algorithms of an efficiency order below $O(n^2)$ is not very promising.

Thus, the alternative is reducing the size n of the set to which instance selection algorithms are applied. In the construction of ensembles of classifiers the problem of learning from huge datasets has been approached by means of learning many classifiers from small disjoint subsets (Chawla et al., 2004). In that paper, the authors showed that it is also possible to learn an ensemble of classifiers from random disjoint partitions of a dataset, and combine predictions from all those classifiers to achieve high classification accuracies. They applied their method to huge datasets with very good results. Furthermore, the usefulness of applying instance selection to disjoint subsets has also been shown in García-Pedrajas et al. (2010). In that work, a cooperative evolutionary algorithm was used. The training set was divided into several disjoint subsets and an evolutionary algorithm was performed on each subset of instances. The fitness of the individuals was evaluated only taking into account the instances in the subset. To account for the global view needed by the algorithm a global population was used. This method is scalable to medium/large problems but cannot be applied to huge problems. Zhu and Wu (Zhu and Wu, 2006) also used disjoint subsets in a method for ranking representative instances.

The usefulness of applying instance selection to disjoint subsets has also been shown in García-Pedrajas et al. (2010). 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/large problems but cannot be applied to huge problems.

There are not many previous works that have dealt with instance selection for huge problems. Cano et al. (2003, 2005) and (Garcia et al., 2008; Derrac et al., 2010) proposed an evolutionary stratified approach for large problems. Furthermore Kim and Oommen (2004) proposed a method based on a recursive application of instance selection to smaller datasets.

4.3 Recursive method

In previous sections we have thoroughly explained the scaling problems of most instance selection algorithms when applied to large to huge datasets due to their high computational complexities. Therefore the alternative is to reduce the number of instances n of the set to which instance selection algorithms are applied. Following that philosophy, we can develop a methodology based on applying the instance selection algorithm to subsets of the whole training set. A simple approach consists of using a stratified random sampling (Liu and Motoda, 2002) (Cano et al., 2005), where the original dataset is divided into many disjoint subsets, and then apply instance selection over each subset independently. However, due to the fact that to select the nearest neighbor of an instance we need to know the whole dataset, this method is not likely to produce good results. In fact, in practice its performance is poor. However, the divide-and-conquer principle of this method is an interesting idea for scaling up instance selection algorithms. Furthermore, divide-and-conquer methodology has the additional advantage that we can adapt the size of the subproblems to the available resources.

One way to improve that method is, instead of using a random partition of the dataset, to construct the subsets considering adjacent regions. In this way, the locality of nearest neighbor rule may work in our favor. This was our first choice and although it produces good results in terms of testing error, the storage reduction is still too small [2]. Consequently we decided to apply the above idea in a recursive manner so as to improve the storage reduction. After the first application of the instance selection algorithm the subsets of selected instances are rejoined and the method is repeated. This is the methodology we propose.

Our method is applicable to any instance selection algorithm, as the instance selection algorithm is a parameter of the method. First, our method divides the whole training set, T, into disjoint subsets, t_i, of size s such as $T = \bigcup_i t_i$. s is the only parameter of the algorithm. The way the dataset is divided is relevant, and it is explained in Section 4.3.2. Then, the instance selection algorithm of our choice is performed over every subset independently. The selected instances in each subset are joined again. With this new training constructed with the selected instances, the process is repeated until a certain stop criterion is fulfilled. The process of combining the instances selected by the execution of the instance selection algorithm over each dataset can be performed in different ways. We can just repeat the partition process as in the original

[2] Experiments with this method, not shown in this section, obtained testing error results better than standard instance selection algorithms but with a very limited storage reduction.

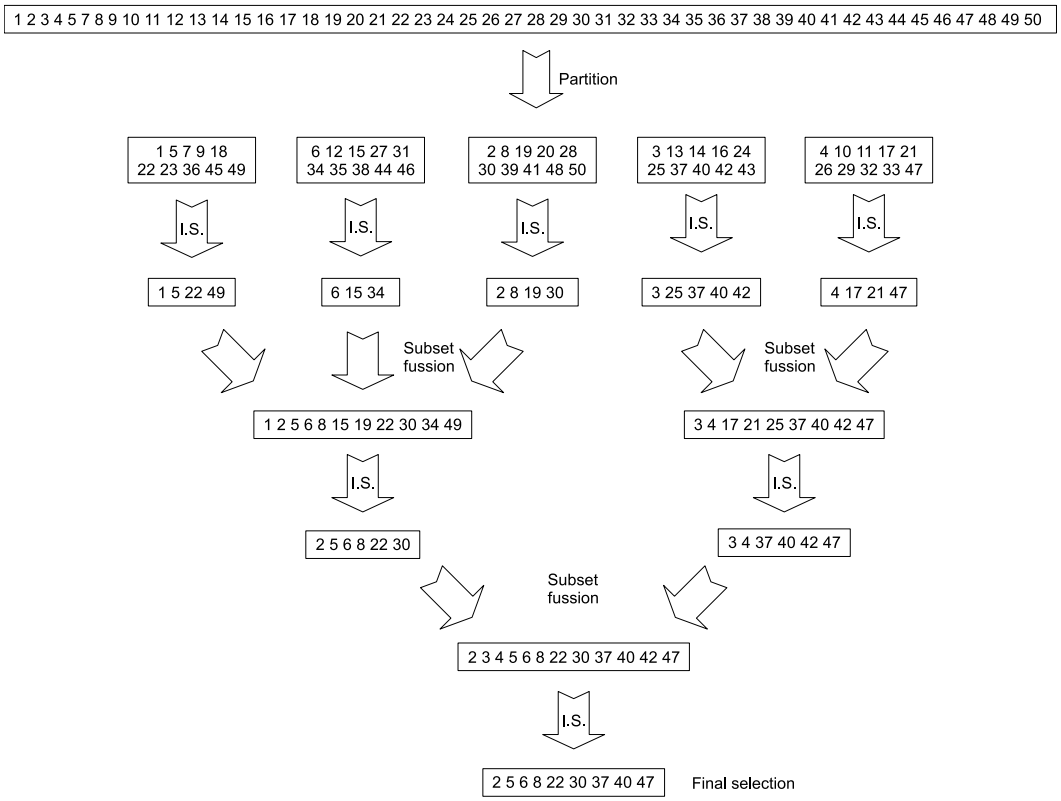


Fig. 3. Example of the method for a dataset of 50 instances and subsets of 10 instances. The instance selection (I.S.) algorithm can be any one of the many available methods.

dataset. However, as the first partition is performed using spatial properties of the instances (see Section 4.3.2) we can take advantage of this performed task. In this way, instead of repeating the partitioning process, we join together the subsets of selected instances until new subsets of approximately size s are obtained. An example of the whole algorithm is shown in Figure 3, and the detailed process is shown in Algorithm 1.

4.3.1 Stop criterion.

The stop criterion is of importance for the method. If the recursive process is repeated too many times, the reduction is too large and the testing error very poor. Fixing a number of iterations for every dataset is difficult, as it depends on the specific features of each problem. Thus, we use a cross-validation approach. We divide the training set into two parts, using one of them for performing the instance selection algorithm and the other one for obtaining the validation error. The number of iterations is obtained as the last iteration before the validation error starts to grow. Then, we perform the algorithm using the whole training set for the number of iterations obtained in the cross-validation process.

Algorithm 1: Recursive instance selection algorithm

Data : A training set $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, and subset size s.

Result : The reduced selector $S \subset T$.

1 $S = T$

2 Partitionate instances into disjoint subsets $t_i : \bigcup_i t_i = S$ of size s

repeat

  foreach *subset* $t_i \subset S$ **do**

3 Apply instance selection algorithm in t_i to obtain selected subset $s_i \subset t_i$

4 Remove from S instances removed from t_i

  end

5 Fusion subsets s_i to obtain new subsets t_j of size s

until *until stop criterion*

6 Return S

The most important advantage of our method is the large reduction in the execution time. The experiments show a large difference when using standard widely used instance selection algorithms. Additionally, the method is easy to implement in a parallel environment, as the execution of the instance selection algorithm over each subset is performed independently. Moreover, due to the fact that the disjoint subsets are of fixed size and independent from the actual size of the whole dataset, the complexity of our recursive method remains linear.

There is a last useful feature of the proposed approach. As the problem size grows the requirements for memory use also grow. For some problems it is not feasible to keep all data in memory due to the need to use the disk for memory swapping. However, existing methods need all the training set in memory during the execution of the algorithm. Our method requires each subset to be in memory only while it is processed, but it is not needed during the processing of the remaining subsets. In this way, our method is scalable both in time and storage requirements.

4.3.2 Partition of the dataset

The first step of our method is to partition 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 to avoid large execution time. Furthermore, the time spent by the algorithm highly depends on the size of the larger subset, so it is important that the partition algorithm would produce subsets of approximately equal size.

The most simple method is just a random partition, where each instance is randomly assigned to one of the subsets. However, k-NN is a local learning

algorithm, so a partition that produces, at least partially, subsets of instances near to each other is likely to produce better results. A simple method is dividing the input space into regions of equal size, and using the instances within each region as subsets. However, this method does not produce subsets of equal size. Another procedure based on the same idea can be constructed that produces subsets of equal size.

This procedure selects a random input, without replacement, and divides the set into two halves using the median of the values of the input, thus assuring that the two subsets are of the same size. The number of subsets must be a power of two. As obtaining the median is of $O(N)$, the partition can be made efficiently. The process is repeated using a new selected random variable until the desired number of subsets is produced.

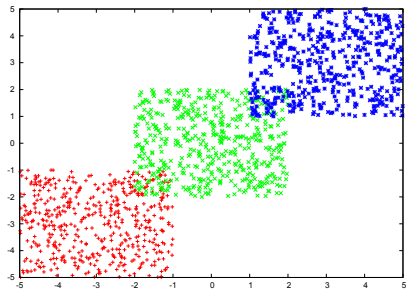
Algorithm 2: Algorithm for partitioning the training sets into disjoint subsets

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 $t_i : \bigcup_i t_i = T$.

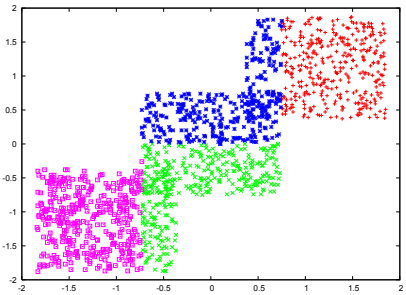
1 Adjust the number of subsets n_s to a power of 2 using size s
foreach *Class* **do**
  for $i = 0$ *to* $\log_2(n_s)$ **do**
2   Select an input j randomly without replacement
    for $k = 0$ *to* 2^i **do**
3   Divide every subset into two halves using median of input j within the subset
    end
  end
end
4 Return $t_i : \bigcup_i t_i = T$

The partition performed in a training set of 2000 instances is depicted in Figure 4 (b). The dataset, which is made up of three classes is shown in Figure 4 (a). The figure shows how the partition is local, in the sense that the instances in each subset correspond to adjacent instances in the space. This partition is performed only in the first iteration of the algorithm. In the subsequent steps (see Algorithm 1 step 5) the subsets are joined into new subsets. The figure also shows a problem with the proposed method, as the partition is performed without using class labels, some subsets have only instances of one class. This fact happens in the real-world datasets of our experiments. When an instance selection algorithm receives a subset with instances of only one class its performance is poor, as removing instances has no effect on the error of the nearest neighbor classifier.

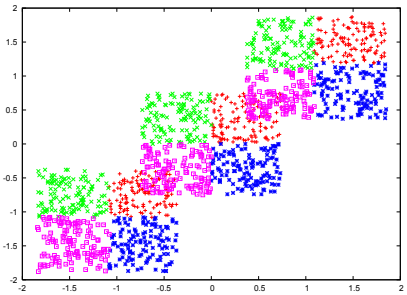
To avoid this effect we perform the partition taking class labels into account. The training set is divided into subsets that have approximately the same distribution of classes of the original set. This is accomplished applying the described algorithm to each class separately. Algorithm 7 shows the complete



(a) Original training set with three classes.



(b) Partition ignoring class labels into four subsets.



(c) Partition using class labels into four subsets.

Fig. 4. Example of the method for partitioning a dataset with 2000 instances, three classes and two features.

procedure for performing the partition. The result of such a partition, which is used in all the reported experiments, is shown in Figure 4 (c).

Partitioning in a similar way, using just one variable, has been used before in for learning ensembles of classifiers (Banfield et al., 2005).

4.3.3 Experimental setup

The evaluation of a certain instance selection algorithm is not a trivial task. We can distinguish two basic approaches: direct and indirect evaluation (Liu and Motoda, 2002). Direct evaluation evaluates a certain algorithm based exclusively on the data. The objective is to measure at which extent the selected instances reflect the information present in the original data. Some proposed measures are entropy (Cover and Thomas, 1991), moments (Smith, 1998), and histograms (Chaudhuri et al., 1998).

Indirect methods evaluate the effect of the instance selection algorithm on the task at hand. So, 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 learning, the most usual way of evaluation is estimating the performance of the algorithms on a set of benchmark problems. In those problems several criteria can be considered, such as Wilson and Martinez (2000): storage reduction, generalization accuracy, noise tolerance, and learning speed. Speed considerations are difficult to measure, as we are evaluating not only an algorithm but also a certain implementation. However, as the main aim of our work is scaling up instance selection algorithms, execution time is a basic issue. To allow a fair comparison, we have performed all the experiments in the same machine, a bi-processor computer with two Intel Xeon QuadCore at 1.60GHz.

To estimate the storage reduction and generalization error we used a k-fold cross-validation (cv) method. In this method, the available data are divided into k approximately equal sized subsets. Then the method is learned k times, using each of the k subsets in turn as the 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 $k = 10$.

We considered several options for the main statistical test choice: t-tests, sign tests and the Wilcoxon test. In the following lines we will explore the advantages and disadvantages of each choice and we will explain why the Wilcoxon test is the best option to fulfil our needs on this subject.

The use of t-tests (Anderson, 1984) for the comparison of several methods has been criticized in several papers (Dietterich, 1998). This test can provide an accurate evaluation of the probability of obtaining the observed outcomes by chance, but it has limited ability to predict relative performance even on further dataset samples from the same domain, let alone on other domains. Moreover, as more datasets and algorithms are used, the probability of type I error, a true null hypothesis incorrectly rejected, increases dramatically. Multiple comparison tests can be used in order to circumvent this last problem, but these tests are not usually able to establish differences between the algorithms.

To avoid these problems several authors perform a sign test on the win/draw/loss record of the two algorithms across all datasets. If the probability of obtaining the observed results by chance, the p-value of the sign test, is below 5%, they conclude that the observed performance is indicative of a general underlying advantage to one of the algorithms with respect to the type of learning task used in the experiments.

Nevertheless, the comparison using sign tests has two problems: Firstly, the differences between the two algorithms compared must be very marked for the test to find significant differences (Demšar, 2006); secondly, on some occasions the p-value of the test can be above or below the critical value due to a single modification of the outcome of one experiment, making the result of the test

less reliable. So, as the main test we have used the Wilcoxon test for comparing pairs of algorithms for several reasons (Demšar, 2006). The Wilcoxon test assumes limited commensurability. It is safer than parametric tests since it does not assume normal distributions or homogeneity of variance. Thus, it can be applied to error ratios, storage requirements and execution time. Furthermore, empirical results (Demšar, 2006) show that it is also stronger than other tests.

The formulation of the test (Wilcoxon, 1945) is the following: Let d_i be the difference between the results of the two methods on i-th dataset. These differences are ranked according to their absolute values; in case of ties an average rank is assigned. Let R^+ be the sum of ranks for the datasets on which the second algorithm outperformed the first, and R^- the sum of ranks where the first algorithm outperformed the second. Ranks of $d_i = 0$ are split evenly among the sums:

$$R^+ = \sum_{d_i>0} \text{rank}(d_i) + \frac{1}{2} \sum_{d_i=0} \text{rank}(d_i), \tag{1}$$

and,

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

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

$$z = \frac{T - \frac{1}{4}N(N+1)}{\sqrt{\frac{1}{24}N(N+1)(2N+1)}} \tag{3}$$

is distributed approximately according to $N(0,1)$.

It is worth to note that we perform a pairwise comparison between each standard instance selection algorithm and its recursive counterpart. We have chosen this strategy because we think it provides very direct comparative results between algorithms.

Our approach is based on applying data mining algorithms (specifically a feature selection algorithm in this situation) to subsets of the training set, so to perform sound experiments, the algorithm used for the whole training set and the algorithm used in our method are exactly the same. That is, when we applied our method using a feature selection algorithm and when we perform the feature selection algorithm for the whole training set, the implementation is the same in both cases. The source code, written in C and licensed under

the GNU General Public License, that is used for all methods as well as the partitions of the datasets are freely available upon request to the authors.

In order to make a comprehensive comparison between the standard algorithms and our proposal we have selected a set of 30 problems from the UCI Machine Learning Repository (Hettich et al., 1998). A summary of these datasets is shown in Table 1. We have selected datasets with, at least, 1000 instances.

Table 1
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 as it depends not only on the number of features but also on their type.

	Data set	Cases	Features			Classes	Inputs	1-NN error
			C	B	N			
1	abalone	4177	7	-	1	29	10	0.8034
2	adult	48842	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	28056	6	-	-	18	6	0.4356
9	kr vs. kp	3196	-	34	2	2	38	0.0828
10	letter	20000	16	-	-	26	16	0.0454
11	magic04	19020	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	12960	-	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	10992	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	58000	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

For estimating the storage reduction and generalization error we used a k-fold cross-validation 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 $k = 10$. The table shows the generalization error of a 1-NN classifier, which can be considered a baseline measure of the error of each dataset. These datasets are representative of problems from medium to large size.

4.3.4 Standard algorithms for the comparison

Our model is tested against three of the most successful state-of-the-art algorithms. We have used the classical algorithms DROP3 (Wilson and Martinez, 2000), and ICF (Brighton and Mellish, 2002). DROP3 (*Decremental Reduction Optimization Procedure 3*) is shown in Algorithm 3. This algorithm represents one of the examples of a new generation of algorithms that were designed taking into account the effect of the order of removal on the performance of the algorithm. So, this algorithm is designed to be insensitive to the order of presentation of the instances. It includes a noise filtering step using a method similar to Wilson's *Edited Nearest-Neighbor* Rule (Wilson, 1972). Then, the instances are ordered by the distance to their nearest neighbor. The instances are removed beginning with the instances furthest from its nearest neighbor. This tends to remove the instances furthest from the boundaries first.

ICF is shown in Algorithm 4. For ICF algorithm *coverage* and *reachability* are defined as follows:

$$Coverage(c) = \{c' \in T : LocalSet(c)\} \tag{4}$$
$$Reachable(c) = \{c' \in T : LocalSet(c')\}. \tag{5}$$

The Local-set of a case c is defined as "the set of cases contained in the largest hypersphere centred on c such that only cases in the same class as c are contained in the hypersphere" (Brighton and Mellish, 2002). In Case Base Reasoning (CBR) framework (Smyth and Keane, 1995) a case c can be adapted to a case c' if c is relevant to the correct prediction of c'. That means that c is a member of the neighborhood of c', bounding the neighborhood of c' by the first instance of a different class (see (Brighton and Mellish, 2002) for details). The algorithm is based on repeatedly applying a deleting rule to the set of retained instances until no more instances fulfill the deleting rule.

The concept of reachable and coverage sets used by ICF are similar to the neighborhood and associate sets used by RT algorithms (Wilson and Martinez,

Algorithm 3: DROP3 algorithm

Data : A training set $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$.
Result : The reduced selector $S \subset T$.

$S = T$
Noise filtering: Remove any instance in S missclassified by its k neighbors
Sort instances in S by distance to their nearest enemy
foreach *Instance* $P \in S$ **do**
    Find $P.N_{1..k+1}$, the $k+1$ nearest neighbors of P in S
    Add P to each of its neighbors' list of associates
end
foreach *Instance* $P \in S$ **do**
    Let `with` = # of associates of P classified correctly with P as a neighbor
    Let `without` = # of associates of P classified correctly without P
    if `without` $\geq$ `with` **then**
        Remove P from S
        foreach *Associate* A *of* P **do**
            Remove P from A's list of nearest neighbors
            Find a new nearest neighbor for A
            Add A to its new neighbor's list of associated
        end
    end
end

1997). The difference is that the sets defined in ICF are not of fixed size, but bounded by the first instance belonging to another class. This difference is considered *crucial* by the authors of ICF.

We have chosen ICF and Drop3 as representative of the methods we can consider "classical", as they have been around for quite a long time and are widely used. In the experimental section we compare the performance of these two methods when they are applied to the whole training set, we will call this application just ICF and Drop3, with the application of both methods using our recursive approach.

As an alternative to these "classical" methods, evolutionary computation algorithms have been applied for instance selection, considering this task to be a search problem. Evolutionary computation (EC) (Holland, 1975) (Goldberg, 1989) (Michalewicz, 1994) is a set of global optimization techniques that have been widely used in the last few years for almost every problem within the field of Artificial Intelligence. In evolutionary computation a population (set) of individuals (solutions to the problem faced) are codified following a code similar to the genetic code of plants and animals. This population of solutions is evolved (modified) over a certain number of generations (iterations) until the defined stop criterion is fulfilled. Each individual is assigned a real value that measures its ability to solve the problem, which is called its *fitness*.

Algorithm 4: Iterative Case Filtering (ICF) algorithm.

Data : A training set $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$.
Result : The reduced selector S.

```
1  S = T
2  Noise filtering: Remove any instance in S missclassified by its k neighbors
   repeat
       forall x ∈ S do
3          Compute reachable(x)
4          Compute coverage(x)
       end
5      progress = false
       forall x ∈ S do
           if |reachable(x)| > |coverage(x)| then
6              Flag x for removal
7              progress = true
           end
       end
       forall x ∈ S do
           if x flagged for removal then
8              S = S − {x}
           end
       end
   until not progress
```

In each iteration new solutions are obtained combining two or more individuals (crossover operator) or randomly modifying one individual (mutation operator). After applying these two operators a subset of individuals is selected to survive to the next generation, either by sampling the current individuals with a probability proportional to their fitness, or by selecting the best ones (elitism). The repeated processes of crossover, mutation and selection are able to obtain increasingly better solutions for many problems of Artificial Intelligence.

The application of evolutionary computation to instance selection 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 Kuncheva (1995), Ishibuchi and Nakashima (2000) and Reeves and Bush (2001).

Brighton and Mellish (2002) argued that the structure of the classes formed by the instances can be very different, thus, an instance selection algorithm can have a good performance in one problem and be very inefficient in another. They state that the instance selection algorithm must gain some insight into

the structure of the classes to perform an efficient instance selection. However, this insight is not usually available or very difficult to acquire, especially in real-world problems with many variables and complex boundaries between the classes. In such a situation, an approach based on EC may be of help. The approaches based on EC do not assume any special form of the space, the classes or the boundaries between the classes, they are only guided by the ability of each solution to solve the task. In this way, the algorithm learns the relevant instances from the data without imposing any constraint in the form of classes or boundaries between them.

Thus, one of the most interesting advantages of the application of evolutionary computation to instance selection is that evolutionary approaches do not depend on specific classifiers, and can be used with any instance based classifier. This is in contrast with most standard instance selection algorithms that are specifically designed for k-NN classifiers. For instance, Reeves and Bush (2001) used a genetic algorithm to select instances for RBF neural networks.

Cano et al. (2003) performed a comprehensive comparison of the performance of different evolutionary algorithms for instance selection. They compared a generational genetic algorithm (Goldberg, 1989), a steady-state genetic algorithm (Whitley, 1989), a CHC genetic algorithm (Eshelman, 1990), and a population based incremental learning algorithm (Baluja, 1994). 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 major problem that has to be 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 datasets. As we are mainly concerned with this problem, we have used as third instance selection method a genetic algorithm using CHC methodology. The execution time of CHC is clearly longer than the time spent by ICF and Drop3, so it gives us a good benchmark to test our methodology on an algorithm that has an important scalability problem.

4.3.5 Experimental results

The same parameters were used for the standard version of every algorithm and its application within our methodology. For Drop3 and ICF we used $k = 3$ neighbors, and for CHC we used $k = 1$. For our model we used a subset size of 100 instances. Cross-validation was used for stopping the selection using a random 10% of the training set as validation set. For CHC we used a population of 100 individuals that were evolved over 100 generations. The

evaluation of the individuals was made considering two criteria: reduction of storage, ρ, and classification error, ϵ. The fitness of individual j, F_j, is given by:

$$F_j = w(1 - \epsilon) + (1 - w)\rho, \tag{6}$$

where $0 \leq w \leq 1$. The weight w is needed to avoid a negative effect that may occur due to the asymmetry of the two values of the fitness function: the reduction value can be made arbitrarily high, up to the maximum value 1, by just removing more instances. In our experiments $w = 2/3$. The fitness value is the usual one in applying genetic algorithms to instance selection (Cano et al., 2003).

Table 2
Testing error, storage requirements and execution time (in seconds) for standard Drop3 algorithm and our approach. Mean and standard deviation values are shown.

Dataset	Drop3					Recursive Drop3				
	Storage		Error		Time	Storage		Error		Time
	Mean	SD	Mean	SD		Mean	SD	Mean	SD	
abalone	0.3069	0.0027	0.7782	0.0979	1.8	0.0730	0.0326	0.7767	0.0804	0.6
adult	0.1248	0.0011	0.1714	0.0051	22853.9	0.0570	0.0023	0.1977	0.0158	112.3
car	0.2668	0.0135	0.2378	0.0955	1.9	0.1665	0.0232	0.2680	0.0886	0.2
gene	0.3877	0.0067	0.2776	0.0230	35.6	0.0989	0.0231	0.3186	0.0295	1.2
german	0.3073	0.0080	0.2870	0.0743	0.9	0.0500	0.0228	0.3170	0.0508	0.3
hypo	0.0514	0.0041	0.0610	0.0111	11.8	0.0113	0.0033	0.1180	0.0812	0.9
isolet	0.2852	0.0025	0.1770	0.0403	208.9	0.1956	0.0054	0.2233	0.0494	2.3
krkopt	0.4431	0.0032	0.4803	0.0098	1533.0	0.1950	0.0036	0.5178	0.0091	13.1
kr vs. kp	0.2229	0.0069	0.1016	0.0171	10.1	0.1243	0.0076	0.1693	0.0159	0.7
letter	0.1744	0.0008	0.1037	0.0053	1849.8	0.3675	0.0128	0.1161	0.0116	17.4
magic	0.1789	0.0083	0.1978	0.1333	199.8	0.0599	0.0248	0.2533	0.0807	4.8
mfeat-fac	0.1208	0.0052	0.0600	0.0175	39.3	0.1499	0.0229	0.0630	0.0123	2.3
mfeat-fou	0.2473	0.0077	0.2320	0.0423	5.6	0.1347	0.0282	0.2815	0.0250	0.9
mfeat-kar	0.1655	0.0064	0.0835	0.0292	6.5	0.1594	0.0258	0.0930	0.0310	1.0
mfeat-mor	0.2062	0.0043	0.2885	0.0315	1.4	0.1138	0.0368	0.3435	0.0303	0.3
mfeat-pix	0.1095	0.0031	0.0480	0.0155	76.5	0.1442	0.0260	0.0560	0.0171	2.6
mfeat-zer	0.2231	0.0041	0.2375	0.0236	4.0	0.1467	0.0301	0.2390	0.0231	0.8
nursery	0.2934	0.0049	0.3327	0.0454	337.4	0.1274	0.0191	0.2612	0.0660	3.0
optdigits	0.0911	0.0027	0.0420	0.0069	161.0	0.1825	0.2453	0.0453	0.2865	3.0
page-bl	0.0430	0.0020	0.0437	0.0078	15.7	0.0170	0.0035	0.0859	0.0247	1.0
pendigits	0.0451	0.0017	0.0168	0.0028	175.0	0.1737	0.0073	0.0135	0.0044	3.2
phoneme	0.1852	0.0020	0.1383	0.0147	11.5	0.0675	0.0089	0.2645	0.0266	0.8
satimage	0.1366	0.0034	0.1101	0.0130	57.7	0.0828	0.0035	0.1608	0.0162	1.8
segment	0.1219	0.0076	0.0784	0.0204	4.1	0.1261	0.0206	0.1087	0.0314	0.6
shuttle	0.0028	0.0008	0.0016	0.0006	7543.4	0.0197	0.0047	0.0063	0.0017	20.9
sick	0.0625	0.0045	0.0509	0.0127	14.8	0.0216	0.0091	0.1056	0.0657	1.0
texture	0.0878	0.0018	0.0329	0.0062	97.0	0.2411	0.0073	0.0320	0.0075	2.2
waveform	0.2961	0.0043	0.2276	0.0286	28.8	0.0951	0.0287	0.2352	0.0223	1.5
yeast	0.3193	0.0087	0.4500	0.0253	0.6	0.0777	0.0243	0.4642	0.0274	0.3
zip	0.1040	0.0022	0.0497	0.0062	601.7	0.1824	0.0064	0.0532	0.0074	4.9

Table 2 shows the results using Drop3 as base algorithm. These results are plotted in Figure 5. The testing error of our approach is slightly worse than the error obtained using the standard algorithm. The results of storage re-

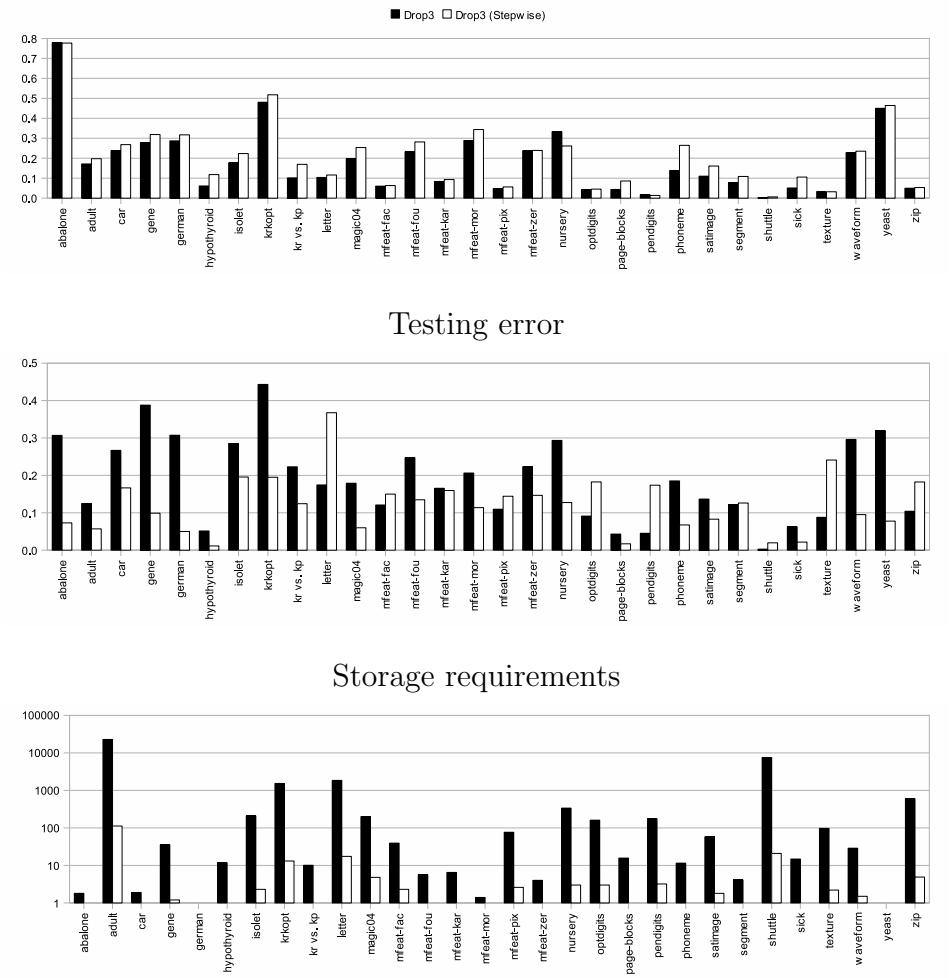


Fig. 5. Testing error, storage requirements and execution time (in seconds) for standard Drop3 algorithm and our approach.

quirements are favorable to our method which is able to significantly improve the results of standard Drop3. In fact, for abalone, car, gene, german, isolet, krvskp, krkopt, magic, nursery, phoneme, waveform and yeast datasets, our method is even able to show a clear improvement over standard Drop3. However, it is in execution time that our algorithm shows its better face. The reduction in the time spent is very marked, with an extreme case in shuttle dataset where our algorithm took less than 0.3% of the time needed by standard Drop3. As a summary, we can say that for Drop3 our procedure is able to improve the storage reduction of Drop3, with a worse testing error but with a large reduction of execution time. Wilcoxon test finds significant differences in terms of testing error (p-value of 0.0001) in favor of standard Drop3, and in terms of storage (p-value of 0.0104) in favor of our approach. Regarding standard deviation, our method has higher values, but still within moderate limits.

Table 3 shows the results using ICF as base algorithm. These results are plotted in Figure 6. Our method is able to improve the testing error of standard ICF, with a better average error in 18 of the 30 datasets. However, the differences are not statistically significant (p-value of 0.8774). On the other hand, storage requirements are clearly better in our approach. In this way, Wilcoxon test finds significant differences between both algorithms at a confidence level of 95% (p-value of 0.0148). The differences in execution time are, as in the previous case, clearly marked as is shown in Figure 6. As a summary, we can say that for ICF our procedure is able to match the testing error of ICF, with better average performance in terms of storage reduction, and with a large reduction of execution time. As was the case for Drop3, there are several datasets, namely abalone, adult, car, gene, german, krkopt, krvskp, mfeat-mor, nursery, waveform and yeast, for which the increment in storage reduction is clearly marked. In terms of standard deviation, for storage reduction our algorithm achieves worse results, although the differences are small, and for testing error the deviations of both methods are similar.

Table 3
Testing error, storage requirements and execution time (in seconds) for standard ICF algorithm and our approach. Mean and standard deviation values are shown.

Dataset	ICF					Recursive ICF				
	Storage		Error		Time	Storage		Error		Time
	Mean	SD	Mean	SD		Mean	SD	Mean	SD	
abalone	0.2510	0.0047	0.8082	0.0865	1.7	0.0461	0.0187	0.7990	0.0680	0.8
adult	0.1082	0.0009	0.2194	0.0032	9170.8	0.0275	0.0119	0.2125	0.0420	85.0
car	0.3813	0.0557	0.2709	0.1407	1.1	0.1787	0.0239	0.3180	0.0891	1.0
gene	0.2508	0.0080	0.3527	0.0146	26.1	0.0634	0.0048	0.4641	0.0273	2.0
german	0.1485	0.0093	0.3260	0.0544	0.4	0.0116	0.0044	0.3943	0.0869	1.4
hypo	0.0398	0.0029	0.1156	0.0356	3.7	0.0129	0.0714	0.0696	0.0100	0.9
isolet	0.1713	0.0028	0.2648	0.0541	103.1	0.0949	0.0040	0.2751	0.0578	2.3
krkopt	0.5290	0.0073	0.4032	0.0068	1109.8	0.0786	0.0027	0.6210	0.0096	7.4
kr vs. kp	0.2707	0.0080	0.1267	0.0330	5.3	0.0463	0.0045	0.2132	0.0125	1.0
letter	0.1362	0.0025	0.2018	0.0154	760.3	0.1650	0.0062	0.1947	0.0084	7.3
magic	0.1160	0.0073	0.2395	0.1499	138	0.0214	0.0029	0.2581	0.1290	2.9
mfeat-fac	0.0896	0.0052	0.0905	0.0171	17.5	0.1296	0.0125	0.0828	0.0176	3.2
mfeat-fou	0.1395	0.0055	0.3280	0.0256	2.4	0.0681	0.0078	0.4194	0.0340	1.6
mfeat-kar	0.1035	0.0056	0.1725	0.0334	2.5	0.1154	0.0084	0.1580	0.0328	1.2
mfeat-mor	0.2008	0.0088	0.3685	0.0563	0.6	0.0964	0.0201	0.3229	0.0384	0.7
mfeat-pix	0.0864	0.0047	0.1000	0.0218	27	0.1172	0.0089	0.0669	0.0160	2.9
mfeat-zer	0.1503	0.0062	0.2715	0.0193	1.7	0.0793	0.0074	0.2970	0.0223	1.3
nursery	0.8752	0.0095	0.2414	0.1407	287.2	0.2932	0.0446	0.2291	0.1139	3.7
optdigits	0.0606	0.0023	0.1103	0.0258	82.8	0.1104	0.0061	0.0626	0.0083	3.7
page-bl	0.0307	0.0027	0.2185	0.0128	5.6	0.0557	0.0482	0.0569	0.0133	1.3
pendigits	0.0348	0.0018	0.0651	0.0094	70.6	0.2071	0.0113	0.0251	0.0097	2.7
phoneme	0.1392	0.0031	0.1941	0.0192	4.6	0.0391	0.0102	0.2774	0.0199	0.7
satimage	0.0713	0.0023	0.1677	0.0259	25.4	0.0699	0.0039	0.1622	0.0216	2.2
segment	0.1077	0.0060	0.1394	0.0285	1.6	0.1118	0.0091	0.1361	0.0258	0.7
shuttle	0.0229	0.0026	0.0473	0.0129	2640	0.1902	0.0814	0.0034	0.0007	102.1
sick	0.0452	0.0045	0.0912	0.0190	4.7	0.0103	0.1325	0.1244	0.0101	1.1
texture	0.0725	0.0030	0.0973	0.0149	46.8	0.1695	0.0100	0.0578	0.0124	2.0
waveform	0.1211	0.0050	0.2840	0.0231	15	0.0240	0.0112	0.2818	0.0264	1.6
yeast	0.2137	0.0079	0.5095	0.0358	0.3	0.0259	0.0102	0.5453	0.0771	0.1
zip	0.0497	0.0019	0.2549	0.0182	219.8	0.0923	0.0067	0.1793	0.0224	4.5

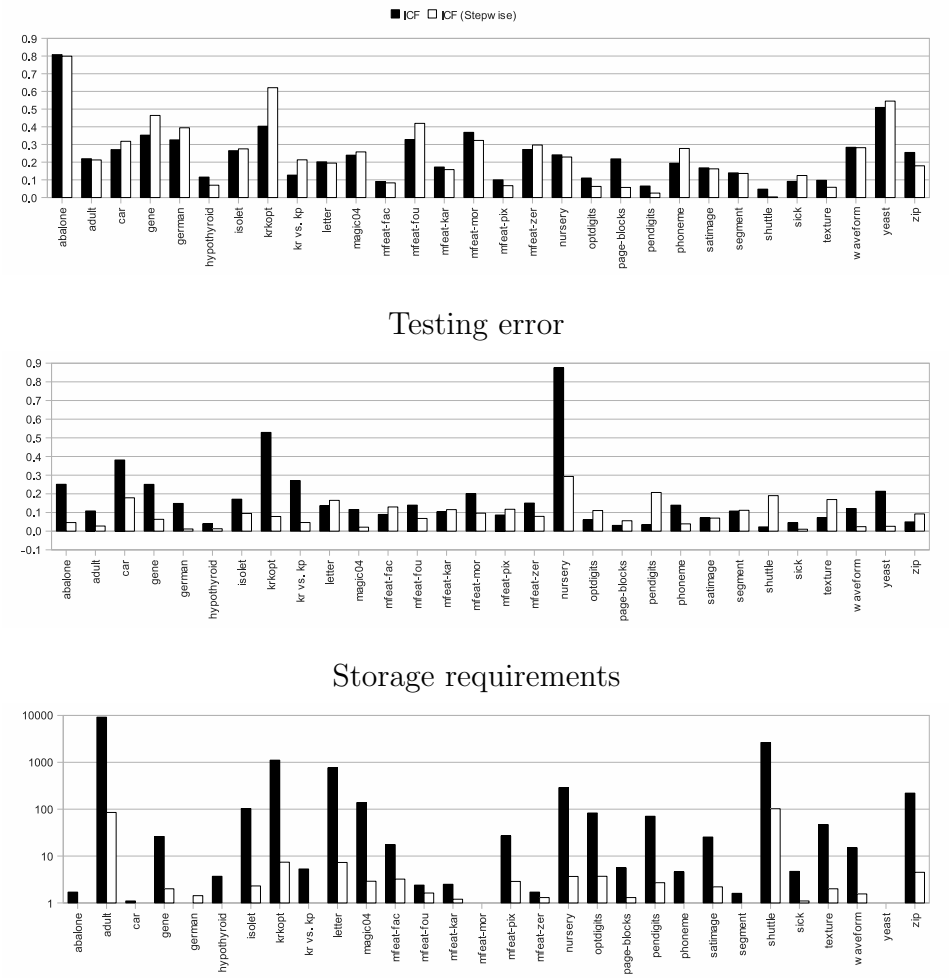


Fig. 6. Testing error, storage requirements and execution time (in seconds) for standard ICF algorithm and our approach.

Table 4 shows the results using CHC genetic algorithm as base method. These results are plotted in Figure 7. A first interesting result is the problem of scalability of CHC algorithm, which is more accused for this algorithm than for Drop3 and ICF. In other works (Cano et al., 2003) (García-Pedrajas et al., 2010), CHC algorithm is compared with Drop3 and ICF in small to medium problems. For those problems, the performance of CHC was better than the performance of Drop3 and ICF. However, as the datasets are larger, the scalability problem of CHC manifests itself. In our set of problems, CHC clearly performs worse than Drop3 and ICF and takes considerably more execution time. We must take into account that for CHC we need a 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 10000 bits, making the convergence of the algorithm problematic.

Regarding the comparison with our method, the behavior of CHC is similar to the behavior of Drop3. Our method is able to significantly improve the storage reduction (p-value of the Wilcoxon test of 0.0000), although the testing error is worse (p-value of 0.0001). The reduction in execution time is dramatic, with our method usually running in about 1% of the time needed by the standard method. As a summary, we can say that for CHC our procedure is able to improve the storage reduction of CHC, with a worse testing error, but with a very large reduction in execution time. The storage reduction is specially marked. Our method performs better, and with marked differences, in all 30 datasets. The table also shows that our method has a slightly worse deviation than standard CHC.

Table 4
Testing error, storage requirements and execution time (in seconds) for standard CHC algorithm and our approach. Mean and standard deviation values are shown.

Dataset	CHC					Recursive CHC				
	Storage		Error		Time	Storage		Error		Time
	Mean	SD	Mean	SD		Mean	SD	Mean	SD	
abalone	0.3818	0.0209	0.7998	0.0584	7722.2	0.1718	0.0511	0.8202	0.0387	263.9
adult	0.1988	0.0030	0.2257	0.0060	91096.0	0.1393	0.0025	0.2104	0.0057	1201.0
car	0.4192	0.0181	0.2639	0.0598	5483.6	0.1225	0.0354	0.2663	0.0690	45.1
gene	0.3004	0.0175	0.2968	0.0288	7236.6	0.1364	0.0483	0.3640	0.0324	101.2
german	0.3483	0.0424	0.3290	0.0614	440.1	0.0976	0.2775	0.3430	0.2074	32.2
hypo	0.2613	0.0187	0.0775	0.0100	7183.9	0.0997	0.0485	0.0923	0.0174	84.5
isolet	0.2993	0.0186	0.2026	0.0433	10885.4	0.1824	0.0022	0.2252	0.0401	409.7
krkopt	0.5237	0.0056	0.4711	0.0066	137015.0	0.1701	0.0032	0.5045	0.0053	6258.7
kr vs. kp	0.2712	0.0274	0.1276	0.0195	7107.5	0.1393	0.0085	0.1687	0.0178	79.6
letter	0.2952	0.0243	0.0905	0.0168	51024.0	0.1778	0.0036	0.1106	0.0059	1161.2
magic	0.2952	0.0591	0.1225	0.0275	29327.0	0.1041	0.0556	0.3109	0.1253	644.3
mfeat-fac	0.3933	0.0357	0.0455	0.0136	6518.4	0.1052	0.0443	0.0735	0.0239	57.7
mfeat-fou	0.3384	0.0238	0.2280	0.0361	7026.9	0.1552	0.0294	0.2810	0.0296	80.1
mfeat-kar	0.3838	0.0448	0.0650	0.0170	7010.8	0.1500	0.0092	0.0945	0.0151	66.5
mfeat-mor	0.3573	0.0480	0.3265	0.0364	7054.2	0.1296	0.0339	0.3410	0.0335	66.6
mfeat-pix	0.3759	0.0037	0.0440	0.1058	6441.9	0.1259	0.0347	0.0645	0.0121	62.9
mfeat-zer	0.3439	0.0080	0.2205	0.0045	7076.3	0.1422	0.0276	0.2515	0.0424	73.0
nursery	0.2941	0.0147	0.2427	0.0073	22397.3	0.1183	0.0417	0.2752	0.0815	326.0
optdigits	0.2755	0.0136	0.0404	0.0048	7846.4	0.1395	0.0030	0.0504	0.0090	170.7
page-bl	0.2786	0.0126	0.0408	0.0113	7662.8	0.1344	0.0250	0.0558	0.0116	126.1
pendigits	0.2903	0.0033	0.0121	0.0128	11636.6	0.1357	0.0029	0.0170	0.0073	288.6
phoneme	0.2846	0.0633	0.1457	0.0198	7772.9	0.1395	0.0032	0.1726	0.0095	173.0
satimage	0.2825	0.0102	0.1157	0.0235	8491.8	0.1392	0.0029	0.1263	0.0137	127.3
segment	0.3030	0.0293	0.0649	0.0115	7062.4	0.1400	0.0087	0.1078	0.0264	71.5
shuttle	0.2638	0.0169	0.0055	0.0062	77089.0	0.1428	0.0025	0.0016	0.0006	1137.3
sick	0.2578	0.0187	0.0514	0.0113	7179.8	0.1115	0.0336	0.0597	0.0134	80.1
texture	0.2825	0.0260	0.0249	0.0243	7876.1	0.1425	0.0074	0.0415	0.0097	148.8
waveform	0.2911	0.0035	0.2878	0.0062	7926.8	0.0954	0.0532	0.3118	0.0307	206.6
yeast	0.3711	0.0087	0.5014	0.0253	2981.1	0.1363	0.0584	0.5439	0.0465	50.3
zip	0.2871	0.0022	0.0510	0.0062	9748.2	0.1539	0.0052	0.0574	0.0065	371.7

The behavior of the standard algorithms and our approach in terms of execution time in function of the number of instances is illustrated in Figure 8. We plot the time spent by the algorithms as a function of the number of instances. The figure shows that standard methods have an execution time that is approximately quadratic with respect to the number of instances, for Drop3 and

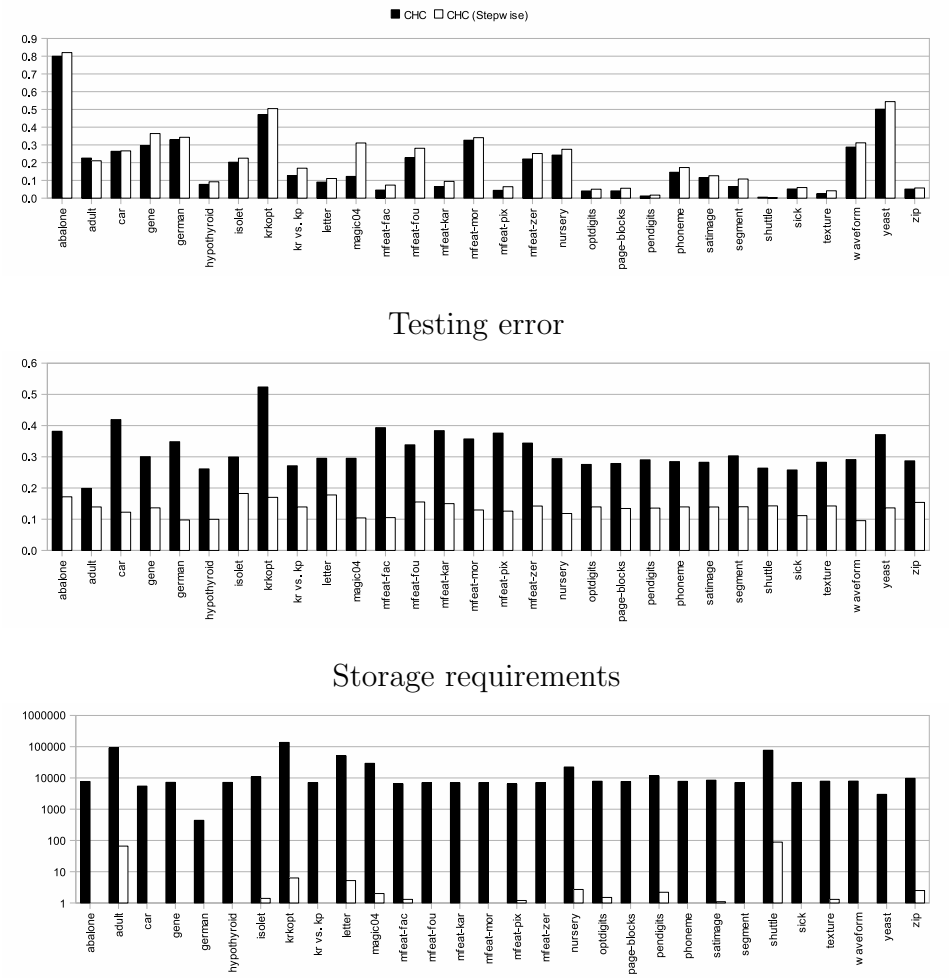
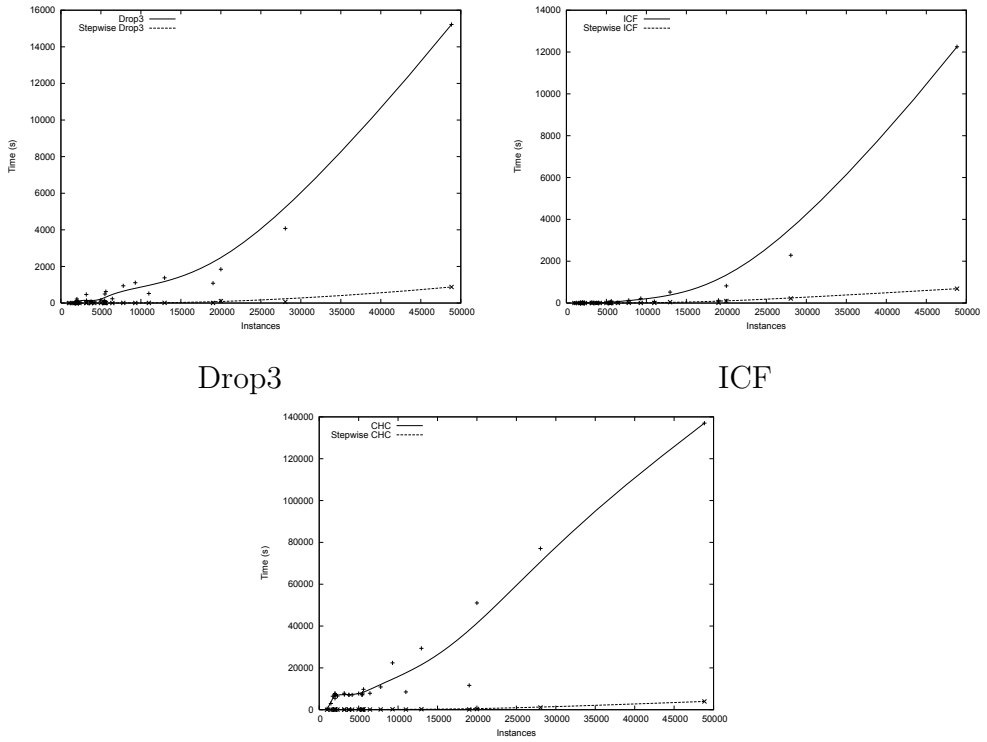


Fig. 7. Testing error, storage requirements and execution time (in seconds) for standard CHC genetic algorithm algorithm and our approach.

ICF, and even higher for CHC. So, for large problems the necessary time is substantial. On the other hand, our proposal is approximately linear, allowing the use of the methods even with hundreds of thousands of instances, as will be shown in Section 4.3.5.3.

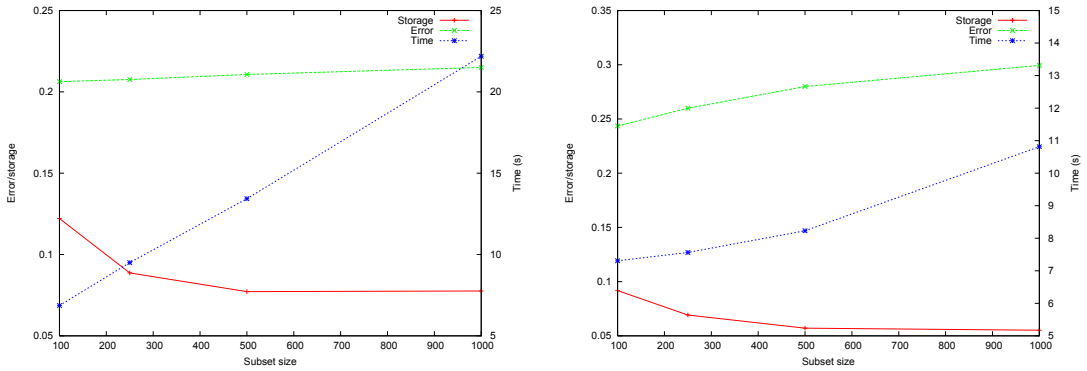
4.3.5.1 Study of subset size effect We have stated that the size of the subset is not relevant provided it is kept small, that is, of about a few hundreds of instances. However, this statement must be corroborated. We have performed experiments using Drop3 and ICF and subsets sizes of 250, 500 and 1000 instances. Figure 9 shows the average values of testing error, storage requirements and execution time with the four sizes. The trend is similar for both algorithms. As the size grows, the algorithm is more efficient in removing instances, achieving larger reductions to the storage. However, that reduction



Drop3 ICF

CHC

Fig. 8. Execution time (in seconds) for standard methods and our approach in function of the number of instances.



Drop3 ICF

Fig. 9. Average testing error, storage requirements and execution time (in seconds) as a function of subset size.

comes at the cost of deteriorating testing error, which is larger. Nevertheless, the increment in the reduction is larger than the increment in the error, so if we are mainly interested in storage reduction we can use a larger subset size, trading testing error for reduction.

Regarding execution time, the effect is clear. As the subset size grows the execution time grows. As the size is larger, the $O(N^2)$ of the algorithms begins

to be relevant, and the processing time of each subset is more important than the reduction of the number of subsets to process. However, even for 1000 instances as subset size the reduction in execution time is still dramatic when compared to the standard algorithm.

4.3.5.2 Improving testing error In previous sections, we have shown that our methodology is able to improve the performance of standard instance selection algorithms in terms of storage requirements with a very significant reduction in execution time. However, for Drop3 and CHC algorithms, there is also a worsening in terms of testing error. We have developed two mechanisms for improving testing error. These mechanisms try to ameliorate the effect that the execution of the algorithm over disjoint subsets may have.

The first method is very simple. Before each step, we add a certain percentage of the instances removed in the previous step to the pool of selected instances. In our method this percentage is of 10%. With this method we try to avoid the damaging effect of removing too many instances.

The second method is more elaborated. The main source of deteriorated testing error in our method is the limited view that each execution of the algorithm has, due to the fact that it is applied to a small subset of the whole dataset. In this way, useful instances may be removed if they are not relevant in the subset they belongs to in a step of the algorithm. To avoid this effect we use a *"second chance"* approach. We maintain for the whole dataset a list of marked instances. When the instance is selected for removing by the selection algorithm, it is removed only if it is marked in the list. Otherwise, the instance is marked but not removed. If the instance is marked, and not selected to be removed in the last step of the algorithm, it is unmarked. In this way, an instance is removed only if two consecutive steps of the method select it for removing. The chances of removing useful instances are decreased, as they must be selected for removal in two consecutive steps of the algorithms. The process, combining both methods, is shown in Algorithm 5.

In the first step, due to the large number of initial instances, the possibility of removing useful instances is small, so initially all the instances are marked. It means that all instances marked for removal in the first step of the algorithm are actually removed. Table 5 shows the results using both mechanisms, random addition of removed instances and second chance. These mechanisms are specially efficient for Drop3 and ICF. In this set of experiments our method is as good as Drop3 in terms of testing error and significantly better than ICF (p-values for the Wilcoxon test of 0.1650 and 0.0387 respectively). However, the cost is a worse storage reduction. For CHC, the techniques have a more limited effect. The testing error of our method is improved, but it is still significantly worse than the standard CHC algorithm.

Algorithm 5: Recursive instance selection algorithm with second chance and random addition of instances

Data : A training set $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, percentage of random addition p, and subset size s.

Result : The reduced selector $S \subset T$.

```
1  S = T
2  Partitionate instances into disjoint subsets t_i : ⋃_i t_i = S of size s
3  Mark all instances
   repeat
4      Add random p% of unselected instances to the set of selected instances
       foreach subset t_i ⊂ S do
5          Apply instance selection algorithm in t_i to obtain selected subset s_i ⊂ t_i
           forall Instances x selected for removing in t_i do
               if x is marked then
6                  Remove x from t_i
               else
7                  Mark x
               end
           end
           forall Instances x NOT selected for removing in t_i do
8              Unmark x
           end
9          Remove from S instances removed from t_i
       end
10     Fusion subsets s_i to obtain new subsets t_j of size s
   until until stop criterion
11 Return S
```

The experiments show that a better testing error can be achieved using second chance and a percentage of random addition of instances. It is up to the researcher to decide whether it is better for his/her application to focus either on testing error or storage reduction. Furthermore, other techniques may be developed, such as limiting the reduction performed by the application of the instance selection algorithm to a subset.[3] Table 6 shows the standard deviation of this method using the three algorithms of instance selection. In the table we can see that not only the testing error is improved, but also the deviation of the experiments is smaller. In fact, using second chance and random addition of instances, the deviation is very close to the results using the standard algorithms.

[3] In fact, this procedure has been used with results similar to the presented second chance method.

Table 5
Testing error, storage requirements and execution time (in seconds) for the recursive approach using both mechanisms, random addition of removed instances and second chance, of improving testing error.

Dataset	Recursive Drop3			Recursive ICF			Recursive CHC		
	Stor.	Error	t(s)	Stor.	Error	t(s)	Stor.	Error	t(s)
abalone	0.2553	0.7935	1.0	0.1355	0.7868	0.4	0.3196	0.8086	237.4
adult	0.1769	0.1803	335.3	0.0547	0.2020	223.1	0.2553	0.2013	1461.2
car	0.2269	0.2360	0.4	0.1900	0.2936	0.2	0.2501	0.2314	40.6
gene	0.2438	0.2855	2.5	0.1115	0.4205	2.9	0.2770	0.3398	91.1
german	0.2641	0.3060	0.3	0.0277	0.3270	0.3	0.2635	0.3630	31.4
hypo	0.0857	0.0682	0.9	0.0308	0.0743	1.0	0.2499	0.0836	74.7
isolet	0.4329	0.1743	4.4	0.1443	0.2497	5.7	0.3080	0.1982	383.6
krkopt	0.2943	0.4810	30.4	0.1298	0.5737	19.7	0.2906	0.4698	6494.1
krvs.kp	0.2562	0.1282	1.0	0.0794	0.1806	0.8	0.2568	0.1367	83.4
letter	0.4919	0.0925	21.0	0.2429	0.1526	16.0	0.2962	0.0879	1176.2
magic	0.1294	0.2309	7.1	0.0413	0.2338	5.4	0.2529	0.2466	593.1
mfeat-fac	0.2496	0.0465	2.7	0.2025	0.0620	2.9	0.2454	0.0560	52.9
mfeat-fou	0.3137	0.2330	1.1	0.0986	0.3805	0.7	0.2776	0.2420	75.1
mfeat-kar	0.3055	0.0700	1.1	0.1782	0.1220	0.8	0.2628	0.0755	64.4
mfeat-mor	0.2478	0.3025	0.3	0.1821	0.3030	0.1	0.2601	0.3185	61.5
mfeat-pix	0.2754	0.0385	3.1	0.1833	0.0565	3.4	0.2563	0.0445	59.0
mfeat-zer	0.2735	0.2150	0.7	0.1222	0.2500	0.6	0.2637	0.2290	68.1
nursery	0.1684	0.2515	4.0	0.2763	0.2202	5.5	0.2511	0.2394	369.3
optdigits	0.2979	0.0361	3.5	0.1727	0.0571	3.7	0.2414	0.0468	185.3
page-bl	0.0504	0.0634	1.0	0.1257	0.0559	0.4	0.2529	0.0479	132.1
pendigits	0.2137	0.0117	4.3	0.2753	0.0162	4.1	0.2473	0.0133	293.2
phoneme	0.1077	0.2143	0.9	0.0580	0.2361	0.5	0.2507	0.1546	180.4
satimage	0.1304	0.1387	2.4	0.1348	0.1460	2.7	0.2475	0.1162	143.8
segment	0.2037	0.0853	0.5	0.1623	0.1117	0.5	0.2526	0.0732	70.5
shuttle	0.0333	0.0045	81.4	0.3029	0.0016	56.3	0.2613	0.0014	1057.4
sick	0.0828	0.0533	0.6	0.0272	0.0759	1.0	0.2484	0.0520	72.1
texture	0.2373	0.0307	2.6	0.2321	0.0420	3.4	0.2455	0.0295	150.6
wavef	0.2958	0.2458	2.2	0.0578	0.2642	2.1	0.2583	0.2800	200.2
yeast	0.2385	0.4676	0.2	0.0841	0.4899	0.1	0.2949	0.5372	42.5
zip	0.3225	0.0389	8.6	0.1354	0.1191	6.6	0.2620	0.0450	359.1

4.3.5.3 Huge problems In previous experiments we have shown the performance of our methodology in problems that can be considered of medium to large size. In this section, we considered huge problems, from several hundreds of thousands of instances to more than a million. Table 13 shows the problems that are considered. These datasets will show whether our methodology allows scaling up standard algorithms to huge problems. We have tested our method using the three instance selection algorithms of the previous sections. As in the previous sections, for estimating the storage reduction and generalization error we used a 10-fold cross-validation method. The size of the datasets pre-

Table 6
Standard deviation of testing error and storage requirements for the recursive approach using both mechanisms, random addition of removed instances and second chance, of improving testing error.

Dataset	Recursive Drop3		Recursive ICF		Recursive CHC	
	Storage	Error	Storage	Error	Storage	Error
abalone	0.0124	0.0811	0.0167	0.0848	0.0075	0.0583
adult	0.0047	0.0057	0.0096	0.0120	0.0030	0.0061
car	0.0296	0.0631	0.0297	0.1041	0.0122	0.0659
gene	0.0109	0.0275	0.0062	0.0331	0.0106	0.0187
german	0.0137	0.0709	0.0084	0.0603	0.0177	0.0585
hypo	0.0042	0.0103	0.0073	0.0152	0.0100	0.0139
isolet	0.0063	0.0442	0.0038	0.0558	0.0055	0.0416
krkopt	0.0079	0.0077	0.0033	0.0076	0.0025	0.0062
krvs.kp	0.0072	0.0181	0.0083	0.0249	0.0085	0.0150
letter	0.0051	0.0045	0.0061	0.0077	0.0035	0.0072
magic	0.0098	0.0631	0.0043	0.1317	0.0031	0.1147
mfeat-fac	0.0050	0.0105	0.0060	0.0119	0.0163	0.0097
mfeat-fou	0.0107	0.0222	0.0102	0.0342	0.0097	0.0309
mfeat-kar	0.0063	0.0206	0.0123	0.0200	0.0103	0.0192
mfeat-mor	0.0180	0.0265	0.0141	0.0260	0.0077	0.0292
mfeat-pix	0.0070	0.0134	0.0141	0.0092	0.0110	0.0165
mfeat-zer	0.0055	0.0253	0.0075	0.0160	0.0105	0.0314
nursery	0.0198	0.0842	0.0383	0.0915	0.0054	0.1097
optdigits	0.0223	0.0065	0.0059	0.0121	0.0042	0.0115
page-bl	0.0020	0.0112	0.0199	0.0091	0.0106	0.0073
pendigits	0.0040	0.0049	0.0129	0.0043	0.0058	0.0038
phoneme	0.0076	0.0302	0.0075	0.0221	0.0094	0.0117
satimage	0.0034	0.0185	0.0065	0.0154	0.0078	0.0161
segment	0.0075	0.0201	0.0150	0.0137	0.0126	0.0140
shuttle	0.0010	0.0014	0.0127	0.0006	0.0019	0.0006
sick	0.0081	0.0069	0.0120	0.0265	0.0069	0.0101
texture	0.0194	0.0066	0.0069	0.0065	0.0091	0.0060
wavef	0.0082	0.0180	0.0034	0.0222	0.0094	0.0157
yeast	0.0118	0.0339	0.0064	0.0421	0.0093	0.0257
zip	0.0052	0.0052	0.0093	0.0117	0.0071	0.0052

vents the execution of the standard algorithms in a reasonable time, so the validity of our approach will be tested against 10-fold cross-validation 1-NN testing error, which is shown in the table, together with the time needed to obtain that error.

In the previous experiments the number of steps of the algorithm was obtained by means of a cross-validation method which was described in Section 4.4 in the paragraph devoted to the stop criterion. However, due to the size of the datasets, and to make the algorithm faster, no cross-validation is used for

selecting the number of steps. Instead, we always perform two steps of the algorithm. This number of two steps was chosen as it was the most commonly chosen by the stop criterion in the previous experiments using medium to large datasets.

Table 7
Summary of datasets. The features of each dataset can be C(continuous), B(binary) or N(nominal). The Variables column shows the number of variables, as it depends not only on the number of features but also on their type.

Data set	Cases	Features			Classes	Variables	1-NN	
		C	B	N			Error	Time (s)
census	299285	7	-	30	2	409	0.0743	8723.2
covtype	581012	54	-	-	7	54	0.3024	16980.3
kddcup99	494021	33	4	3	23	118	0.0006	12649.4
kddcup991M	1000000	33	4	3	21	119	0.0002	37093.5
poker	1025010	5	-	5	10	25	0.4975	35460.7

Table 8
Testing error, storage requirements and execution time (in seconds) for our approach for huge problems.

Dataset	Storage	Error	Time
	Recursive Drop3		
census	0.1205	0.1101	510.7
covtype	0.1381	0.3969	213.6
kddcup99	0.0499	0.0179	534.6
kddcup991M	0.0337	0.0087	1249.0
poker	0.0941	0.4977	305.5
	Recursive ICF		
census	0.6047	0.0621	188.5
covtype	0.1179	0.4047	134.5
kddcup99	0.2870	0.0181	222.0
kddcup991M	0.2918	0.0010	746.0
poker	0.0474	0.5114	280.0
	Recursive CHC		
census	0.1410	0.0814	4200.7
covtype	0.1416	0.3323	9682.3
kddcup99	0.1427	0.0009	5308.0
kddcup991M	0.1436	0.0004	9847.0
poker	0.1671	0.5358	26468.0

Results are shown in Table 8. The first noticeable fact is the scalability of our method. The execution times are very low even for the two datasets that have more than a million instances. Even for CHC the time spent is within

moderate limits. If we compare these values with the estimation we can get from Figure 8 of the time Drop3, ICF or CHC would need, the advantage of our proposal is clearly stated.

Regarding the performance, Drop3 is specially efficient in reducing storage. It achieves a large reduction for the five datasets, without damaging the performance very significantly. In fact, the increment of the testing error is similar to the increment experimented by the original Drop3 algorithm when applied to the 30 datasets of the previous experiments. In terms of testing error, CHC is the best one, although the reduction it achieves is worse than Drop3. ICF performs worse than CHC and Drop3, both in terms of testing error and storage reduction. It is, however, the fastest one.

4.4 Democratic instance selection method

In this section we propose a methodology that uses 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) A very 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 partitioning solution was previously applied to the recursive method (please refer to Section 4.3.2 to a detailed description of the partitioning step performed in the recursive method). As previously explained, this partition method did 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 as 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".Additionally, our method is straightforwardly paralellizable without significant modifications.
(3) Following the philosophy of classifier ensembles we can carry out 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.

The proposed scaling algorithm, called DEMOIS., consists of dividing the original dataset into several disjoint subsets of approximately the same size that cover all the dataset. 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 made, the instances which have received a number of votes above a certain threshold are removed. An outline of the method is shown in Algorithm 6. 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 (Schapire et al., 1998). Figure 10 shows an example of the algorithm for 10 rounds of votes and a dataset of 50 instances.

Algorithm 6: 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 $i = 1$ ***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 $\geq v$
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, as 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 one of the processors.

However, as stated so far, the method still has two important issues to be addressed before we can obtain a useful algorithm. Firstly, the partition method is not trivial, as a strictly random partition would not perform well. Secondly, the determination of the number of votes is problem-dependent. We made 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. For instance, using 10 rounds if we set a threshold of 5 votes to remove an instance, there are datasets for which that means removing almost all the instances, while there are other datasets for which that threshold results in keeping almost all instances. Thus a method must be developed for

the automatic determination of the number of votes needed to remove an instance from the training set. The 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, as the instance selection algorithm is a parameter of the method.

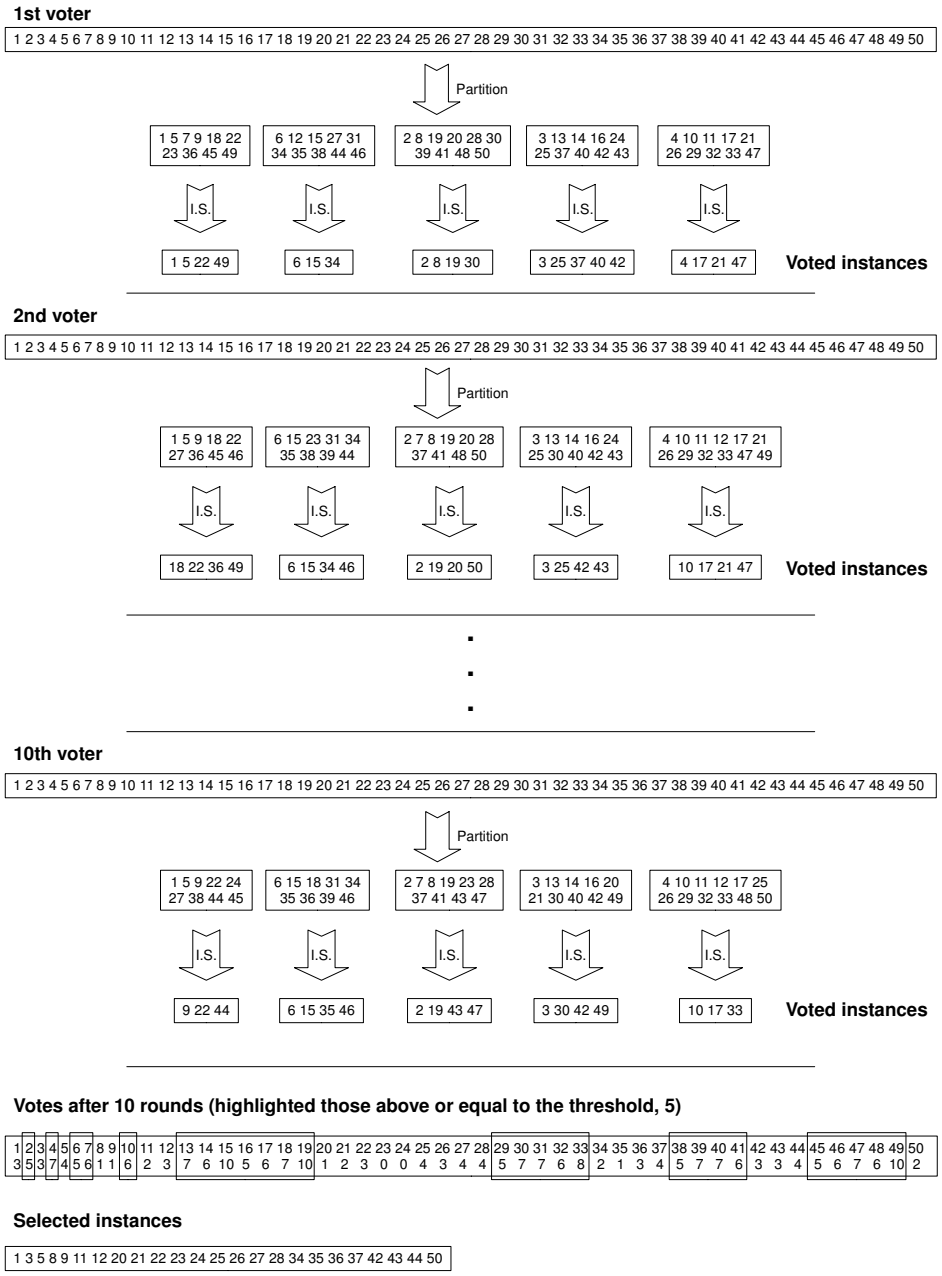


Fig. 10. Example of democratic instance selection for a dataset of 50 instances and subsets of 10 instances. The instance selection algorithm can be any one of the many available.

4.4.1 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. In the experimental section we will show a study of the influence of the subset size on the performance of the algorithm.

In the recursive method described in Section 4.4 we applied a simple partitioning method whose improvement resulted in the partitioning technique described in this section.

The first thing that we must take into account is the fact that we need several different partitions of the dataset, as each round of votes needs a different partition. Otherwise, the votes cast will be the same as most instance selection algorithms are deterministic. The simplest method would be just 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 García-Osorio and Fyfe (2005) 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 as 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. But there is an additional, and more subtle point, about the partition that is of the utmost importance for the performance of the method.

Each partition represents a different optimization problem, and so 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. [4]

These two previous requirements, partitions that keep certain spatial consis-

[4] In fact, we performed experiments with random partitions with poor results. However, if the different random partitions are varied smoothly, for example, performing and initial random partition and then exchanging a few instances between subsets at each round, the performance is clearly improved. This result corroborates the necessity to have in subsequent rounds of the algorithm partitions that are not very far apart.

tency and that vary smoothly are obtained using the theory of Grand Tour (Asimov, 1985). The idea of the grand tour method, introduced by Asimov (1985) and Buja and Asimov (1986), 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 angles. The method rotates a plane in the high dimensional space. The data is projected onto this plane for each of its orientations, and when the sequence of projections is visualized in a computer screen, an animation is obtained which is useful for identifying structure in the dataset, such as clusters an outliers. The grand tour shares a common objective with exploratory projection pursuit techniques. In both cases the human ability for visual pattern recognition is exploited.

4.4.1.1 Algorithms When the grand tour is used for visualization the sequence of planes in must hold two conditions:

a. It should be dense in the set of all planes in the high dimensional space.
b. It 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" (Buja et al., 2005) and are based on the interpolation of a sequence of randomly generated planes. In the context of our algorithm is enough to use a one dimensional grand tour, we project the data onto a rotating vector and then we use this projection to divide the dataset into the subsets that we will pass to the underlying instance selection algorithm. As well as this, we are more concerned with the simplicity of the algorithm so, instead of an interpolation class algorithm, we have chosen a parametrization class algorithm, the torus method (Asimov, 1985), based on obtaining a sequence of rotation matrices, so the problem now is how to obtain these matrices.

We want to obtain a generalized rotation matrix Q that we will use to rotate the vector onto 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 for being 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 ($p = (d-1)d/2$, and 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:

$$\beta(\theta_{1,2}, \theta_{1,3}, \ldots, \theta_{d-1,d}) = R_{1,2}(\theta_{1,2}) \times R_{1,3}(\theta_{1,3}) \times \cdots \times R_{d-1,d}(\theta_{d-1,d}) \quad (7)$$

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

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

Summing up, 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 of getting the curve through the p-torus (Wegman and Solka, 2002): the Asimov-Buja winding algorithm, the random curve algorithm, the fractal curve algorithm. In the experiments we use the random curve algorithm. First, we just randomly take two points $\mathbf{s}_i, \mathbf{s}_j$ in $[0, 2\pi]^p$ and create a linear interpolant between them 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.

4.4.1.2 Some implementation details Obtaining the curve strictly through the shortest path in the p-torus adds a burden of complexity that seems not to give any extra advantage in our algorithm. So, instead we just interpolate the points through the hypercube $[0, 2\pi]^p$. If we have a point near by the p-dimensional point $(2\pi, 2\pi, \ldots, 2\pi)$ and a point near by 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 $[0, 2\pi]^p$, but in our current implementation, we just interpolate the point using the path strictly inside the hypercube (which length is approximately $\sqrt{p(2\pi)^2}$). Besides, most of the time with the interpolation of the first two points in $[0, 2\pi]^p$ we already obtain enough orientations to get all the partitions required by the algorithm.

As the denseness 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 usually ten to fifteen steps of grand tour is enough), we can use even simpler

ways of obtaining the sequence of projections. In the case of uni-dimensional projections we could have used just the pseudo grand tour obtained by using Andrews curves (Andrews, 1972). If we want a sequence of bi-dimensional projections we can use the orthogonal vectors given by the Wegman curves (Wegman and Shen, 1993).

One concern when the grand tour is used for dynamic data visualization is that, in general, the mapped curve on $SO(d)$ could not be uniformly distributed even when the curve on the p-torus is equi-distributed. Here, we are also interested in uniformly rotating the projection vector, and we simply solve this by 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.

We used as projection vectors: arbitrary canonical basis vectors $\mathbf{e}_i$, the first principal component direction of the dataset, random vectors. The results seems not to depend on this choice, so the simplest one, $\mathbf{e}_1$, is the one used in the experiments.

4.4.2 Partition algorithm

Following this idea we obtain the first partition projecting our dataset into 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 7 shows the method for performing the partition based on this methodology.

Algorithm 7: 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 is divided into four subsets, is depicted in Figure 11. 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 different rounds of votes are performed.

This partition is specially designed for k-NN instance selection algorithms. It we apply our methodology to other classifier a random partition of the dataset

can be used as it will be shown in Section 4.4.6.11.

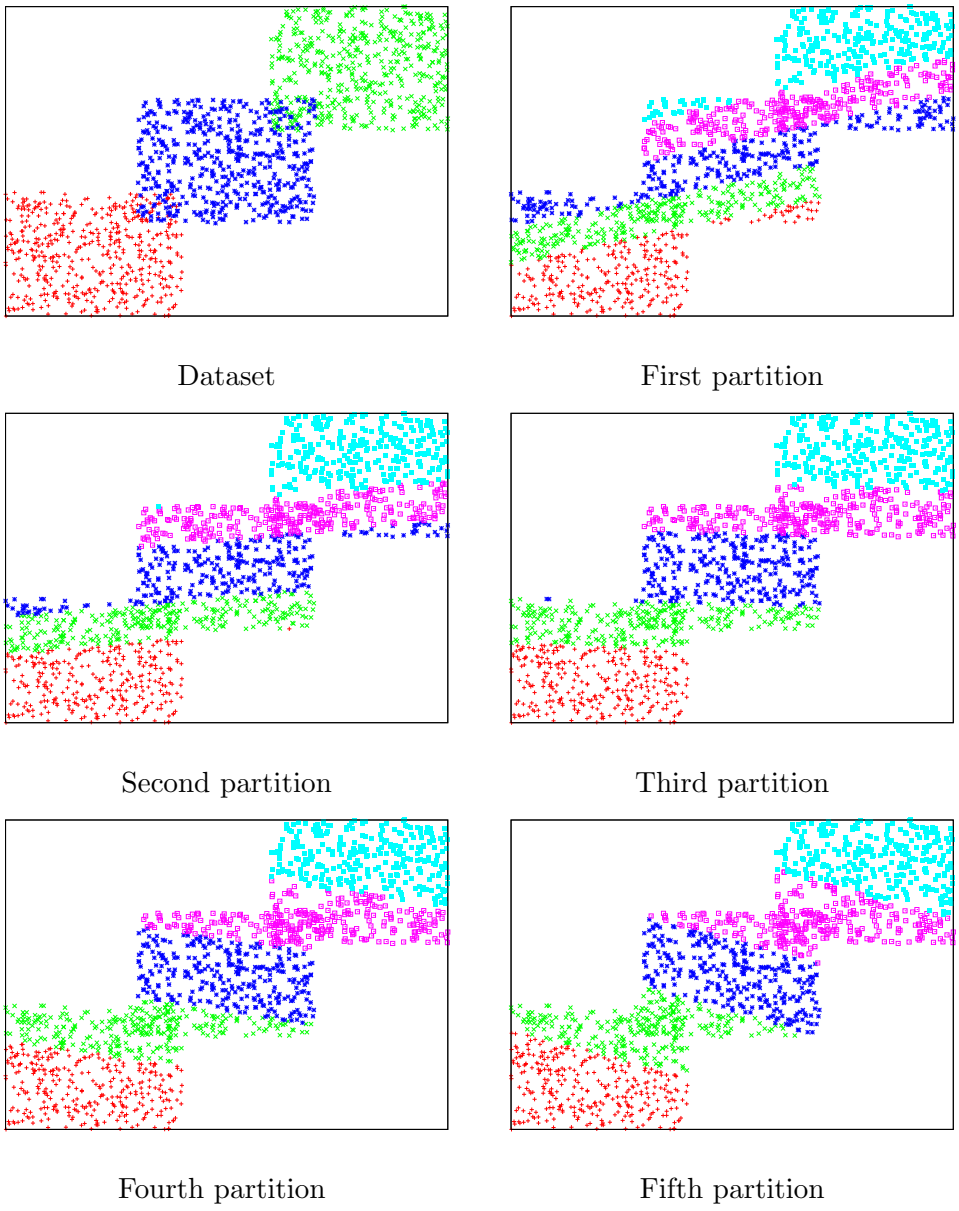


Fig. 11. Example of the method for partitioning the dataset with 1500 instances, three classes, and two features, with five rounds of votes. Four subsets are created each round.

4.4.3 Determining the number of votes

An important issue in our method is determining the number of votes needed to remove an instance from the training set. Preliminary experiments showed that this number highly 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 very time consuming. A second choice is estimating the best value for the number of votes from the effect on the training set. This latter method is the one we have chosen. The election of the number of votes must take into account two different criteria: training error, ϵ_t, and storage, or memory, requirements m. Both values must be minimized as much as possible. Our method of choosing the number of votes needed to remove an instance is based on obtaining the threshold number of votes, v, that minimizes a fitness criterion, $f(v)$, which is a combination of these two values:

$$f(v) = \alpha\epsilon_t(v) + (1 - \alpha)m(v), \tag{8}$$

where α is a value in the interval $[0, 1]$ which measures the relative relevance of both values. In general, the minimization of the error is more important than storage reduction, as we prefer a lesser error even if the reduction is smaller. Thus, 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. m is measured as the percentage of instances retained, and ϵ_t is the training error.

However, 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 is the following: We perform r rounds of the algorithm and store the number of votes received by each instance. Then, we must obtain the threshold number of votes, v, to remove an instance. This value must be $v \in [1, r]$. We calculate the criterion $f(v)$ (eq. 9) for all the possible threshold values from 1 to r, and assign v to the value which minimizes the criterion. After that, we perform the instance selection removing the instances whose number of votes is above or equal to the obtained threshold v.

It is worth noting that we can also make a *democratization* of the estimation of the threshold of votes. In fact, that democratic version has been used in the parallel implementation of the method (see Section 4.4.7).

A more advanced solution is to parallelize the estimation of the threshold of votes over the disjoint subsets, as these are independent processes that can be calculated at the same time with no need of interaction. Moreover this solution is more accurate as it does not use a percentage of the data available but the whole dataset to estimate this parameter.

4.4.4 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 a must. 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 from 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, as 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, as K is a constant value. Figure 12 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 $r = 10$ rounds of votes.

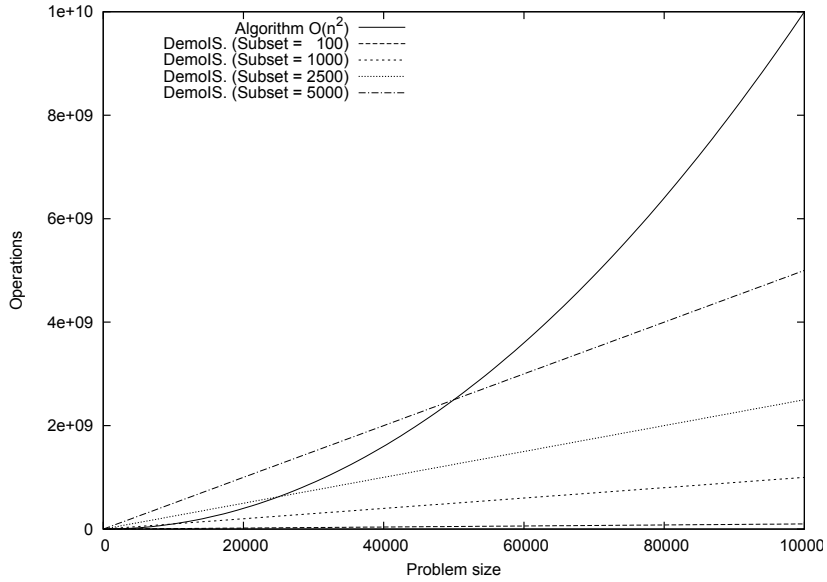


Fig. 12. Computational cost of our method and a base instance selection algorithm of $O(n^2)$.

Thus, the gaining in execution time would be greater as the size of the datasets is larger. 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. As the application of

the instance selection algorithm to each subset is independent from 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. Regarding the determination of the number of votes, the process can be made 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 for determining the number of votes, with a limit on the maximum size of this subset that is fixed for any dataset. In this way, from medium to large datasets we use the 10% of the training set, for huge problems the 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 [5] in the number of instances without damaging the performance of the algorithm. With this method the complexity of this step is $O(1)$ as the number of instances used is bounded regardless the size of the dataset.

Finally, we consider the partition of the dataset apart from the algorithm as many different partition methods can be devised. The partition described in Section 4.4.2 can be implemented with a complexity $O(nlog(n))$, 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 sized intervals. Both methods achieve the same performance as the obtained partition is very similar, and in our experiments we have used the latter to keep the complexity of the whole procedure linear. However, this partition is specially designed for k-NN classifier. 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 C4.5 and SVMs, we have used a random partition of the dataset.

4.4.5 Experimental setup

The experimental setup used in this work is the same as the one previously detailed in the recursive instance selection Section 4.3.3, except for some minor details we proceed to discuss in the following lines.

In order to make a fair comparison between the standard algorithms and our proposal, we have selected the same datasets as in the recursive method. The features of the 30 selected problems from the UCI Machine Learning

[5] This *reasonable* bound can be from a few hundreds to a few thousands, even for huge datasets.

Repository (Hettich et al., 1998) can be consulted on the previous Table 1.

4.4.5.1 Instance selection algorithms In order to obtain an accurate view of the usefulness of our method, we must select some of the most widely used instance selection algorithms. We have chosen to test our model using several of the most successful state-of-the-art algorithms. Initially, we used the algorithms DROP3 (Wilson and Martinez, 2000), and ICF (Brighton and Mellish, 2002). These two methods were previously applied in the recursive method, their description can be consulted in Section 4.3.4 and their pseudocodes are detailed in Algorithms 3 and 4, respectively.

In addition to these two methods that can be considered "classical", as they have been around for quite a long time and are widely used, it is worth mentioning Reduced Nearest Neighbor (RNN) rule (Gates, 1972). This method is extremely simple, but it also shows an impressive performance in terms of storage reduction. In fact, it is the best of the methods used in reducing storage requirements, as will be shown in the next section. However, it has a serious drawback, its computational complexity. Among the standard methods used this is the one that shows a worst scalability, taking several hundreds hours in the worst case. Therefore, RNN is the perfect target for our methodology, an instance selection method highly efficient but with a serious scalability problem. So we have also tested our approach using RNN, which is shown in Algorithm 8, as base instance selection method.

Algorithm 8: RNN algorithm

Data : A training set $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, a selector $S = \varnothing$
Result : The set of selected instances $S \subset T$.

```
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
  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
```

We have also used one of the most recent algorithms for instance selection, the Modified Selective Subset (MSS) method (Barandela et al., 2005). The procedure is shown in Algorithm 9. We chose this algorithm as an example of a recent and fast algorithm. With MSS we want to test whether our method

is also able to improve the execution time of algorithms that are not so time demanding as the previous ones.

Algorithm 9: MSS algorithm

Data : A training set $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, a selector $S = \varnothing$
Result : The set of selected instances $S \subset T$.

```
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
```

As an alternative to these standard methods, genetic algorithms have been applied 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 Kuncheva (1995), Ishibuchi and Nakashima (2000) and Reeves and Bush (2001). Cano et al. (2003) performed a comprehensive comparison of the performance of different evolutionary algorithms for instance selection. They compared a generational genetic algorithm (Goldberg, 1989), a steady-state genetic algorithm (Whitley, 1989), a CHC genetic algorithm (Eshelman, 1990), and a population based incremental learning algorithm (Baluja, 1994). 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 major 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. As we are concerned with this problem, we have used as 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 and MSS, so it gives us a good benchmark to test our methodology on an algorithm that, as RNN, has a big scalability problem.

4.4.6 Experimental results

The same parameters were used for the standard version of every algorithm and its application within our methodology. All the standard methods have no relevant parameters, the only value we must set is k, the number of nearest neighbors. For DROP3, ICF, RNN and MSS we used $k = 3$ neighbors. 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 (Cano et al., 2003). Our method has two parameters: subset size, s, and number of rounds, r. Regarding subset size we must use a value large enough to allow for a meaningful application of the instance selection algorithm on the subset, and small enough to allow a fast execution, as the time used by our method grows with s. As a compromise value we have chosen $s = 1000$, and a minimum of two subsets if the dataset has 1000 or less instances. For the number of rounds we have chosen a small value to allow for a fast execution, $r = 10$. Furthermore, in Section 4.4.6.9 we perform 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 9 and 10, for standard and DEMOIS. methods.

Table 9. 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	22853.9	0.1082	0.2194	9170.8	0.2950	0.2281	2990.8	0.0333	0.1951	1896540.3	0.1988	0.2257	91096.0
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	10885.4
krkopt	0.4431	0.4803	1533.0	0.5290	0.4032	1109.8	0.6565	0.4323	356.7	0.0425	0.5678	1057715.5	0.5237	0.4711	137015.0
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	21394.0	0.2952	0.0905	51024.0
magic04	0.1789	0.1978	199.8	0.1160	0.2395	138	0.3204	0.2440	23.4	0.0293	0.1805	50817.2	0.2952	0.1225	29327.0
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	22397.3
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	11636.6
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	77089.0
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 10. 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	12144.9	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

4.4.6.1 DROP3 vs. DEMOIS.*drop3* The results using standard DROP3 algorithm and our method with DROP3 as base algorithm are plotted in Figure 13. The figure shows results for testing error, storage requirements and execution time. We will use a graphic representation based on the kappa-error relative movement diagrams (Maudes-Raedo et al., 2008), but 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 graphics are a very convenient way of summarizing the results. For example, arrows pointing down-left represent datasets for which our method outperform the standard algorithm in both error and storage, arrows pointing up-left indicate that our algorithm improves the storage but with a worse testing error, and so on. Numerical results are shown in Tables 9 and 10. In terms of error, our method is able to match the results of 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, it 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 very significant. For small problems there is a small overload due to the 10 rounds of votes performed, however as the problem grows in complexity 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 a better storage reduction.

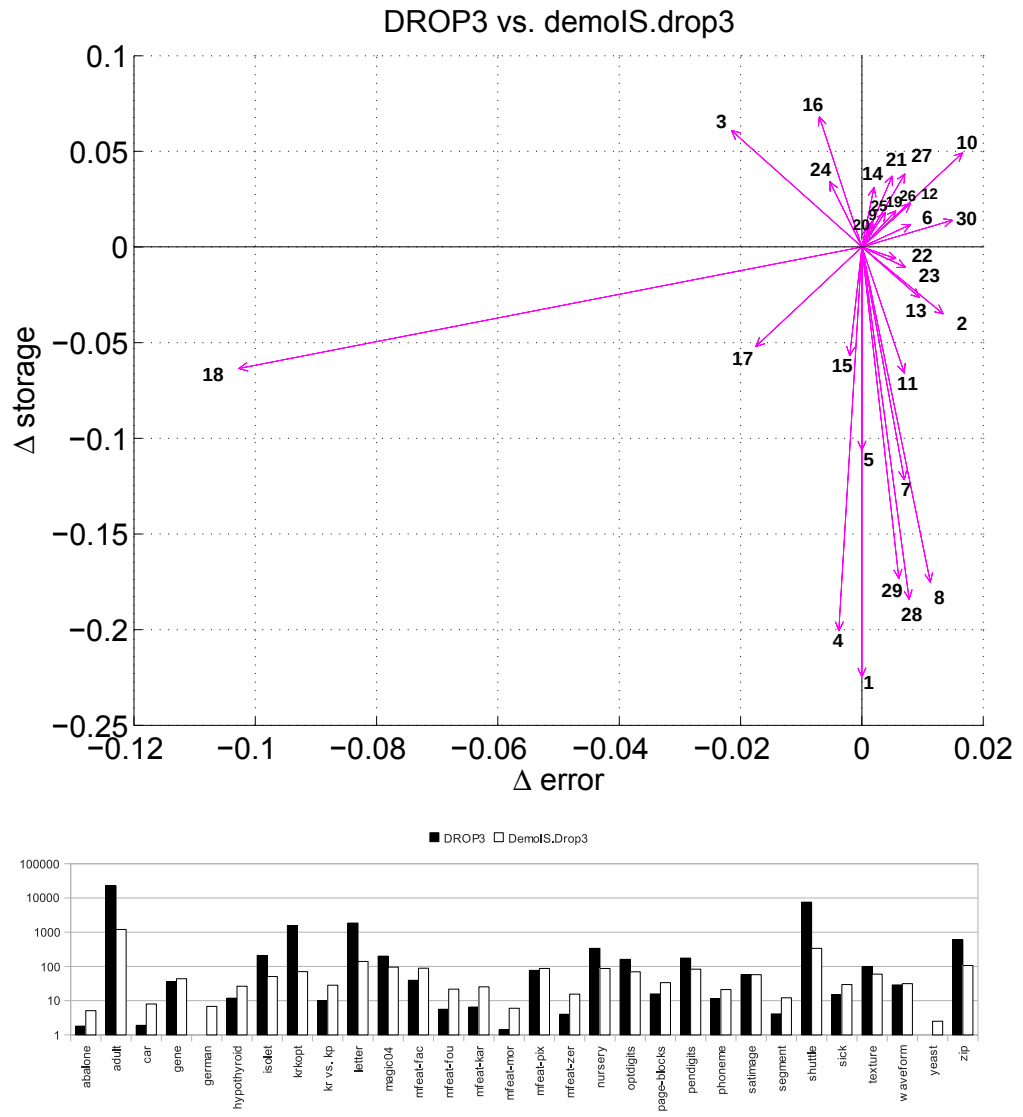


Fig. 13. Storage requirements/testing error (*top*) and execution time in seconds (*bottom* – using a logarithmic scale) for standard DROP3 algorithm and our approach.

4.4.6.2 ICF vs. DEMOIS.*icf* Results for ICF and DEMOIS.icf are plotted in Figure 14, and the numerical results are shown in Tables 9 and 10. 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 exception of 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.

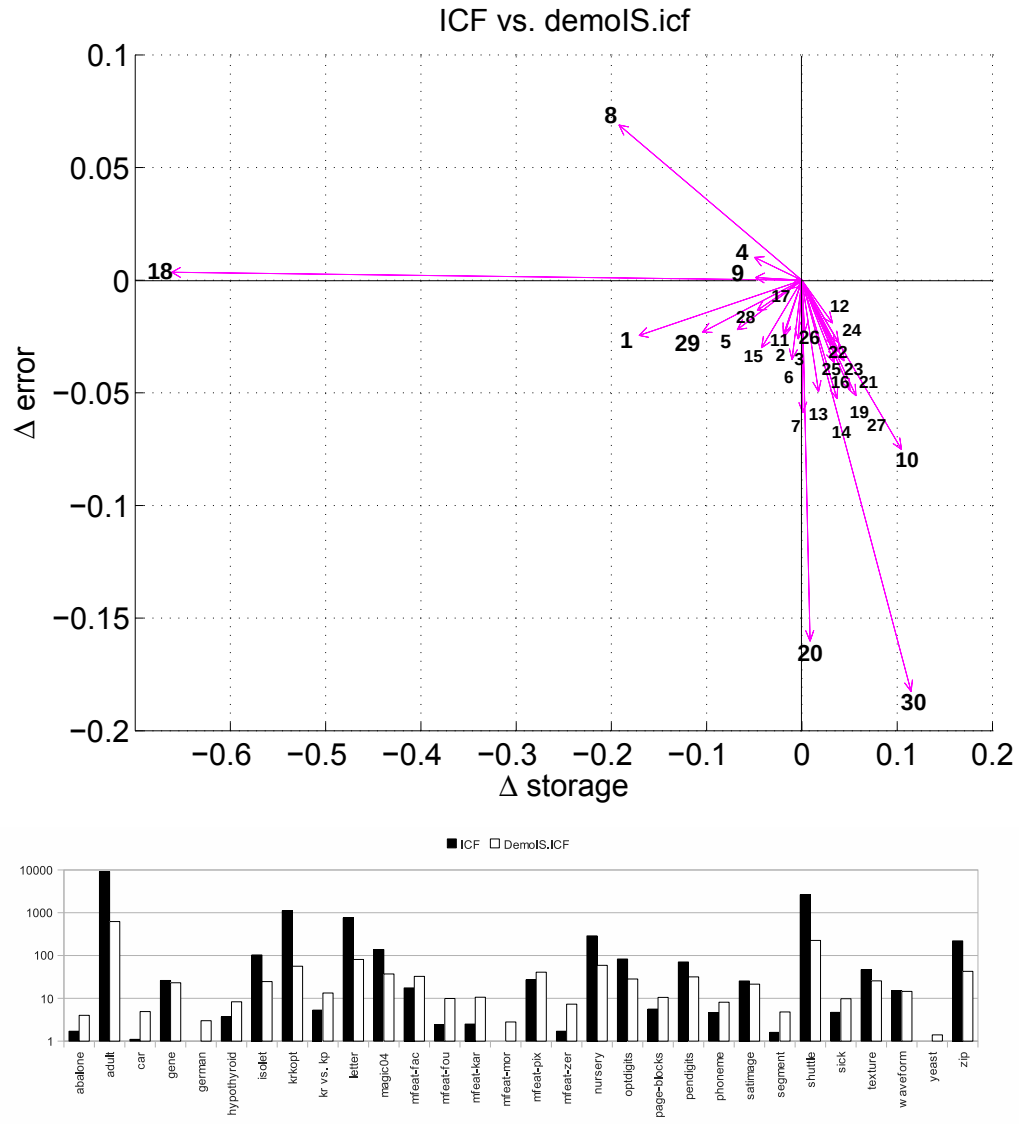


Fig. 14. Storage requirements/testing error (*top*) and execution time in seconds (*bottom* – using a logarithmic scale) for standard ICF algorithm and our approach.

4.4.6.3 MSS vs. DEMOIS.*mss* Results for MSS and DEMOIS.mss are plotted in Figure 15, and the numerical results are shown in Tables 9 and 10. In terms of both, testing error and storage reduction, the performance of DEMOIS.mss and MSS are very 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, the improvement on the execution time is still significant.

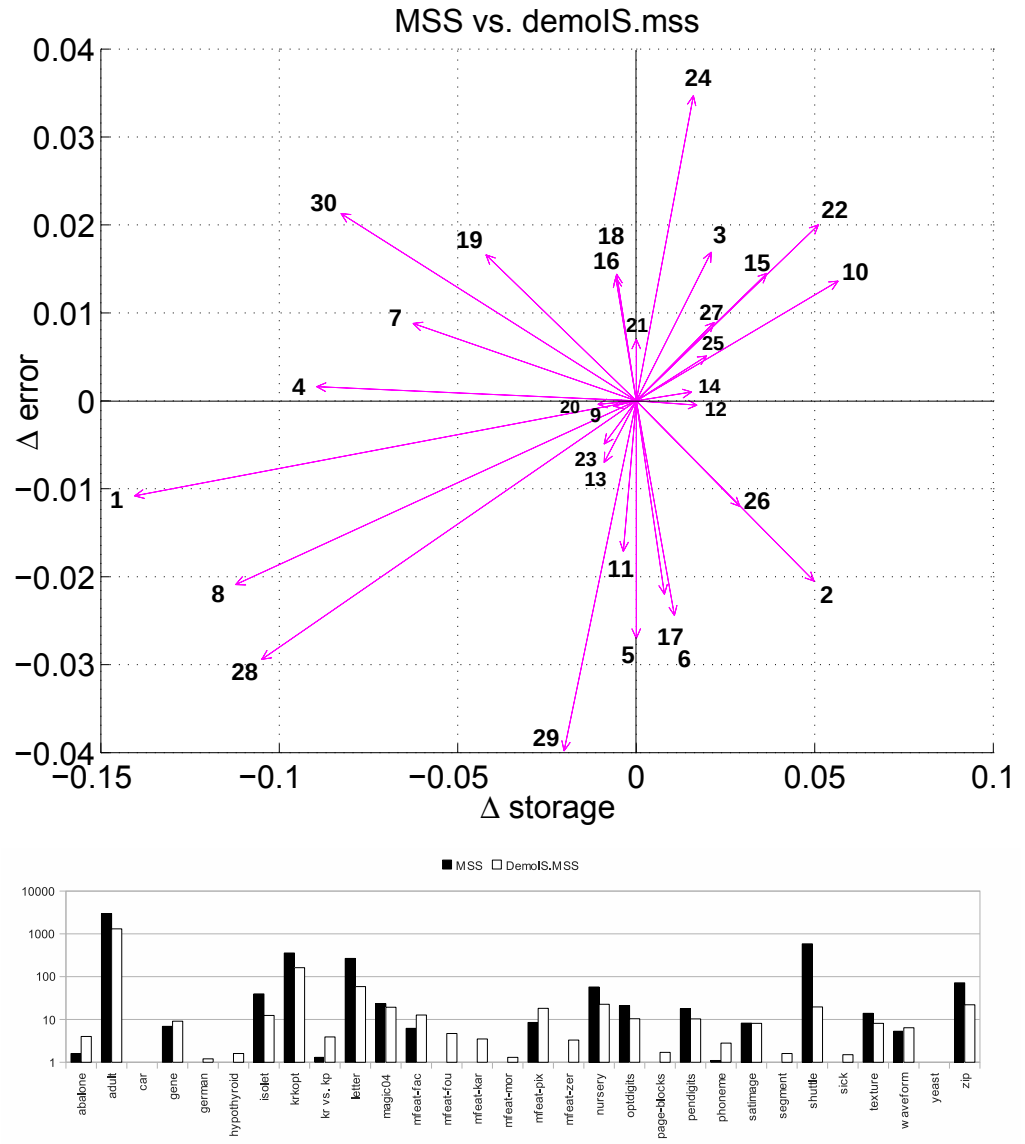


Fig. 15. Storage requirements/testing error (*top*) and execution time in seconds (*bottom* – using a logarithmic scale) for standard MSS algorithm and our approach.

4.4.6.4 RNN vs. DEMOIS.*rnn* The next experiment is conducted using as base instance selection algorithm RNN. The results are plotted in Figure 16 with numerical values shown in Tables 9 and 10. 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 very serious problem of scalability. As an extreme example, for adult problem it took more than 500 hours per experiment. This scalability problem prevents is 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, it 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 very 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 a small worsening of the storage reduction. In terms of execution time the results are remarkable, the reduction of the time consumed by the selection process is large, with the extreme example of the two most time consuming datasets, adult and krkopt, where the speed-up is more than a hundred times.

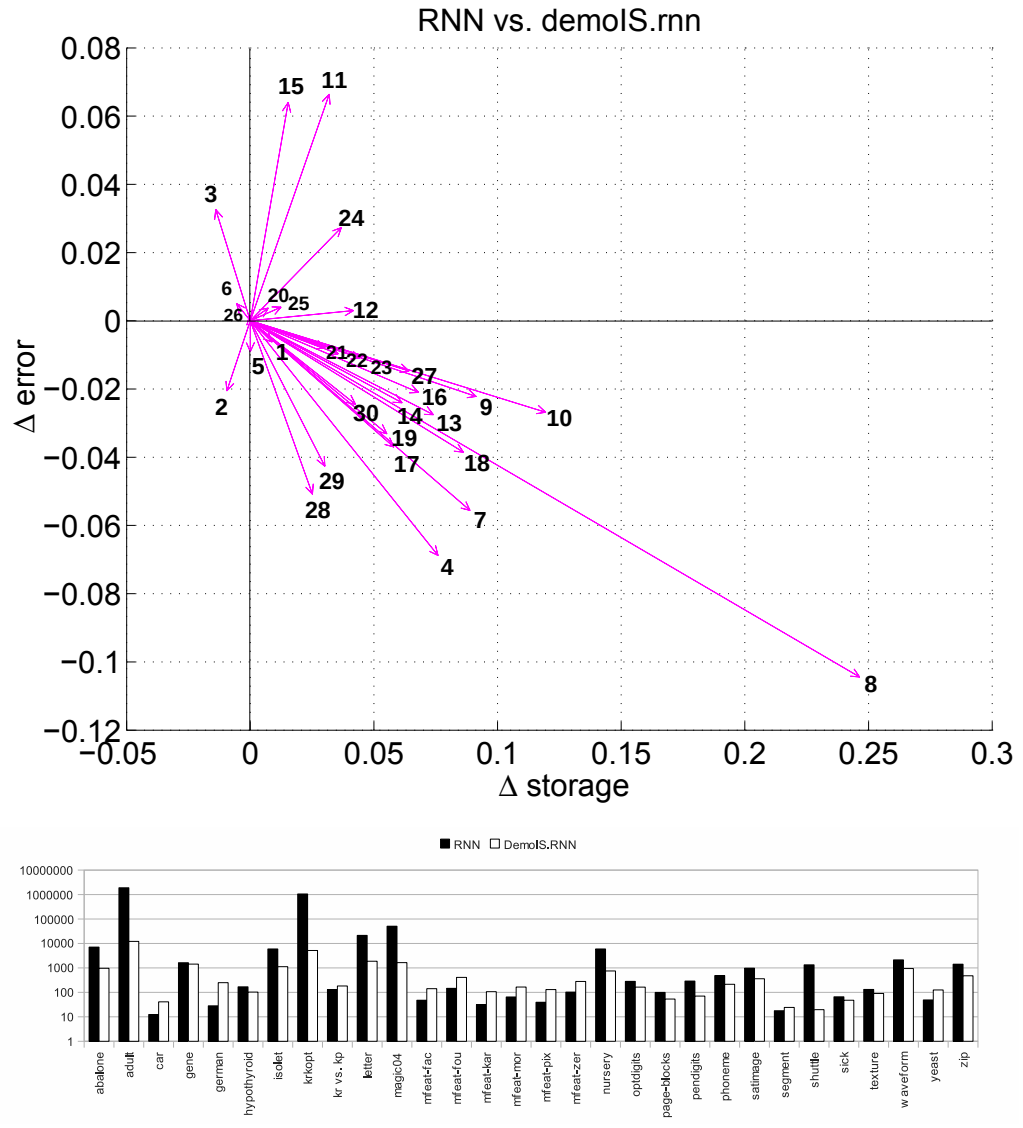


Fig. 16. Storage requirements/testing error (*top*) and execution time in seconds (*bottom* – using a logarithmic scale) for standard RNN algorithm and our approach.

4.4.6.5 CHC vs. DEMOIS.*chc* Figure 17 plots the results of CHC algorithm, with numerical values shown in Tables 9 and 10. Due to the high computational cost of CHC we have chosen for this algorithm a smaller subset size of $s = 250$. A first interesting result is the problem of scalability of CHC algorithm, which is more marked for this algorithm than for the previous ones. In other works, (Cano et al., 2003) (García-Pedrajas et al., 2010), 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, as the datasets are larger, the scalability problem of CHC manifests itself. In our set of problems, CHC clearly performs worse than DROP3, ICF, MSS and RNN in terms of storage reduction. We must take into account that for CHC we need a 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 10000 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 bad side effect is a worse testing error, which is however not very marked and compensated by the improvement in running time and storage reduction.

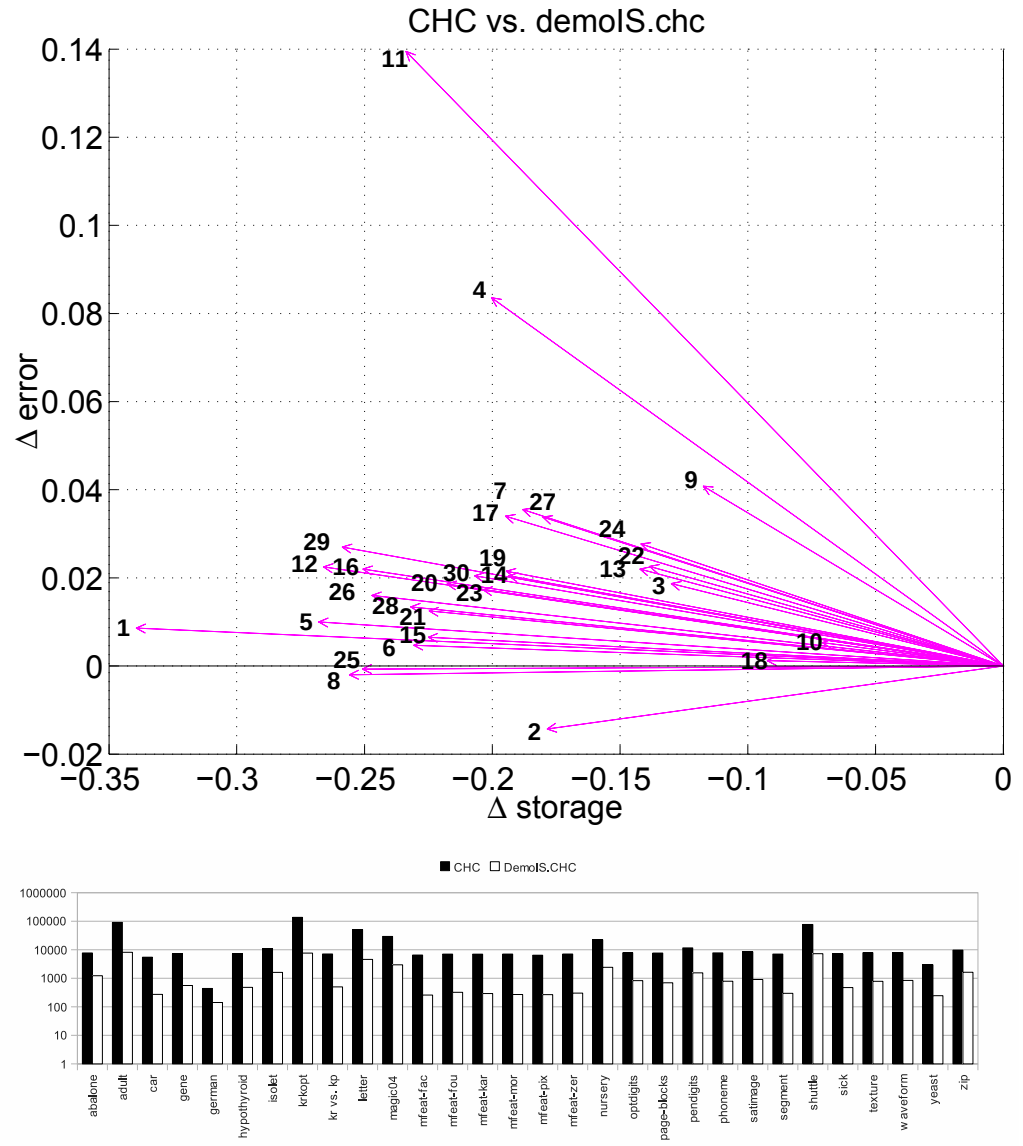


Fig. 17. Storage requirements/testing error (*top*) and execution time in seconds (*bottom* – using a logarithmic scale) for standard CHC algorithm and our approach.

4.4.6.6 Control experiments The previous experiments showed that our method is able to, at least, match the performance of standard methods with a very 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 forget that since random sampling does not determine the number of instances to retain in the subset, it only solves part of the problem (Wilson and Martinez, 2000), 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 the random sampling. Table 11 shows the comparison for all methods.

Table 11
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 Wilcoxon test are marked 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✓

The experiment shows interesting results. Firstly, 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 them 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, as MSS and CHC do not show a significantly better behavior than random sampling.

Nevertheless, we must not rule out the use of these algorithms, as the comparison is made using as random sampling rate the value obtained by the corresponding instance selection algorithm. If we consider random sampling alone, we will not be able to know 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.

4.4.6.7 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 in function of the number of instances in Figure 30. We plot the time spent by the algorithms as the number of instances increases. A Bezier line is drawn using those points to allow a better plot. The figure shows that 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. In general, the standard methods need a long execution time for large problems. On the other hand, 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 4.4.4.

4.4.6.8 Summary of results As a summary of the previous experiments, Table 12 shows the comparison of the behavior 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 the 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 all the remaining algorithms. In terms of execution time, as showed by Figure 30 the behavior is excellent for the five algorithms.

Table 12
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 marked with 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

4.4.6.9 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, of about a few hundreds or thousands of instances. Thus, we chose a subset size of 1000 instances as a good compromise between a significant enough big subset of instances and a small enough set. In this section we study the effect

of subset size in the behavior of our method. We have performed experiments using DROP3 and ICF and subset sizes of 100, 250, 500, 1000, 2500 and 5000 instances and 10 rounds of votes. Figure 19 and 20 show the results for testing error, storage requirements and execution time[6] with the six different sizes using DROP3 and ICF respectively. For DROP3 the reduction is kept very 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, as DROP3 is the base method the time grows approximately quadratically as the subset size grows.

[6] To avoid too large figures, in the following plots we show the number of each dataset, as it listed in Table 1, instead of its name.

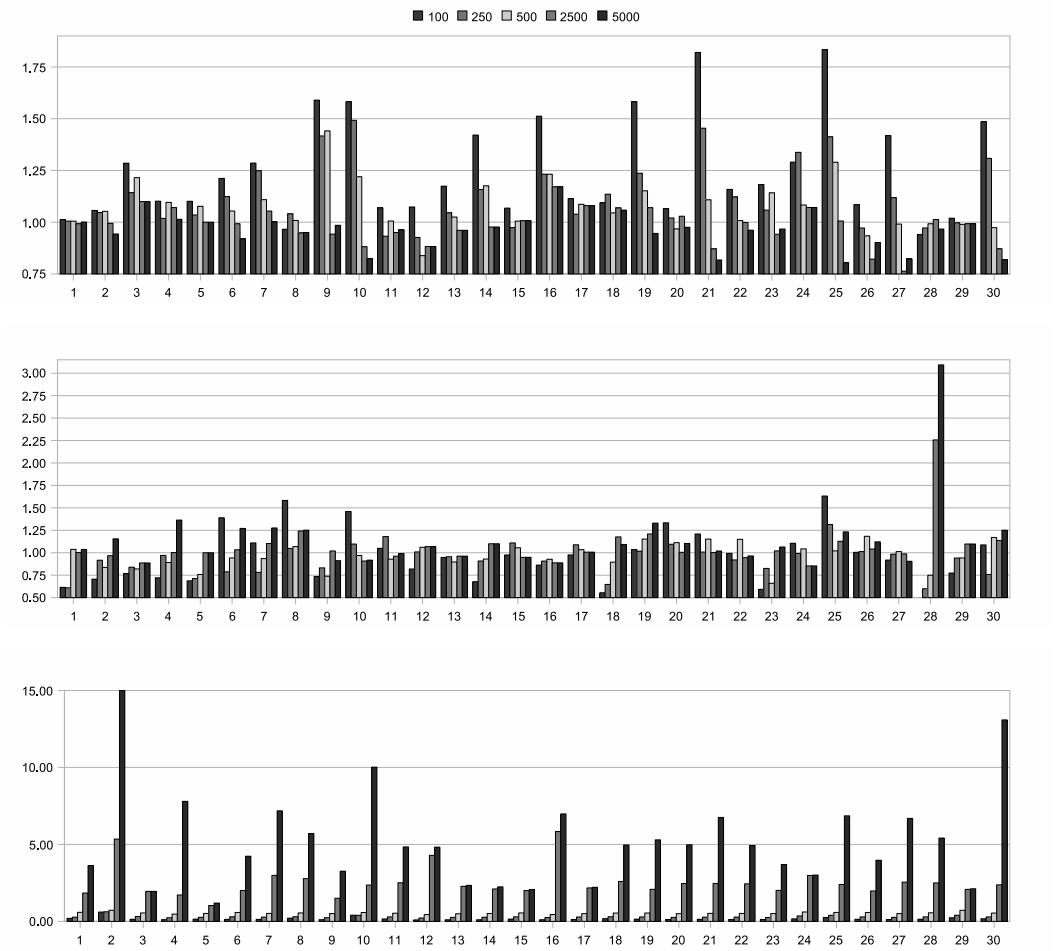


Fig. 19. Average testing error (*top*), storage requirements (*middle*) and execution time (*bottom*) as a function of subset size for DROP3 algorithm. The plots show relative values with respect to the results using a subset of 1000 instances.

The behavior for ICF is similar. In this case, there is a more significant reduction in storage requirements as the subset size is 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 is larger, the $O(n^2)$ of the algorithm begins to be relevant, and the processing of each subset is more important than the reduction of the number of subsets to process. The results corroborate that 1000 instances is a good compromise among the sizes that favor storage reduction, testing error and execution time.

A similar study was performed to test the effect of the number of rounds on the performance of the method. We run the method using 5, 10, 25 and 100 rounds. The results for DROP3 and ICF are shown in Figures 21 and 22 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, from 25 and on, many of the cast votes are redundant and there is no advantage in having so many rounds. In this way, a value around 10 rounds is enough. The effect of adding more voters in the testing error is marginal, and each new round add more execution time. This behavior is similar to the case of classifier ensembles, where little gain is obtained after the first few classifiers are added (García-Pedrajas et al., 2007).

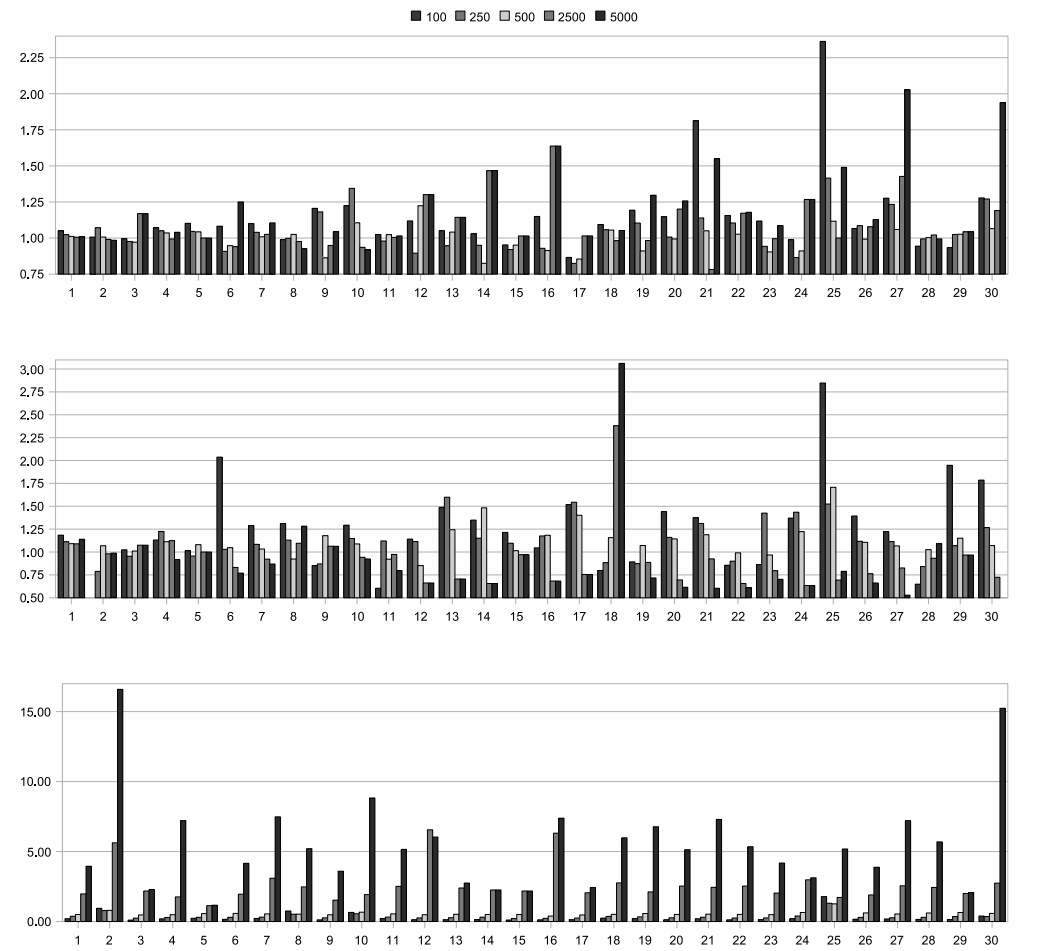


Fig. 20. Average testing error (*top*), storage requirements (*middle*) and execution time *bottom*) as a function of subset size for ICF algorithm. The plots show relative values with respect to the results using a subset of 1000 instances.

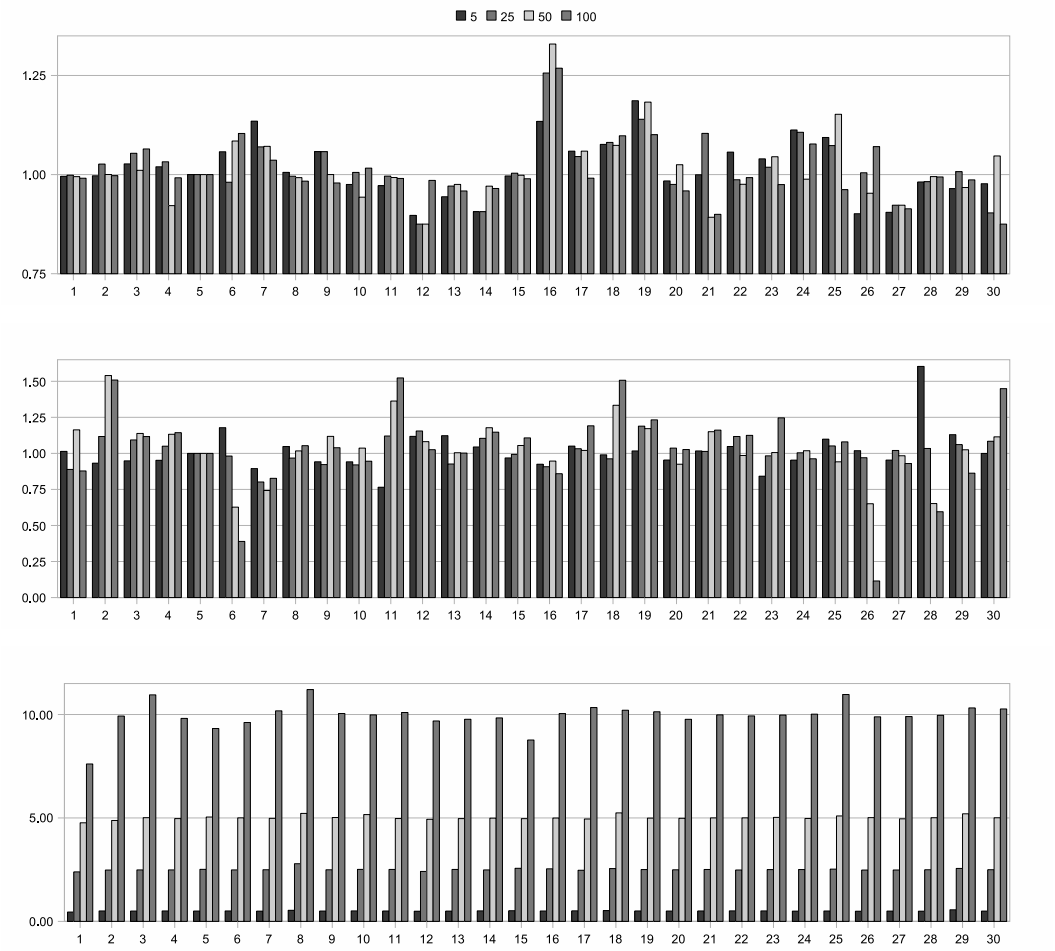


Fig. 21. Average testing error (*top*), storage requirements (*middle*) and execution time (*bottom*) as a function of number of rounds for DROP3 algorithm. The plots show relative values with respect to the results using 10 rounds.

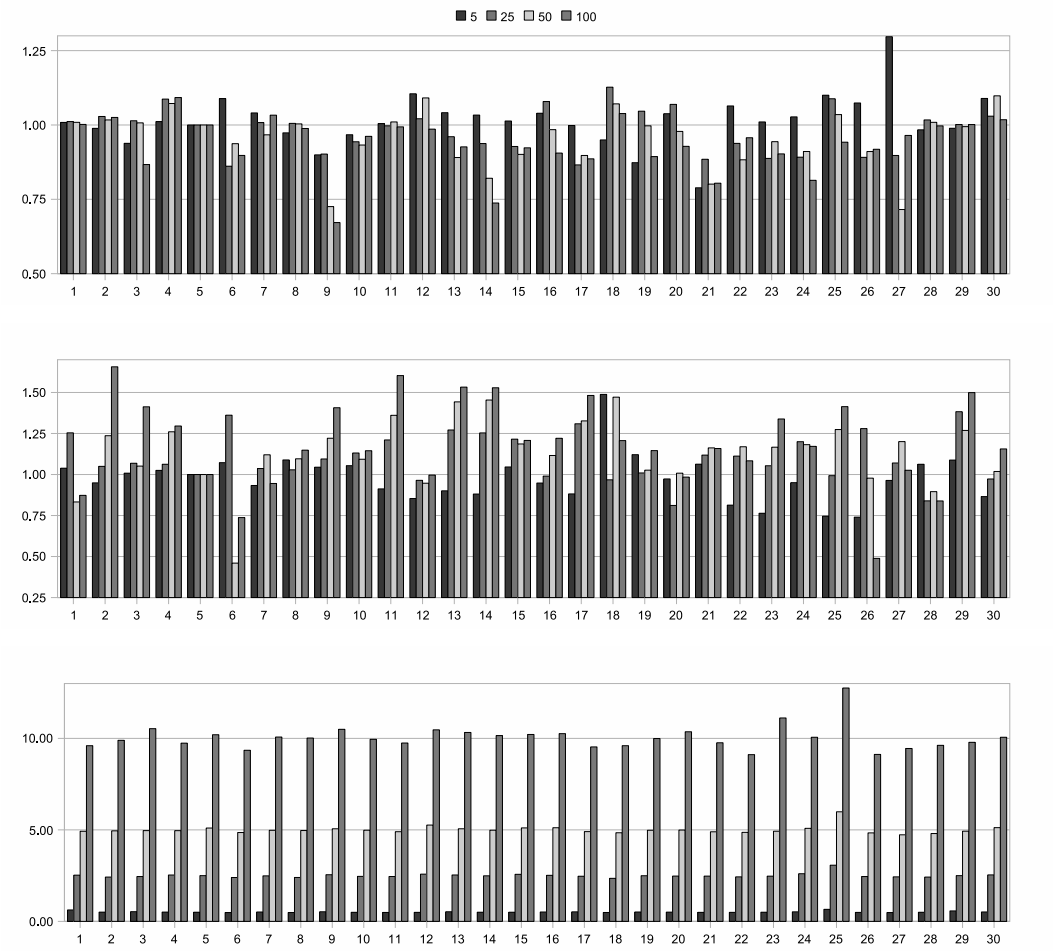


Fig. 22. Average testing error (*top*), storage requirements (*middle*) and execution time (*bottom*) as a function of number of rounds for ICF algorithm. The plots show relative values with respect to the results using 10 rounds.

4.4.6.10 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, from a few hundreds of thousands of instances to more than a million. Table 13 shows the problems that are used. These datasets will show whether our methodology allows scaling up standard algorithms to huge problems. As in the previous problems, the testing error and storage reduction is obtained using 10-fold cross-validation. The size of the datasets prevents the execution of the standard algorithms in a reasonable time, so the validity of our approach will be tested using the 1-NN 10-fold cv testing error shown in the table. For these problems we have used DEMOIS.drop3, DEMOIS.icf and DEMOIS.rnn.

Table 13
Summary of datasets. The features of each dataset can be C(continuous), B(binary) or N(nominal). The Inputs column shows the number of inputs of the network as it depends not only on the number of input variables but also on their type.

Data set	Cases	Features			Classes	Inputs	1-NN error
		C	B	N			
census	299285	7	-	30	2	409	0.0743
covtype	581012	54	-	-	7	54	0.3024
kddcup99	494021	33	4	3	23	118	0.0006
kddcup991M	1000000	33	4	3	21	119	0.0002
poker	1025010	5	-	5	10	25	0.4975

Table 14
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	20894.3
covtype	0.1627	0.3333	7352.0
kddcup99	0.0123	0.0066	44198.0
kddcup991M	0.0114	0.0019	89547.0
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	15120.0
poker	0.0483	0.5099	5265.3
	DEMOIS.rnn		
census	0.0006	0.0623	75181.0
covtype	0.2653	0.2955	190903.0
kddcup99	0.0063	0.0036	112947.0
kddcup991M	0.0026	0.0037	229273.0
poker	0.0001	0.4990	335141.7

Results are shown in Table 14. 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, DEMOIS.rnn for poker dataset, our approach spent 93 hours. This value is very good if we take into account that standard RNN took more than 500 hours in adult dataset, a problem with 48,842 instances whereas poker dataset has 1025,010 instances. Regarding the effectiveness of

the scalability the results are very good. The achieved testing error is close to the 1-NN error for all problems and methods, with the only exception of the covtype problem with 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. For covtype, the reduction is still large, although not so impressive.

4.4.6.11 Huge problems: Application to other methods of classification We have stated that our approach can be applied to other classifiers as well. Other learners can be benefited 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 as the constructed classifier would be simpler (Cano et al., 2007)(Sebban et al., 2000). When the instances selected are used as a training set for an instance base learner, such as an SVM or a decision tree, the term prototype selection is more often used 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 6. As it is the case for the described instance selected methods, prototype selection algorithms for decision trees, neural networks, or SVMs, suffer a problem of scalability. Thus, our method can contribute to scale up these algorithms as it 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 have chosen these two classifiers as their complexity depends on the quality of the training set (Sebban et al., 2000) 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 4.4.2 is specially designed for k-NN method. Thus, for the experiments with decision trees and SVMs we have used a simple random partition of the training set into disjoint subsets of approximately the same size.

As prototype selection algorithm, we can use any of the previously described. However, as 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 (Brodley and Friedl, 1999) is a filter

approach based on using a set of different classifiers as noise filters. These classifiers should detect the noisy, mislabeled, etc. instances and then remove them from the training set. The procedure is shown in algorithm 10. The authors propose 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. 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 algorithm of our choice is trained on the remaining instances. In majority vote the procedure is the same, but instances are discarded just if a majority of the classifiers misclassify them. In our experiments we have used the latter approach as the former resulted in removing very few instances. We will use the term Majority Vote Filter (MVF) algorithm to refer to this method.

Algorithm 10: Majority vote filter algorithm

Data : A training set $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ and a set of learners D
Result : The subset of selected instances S
foreach $d_i \in D$ **do**
1 | Train d_i on T
end
2 $S = T$
foreach $\mathbf{x}_i \in S$ **do**
| **if** $\mathbf{x}_i$ *is misclassified by a majority in* D **then**
3 | | Remove $\mathbf{x}_i$ from S
| **end**
end

As classifiers for D we have chosen a 1-NN classifier, a k-NN classifier where k is obtained by cross-validation, a C4.5 decision tree (Quinlan, 1993), an SVM with a linear kernel, and an SVM with a Gaussian kernel. Decision trees and SVMs are sensitive to parameters, so we have performed our experiments using cross-validation for setting the values of the parameters. For each one of the classifiers used we have obtained the best parameters from a set of different values. For SVM with a linear kernel we tried $C \in \{0.1, 1, 10\}$, 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 the 18 possible combinations. For C4.5 we tested 1 and 10 trials and softening of thresholds trying all the 4 possible combinations. The parameters chosen by cross-validation are then used to learn the classifier using all the available training data. This process is repeated each time a classifier is trained. Although this method does not assure an optimum set of parameters, at least we can be sure that a good set of parameters is obtained in a reasonable time. The SVM learning algorithm was programmed using functions from the LIBSVM library (Chang and Lin, 2001).

The experiments were performed with the same experimental setup of the previous ones. There is only a change that can be performed in the *democratic*

algorithm. The evaluation of the vote threshold (see Section 4.4.3) can be made using as evaluator a k-NN algorithm, as in the previous results, or we can use the classifier we are going to learn. The later method is more time consuming, but it is likely to produce better results. In this section we show results with the latter method, the former is faster but the results are slightly worse.

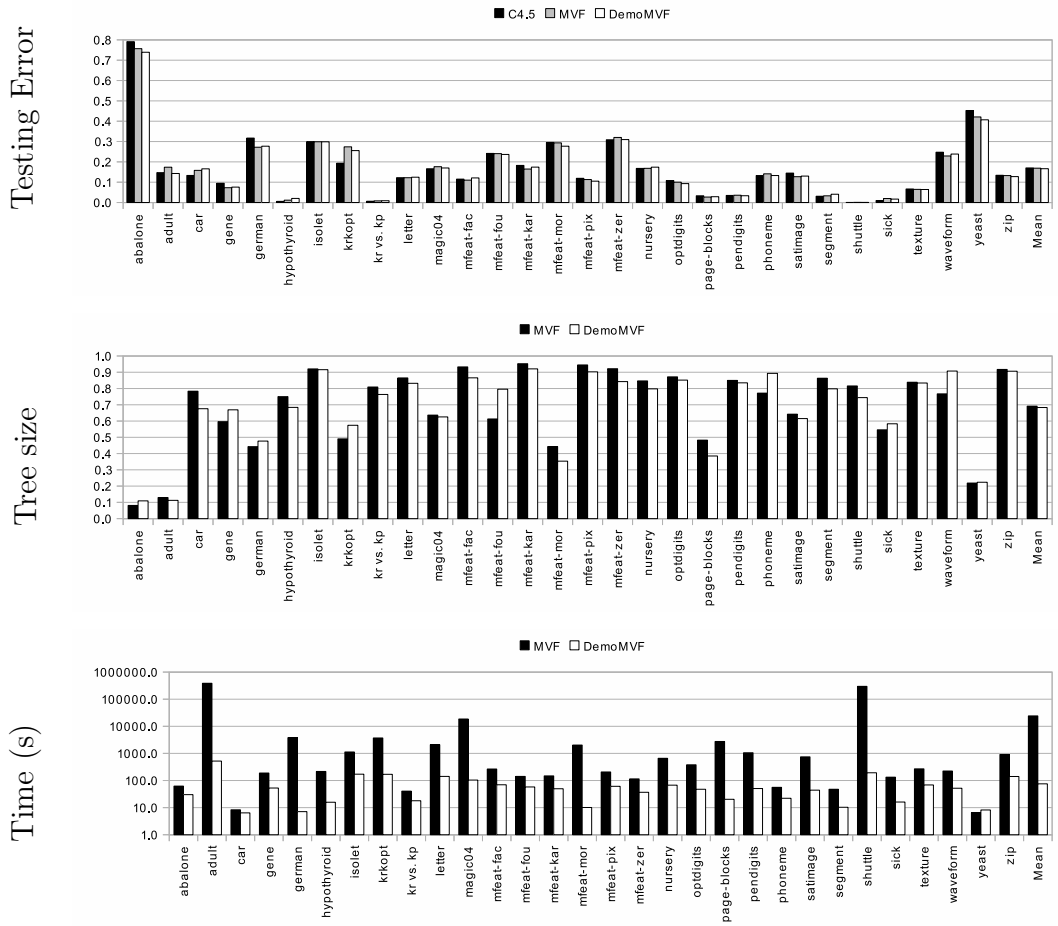


Fig. 23. Testing error, relative tree size, measured as the ratio of the number of nodes with respect to C4.5 applied to the whole dataset, and execution time in seconds (using a logarithmic scale) for standard MVF algorithm and our approach, compared with C4.5 algorithm applied to the whole dataset.

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, and finally we performed the same experiment using the *democratic* version of MVF, DEMOIS.MVF. Results for C4.5 classifier are shown in Table 15 and for SVM are shown in Table 16. These results are plotted in Figures 23 and 24 respectively.

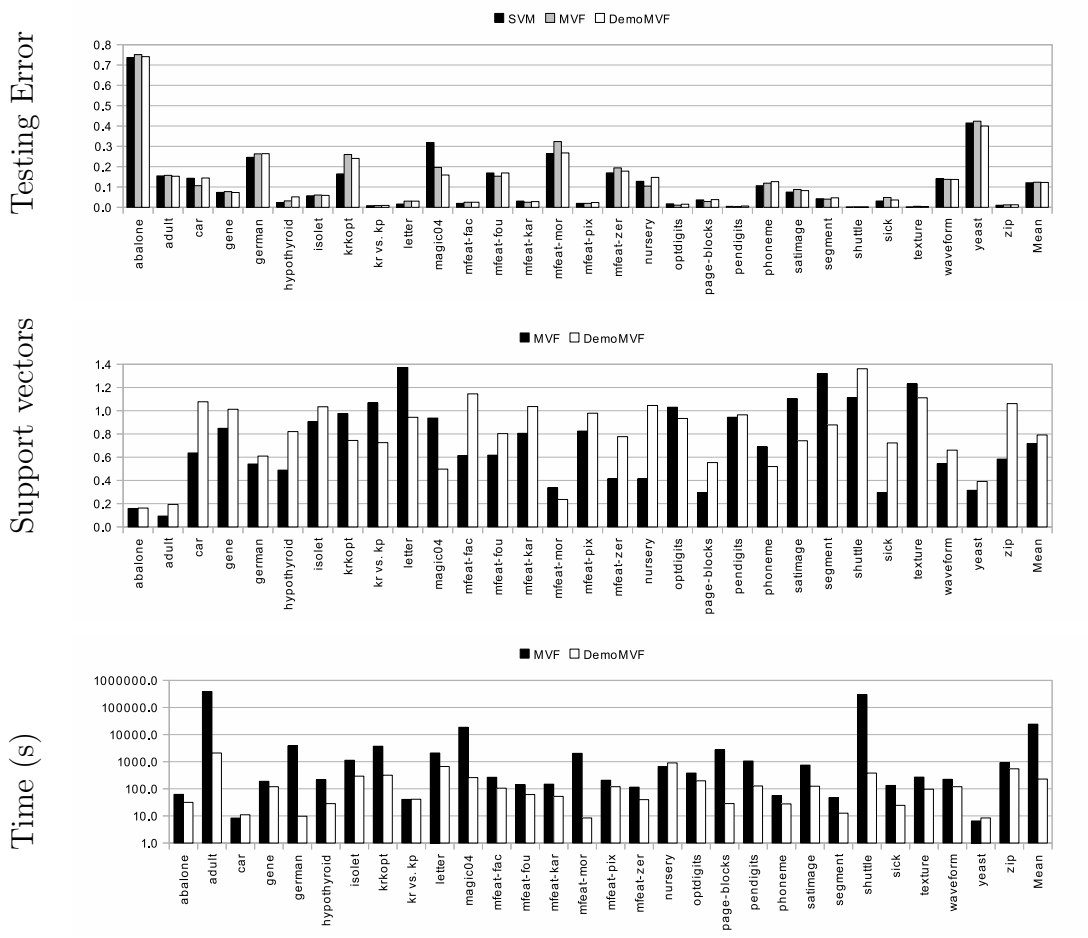


Fig. 24. Testing error, relative tree size, measured as the ratio of the number of support vectors with respect to SVM applied to the whole dataset, and execution time in seconds (using a logarithmic scale) for standard MVF algorithm and our approach, compared with SVM applied to the whole dataset.

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 the 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 is specially noticeable for adult and shuttle datasets. DEMOIS.MVF is able to keep the performance of MVF but with a very significant reduction in the execution time. As it was the case for the experiments using k-NN the reduction is more significant as the problem is larger, supporting our claim that the proposed method is able to scale up prototype selection algorithms efficiently as well as instance selection methods.

Table 15
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	387406.1	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	18577.2	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	298567.8	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 16
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	15175.3	0.1572	1415.5	387406.1	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	18108.7	0.2598	17658.3	3708.7	0.2404	13472.7	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	10112.7	2089.6	0.0305	6952.7	664.5
magic04	0.3190	4822.7	0.1964	4517.5	18577.2	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	298567.8	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

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 (Sebban and Nock, 2000) and SiS (Sane and Ghatol, 2007), can be used as well.

4.4.6.12 Huge problems: Noise tolerance Instance selection algorithms, as any other learning algorithm (Bauer and Kohavi, 1999), degrade their performance in the presence of noise. In the field of ensembles of classifiers, Dietterich

(2000a) tested the effect of noise on learning algorithms introducing artificial noise in the class labels of different datasets. Real-world problems do have noise, thus, it is relevant to study the behavior of any learning algorithm in presence of noise. In this section we study the sensitivity of our method to noise and compare it with standard algorithms.

In order to add noise to the class labels, we follow the method of Dietterich (2000a). In order 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 the DROP3 and ICF, and their democratic counterparts DEMOIS.drop3 and DEMOIS.icf. Figure **??** shows the results for the four methods at a noise level of 5%, 10% and 20%, and DROP3 and ICF algorithms. The figure shows 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 keep a good performance in presence of noise, as it uses partial views of the dataset which might be more sensitive to noise. The figures show how our method is able to keep its relative behavior with respect to the original algorithms as noise is added.

4.4.7 Parallel implementation of the DEMOIS. *algorithm: the* FEDERAL *algorithm*

In this section we show a parallel algorithm called the FEDERAL algorithm (Pedrajas et al., 2011) based on the described *democratization* approach, DEMOIS. algorithm. The FEDERAL algorithm is able to achieve an enormous reduction in the execution time of any instance selection algorithm while keeping its performance.

The parallel implementation is based on a master/slave architecture. The master performs the partition of the dataset and sends the subsets to each slave. Each slave performs the instance selection algorithm using only the instances of its subset and then returns the selected instances to the master. The master stores the votes for each removed instance. The general architecture of the system is shown in Figure 26. This method has the advantage that it is still applicable for very large datasets, as only a small part of the dataset must be kept in memory.

The threshold of votes is obtained using the same parallel FEDERAL approach. Again, we divide the dataset into disjoint subsets and evaluate the application of each threshold on every subset separately. The value of the goodness of a threshold is the average value of evaluating (9) in each subset.

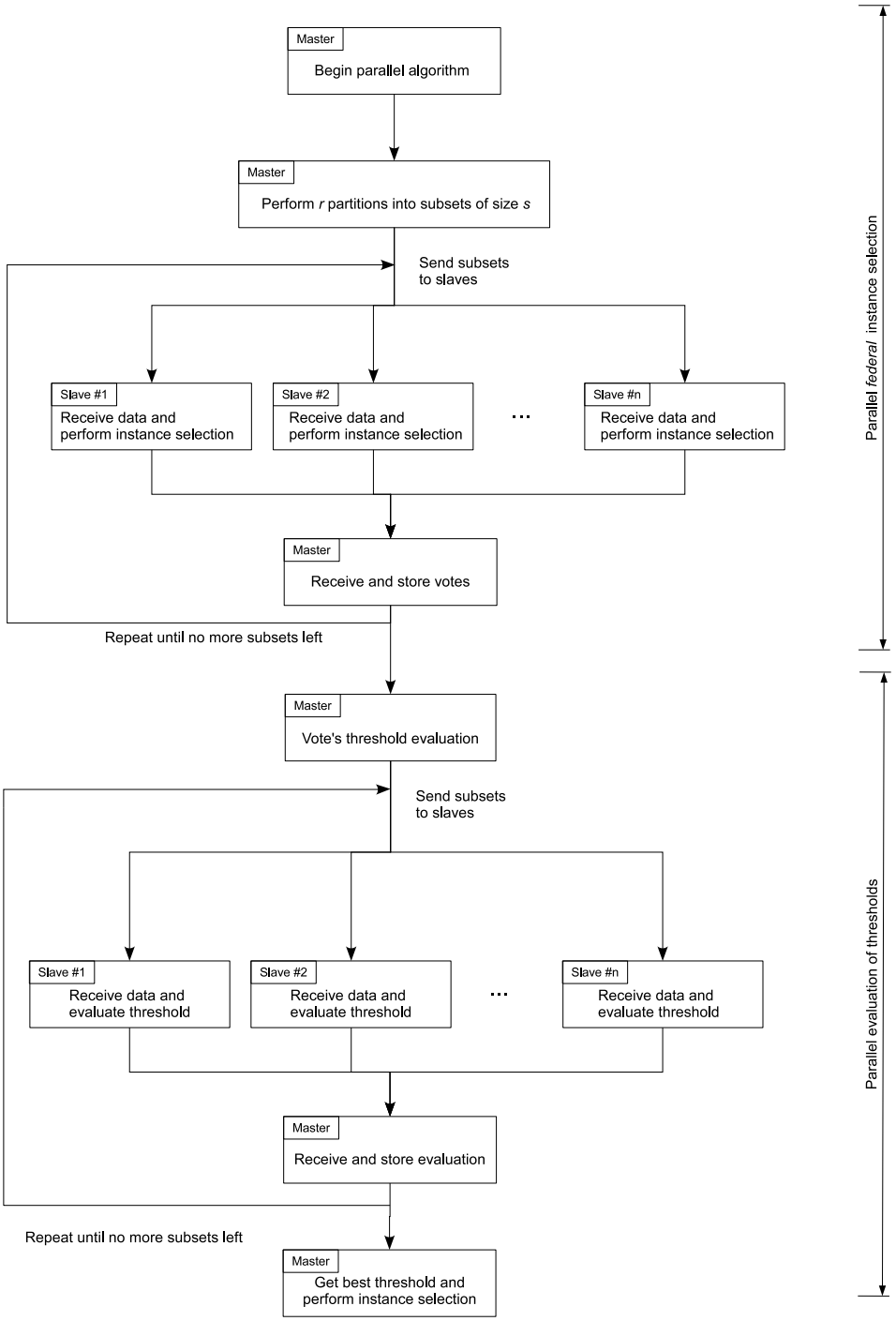


Fig. 26. Parallel implementation of FEDERAL instance selection.

On any parallel implementation, the communication between the different tasks is an important issue as an excess of information exchange harms the performance of the algorithm. In our method, the communication between the master and the slaves occurs only twice. It is interesting to note that during the execution of each instance selection algorithm in each slave no exchange of information between tasks is needed. The communication between the different

slaves and the master is needed in the following steps:

(1) Before each slave initiates its instance selection process, it must receive the subset of data to perform that process. This amount of information is always small as the basis of the method is that each slave takes care of only a small part of the whole dataset. Furthermore, if the slaves can access the disk, they can read the needed data directly from it.
(2) Once the instance selection process is finished, the slaves send the selection performed to the master. This selection consists of a list of the selected instances, which is a small sequence of integers.

When the process of selection is finished, the step of obtaining the best value for the votes threshold must be carried out. As shown, this step is also performed in parallel. The exchange of information between the master and the slaves is similar to the previous one:

(1) Before each slave initiates the evaluation of a certain threshold, it must receive the subset of data to perform that task. This amount of information is always small as the basis of the method is that each slave takes care of only a small part of the whole dataset. As in the previous case, if the slaves can access the disk, they can read the needed data directly from it.
(2) Once the evaluation process is finished, the slaves send the evaluation performed to the master. The evaluation is the error obtained when the corresponding threshold is used, which is a real number.

Thus, in both cases the data exchanged between the master and the slaves is not large, avoiding a bottleneck in the algorithm. The whole parallel procedure is shown in Algorithm 11. First, the algorithm sends a subset to every slave. Then it waits for the first slave to finish, gets the results and sends the finished slave a new subset, until all subsets are processed. The same procedure is performed for evaluating the best votes threshold. The final selection of instances in performed in line 14 of the algorithm.

The usefulness of our FEDERAL method is shown by an extensive comparison using 35 datasets of medium and large sizes from the UCI Machine Learning Repository, including imbalanced-datasets. Detailed results for the standard and *federalized* version are shown in Table 17 for CHC. The comparison between the standard version and the *federalized* one is shown in Table 18.

The results show a very good behavior of FEDIS.chc. In terms of testing error, our method is able to match the results of the standard method, and even improve them for some datasets. In terms of storage reduction, FEDIS is less efficient than standard CHC, but still achieve very good results, and the differences are not large. In terms of testing time, the improvement is very significant. In the most extreme case, `krkopt` dataset, FEDIS.chc is more

Algorithm 11: *Federal* instance selection (FEDIS) algorithm.

Data : A training set $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, subset size s, number of partitions r and number of processors p.

Result : The set of selected instances $S \subset T$.

for $i = 1$ ***to*** r **do**

1 Divide instances into n_s disjoint subsets $t_j^i : \bigcup_j t_j^i = T$ of size s

end

/* The total number of subsets to process is $r \times n_s$ */

/* Send first round of subsets to slaves */ subset := 1

3 partition := 1

for $i = 1$ ***to*** p **do**

4 Send subset $t_{subset}^{partition}$ to slave i

if *subset* $= n_s$ **then**

5 subset := 1

6 partition++;

end

end

/* Record slave selection and send new subsets to task */ **for** $i = 1$ ***to*** $r \times n_s$ **do**

7 Wait for an slave, w, to finish

8 Store votes of removed instances from slave w

if *More subsets to process* **then**

9 Send subset $t_{subset}^{partition}$ to slave w

if *subset* $= n_s$ **then**

10 subset := 1

11 partition++;

end

end

end

12 v = ObtainOptimumThresholdOfVotes (t) (Algorithm 12)

13 $S = T$

14 Final instance selection: Remove from S all instances with a number of votes $\geq v$

15 **return** S

than 2,400 times faster than standard CHC, from an average execution time of 128418.3 seconds to an average execution time of 52.8 seconds.

Table 18
Comparison of standard CHC algorithm and its *federalized* counterpart. The table shows the win/draw/loss ($w/d/l$) of the algorithm in columns against the algorithm in the row and the p-value of the Wilcoxon test (p_w).

		FEDIS.chc		
		Testing error	Storage	Time
	$w/d/l$	18/0/17	7/0/28	35/0/0
CHC	p_w	0.5888	0.0036	0.0000

Additionally, our method is applied to eight very large datasets (the largest set of 50 million instances and 800 features) with very good results and fast exe-

Algorithm 12: Algorithm for obtaining the optimum threshold of votes.

Data : A training set $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, subset size s, number of partitions r, partitions t, and number of processors p.

Result : The optimum threshold of votes v_{best}.

```
/* Send first round of subsets to slaves */ subset:= 1
partition := 1
for i = 1 to p do
    Send subset t_subset^partition to slave i
    if subset = n_s then
        subset := 1
        partition++;
    end
end
/* Record slave evaluation of f(v), v ∈ {1, 2, ..., r} */ for i = 1 to r × n_s do
    Wait for an slave, w, to finish
    Store f_w(v), v ∈ {1, 2, ..., r} from slave w
    if More subsets to process then
        Send subset t_subset^partition to slave w
        if subset = n_s then
            subset := 1
            partition++;
        end
    end
end
Set f(v), v ∈ {1, 2, ..., r} as the average evaluation for all slaves
v_best = argmax_{1,...,r} f(v)
return v_best
```

cution time. In most of the cases our method is able to match the performance of the original algorithm, even sometimes improving it, with a considerable reduction in execution time. Moreover, comparative results of our parallel implementation with stratification algorithm will be detailed in Section 4.4.8.2.

Table 17
Summary of results for standard CHC algorithm and our parallel implementation FedIS.chc.

Dataset	CHC			FedIS.chc		
	Storage	Error	Time (s)	Storage	Error	Time (s)
abalone	0.0499	0.7681	678.3	0.0392	0.7696	39.1
adult	0.2101	0.2356	93567.1	0.0191	0.1820	60.4
car	0.0433	0.1696	97.3	0.0435	0.1806	25.4
euthyroid	0.0104	0.0683	339.4	0.0129	0.0734	21.2
gene	0.0537	0.3259	356.1	0.0547	0.3388	23.5
german	0.0313	0.2920	29.8	0.0295	0.3240	7.9
hypothyroid	0.0023	0.0626	550.7	0.0058	0.0608	32.2
isolet	0.0360	0.1917	2477.2	0.1471	0.1397	46.7
krkopt	0.5337	0.4723	128418.3	0.0720	0.5077	52.8
kr vs. kp	0.0334	0.1226	333.8	0.0333	0.1517	21.3
letter	0.0611	0.1148	64400.0	0.2377	0.0702	25.1
magic04	0.0182	0.1803	48162.2	0.0234	0.1850	22.8
mfeat-fac	0.0395	0.0745	108.8	0.0428	0.0620	30.7
mfeat-fou	0.0565	0.2680	122.5	0.0649	0.2460	32.7
mfeat-kar	0.0607	0.1055	113.7	0.0708	0.1010	28.8
mfeat-mor	0.0247	0.2935	128.1	0.0310	0.3025	32.1
mfeat-pix	0.0442	0.0590	106.7	0.1051	0.0480	31.4
mfeat-zer	0.0528	0.2550	118.8	0.0574	0.2265	30.9
mushroom	0.0028	0.0011	3763.9	0.0508	0.0016	42.9
nursery	0.0463	0.2529	9955.8	0.0502	0.2697	26.4
optdigits	0.0301	0.0603	1211.1	0.1044	0.0461	14.8
ozone1hr	0.0011	0.0297	268.1	0.0044	0.0320	15.2
ozone8hr	0.0031	0.0636	231.8	0.0100	0.0739	15.0
page-blocks	0.0098	0.0459	1577.3	0.0117	0.0517	19.7
pendigits	0.0158	0.0207	8499.8	0.0276	0.0169	16.8
phoneme	0.0274	0.1518	1548.2	0.0330	0.1563	18.1
satimage	0.0241	0.1179	2057.1	0.0306	0.1123	24.2
segment	0.0347	0.0749	175.7	0.0502	0.0844	10.2
shuttle	0.2638	0.0055	76057.0	0.0048	0.0040	90.2
sick	0.0042	0.0411	557.6	0.0095	0.0448	31.3
texture	0.0345	0.0435	1235.1	0.0403	0.0426	15.7
titanic	0.0031	0.2122	271.8	0.0049	0.2117	52.6
waveform	0.0103	0.2426	818.3	0.0480	0.2266	12.3
yeast	0.0398	0.4514	75.6	0.0382	0.4291	17.6
zip	0.0351	0.0700	3940.8	0.0770	0.0566	13.4
average	0.0557	0.1698	12924.4	0.0482	0.1666	28.6

4.4.8 A comparison of DEMOIS. *algorithm and other approaches to scale up instance selection algorithms*

In this section we are going to compare the results of our best approach to scale by instances, DEMOIS., with its precursor the RECURSIVE METHOD and also with the high-performance stratification method (Cano et al., 2005) whose main features and similarities with our method have been previously explained in Section 2.

4.4.8.1 A comparison between Recursive and DEMOIS. *approaches* At the beginning of this Section 4 we have discussed how DEMOIS. algorithm is based on the recursive divide-and-conquer method (de Haro-García and Pedrajas, 2009). The original recursive method was able to obtain very good results in terms of execution time and storage reduction. However, the main drawback of that method is in the testing error, which was worse than that obtained if we applied the original method alone. Figure 27 shows a comparison of DEMOIS. and this recursive 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%.

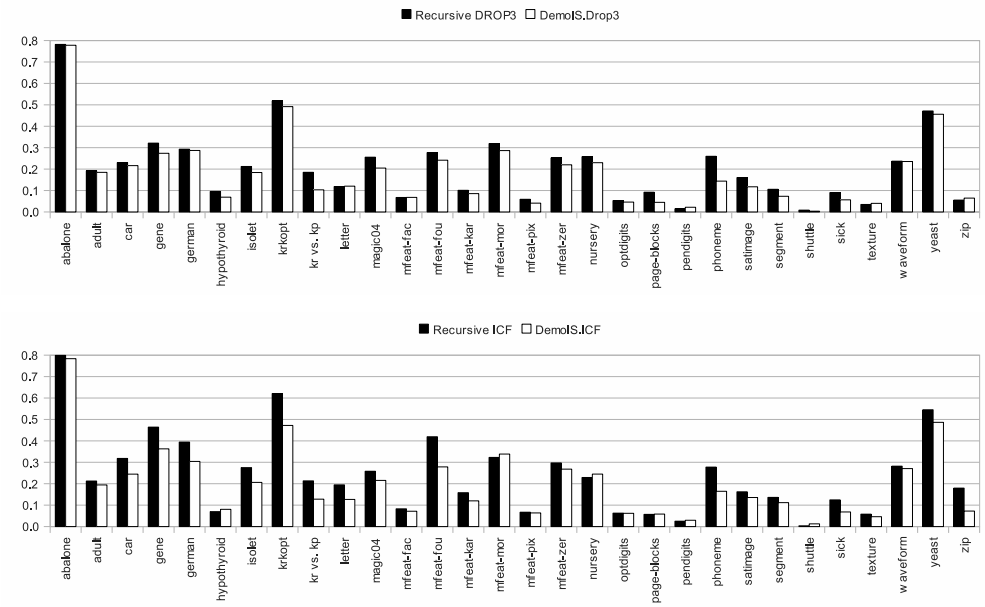


Fig. 27. Testing error for recursive and democratic instance selection using DROP3 (*top*) and ICF (*bottom*) as base methods.

4.4.8.2 A comparison between Stratification and DEMOIS. *approaches* In Section 4.2, we stated that stratification is a closely related method. In consequence we present the comparison of our approach and stratification using CHC as the base method.

For the comparison we have used a parallel implementation of both algorithms so as to save execution time in the experiments. The parallel implementation of DEMOIS. is one of our application papers, called FEDIS (Pedrajas et al., 2011)

Table 19
Comparison of parallel-stratification and FEDIS. The table shows the win/draw/loss ($w/d/l$) of the algorithm in columns against the algorithm in the row, and the p-value of the Wilcoxon test (p_w).

		FEDIS.chc		
		Medium/large sized problems		
		Testing error	Storage	Time
Stratified	$w/d/l$	23/3/9	20/0/15	18/0/17
CHC	p_w	0.0017	0.4128	0.0825
		Class-imbalanced problems		
		G-mean	Storage	Time
Stratified	$w/d/l$	29/0/11	20/0/20	3/0/37
CHC	p_w	0.0005	0.8088	0.0000
		Very large problems		
		Testing error	Storage	Time
Stratified	$w/d/l$	8/0/0	7/0/1	0/0/8
CHC	p_w	0.0100	0.0100	0.0100

We have used the same architecture of FEDIS to parallelize the stratification algorithm. It is worth to note that times reported show wall clock times including all the stages of the algorithms. All the parameters of the two methods, FEDIS and parallel-stratification, are the same, including a subset size of 1,000 instances.

The comparison between parallel-stratification and FEDIS is shown in Table 19. The table shows results for medium/large sized datasets, class-imbalanced datasets and very large datasets. The results for medium/large datasets are plotted in Figure 28. The comparison shows that FEDIS is significantly better than parallel-stratification in terms of accuracy (p-value of 0.0017) and as good as parallel-stratification in terms of reduction.

The comparison also shows that FEDIS is almost as fast as parallel-stratification. These results need explanation. In medium sized datasets, all the rounds of

voting can be made at once in the 256 cores, when we have less than 256 subsets. This occurs for datasets with less than 25,000 instances. For these datasets, FedIS is even faster than parallel-stratification. The difference in favor of FedIS comes from the stratified partition. Performing a partition that keeps class distribution is slightly more time consuming that our method. For large datasets, not all rounds of voting can be made concurrently, and then parallel-stratification is faster. However, the differences are not large for any dataset, being the largest difference below 54 seconds.

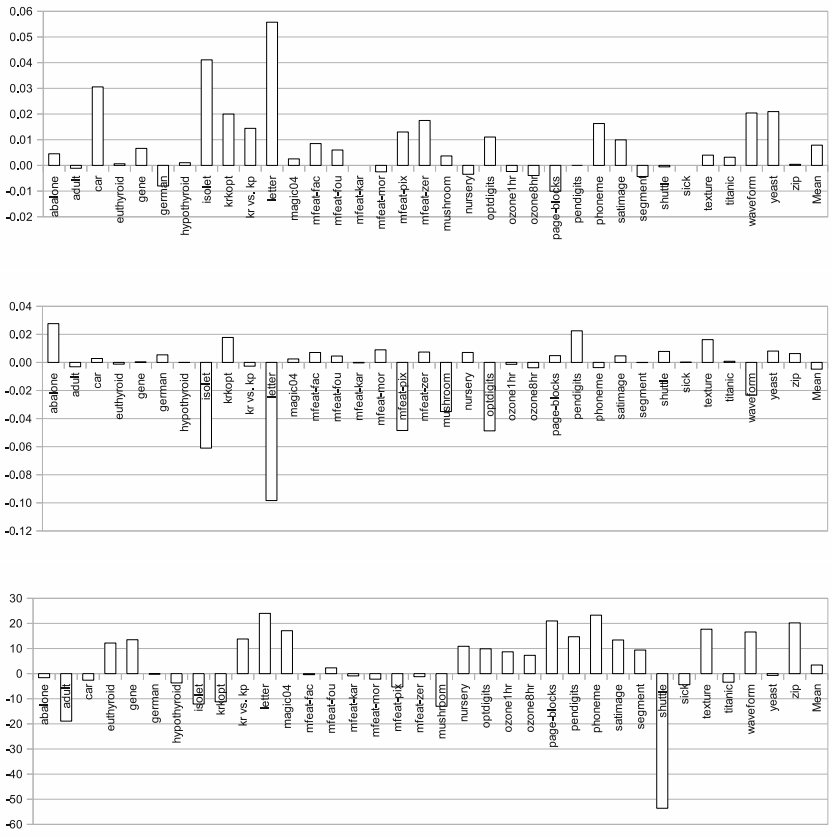


Fig. 28. Stratification method for CHC and FedIS.chc results. Error (top, difference between the parallel version of the stratification method and FedIS) storage requirements (middle, difference between the stratification method and FedIS) and speed-up of the stratification method with respect to the parallel algorithm (bottom).

The results for class-imbalanced problems are plotted in Figure 29. The comparison shows the same behavior of both algorithms.

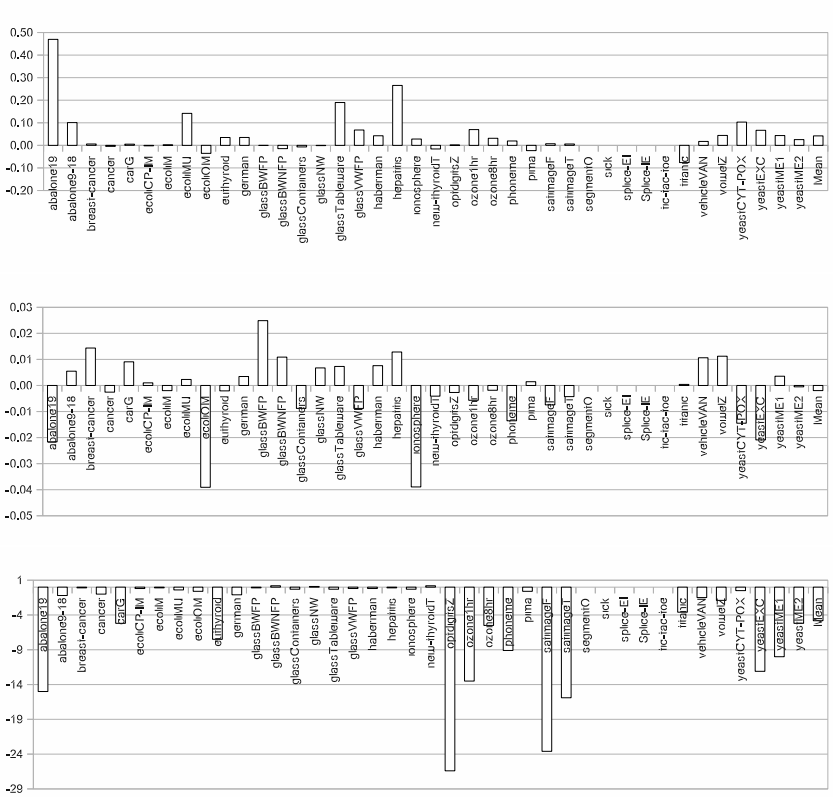


Fig. 29. Stratification method for CHC and FEDIS.chc results for class-imbalances problems. Error (top, difference between the stratification method and FEDIS) storage requirements (middle, difference between the stratification method and FEDIS) and speed-up of the stratification method with respect to the parallel algorithm (bottom).

Table 20
Summary of results for very large datasets using the parallel version of stratification.

Dataset	Storage	Testing error	G-mean	Time (s)
census	0.0318	—	0.7435	85.8
chrom21	0.2937	—	0.3481	375.0
covtype	0.0678	0.3805	—	55.9
dna	0.1225	—	0.1966	10905.0
kddcup99	0.0312	0.0040	—	50.6
kddcup991M	0.0290	0.0044	—	91.7
kddcup99all	0.0196	0.0084	—	520.0
poker	0.0558	0.5031	—	66.0

The results for very large datasets are shown in Table 20. A first interesting result is that stratification is faster, as it should be expected, but the differences are not large. This is because, although FEDIS performs 10 rounds of instance selection in different partitions, all these rounds can be done simultaneously. In fact, with a larger cluster, both stratification and FEDIS would

have constant time complexity. In terms of performance, FEDIS is better than stratification in testing error and reduction in all datasets with the only exception of `census`, where stratification achieved a better storage reduction. Wilcoxon test finds significant differences in both cases (see Table 19). In fact, in very large class-imbalanced datasets, the performance of FEDIS is clearly better than stratification.

In Section 4.4.4, we presented theoretical arguments for the linear time complexity of our approach, or even constant time complexity if we have enough processors. To illustrate this property, Figure 30 shows the behavior of the stratification method and our approach in terms of execution time for very large problems. We plot the time spent by the algorithms as the complexity of the problem increases. The plot corroborates our theoretical arguments and shows the linear time complexity of our approach even for very large datasets.

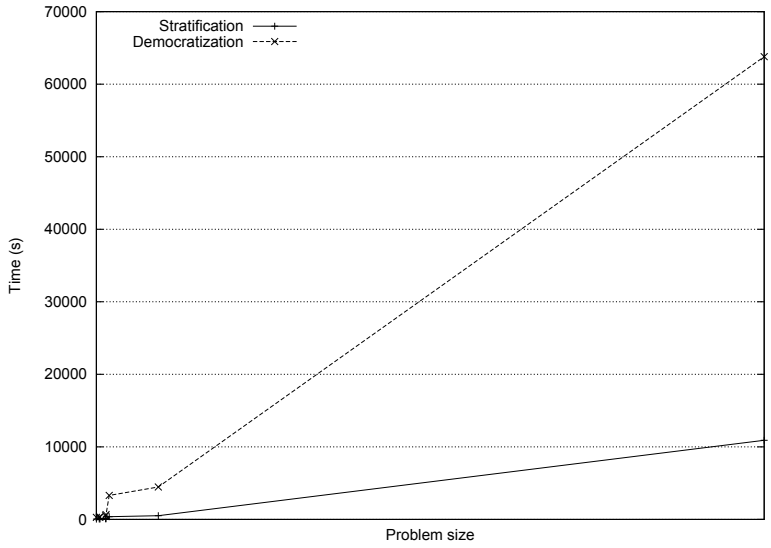


Fig. 30. Wall clock time spent by stratification and FEDIS using CHC as base algorithm and very large datasets

4.5 Summary

In this section we have described the application of our generic scaling methodology to instance selection algorithms.

Firstly we have presented a recursive approach which is the precursor idea of our main methodology to scale up instance selection algorithms, called DEMOIS.. As explained the recursive procedure partitions the original dataset into disjoint subsets, an instance selection algorithm is applied to each subset, and then the selected instances are rejoined to repeat the process. The

recursive method is able to improve the storage requirements of the original algorithms with a considerable reduction in execution time. However, the accuracy of the recursive method worsens the one of the original method alone.

In a next stage we designed the DEMOIS. algorithm, based on the previous method and taking into account the experimental results obtained so as to improve them. The method is applicable to any instance selection method without any modification. The DEMOIS. method consists of performing several rounds of applying instance selection on disjoint subsets of the original dataset. In this case, instead of recursively reducing the original set of instances, we perform independent selection phases and combine their results by means of a voting method. Using five well known instance selection algorithms, we showed that our method is able to improve the results of the recursive method and matches 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. For a more detailed comparison between DEMOIS. and the recursive procedure we refer the reader to Section 4.4.8.1. Moreover, DEMOIS. is able to scale up to huge problems with hundreds of thousands of instances, executing fast and achieving a very significant reduction of storage while keeping the testing error similar to the 1-NN error using the whole datasets.

Both recursive and demois methods are straightforwardly parallelizable without modifications. Consequently we have finally described how to parallelize our most efficient approach, DEMOIS., to benefit from an even more impressive reduction in the execution time of the scaling process. As detailed, the parallel version of DEMOIS. is called FEDERAL instance selection and it is able to match the performance of the original algorithm with a considerable reduction in execution time.

The results show that our parallel approach allows the scaling up of instance selection algorithms to problems of almost any size. Two features of our method are able to guarantee that scalability: firstly, the selection of instances is always performed over small datasets, keeping the time spent by the process low; secondly, only those small subsets of instances must be kept in memory, removing any scalability constraint due to memory limits.

A comparison with a parallel implementation of the stratification method, which shares the philosophy of our approach, showed that our proposal is better in terms of achieved accuracy whereas keeps the same reduction of stratification. The cost is a higher running time. In Section 4.4.8.2 the results of a complete set of comparative experiments are shown. This pattern of behavior has been shown in medium/large datasets, class-imbalanced problems and huge problems (up to 50 millions of instances).

5 Scaling up feature selection algorithms

Many recent pattern recognition problems involve highly complex datasets with large numbers of possible explanatory variables. For many reasons, this abundance of variables significantly hinders classification and recognition tasks. There are also efficiency issues, as the speed of many classification algorithms is significantly improved when the complexity of the data is reduced. One of the approaches to address the problems linked to having too many features is feature selection. However, feature selection algorithms have scalability issues when the number of features or instances is large.

After the successful application of our *democratization* methodology to scale up instance selection algorithms, in this section we are going to describe the application of our scaling approach to state-of-the-art feature selection algorithms. The resulting methodology consists of several rounds of "weak" feature selection processes whose outputs are combined into a single subset of relevant features. This method shares the design philosophy of the ensembles of classifiers, combining weak classifiers to obtain a strong one. We combine feature selectors instead of classifiers. The methodology can be used with any feature selection method.

We performed extensive comparative experiments proving that our method is especially efficient when we use feature selection algorithms that suffer from high computational cost or when we have to deal with very large datasets. During the fruitful collaboration with acknowledged Dr. Ludmila Kuncheva, we designed a simpler algorithm (de Haro-García et al., 2011) based on the described approach in order to benefit from the previously mentioned advantages when selecting relevant features in real world medical datasets. Specifically we worked with Functional Magnetic Resonance Imaging (fMRI) datasets, characterized by suffering from serious problems due to their huge amount of features. Each voxel in the 3-D image is taken to be a feature, and each image is an instance. The class label of the instance is determined by the type of stimulus presented at the time of the scan. Different images are shown to a set of individuals and the classification task is to determine the stimulus type.

This application will be summarized in section 5.6 and confirms the usefulness and practical applicability of our method.

5.1 Preliminary concepts on feature selection problems

Data mining (Agrawal et al., 1993), as a multidisciplinary joint effort involving databases, machine learning, and statistics, has been highly successful at turning mountains of data into nuggets. To use data mining tools effectively, data preprocessing is essential. Feature selection is one of the most important and frequently used techniques in data preprocessing for data mining (Blum and Langley, 1997) (Liu and Motoda, 2001). In contrast to other dimensionality reduction techniques, it preserves the original semantics of the variables, hence offering the advantage of interpretability by a domain expert (Saeys et al., 2007).

In real-world situations, the relevant features are often unknown a priori. Therefore, many candidate features are introduced to better represent the domain. Unfortunately many of these are either partially or completely irrelevant or redundant to the target concept. An irrelevant feature does not affect the target concept in any way, a redundant feature does not add anything new to the target concept, and a relevant feature is neither irrelevant nor redundant to the target concept (John et al., 1994).

Feature selection has been a fertile field of research and development since the 1970s in statistical pattern recognition (Jain and Zongker, 1997), (Mitra et al., 2002), machine learning (Blum and Langley, 1997), (Kohavi and John, 1997), and data mining (Dash et al., 2002), (Kim et al., 2000), and it has been widely applied to many fields, such as text categorization (Leopold and Kindermann, 2002), (Nigam et al., 1999), image retrieval (Rui and Huang, 1999), (Swets and Weng, 1995) customer relationship management (Ng and Liu, 1999), intrusion detection (Lee et al., 2000), and genomic analysis (Xing et al., 2001).

Feature selection can be defined as the selection of a subset of M features from a set of N features, $M < N$, such that the chosen subset optimizes the value of some criterion function over all subsets of size M (Narendra and Fukunaga, 1977). The objectives of feature selection are manifold, the most important ones being the following (Saeys et al., 2007):

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

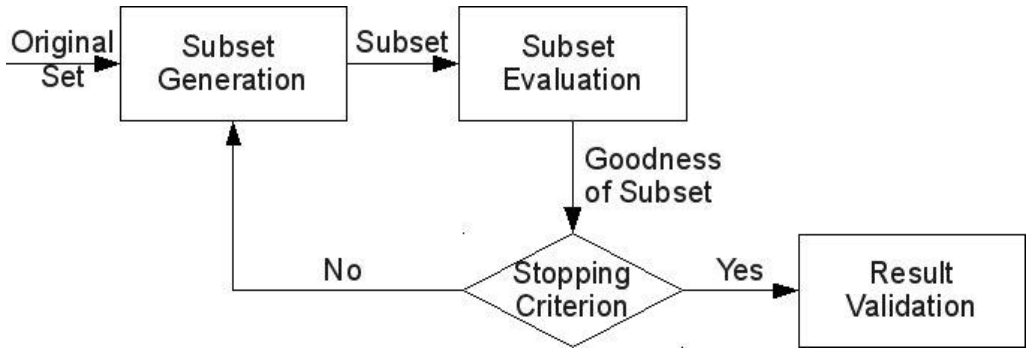


Fig. 31. Four basic steps of a typical feature selection process.

The advantages of feature selection techniques are well known, but we will have to deal with a scalability problem if we apply these techniques to large datasets [7]. The advantages of feature selection come at a certain price, as the search for a subset of relevant features introduces an additional layer of complexity to the modeling task. This new layer increases the memory and running time requirements, making these algorithms very inefficient when applied to problems that involve very large datasets. Ironically, standard feature selection becomes impracticable on large datasets, which are the ones that would benefit most from its application. Therefore, with the increasing size of datasets in all fields of application, the need to scale up feature selection algorithms is also growing.

Instead of scaling a certain feature selection algorithm, we might opt to select a subset of the whole dataset and disregard the remaining data. However, there are reasons for scaling up learning algorithms. The main reason is that increasing the size of the training set often increases the accuracy of the learned models(Provost and Kolluri, 1999). In many cases, the degradation in accuracy when learning from smaller samples stems from overfitting, due to the need to allow the program to learn small disjuncts(Holte et al., 1989), elements of a class description that cover few data items. The existence of noise in the data further complicates the problem because with a small sample, it is impossible to tell the difference between a special case and a spurious data point. Thus, using a small subset of the whole large dataset is not likely to produce usable results, and scaling up the learning method becomes a must.

A typical feature selection process consists of four basic steps (shown in Figure 31), namely, subset generation, subset evaluation, stopping criterion, and result validation (Dash and Liu, 1997).

The generation procedure is a search procedure that produces candidate fea-

[7] Datasets can be large in that they have a large number of instances or a large number of features. A certain feature selection algorithm may have a scalability problem when there are many features, many instances, or both.

Fig. 32. Schematic view of filter methods.

ture subsets for evaluation based on a certain search strategy (Blum and Langley, 1997). The generation procedure can start *(i)* with no features, *(ii)* with all features, or *(iii)* with a random subset of features. In the first two cases, features are iteratively added or removed, whereas in the last case, features are either iteratively added or removed or produced randomly thereafter. Feature selection methods generate candidates randomly or using a deterministic procedure. An evaluation function measures the goodness of the subset produced, and this value is compared with the previous best value. If the new value is found to be better, then the new subset replaces the previous best subset. The process of subset generation and evaluation is repeated until a given stopping criterion is satisfied (e.g., a predefined number of features are selected or a number of iterations reached). The feature selection process ends by outputting a selected subset of features to a validation procedure (Dash and Liu, 1997).

In the context of classification, feature selection techniques can be organized into different categories. The output type divides feature selection algorithms into two groups (Liu and Yu, 2005): ranked list and minimum subset. The essential difference between the two concerns the presence of an order on the selected features. There is no order among the features in a selected subset. One cannot easily remove any more features from a subset, but one can do so for a ranked list by removing the least important one.

If we consider how different methods combine the feature selection search with the construction of the classification model, we can identify three categories (Saeys et al., 2007): filter methods (Dash et al., 2002), (Liu and Setiono, 1996), (Yu and Liu, 2003), wrapper methods (Caruana and Freitag, 1994), (Dy and Brodley, 2000), (Kohavi and John, 1997), and hybrid/embedded methods (Das, 2001), (Xing et al., 2001):

(1) Filter techniques (see Figure 32) rely on the intrinsic properties of the data to evaluate and select feature subsets without involving a mining algorithm. The advantages of filter techniques are that they easily scale to very high-dimensional datasets, they are computationally simple and fast, and they are independent of the classification algorithm. As a result, the feature selection must be performed only once, and then different classifiers can be evaluated.
(2) Wrapper methods (see Figure 33) embed the model hypothesis search within the feature subset search. In this setup, a search procedure in the space of possible feature subsets is defined, and various subsets of features

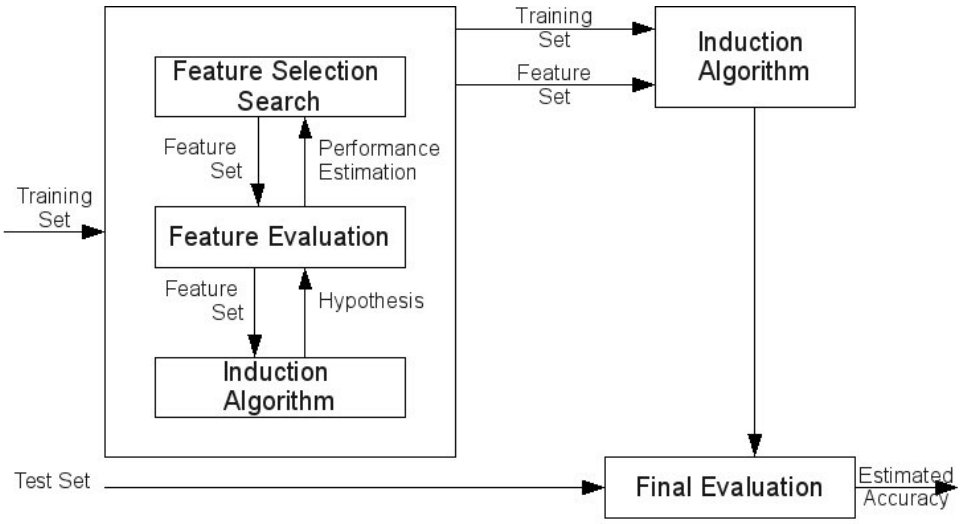


Fig. 33. Schematic view of wrapper methods.

are generated and evaluated. To search the space of all feature subsets, a search algorithm is then *wrapped* around the classification model. However, as the space of feature subsets grows exponentially with the number of features, heuristic search methods are used to guide the search for an optimal subset. The evaluation of a specific subset of features is obtained by training and testing a specific classification model. The advantages of these models include the interaction between feature subset search and model selection and their ability to take into account feature dependencies. A common drawback is that they have a high risk of over-fitting and are very computationally intensive.

(3) Hybrid/embedded techniques attempt to combine the advantages of the first two models by building the search for an optimal subset of features into the classifier construction. Like wrappers, they are specific to a given learning algorithm. Embedded methods have the advantage that they include the interaction with the classification model, while at the same time being far less computationally intensive than wrapper methods.

As previously stated, when the dimensionality of a domain expands, the number of features f increases. In these cases, finding an optimal feature subset is usually intractable (Kohavi and John, 1997), and many problems related to feature selection have been shown to be NP-hard (Blum and Rivest, 1992). The same scalability problems appear when the number of instances is large, as most feature selectors evaluate some criterion on the whole dataset. To be able to work with large datasets that cannot be processed by an ordinary feature selection process (it would be too computationally intensive), we propose a new methodology called *pseudoensembles* of feature selectors, which consists of performing several rounds of fast feature selectors.

5.2 Related Work on Large-Scale Feature Selection Problems

Over the last years, some interesting work has been done on large-scale learning (Sonnenburg et al., 2007) (Breiman, 1999) (Chawla et al., 2004). As we have previously emphasize, scaling up learning algorithms, the issue is not as much one of speeding up a slow algorithm as one of turning an impracticable algorithm into a practicable one. The crucial issue is seldom "how fast" we can run on a certain problem, but it is rather "how large" a problem we can (feasibly) deal with.

Not many previous studies have dealt with feature selection for large problems. It is worth mentioning the work of Sikonja (1998), in which they sped up the Relief algorithm by means of k-d trees. Although the algorithm shows very good performance, k-d trees add new memory requirements and are less efficient if the trees are not balanced. Furthermore, this method is not applicable to other feature selection algorithms.

As a way to overcome scaling up problems, the relevance of the features can also be evaluated individually. Univariate approaches are simple and fast and are therefore appealing. However, they do not consider possible correlations and dependencies between the features. Therefore, multivariate search techniques must be used, which may be computationally too expensive when dealing with a large feature space, as in our case. To solve this problem, Lai et al. (2006) proposed a data partitioning method that partly shares our philosophy. They sample T feature subsets randomly and then apply a feature selection process to each subset by means of SVM-RFE or the Liknon algorithm. Each feature that happens to be in a subset receives a value (possibly a rank) in comparison with the rest of the set. Finally, the feature values are tallied, and a single ranking is produced. Our method has the advantage of having a more complete view of the problem, as it learns from the whole dataset partitioned into subsets, rather than only from several isolated samples. Randomly sampled subsets may miss some features that are key to the learning process.

Gadat and Younes (2007) propose an algorithm that attributes a weight to each feature, proportional to its importance for the classification task. The entire set of weights is optimized by a learning algorithm based on a training set. These weights result in an estimated probability distribution over the features and are optimized by means of a stochastic gradient descent algorithm using the training set. Their method is successfully tested on face detection, handwritten digit recognition, spam classification and micro-array analysis.

Bins and Draper (2001) address large feature selection problems applied to computer vision datasets by proposing a three-step algorithm. The first step

uses a variation of the well-known Relief algorithm to remove irrelevance, the second step clusters features using K-means to remove redundancy, and the third step is a standard combinatorial feature selection algorithm: when possible (less than 110 features), they use the Sequential Floating Backward Selection (SFBS) algorithm. When the number of features remaining after the relevance and redundancy filters exceeds 110, they switch to the less effective Sequential Floating Forward Selection (SFFS) algorithm.

5.3 Pseudoensembles Feature Selection Method

As previously mentioned, when the dimensionality of a domain expands, the number of features N increases and many problems related to feature selection have been shown to be NP-hard (Blum and Rivest, 1992). In these cases, finding an optimal feature subset by an ordinary feature selection process is usually intractable because it would be too computationally intensive (Kohavi and John, 1997). We propose an application of our DEMOCRATIZATION scaling to feature selection, which consists of performing several rounds of fast feature selectors.

More specifically, for each step i, we divide the dataset S into several small disjoint datasets $S_{i,j}$. The feature selection algorithm is applied with no modifications to each one of these subsets, and a selection $C_{i,j}$ is produced from each subset of the data. After all the n_s rounds are applied, (which can be done in parallel, as all of them are independent from each other), the combination method constructs the final selection C as the result of the feature selection process.

As stated, our method can be summarized in three stages:

- Partition of the datasets.
- Application of feature selection to the subsets.
- Combination of the results.

These three stages are detailed in the following sections. There are two different versions of the method depending, on whether the basic feature selection algorithm outputs a subset of instances or a ranking. These two versions are shown in algorithms 13 and 14, respectively. Due to the fact that one of the versions of our method does not use a "democratic" voting process (specifically the one that outputs a ranking of features) we decided to call our methodology: *pseudoensembles* of feature selectors.

As the experiments will show, the most important advantage of our method is the large reduction in execution time. Moreover, our approach has a very competitive complexity, as it is linear in the number of instances or in the number of features (depending on which factor determines the complexity of the feature selection base algorithm).

As previously stated, the method has the additional advantage of allowing an easy parallel implementation. The application of the feature selection algorithm to each small subset of instances or features is independent of all the remaining subsets. All the subsets can be processed at the same time. Even different rounds can be run at once. Moreover, there is little communication between the nodes of the parallel execution, only in the combination step to

compute the final ranking or subset of features. This allows a complete parallelization of the methodology. In fact, if we have a sufficiently large cluster of machines, all the tasks can be performed at the same time. In such a case, the proposed methodology is of constant complexity.

Furthermore, we can choose the complexity of the execution in each one of the processors because the size of the subsets is a parameter of the methodology. Another parameter is the feature selection algorithm, so any feature selection algorithm can be applied in the *pseudoensembles* framework with no modifications.

5.3.1 Partition of the datasets

The partition process consists of dividing the original dataset into several disjoint subsets of approximately the same size that cover the full dataset.

In contrast to the complex partitioning method employed in the previous scaling of instance selection algorithms, we are going to use the simplest method available, strictly a random partition. In the present partitioning method, each feature or instance is randomly assigned to one of the subsets. We need a different partition of the dataset for each round of the algorithm; otherwise, the results will be the same because the subsets will be identical.

It is worth noting that we do not consider the application of the learning algorithm to a subset as a solution to the problem; in that way, the combination is not made at the round level. A standard sampling approach was discarded for the partition because it may never pick some features or instances (depending on which factor is the basis of the partition). Our method takes into account all features or instances in each round. Therefore, we can state that each element of the entire dataset is present the same number of times in the learning/testing process, once per round. These rounds will not select the best features on their own because they only have partial knowledge of the original dataset. They can be seen as weak feature selectors that provide a fast approximate solution that suffer from locality, and moreover, they are more sensitive to noise. Nevertheless, the key to the success of our method comes from the combination of all these rounds based on the quality of the selected features. This approach succeeds in selecting good features globally while keeping the complexity and execution time manageable. The combination of rounds applied in the *pseudoensembles* will be detailed at the end of this section.

The partition of the datasets can be carried out in two ways: we can choose to split datasets by instances or by features [8] . This decision should be closely

[8] It is also possible to partition the datasets by features and instances together.

related to the complexity of the feature selection algorithm selected to be the core of our method. Depending on the design, the complexity of a certain feature selection may depend on the number of features, or, on the other hand, it may come from the number of instances in the dataset. That is the reason why it would be more highly recommended to divide datasets by one or the other, depending on which factor specifically relies on the complexity of the algorithm involved.

Preliminary experiments have shown that the size of the subsets has no significant impact on the results of the method, provided that it is small enough to avoid large execution times and large enough to allow a meaningful application of the selection process. 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. When partitioning by instances, values in the interval $[100, 1000]$ are recommended. When partitioning by features values in the interval $[5, 11]$ are also advisable.

5.3.2 Application of feature selection to the subsets

The feature selection algorithm is applied to all the datasets in several times or iterations, which we call 'rounds'. This repetition ensures that we have gathered enough information for the combination step to be useful. The advantage of our method is that it does not need any modification of the learning algorithm, thus saving the time spent in adapting known algorithms. Although the same feature selection algorithm will be applied to different subsets, it will produce different results that can be advantageously combined. This is a similar philosophy to the construction of ensembles of classifiers (Rokach, 2009).

pseudoensembles deal with the feature selection algorithms as black boxes, so it is possible to include any feature selection in our framework with no modifications. Thus, the output of our method is dynamic, depending on which type of output was originally provided by the feature selection algorithm. If the feature selection method computes a ranking of the features, we will also return a ranking as the final output and let the user establish her/his own threshold to select the most relevant features for her/his problem. However, if the feature selection method returns a subset of the most relevant features, the *pseudoensembles* will also produce a final subset of relevant features. In the following two sections, we explain the application of each of these possibilities.

In such a case, each subset will be formed by a subset of instances and a subset of features. This possibility has yet to be explored experimentally.

5.3.2.1 Pseudoensembles using subset selection The feature selection algorithm is applied to each subset separately. The features 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 made, the features that have received a number of votes above a certain threshold are removed. An outline of the method is shown in Algorithm 13.

Algorithm 13: *Pseudoensembles* using subset selection

Data : $\mathbf{Z} = \{X, Y\}$:A labeled training dataset, with features $X = \{x_1, \ldots, x_n\}$ and class labels $Y = \{y_1, \ldots, y_n\}$
J(S): Evaluation criterion for subset $S \subseteq X$ included in the feature selection method applied
$\boldsymbol{n_s}$: Number of subsets of M instances or M features (depending on whether we are splitting by instances or by features)
r: Number of rounds/iterations performed r

Result : The set of selected features $S \subseteq X$.

```
Initialize the array of votes with n zeros: V(x) = 0, x ∈ X
for i = 1 to r do
    Split Z randomly into n_s disjoint subsets of instances or features of size M
    for i = 1 to n_s do
        Apply feature selection algorithm to S_j
        For all features x ∈ S_j, add a vote to the features to be removed in the
        subset S_j
    end
end
Obtain threshold of votes, v, to remove a feature
S = T
Remove from S all features with a number of votes above or equal to v
return S
```

However, as stated so far, there is still an important issue to be addressed in the subset selection before we can obtain a useful algorithm: the determination of threshold of votes required to remove a feature, which is problem-dependent. Depending on the problem, a certain threshold may be too low or too high. Thus, a method must be developed for the automatic determination of the number of votes needed to remove a feature from the training set. The automatic determination of this threshold has the additional advantage of relieving the researcher of the duty of setting a difficult parameter of the algorithm.

5.3.2.2 Pseudoensembles using ranking of features Ranking the features based on their individual merit has at least two flaws. First, features that are not very relevant on their own but become relevant in a group will receive poor scores and may be dropped off at an early stage of the feature selection

cascade. Second, relevant but possibly redundant features will occupy the top spots. Consequently, ranking methods usually try to evaluate the merit of each feature with respect to the remaining features. This evaluation requires complex algorithms where the scalability problem appears.

In our framework, if we use as the base feature selector a method that outputs a ranking of the variables, the ranking of each variable on every subset will be recorded. The resulting ranks will be combined in the combination step. The whole method when using rankings of variables is outlined in Algorithm 14.

Algorithm 14: *Pseudoensembles* using ranking of features

Data : $\mathbf{Z} = \{X, Y\}$:A labeled training dataset, with features $X = \{x_1, \ldots, x_n\}$ and class labels $Y = \{y_1, \ldots, y_n\}$
J(S): Evaluation criterion for subset $S \subseteq X$ included in the feature selection method applied
$\boldsymbol{n_s}$: Number of subsets of M instances or M features (depending on whether we are splitting by instances or by features)
r: Number of rounds/iterations performed r

Result : Ranking of relevant features

```
Initialize the score array with n zeros: F(x) = 0, x ∈ X
for i = 1 to r do
    Split Z randomly into n_s disjoint subsets of instances or features of size M
    for i = 1 to n_s do
        Apply feature selection algorithm to S_j
        For all features x ∈ S_j, update the scores as F(x) = F(x) + J(S_j)
    end
end
Compute the mean feature scores F̄ = F/r
Sort features by descending F̄_x
return Ranking of relevant features
```

5.3.3 Combination of the results

The combination of results is a key step in our method. We are dealing with simple feature selectors, that we call "weak feature selectors", in the same sense that weak classifiers are combined in an ensemble. We have opted for a simple combination method, following the experience of classifier ensembles, where complex methods have not achieved consistently better performance than simple ones (Kuncheva, 2001).

We must not forget that *pseudoensembles* treat feature selection algorithms as black boxes, and they produce their final output dealing with the original form of the outputs computed after the independent runs of the feature

selection algorithms over different subsets. The following sections specify the combination step for each of the two possible outputs of the *pseudoensembles*.

5.3.3.1 Combination of rankings of features To combine the different rankings of each round into a final ranking, we simply compute the average ranking value for each feature and then sort them into a new rank. This new averaged rank is the final ranking achieved by the pseudoensemble.

It is worth noting that the ranks obtained in each round are related to the weights provided by the feature selection algorithm for each feature. We consider the option of averaging the weights received by each feature in the rounds. Nevertheless, we think that averaging ranks is a better option because the weight provided by the feature selection algorithm for an individual feature is closely attached to which other features are active in the partition being evaluated. Taking into account that the partitions are constructed randomly, the average of the weights would be more sensitive to the noise added by the subset selection.

5.3.3.2 Combination of subsets of features As previously stated, the votes for a feature to be removed will be accumulated over the different rounds. After the rounds take place, we will have a number of votes for each feature. This number represents how many times the feature has been selected to be removed by the application of the feature selector. An important issue in our method is determining the number of votes needed to remove a feature from the training set. Preliminary experiments showed that this number highly 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 very time consuming. A second choice is to estimate the best value for the number of votes from the effect on the training set. This latter method is the one we have chosen.

Our choice is to estimate the best value for the number of votes from its effect on the training set, specifically using 10% of the dataset so as to speed up the process.

The selection of the number of votes must take into account two different criteria: the training error, ϵ_t and the memory requirements (percentage of features retained), m. Both values must be minimized to the extent possible. Our method of choosing the number of votes needed to remove a feature is to calculate the threshold number of votes v that minimizes a fitness criterion $f(v)$:

$$f(v) = \alpha\epsilon_t(v) + (1 - \alpha)m(v), \tag{9}$$

where α is a value in the interval $[0, 1]$ that measures the relative relevance of both values. In general, the error minimization is more important than storage reduction; thus, we have used a value of $\alpha = 0.75$. Different values can be used if the researcher is more interested in storage reduction than in error reduction.

We perform r rounds of the algorithm and store the number of votes received by each feature. Then we must obtain the threshold number of votes v in the interval constituted by the minimum and maximum number of votes received by any feature. We calculate the criterion $f(v)$ (see eq. 9) for all the possible threshold values from 1 to r, and assign to v the value which minimizes the criterion. After that, we perform the feature selection by removing the features whose number of votes is above or equal to the obtained threshold v.

5.4 Experimental Setup

The experimental setup used in this work is the same as the one previously detailed in instance selection Section 4.3.3, except for some minor details we proceed to discuss in the following lines.

In order to make a fair comparison between the standard algorithms and our scaling proposal, we have selected 30 problems from the UCI Machine Learning Repository (Hettich et al., 1998). Most of the datasets selected are the same as in the recursive method and the DEMOIS. algorithm. However, due to the fact that some of the datasets are different, their features can be consulted on Table 21. These datasets can be considered as representative of problems from medium to large size.

When evaluating methods that select a subset of features, we will consider the testing error and the reduction ratio achieved by the methods we are comparing. The case when using methods that output a ranking of features is different, as we do not have a value for the testing error or the reduction as the final result of the algorithm. For these methods, we will use a graphical comparison using plots of the testing error in function of the number of features selected. Additionally, to carry out a numerical comparison, we will take into account two numerical values that summarize the performance of the methods. From the plot of the testing error against the number of features, we will obtain an area under the curve (AUC) measure. Higher values of the AUC mean that the method is selecting better features. However, this value is influenced by the testing error obtained when only a few features are selected. In most cases,

Table 21
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, as it depends not only on the number of features but also on their type.

	Data set	Instances	Features			Classes	Inputs
			C	B	N		
1	abalone	4177	7	-	1	29	10
2	car	1728	-	-	6	4	16
3	gene	3175	-	-	60	3	120
4	german	1000	6	3	11	2	61
5	hypothyroid	3772	7	20	2	4	29
6	isolet	7797	617	-	-	26	617
7	kr vs. kp	3196	-	34	2	2	38
8	letter	20000	16	-	-	26	16
9	magic04	19020	10	-	-	2	10
10	mfeat-fac	2000	216	-	-	10	216
11	mfeat-fou	2000	76	-	-	10	76
12	mfeat-kar	2000	64	-	-	10	64
13	mfeat-pix	2000	240	-	-	10	240
14	mfeat-zer	2000	47	-	-	10	47
15	mushroom	8124	-	6	16	2	117
16	nursery	12960	-	1	7	5	23
17	optdigits	5620	64	-	-	10	64
18	ozone	2536	72	-	-	2	72
19	page-blocks	5473	10	-	-	5	10
20	pendigits	10992	16	-	-	10	16
21	phoneme	5404	5	-	-	2	5
22	satimage	6435	36	-	-	6	36
23	segment	2310	19	-	-	7	19
24	shuttle	58000	9	-	-	7	9
25	sick	3772	7	20	2	2	33
26	soybean	683	-	16	19	19	82
27	texture	5500	40	-	-	11	40
28	waveform	5000	40	-	-	3	40
29	yeast	1484	8	-	-	10	8
30	zip	9298	256	-	-	10	256

that testing error is very poor. We are usually interested in the behavior of the method when the subset of selected features is achieving performance that is at least close to the use of all features. To account for that, we have chosen as a second numerical measure the percentage of features needed to achieve a testing accuracy of at least 90% of the accuracy of the whole set of features.

5.4.1 Feature Selection Algorithms

To obtain an accurate view of the usefulness of our method, we have tested it against two of the most successful state-of-the-art feature selection algorithms. We have chosen to test our model with two high-performance algorithms that are widely used in the feature selection community with very good results, proved in numerous algorithm comparisons: a ranking feature selection method (SVM-RFE) and a subset feature selection method (a genetic algorithm).

5.4.1.1 SVM-RFE The support vector machine recursive feature elimination approach is well-studied for use in gene expression studies (Guyon et al., 2002). This algorithm conducts feature selection in a sequential backward elimination manner, which starts with all the features and discards one feature at a time. Like SVM, SVM-RFE was initially proposed for binary problems. The squared coefficients: $w_j^2(j = 1, ...; p)$ of the weight vector w are employed as feature ranking criteria. Intuitively, those features with the largest weights are the most informative. One iteration of SVM-RFE trains the SVM classifier, computes the ranking criteria w_j^2 for all features, and discards the feature with the smallest ranking criterion. One can consider that the variable that is removed is the one with the least influence on the weight vector norm. Hence, this method is similar to those employed in neural networks in the sense that the ranking criterion is the sensitivity of w^2 with respect to a given variable.

The procedure provides as output a ranking of the features, and by setting an appropriate threshold we can select a small subset of features. Most studies have found RFE to select very good feature sets, but when the number of instances is large, scaling up is needed because SVM has scalability issues with large training sets. Therefore, the complexity of SVM-RFE relies on its core algorithm: SVM has very good scaling by features but has efficiency problems when the number of instances is large. Consequently, our method partitions by instances when we want to select features by means of the SVM-RFE algorithm.

5.4.1.2 Genetic algorithm We decided to implement a simple genetic algorithm in our democratic framework because various works (Yang and Honavar, 1998) have successfully compared genetic algorithms (GA) with other feature selection methods. Among them, it is worth noting the paper of Siedlecki and Sklansky (1989), which compared a genetic algorithms approach with a sequential search (forward and backward) and with a non-optimal variation of branch and bound (Foroutan-Sklansky BB search), which is able to work with a nonmonotonic criterion. On a synthetic 24-dimensional dataset as well as on a real 30-dimensional dataset, the GA outperformed these other feature selection methods (in terms of both classification performance and computational

Algorithm 15: SVM-RFE Algorithm

Data : A training set $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$.
Subset of surviving features $s = [1, 2, ...n]$.
Ranked list of features $r = []$

Result : Ranked list of features r

repeat

1 Restrict training examples to good feature indices: $X = X_0(:, s)$

2 Train the classifier: $\alpha = SVM - train(X, y)$

3 Compute the weight vector of dimension length(s): $w = \Sigma \alpha_k y_k x_k$

4 **for** *alli* **do**

5 Compute the ranking criteria: $c_i = (w_i)^2$

end

6 Find the feature with the smallest ranking criterion: $f = argmin(c)$

7 Update the feature ranked list: $r = [s(f), r]$

8 Eliminate the feature with the smallest ranking criterion: $s = s(1 : f-1, f+1 : length(s))$

until $s = [\,]$

9 **return** r

effort).

In our GA approach, a given feature subset is represented as a binary string (a *chromosome*) of length n, with a zero or one in position i denoting the absence or presence of feature i in the set, where n is the total number of available features. We apply standard genetic operators, such as two-point crossover and mutation. At the beginning of each generation, we perform an elitism step, and we evaluate each individual by means of the fitness function:

$$\begin{aligned} fitness(ind) &= suc\_rate(i) \cdot \alpha + (1 - \alpha) \\ &\cdot ((1 - n\_sel\_feat)/n\_feat), \end{aligned} \tag{10}$$

where *suc_rate* is the wrapper evaluation of the subspace, α is a value in the interval $[0, 1]$, which is set to 0.75, n_sel_feat is the number of selected features and n_feat is the number of all available features.

It is worth noting the scalability problems of a standard application of genetic algorithms when the number of variables is large. In these cases, the search space grows exponentially, so their efficiency and the quality of the results decrease. Nevertheless, these scalability problems can be overcome by partitioning the original dataset into smaller and independent subsets, as proposed in our method. The key is to combine the results of each independent and fast run of the genetic algorithm effectively to produce the final selection of features.

The complexity of our genetic algorithm, because it is a wrapper algorithm, depends on the complexity of the classification model used to evaluate a specific chromosome (subset of activated features) in each generation. As further described in Section 5.5, we tested our genetic algorithm with two different settings, using a C4.5 and a KNN classification algorithm. On the one hand, if the C4.5 algorithm is chosen to evaluate the quality of each chromosome, the complexity of the method mostly depends on the number of features that determines the size of the search space of the genetic algorithm due to the ability C4.5 to deal with datasets with many instances. Therefore, in this setting the set of features is partitioned into smaller subsets to increase its efficiency. On the other hand, if the KNN algorithm is used as evaluator, we use both partitioning by features and instances, as the genetic algorithm will have scalability problems if the number of instances or the number of features is large.

5.5 Experimental Results

In the genetic algorithm, the number of generations was set to 1000. We sped up the process by applying an exhaustive search of all possible combinations of features if the exhaustive search had lesser iterations than the number of generations set in the genetic algorithm. The number of individuals in the population was set to 100. At the beginning of each generation, we applied a 10% of elitism and then obtained the rest of the population by iteratively applying a two-point-crossover operator (in which we keep the two best individuals of each crossover step). The standard mutation percentage was fixed to 10%. These are fairly standard values (Cano et al., 2003).

The SVM-RFE algorithm used a Gaussian kernel, and the parameters of SVM (regularization, C, and kernel width, γ) were estimated using cross-validation during the iterations. The best combination of C and γ was chosen by testing all the possible combinations in the sets: $C \in \{0.1, 1.0, 10.0\}$ and $\gamma \in \{0.0001, 0.001, 0.01, 0.1, 1.0, 10.0\}$. For k-NN, k was also chosen by cross-validation in the interval $[1, 100]$. C4.5 was not sensitive to the parameters. We used pruned trees.

Our method has three parameters: the instance subset size or feature subset size M, the number of rounds r, and α, if we are using *pseudoensembles* to get a subset selection. We used subsets of 7 features and of 100 instances for SVM-RFE and 1000 instances for the genetic algorithm. These values are large enough to allow for a meaningful application of the feature selection algorithm on the subset and yet small enough to allow a fast execution. For both configurations, subset feature selectors and ranking feature selectors, we employed the C4.5, k-NN and SVM classifiers to evaluate the subsets of

features obtained by the methods. For the genetic algorithm, we evolved the population using as evaluators C4.5 and k-NN. SVM is too computationally expensive to be used in the genetic algorithm.

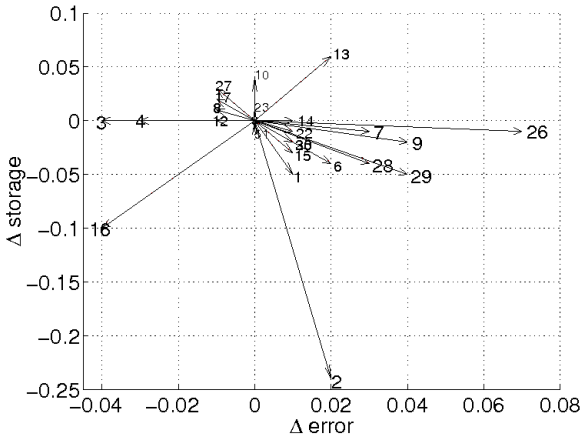
For the number of rounds, we chose a small value to have a fast execution, $r = 10$. Our experiments showed that, as it is the case for ensembles of classifiers (García-Pedrajas and Ortiz-Boyer, 2007), increasing the number of rounds increased the execution time but did not improve the performance. As explained in Section 5.3.3.2, α was set to 0.75 because a lesser error is more important than a smaller storage requirement when directly selecting a subset of features. We will denote as "pseudoE.x" the application of our method with a certain feature selection algorithm X.

5.5.1 Results of pseudoensembles using subset selection

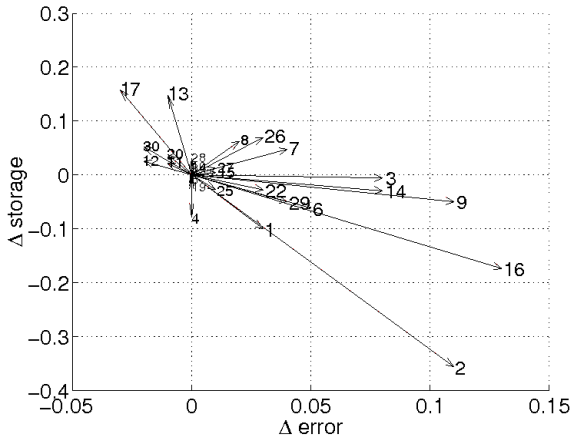
As previously stated in Section 5.4, for the genetic algorithm, we studied the testing error and the reduction ratio as well as the performance in terms of the running time. To show the testing error and reduction ratio, we use a graph based on the kappa-error relative movement diagrams (Rodríguez et al., 2010), but here, instead of the kappa difference value, we will use the storage difference (García-Osorio et al., 2010). These diagrams represent by an arrow the results of two methods applied to the same dataset. The arrow starts at the origin, and the coordinates of the tip of the arrow are given by the difference between the error and storage of our method and the standard feature selection algorithm. The numbers indicate the dataset being represented according to Table 21. These graphics are a very convenient way of summarizing the results. For example, arrows pointing down-left represent datasets for which our method outperforms the standard algorithm in both error and storage, arrows pointing up-left indicate that our algorithm improves the storage but with a worse testing error, and so on. Figure 34 shows the results for testing error and storage requirements in the configurations that use the genetic algorithm as their base algorithm. For GA, we applied our method with 7 features for C4.5 and k-NN as evaluators and with 1000 instances for k-NN. C4.5 is able to scale up fairly well with many instances, so there partitioning by instances is not advisable.

In these storage-error delta diagrams, we can see that our method reduced the storage as much as the standard counterparts of the feature selection algorithms used and maintained a testing error that was not significantly worse, while reducing the time dramatically, as will be shown in Figure 35.

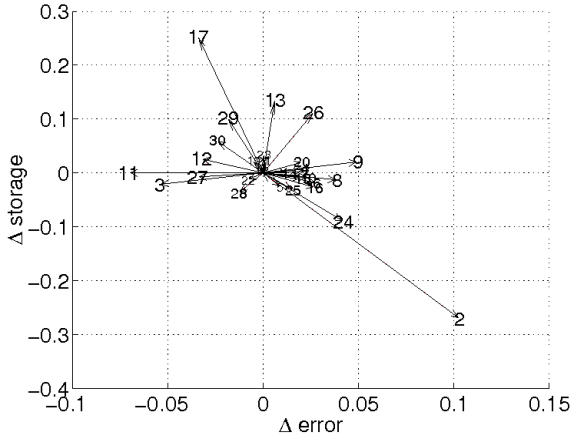
We use the Wilcoxon test to evaluate the statistical relevance of the differences between the standard genetic algorithm and its use in our approach, pseudoE.GA. The results are shown in Table 22. Regarding the testing error,



a)



b)



c)

Fig. 34. Storage requirements/testing error using pseudoE.GA a) pseudoE.GA-KNN-1000inst, b)pseudoE.GA-KNN-7feats, c)pseudoE.GA-C4.5-7feats

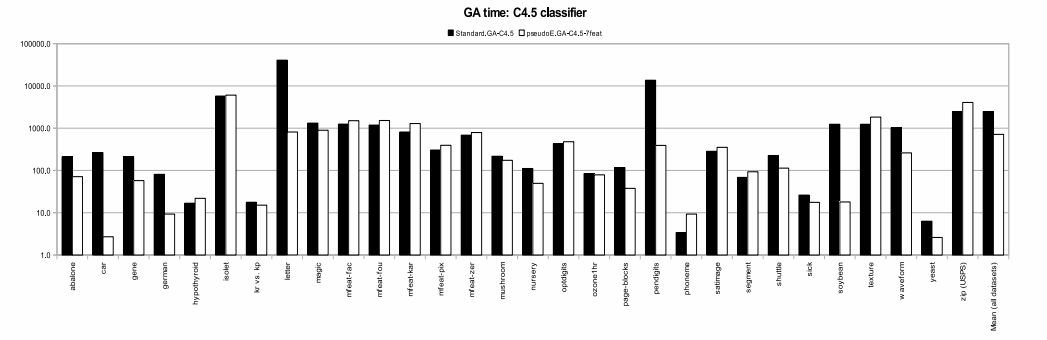
the results are comparable to the ones obtained by the standard version, with the exception of KNN using 7 features, in which our results are significantly worse, whereas the reduction is proved to be similar and even significantly better when partitioning by instances. Furthermore, the tree size values are clearly better using *pseudoensembles*, an average size of 389.8 nodes against 426.1 nodes of the standard configuration, which is better in 27 out of the 30 cases. This improvement had a p-value of 0.0052 in the sign test and of 0.0082 in the Wilcoxon test. Consequently, we can state that fewer and better variables were chosen by our approach.

Table 22
Summary of the performance of the *pseudoensembles* framework in the two possible configurations. The table shows the win/draw/loss record of each algorithm against its standard version. The row labeled p_s is the p-value of a two-tailed sign test on the win/loss record, and the row labeled p_w shows the p-value of the Wilcoxon test. Significant differences at a confidence level of 95% are indicated with a ✓.

	pseudoE.GA		
	Test error		
	C4.5-7f	kNN-1000inst	kNN-7f
Mean all	0.1890 (Std: 0.1811)	0.1462 (Std: 0.1398)	0.1631 (Std: 0.1398)
win/loss	10/20	9/20	7/21
p_s	0.0987	0.0614	0.0125 ✓
p_w	0.1156	0.0612	0.0058 ✓
	pseudoE.GA		
	Reduction		
	C4.5-7f	kNN-1000inst	kNN-7f
Mean all	0.1890 (Std: 0.1966)	0.1763 (Std: 0.2000)	0.1903 (Std: 0.2000)
win/loss	13/17	20/7	13/15
p_s	0.5847	0.0192 ✓	0.8506
p_w	0.5716	0.0093 ✓	0.9672

	pseudoE.SVM-RFE (data over all datasets)		
	AUC		
	C4.5-100inst	kNN-100inst	SVM-100inst
Mean all	0.6313 (Std:0.6091)	0.6167 (Std:0.5831)	0.5720 (Std:0.5474)
win/loss	11/5	13/3	12/4
p_s	0.2101	0.0213 ✓	0.0768
p_w	0.0703	0.0114 ✓	0.0359 ✓
	pseudoE.SVM-RFE (data over all datasets)		
	%Feats for 90% error		
	C4.5-100inst	kNN-100inst	SVM-100inst
Mean all	0.1733 (Std:0.1903)	0.1479 (Std:0.1780)	0.2130 (Std:0.2299)
win/loss	6/2	9/5	5/4
p_s	0.2891	0.4240	1.0000
p_w	0.2623	0.2172	0.6733

pseudoE.GA-c4.5:



pseudoE.GA-KNN:

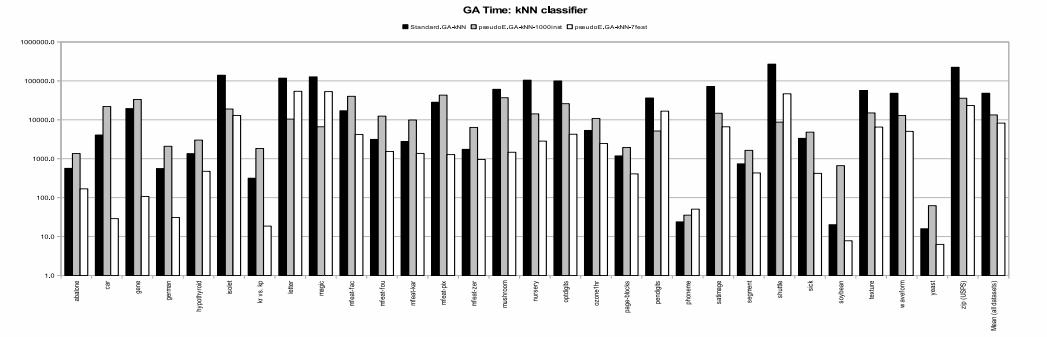


Fig. 35. Comparative study of time between Standard.GA and PseudoE.GA (measured in seconds and using a logarithmic scale)

In terms of execution time, the advantage of *pseudoensembles* over the standard method is significant. The results are plotted in Figure 35 using a logarithmic scale. When the evaluator is the C4.5 algorithm, the time was reduced for most of the datasets. In a few datasets, *pseudoensembles* took slightly more time than their standard counterparts. Examples of these cases are the magic, mfeat-kar and texture datasets. However, the differences in such cases are small, in contrast to the large time advantage obtained using *pseudoensembles* in the majority of datasets. It is worth pointing out that, for datasets like the car dataset, the *pseudoensembles* approach took only 1% of the time required by the standard genetic algorithm. Other representative examples are the letter and pendigits datasets (reducing their time to 2% and 3% of the standard counterparts, respectively).

In the case when the KNN algorithm is chosen as the evaluator, all datasets benefited from *pseudoensembles* (both partitioning by instances and by features), except a few datasets for which the number of instances or features of the dataset is similar to the partition size and that resulted in only two partitions. Thus, the benefits of *pseudoensembles* were masked in these cases. This is the case of the phoneme, car and gene datasets. In the rest of the cases, the advantage of *pseudoensembles* is remarkable, as we can observe in

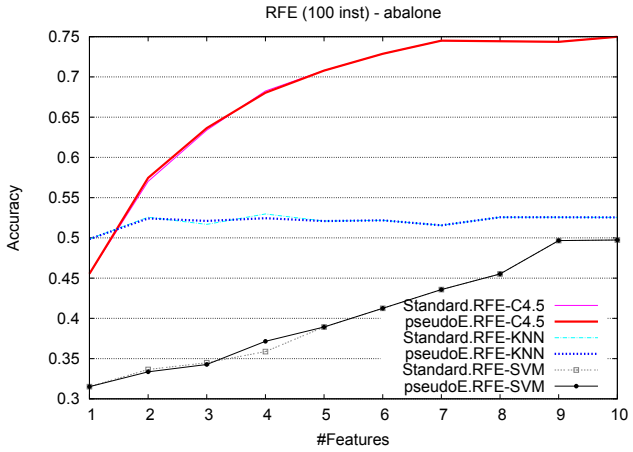
the average results plotted in the last three bars of the graph or in datasets such as isolet, nursery and zip.

5.5.2 Pseudoensembles using ranking of features

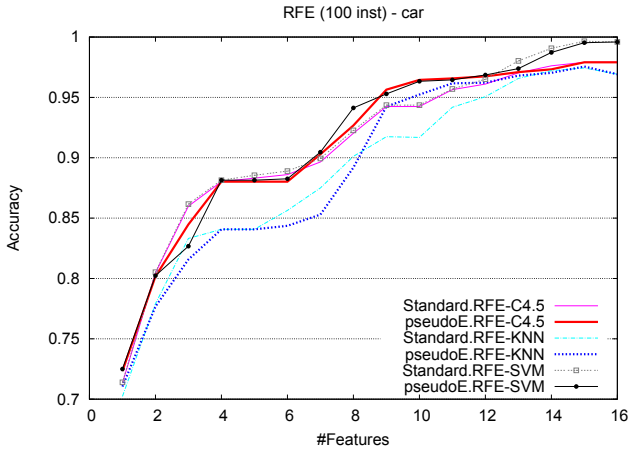
In methods that rank features, it is interesting to study the evolution of the testing accuracy as the number of retained features increases. In Figures 36 to 45, we plot the testing accuracy of each method for a dataset against the number of relevant features in descending order of quality (the first point of all curves is the accuracy corresponding to the method selecting the best feature, the second point is that corresponding to the method selecting the best feature and the second best feature, and so on).

The overall behavior that can be observed in Figures 36 to 45 is that all the methods reached the highest accuracies for all the datasets with a fairly small subset of all the features. The issue is which of them got the highest accuracy with the least number of relevant features. We have plotted the results of six representative datasets that constitute a diverse sample of the thirty datasets studied. In the mushroom dataset, both of the configurations, standard and *pseudoensembles*, got very good accuracy values of above 94%. However, slightly better results were achieved by *pseudoensembles* specially selecting a small number of relevant features. The segment dataset showed a clear advantage for all the configurations of *pseudoensembles*, succeeding in getting better accuracies with less features as they start the curve higher than the standard counterparts. Similar behavior can be observed in the sick dataset, where the difference between the starting points of the curves using KNN is clear. Whereas the soybean graph plots very similar starting accuracies values for all configurations, the *pseudoensembles* increased their accuracy faster than did standard ones around 10 features, reaching competitive accuracies over 90% with fewer features. The *pseudoensembles* configurations using the shuttle dataset also started from a very good accuracy, higher than 0.92. It is worth noting the remarkable difference of 20% between our approach and the respective standard methods using KNN as the evaluator. For the texture dataset, all approaches had quite low starting values of accuracy. However, *pseudoensembles* began to increase their accuracy right away, reaching a 90% level of accuracy when the standard counterparts still had low accuracy values.

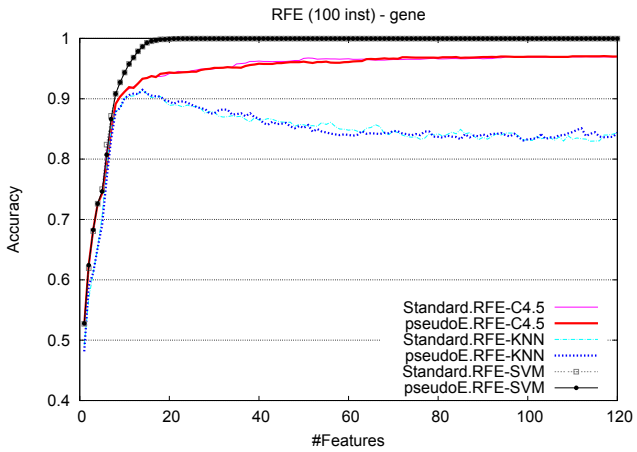
As previously described in Section 5.4, we have chosen as comparison measures between feature ranking methods the area under the accuracy curve (AUC) and the percentage of relevant features needed to achieve a competitive accuracy (in our case, 90% of the best one obtained). The results obtained for these measures are plotted in Figure 46. In the AUC graph, the *pseudoensembles* area values are subtracted from the standard ones, and the goal is to maximize the area under the curve, meaning higher values of accuracy with



abalone

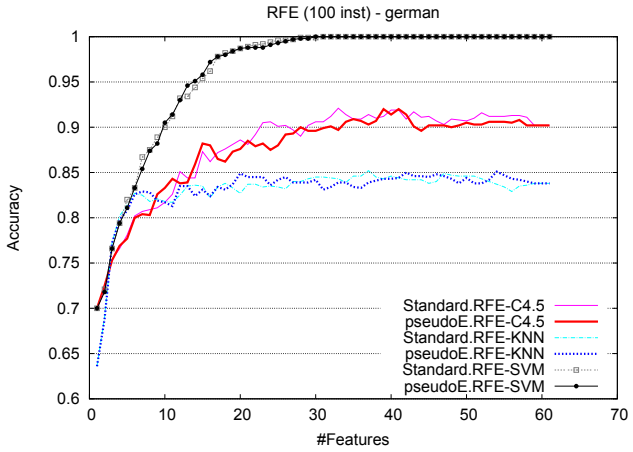


car

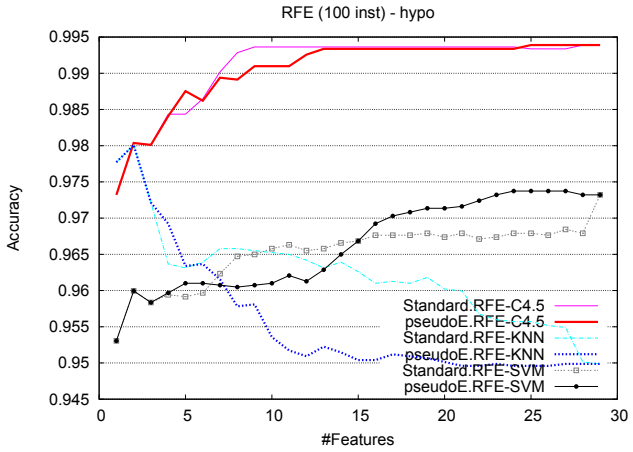


gene

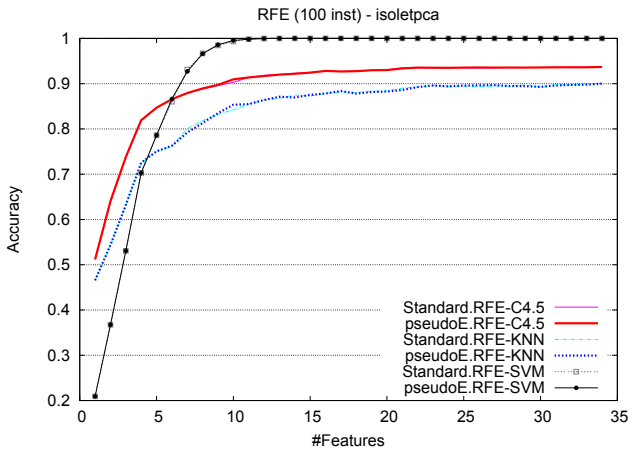
Fig. 36. Evolution of the classification accuracy selecting a different number of features using SVM-RFE: abalone, car and gene datasets.



german

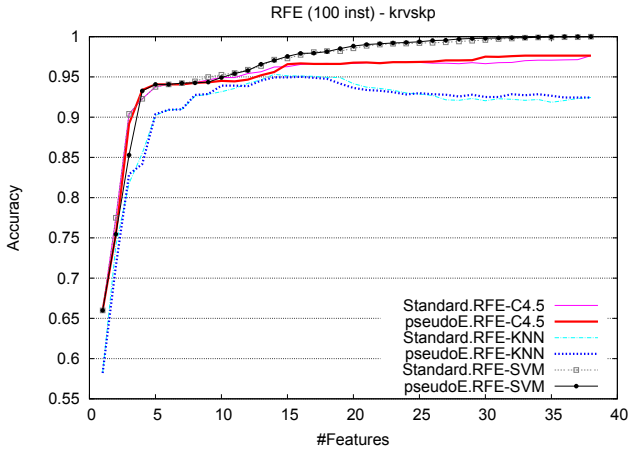


hypo

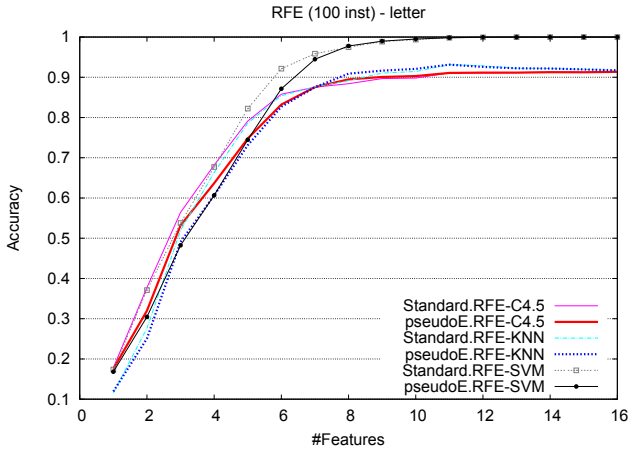


isoletpca

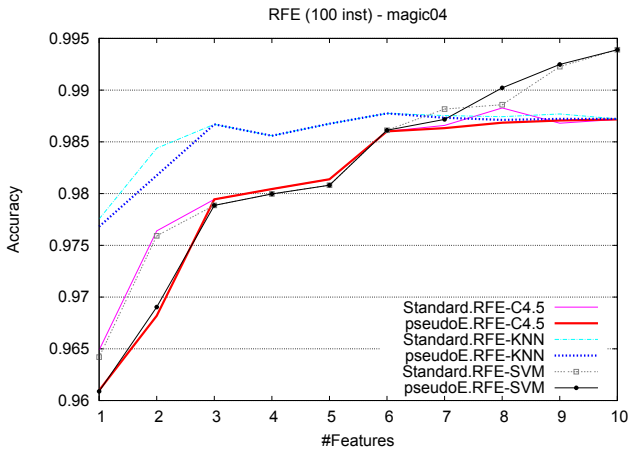
Fig. 37. Evolution of the classification accuracy selecting a different number of features using SVM-RFE: german, hypo and isoletpca datasets.



krvskp

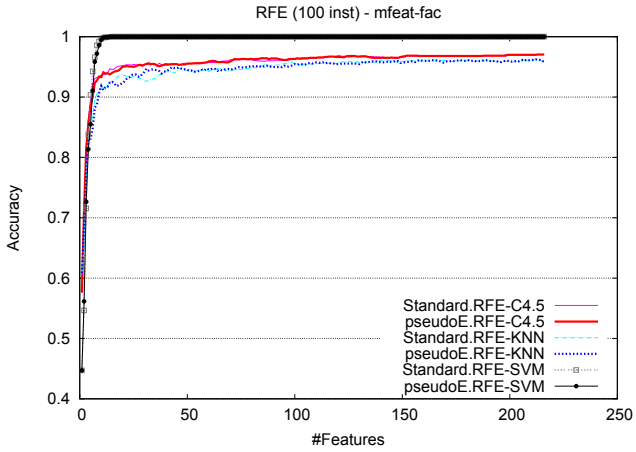


letter

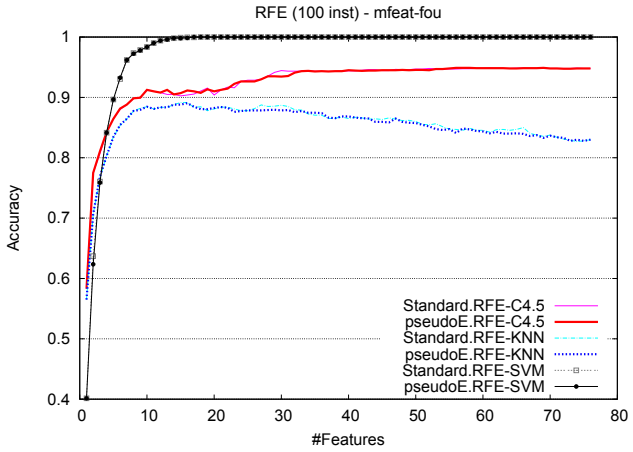


magic04

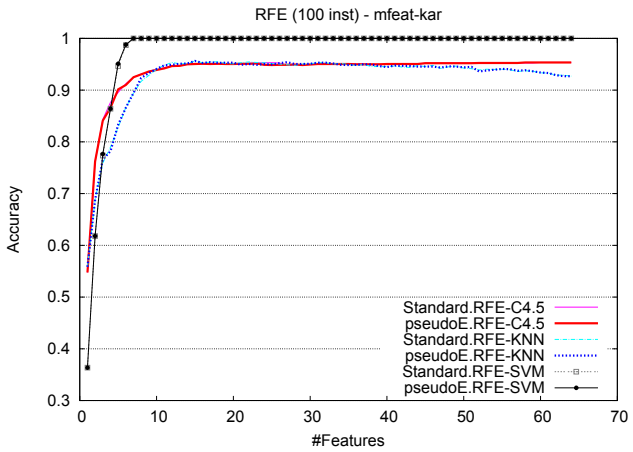
Fig. 38. Evolution of the classification accuracy selecting a different number of features using SVM-RFE: krvskp, letter and magic04 datasets.



mfeat-fac

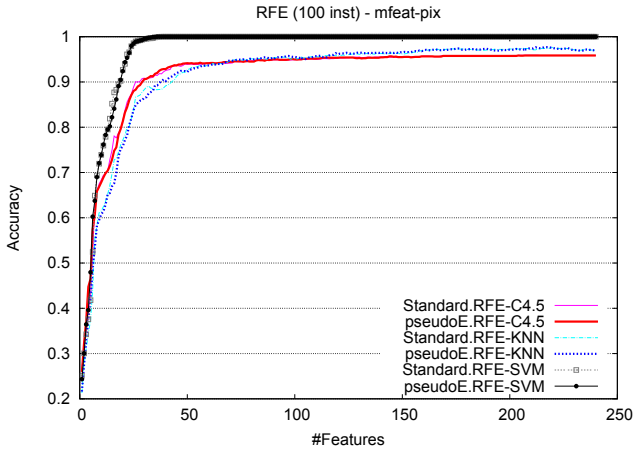


mfeat-fou

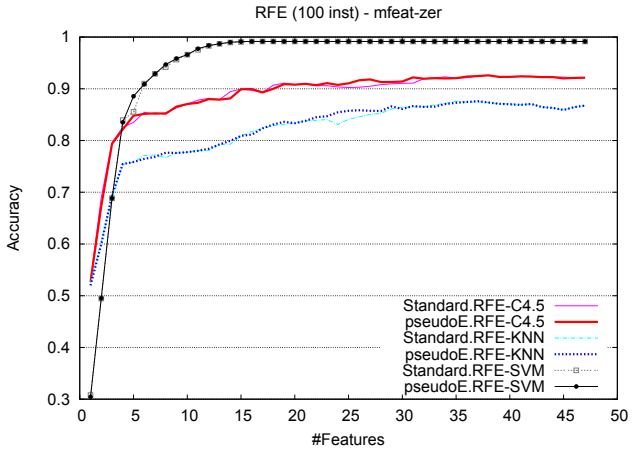


mfeat-kar

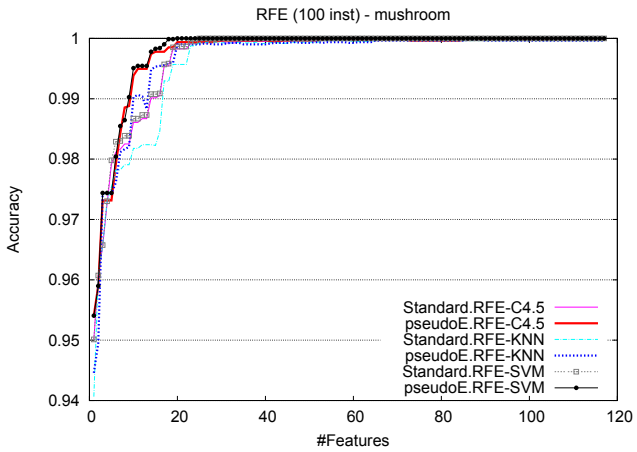
Fig. 39. Evolution of the classification accuracy selecting a different number of features using SVM-RFE: mfeat-fac, mfeat-fou and mfeat-kar datasets.



mfeat-pix

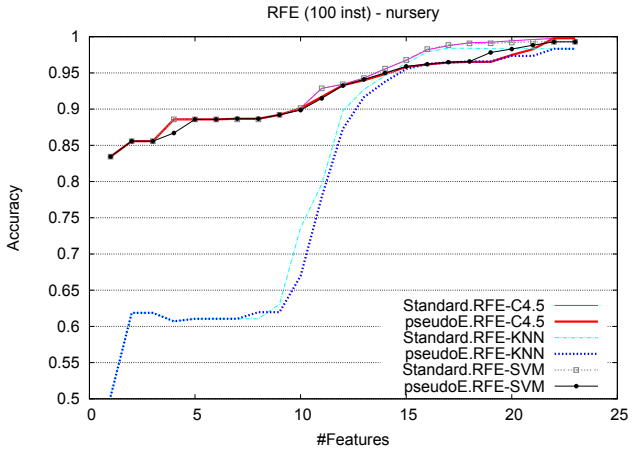


mfeat-zer

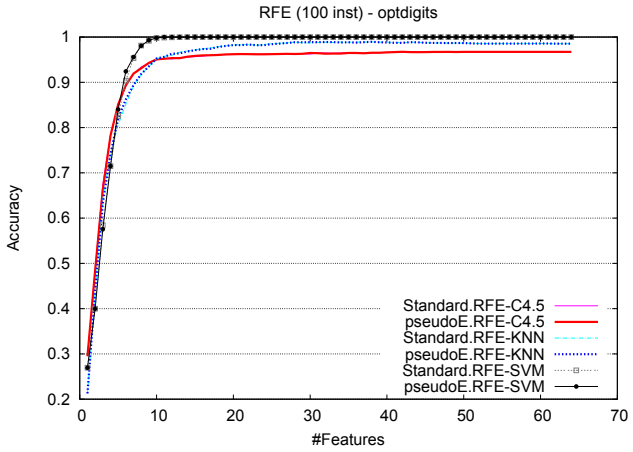


mushroom

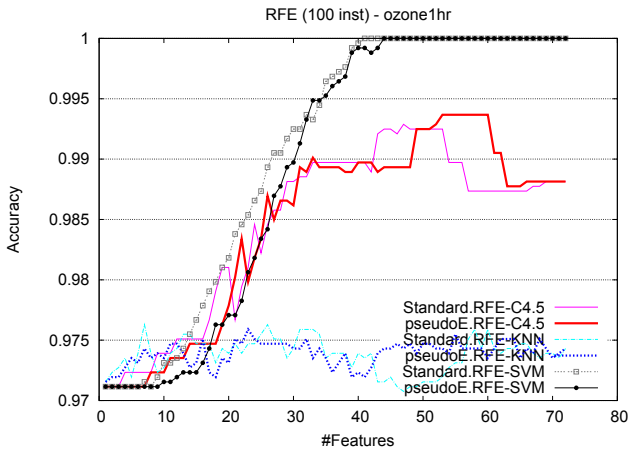
Fig. 40. Evolution of the classification accuracy selecting a different number of features using SVM-RFE: mfeat-pix, mfeat-zer and mushroom datasets.



nursery

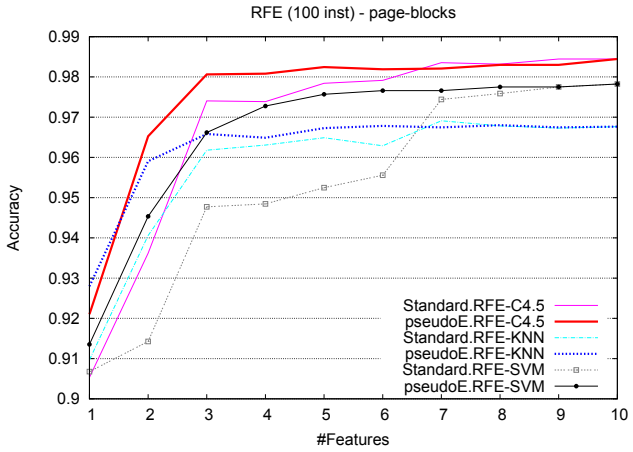


optdigits

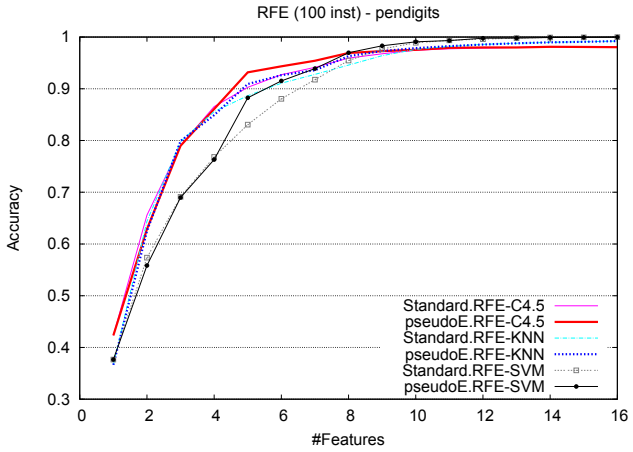


ozone1hr

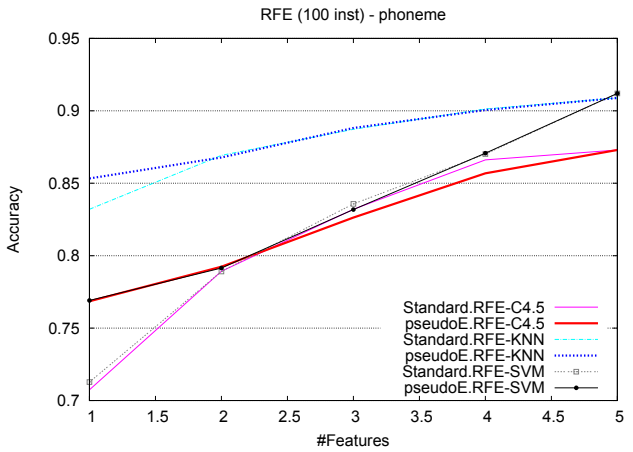
Fig. 41. Evolution of the classification accuracy selecting a different number of features using SVM-RFE: nursery, optdigits and ozone1hr datasets.



page-blocks

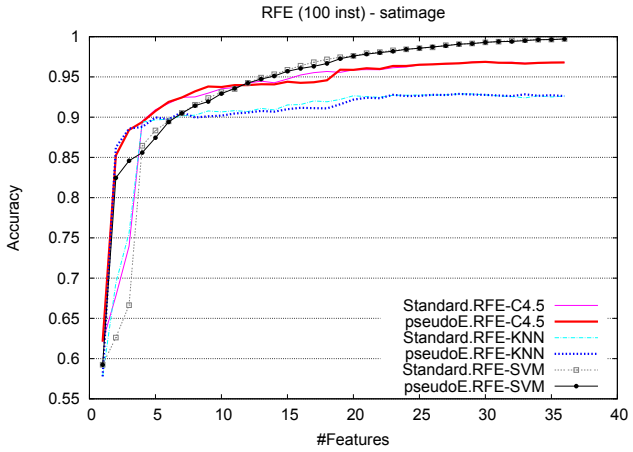


pendigits

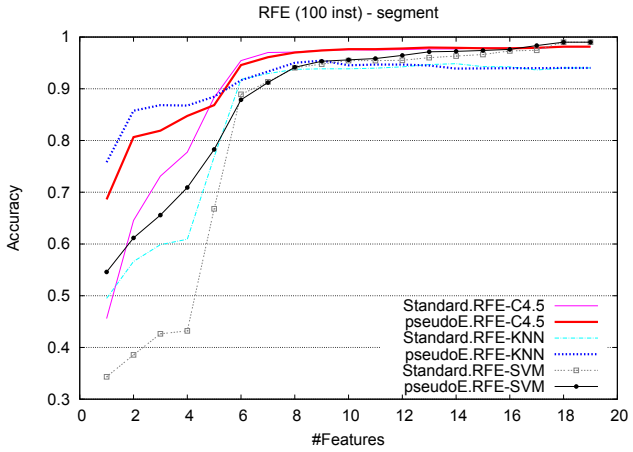


phoneme

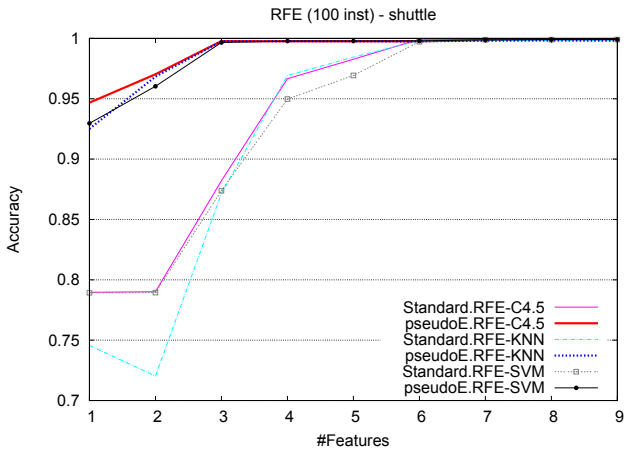
Fig. 42. Evolution of the classification accuracy selecting a different number of features using SVM-RFE: page-blocks, pendigits and phoneme datasets.



satimage

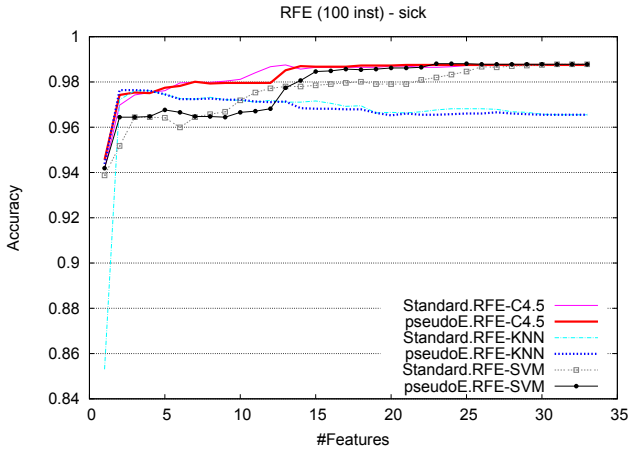


segment

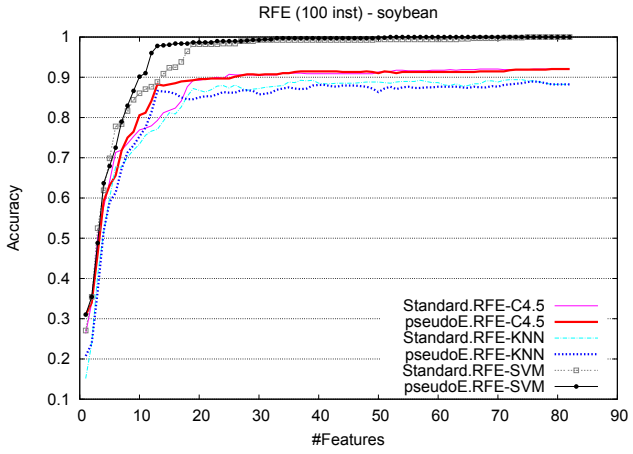


shuttle

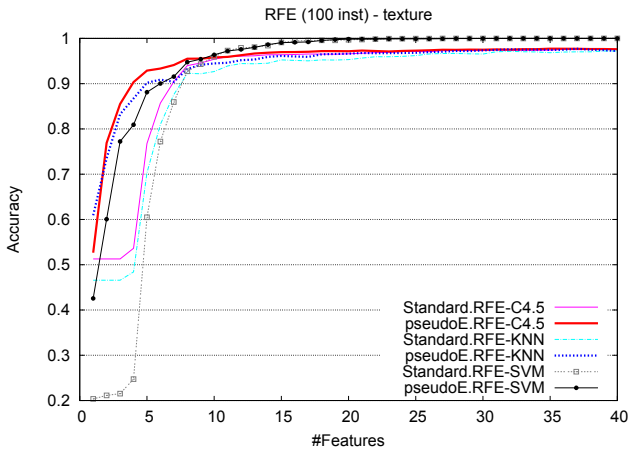
Fig. 43. Evolution of the classification accuracy selecting a different number of features using SVM-RFE: satimage, segment and shuttle datasets.



sick

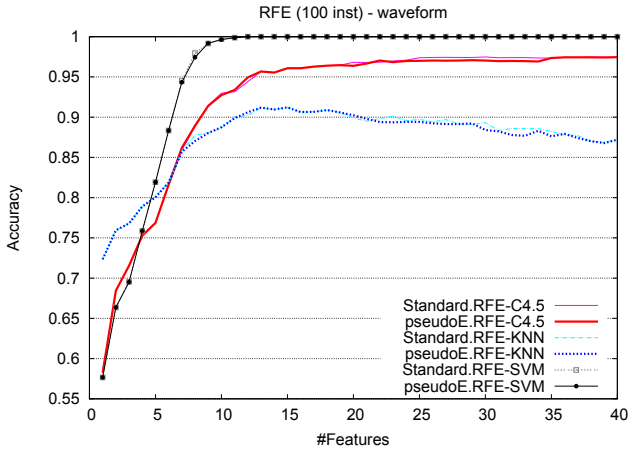


soybean

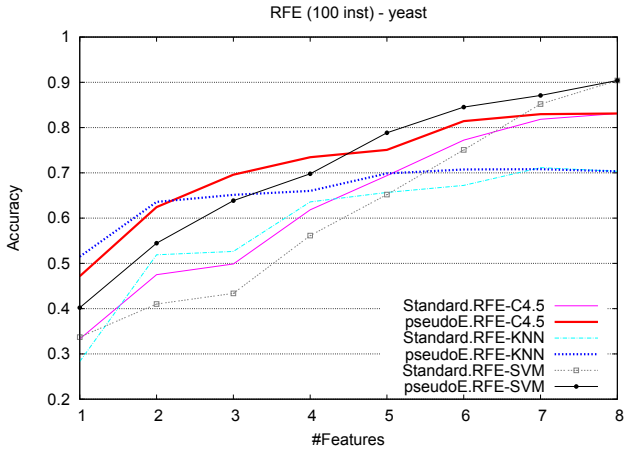


texture

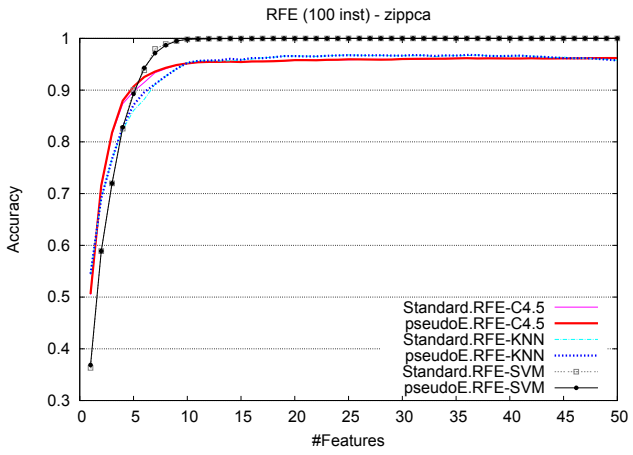
Fig. 44. Evolution of the classification accuracy selecting a different number of features using SVM-RFE: sick, soybean, texture datasets.



waveform



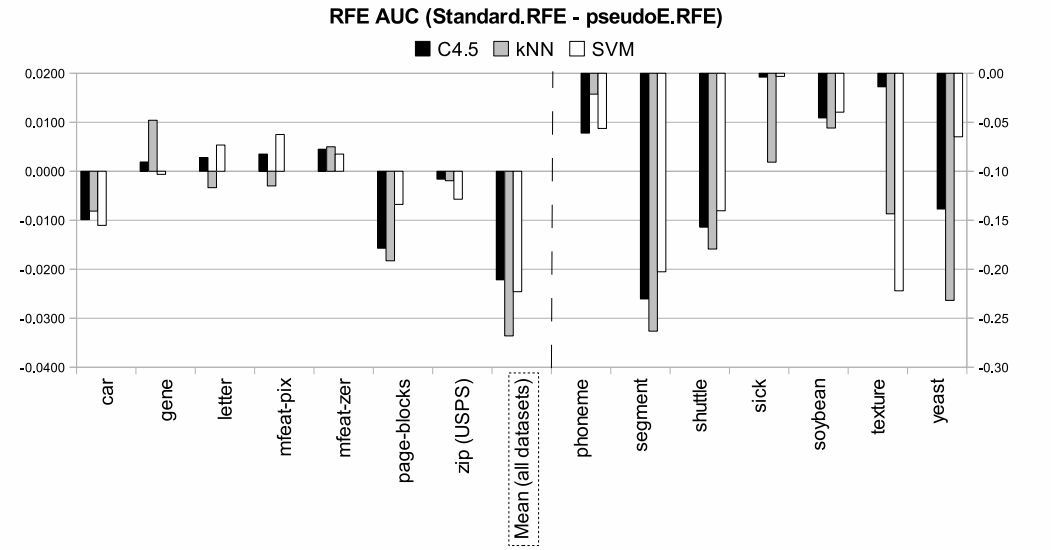
yeast



zippca

Fig. 45. Evolution of the classification accuracy selecting a different number of features using SVM-RFE: waveform, yeast and zippca datasets.

Difference of the AUC



Difference of percentages of retained features to get a 90 %

of the best accuracy

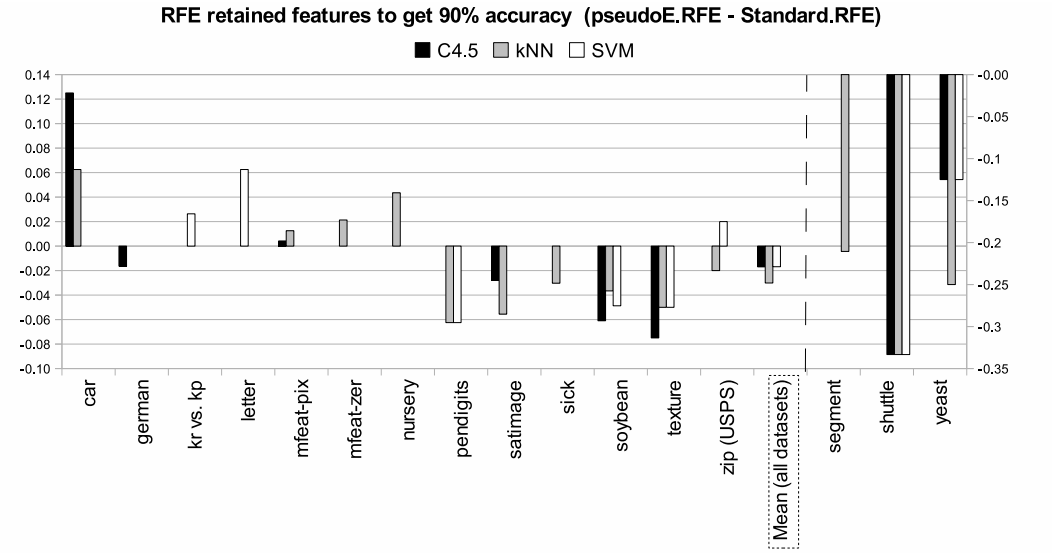


Fig. 46. Comparison of the AUC and the percentage of retained features to get a 90% of the best accuracy with RFE configurations.

less features on average, so negative values indicate better performance of our approach. The good average values are particularly remarkable, as well as the advantage in datasets such as segment, shuttle, texture or yeast. In the percentage of needed features graph, the standard values are subtracted from the *pseudoensembles* values, as the objective is to minimize the number of selected features, and we wanted to keep negative values as the desired ones. Again, the segment, texture and yeast stand out positively along with the average values.

Datasets omitted in these two graphics are those that have a very similar AUC or percentage values for both *pseudoensembles* and standard configurations (thus, the difference between them is almost zero). In summary, we can refer to the Wilcoxon values in Table 22 to state that our approach provided better or at least equally good results for both measures. The results are especially positive for AUC using a partition by instances and k-NN as the evaluator. For that configuration, the sign test and the Wilcoxon test prove that it is better than the standard approach.

Regarding the execution time, Figure 47 plots the difference of the standard configuration values from their *pseudoensembles* counterparts. Bars under the x-axis imply an advantage of *pseudoensembles* over the standard methods. We show datasets that take more than 400 seconds to extract relevant conclusions from the analysis. For datasets with a significant number of features, our approach takes more time (mfeat-fac and mfeat-zer) due to the good scaling properties of SVM-RFE in the number of variables and the fact that these datasets have a manageable number of instances (2000 instances) and do not give our method the chance to benefit from the partition by instances. However, as the number of instances increases, *pseudoensembles* provided more-competitive results, as in the case of shuttle, zip or mushroom datasets (58000, 9300 and 8100 instances, respectively). The gain in execution time will be more clearly visible for huge datasets in Section 5.5.3.

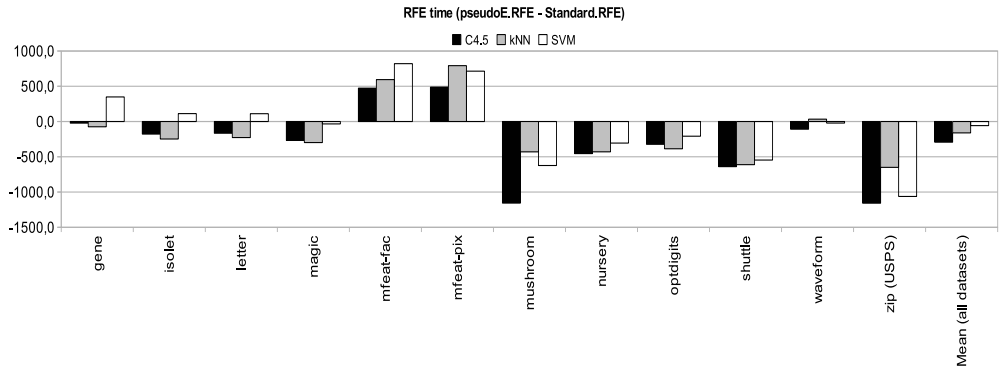


Fig. 47. Running time comparison between the standard RFE algorithm and our approach (in datasets over 400 secs of execution time).

5.5.3 Results for huge datasets

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, from a few hundreds of thousands of instances to almost five millions, with several thousands of features. Table 23 describes the problems that are used, grouping them into datasets with large numbers of instances and datasets with large numbers of features. These datasets will

show whether our methodology allows us to scale up standard algorithms to huge problems. The last column in the table is the 1-NN error using all the features, as a general measure of the complexity of the problem. Due to the huge size of the datasets, we have used C4.5 as the only evaluator. Tables 24 presents the results obtained for these huge datasets using a genetic algorithm, and Table 25 shows the results for SVM-RFE method.

Table 23
Summary of the huge datasets used in our experiments.

Datasets with a large number of instances								
	Data set	Cases	Features			Classes	Inputs	1-NN error
			C	B	N			
1	census	299285	7	-	30	2	409	0.0743
2	covtype	581012	54	-	-	7	54	0.3024
3	kddcup99	494021	33	4	3	23	118	0.0006
4	kddcup99all	4898431	33	4	3	23	122	0.0002
5	kddcup991M	1000000	33	4	3	21	119	0.0002
6	poker	1025010	5	-	5	10	25	0.4975

Datasets with a large number of features								
	Data set	Cases	Features			Classes	Inputs	1-NN error
			C	B	N			
1	arabidopsis	33971	-	-	1003	2	4012	0.3997
2	ccds	36497	-	-	1003	2	4012	0.0742
3	ustilago	55279	-	-	1003	2	4012	0.0559

For the genetic algorithm, we tested the partition by features and by instances. A first result, shown in Table 24, is that the standard method is not applicable to huge problems. In four of the six datasets with many instances, the standard method was not able to finish after 300 hours running. Therefore, our approach allows the application of feature selection methods to those huge datasets. The genetic algorithm configurations are tested on datasets with large numbers of instances and also on datasets with large numbers of features. For the two datasets with many instances for which the standard approach terminated, our approach reduced their execution time by 33%. kddcup991M has 1,000,000 instances and took 478,350 seconds to finish in the standard configuration and less than 160,000 seconds with *pseudoensembles*. It is remarkable that kddcupp99all, which has almost five million instances, took 984,750 seconds, whereas the standard approach did not finish. The comparison with 1-NN error using all features is favorable. *Pseudoensembles* achieved very significant reductions, while keeping the testing error very similar to that obtained using all features. This error is even improved for the census and poker datasets. We would like to draw attention to the experimental validation of the linear complexity of our approach, which can be examined in the execution time of the kddcup99, kddcup991M and kddcup99all datasets. The kddcup99 dataset is approximately half the size of kddcup991M, and the size

Table 24
Reduction, testing error and execution time using huge datasets in standard algorithms and *pseudoensembles*.

Datasets with a large number of instances							
		Standard.GA-C4.5			pseudoE.GA-C4.5-5feat		
	Data set	Reduction	Testing error	Time	Reduction	Testing error	Time (s)
1	census	-	-	>300 h	0.0154	0.0488	451845.2
2	covtype	-	-	>300 h	0.2852	0.3352	288311.5
3	kddcup99	0.0449	0.0040	269649.5	0.0534	0.0048	86708.3
4	kddcup99all	-	-	>300 h	0.0295	0.0064	984750.4
5	kddcup991M	0.0546	0.0036	478350.5	0.0487	0.0037	159439.3
6	poker	-	-	>300 h	0.5120	0.2652	32039.9
	Average	-	-	-	0.1574	0.1107	333849.1

Datasets with a large number of features							
		Standard.GA-C4.5			pseudoE.GA-C4.5-7feat		
	Data set	Reduction	Testing error	Time	Reduction	Testing error	Time (s)
1	arabidopsis	0.0069	0.0092	192497.7	0.0003	0.0080	73735.2
2	ccds	0.0071	0.0667	242004.3	0.0006	0.0381	84379.1
3	ustilago	0.0069	0.0169	502749.9	0.0004	0.0106	158404.4
	Average	0.0070	0.0275	238030.9	0.0006	0.0168	80926.4

Table 25
Reduction, testing error and execution time using huge datasets in standard algorithms and *pseudoensembles*.

Datasets with a large number of instances			
	Data set	% vars to get a 90% of acc	Time (s)
1	census	0.0024	212945.2
2	covtype	0.2407	22782.4
3	poker	0.3200	16367.1
4	kddcup99	0.0339	9767.3
5	kddcup99all	0.0328	76986.5
6	kddcup991M	0.0252	16134.9
	Average	0.1092	59163.9

of kddcup99all is approximately five times the size of the kddcup991M dataset. Their execution times grow in proportion to their size (87,000, 160,000 and 985,000 seconds, respectively). For datasets with large numbers of features, we notice an impressive improvement in the storage requirements and the testing error with *pseudoensembles*, as well as a reduction of up to 34% in the execution time taken by the standard approach.

The complexity of SVM-RFE relies on the number of instances (see Section 5.4.1), so it is more interesting to analyze its behavior in dealing with datasets with large numbers of instances, as it is able to manage datasets with many features reasonably well. Table 25 shows the results using *pseudoensembles*

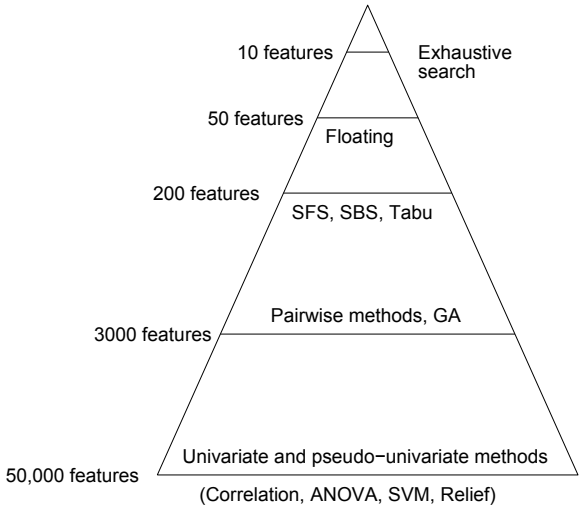


Fig. 48. The "feasibility pyramid" for feature selection.

for SVM-RFE and problems with many features. There are no results for the standard SVM-RFE configuration with these datasets because they surpassed the limit of 300 hours of execution time that we set. The *pseudoensembles* provide competitive values for both of them, with an average of 11% of the features required and less than 60,000 seconds on average to end the feature selection process. These results are particularly impressive if we take into account that, after 300 hours, which is more than one million seconds, the standard method was not able to achieve a result.

5.6 Extension of Pseudoensembles to medical problems. The feasibility pyramid

As previously explained, our scaling methodology has a competitive performance with a low complexity even in huge datasets(de Haro-García and Pedrajas, 2010). Due to this fact, our method can be used to model jump from 3000 features to 200 features, where other algorithms will be overloaded for computational reasons. In de Haro-García et al. (2011) we propose a conceptual structure for the feature selection problem called "feasibility pyramid", that suggests the use of simple and feasible feature selection methods, like our described scaling approach in (de Haro-García and Pedrajas, 2010), to "jump" from larger to smaller feature subsets (up the levels of the pyramid). It is worth highlighting that no other feature selection method offering a demanding performance like in our scaling methodology would be applicable in such a low level of the pyramid because of its complexity. However, the objective in this case is to pre-process an original huge dataset instead of producing a final feature selection, so as to reduce the feature set to a smaller one in which we can apply a more complex and accurate feature selection algorithm.

Note that ranking the features and cutting off the tail is suitable for a jump at any level of the pyramid. Ranking of the features can be done according to different criteria. Arguably, the pseudo-multivariate methods should be preferred to the true univariate methods. These methods are not without flaws, however. The most popular choice, SVM, is known to penalize correlated features by decreasing their weights, even though the features may be highly relevant. This has been noted as a potential problem in fMRI feature selection (Pereira et al., 2009).

We recommend applying the same methodology as in the *pseudoensembles* approach (de Haro-García and Pedrajas, 2010), specifically the one whose output is a rank of features sorted by their relevance. Although this methodology is based on one of the specific types of *pseudoensembles*, the objective of its application is completely different. In this case we do not apply *pseudoensembles* to produce the final feature selection but to pre-reduce an original huge dataset of features to a smaller one in which we can apply a more accurate and complex feature selection method.

Therefore we intend to reduce a subset of few thousand features to a subset of few hundred by means of randomly splitting the original feature set into disjoint subsets of equal cardinality. Each subset is evaluated and the accuracy is taken to measure the quality of the features within that set. The features are ranked based on their individual accuracies, averaged over the T estimates.

The proposed random splitting is formally described in Figure 16. Note that the evaluation criterion is not specified within; this could be a filter or a wrapper criterion. We have performed an experimental study using one of the most popular state-of-the-art wrapper algorithms: SVM-RFE (Guyon et al., 2002)

We examined the behaviour of the random splitting as a step of the feature selection cascade and we compared it with two alternative methods for "pre-selection": ANOVA and SVM, both widely used in the Functional Magnetic Resonance Imaging (fMRI) field due to their ability to produce fast, accurate and reliable ranking of the features. In addition we tested a random permutation as a chance ranking method.

Since we are modelling the jump from the second to the third level up the feasibility pyramid (Figure 48), to jump from 3000 features to 200 features, we selected seven datasets with 3000 features and less than 300 instances to match the scale of a typical fMRI dataset, which is characterized by excessive

[9] *Note: If there is an incomplete set with cardinality less than M, either ignore, and use $\lfloor n/M \rfloor$ sets, of complete with random features and use $\lceil n/M \rceil$ sets. In both cases, care should be taken when averaging the feature scores in F. The score $F(x)$ should be divided by the number of updates of $F(x)$.

Algorithm 16: Random splitting for feature ranking

Data : A labelled dataset, $\mathbf{Z}$, with features $X = \{x_1, \ldots, x_n\}$
An evaluation criterion $J(S)$ for subset $S \subseteq X$, including the estimation protocol (cross-validation, hold-out, etc.)
The number of features in the subsets, M
The number of iterations, T

Result : The mean feature scores $\bar{F} = F/T$.

1 Initialize the feature score array with n zeros: $F(x) = 0$, $x \in X$
for $i = 1$ ***to*** T **do**
2 Split X randomly into disjoint subsets: $S_1, \ldots, S_{n/M}$ of size M.[9]
for $j = 1$ ***to*** n/M **do**
3 Evaluate $J(S_j)$
4 For all features $x \in S_j$, update the scores as: $F(x) = F(x) + J(S_j)$
end
end
5 Obtain threshold of votes, v, to remove a feature
6 $S = T$
7 Remove from S all features with a number of votes above or equal to v
8 **return** $\bar{F} = F/T$.

number of features and relatively small number of instances. We chose data with a small number of fairly balanced classes, numerical features (real-valued or integer), and no missing values. The chosen datasets are: a toy experiment named "'catface data"', two datasets that belong to the collection used for the NIPS 2003 feature selection challenge (Guyon et al., 2004), the "'leukemia dataset"' which is one of the most famous gene expression cancer datasets (Golub et al., 1999), containing information on gene-expression in samples from human acute myeloid (AML) and acute lymphoblastic leukemias (ALL) and three datasets from the fMRI field ("'haxby"' (Haxby et al., 2001) and two datasets (Kuncheva et al., 2010) collected at the School of Psychology, Bangor University, UK).

We have collected some of the most representative results of the experimental study developed in de Haro-García et al. (2011).

Figure 49 displays the classification accuracy as a function of the number of retained features. Each dataset is presented on a separate plot. The curves show the nearest mean accuracies averaged over the 100 testing sets for the four competing ranking methods.

Since all methods were run on identical 90/10 splits of the data, paired t-test was carried out to check for a significant difference in the accuracies of pairs of methods. Table 26 gives a summarized view of the statistical significance of the differences.

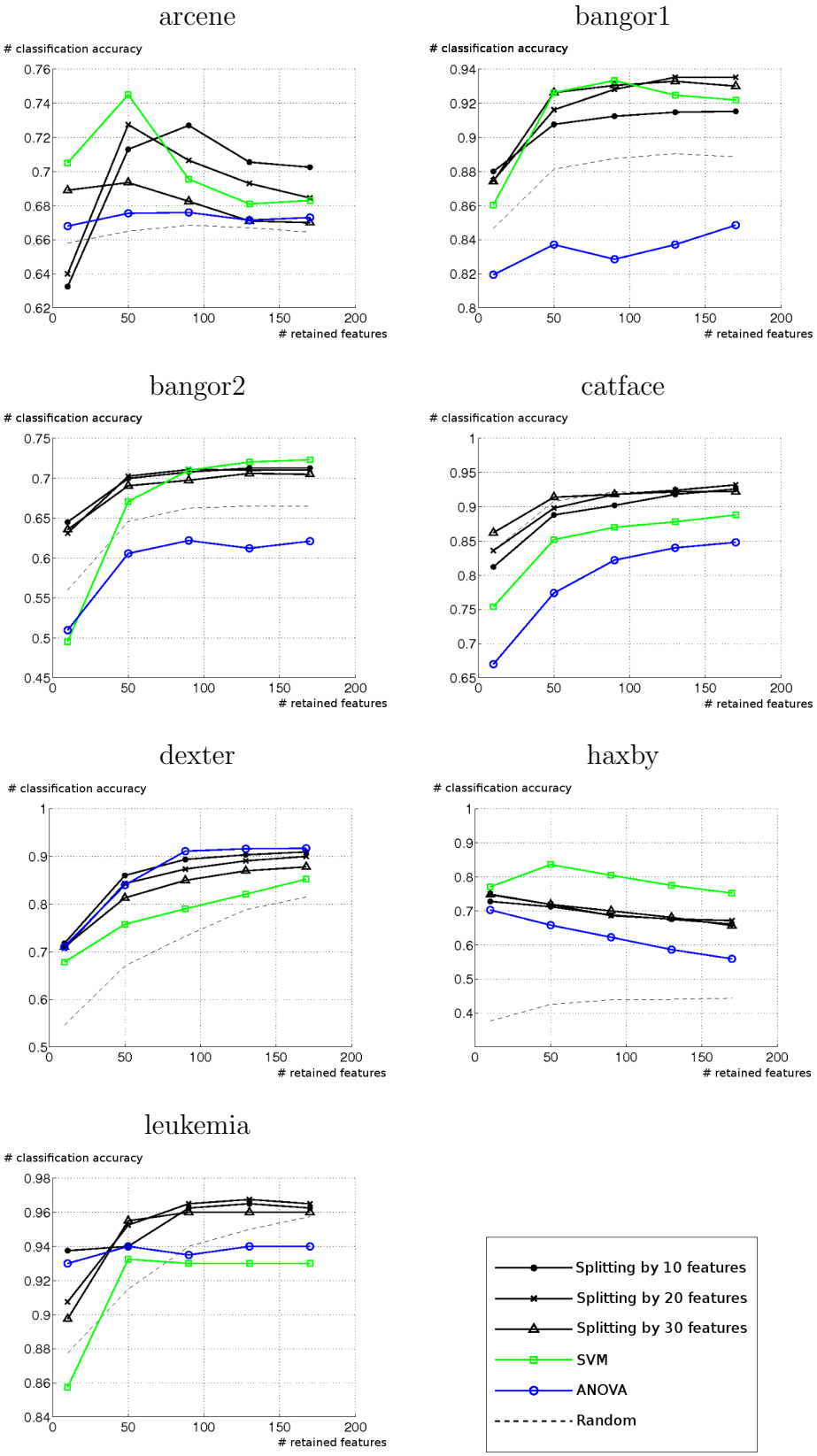


Fig. 49. Classification accuracy with the selected feature sets.

Table 26
Statistical significance of the differences between the AUC of the splitting and the random arranging (R), SVM (S) and ANOVA (A).

	AUC									Last value (170 features)								
	$M = 10$			$M = 20$			$M = 30$			$M = 10$			$M = 20$			$M = 30$		
	R	S	A	R	S	A	R	S	A	R	S	A	R	S	A	R	S	A
arcene	•	–	•	•	○	•	•	○	–	•	•	•	•	–	•	•	○	–
bangor1	•	–	•	•	–	•	•	–	•	•	–	•	•	•	•	•	–	•
bangor2	•	•	•	•	•	•	•	•	•	•	–	•	•	–	•	•	○	•
catface	–	•	•	–	•	•	–	•	•	–	•	•	–	•	•	–	–	•
dexter	•	•	–	•	•	○	•	•	○	•	•	–	•	•	○	•	•	○
haxby	•	○	•	•	○	•	•	○	•	•	○	•	•	○	•	•	○	•
leukemia	–	•	•	–	•	–	–	•	–	–	•	•	–	•	–	–	•	–

Notes:
• Splitting is significantly better
○ Splitting is significantly worse
– No significant difference

Table 27
Average time (in seconds) for one ranking of the features.

	Splitting			R	S	A
Data set	$M = 10$	$M = 20$	$M = 30$			
arcene	7.5	5.1	4.4	0.0	16.8	3.8
bangor1	7.8	5.2	4.5	0.0	17.1	3.8
bangor2	13.1	9.9	8.2	0.0	75.6	4.3
catface	4.1	3.0	2.2	0.0	0.5	3.8
dexter	9.8	6.3	5.1	0.0	52.7	3.8
haxby	11.1	9.2	7.5	0.0	39.8	4.5
leukemia	4.3	4.0	3.4	0.0	0.3	4.4
Average	8.2	6.1	5.1	0.0	29.0	4.0

The results indicate that splitting is generally better than the other methods. It scores the best ranks for AUC with all three values of M (Table 26). The corresponding curves in Figure 49 are often higher than two of the three competing curves (all three for the leukemia data) and on a par with the third curve. The results with haxby data are a curious exception. SVM ranking outperforms all competitors but the accuracies decline with the number of retained features, which represents an early peak effect, unexpected for the nearest mean classifier. This may be also an indication that there is genuinely a very small number of relevant features, and adding "clutter" manages to spoil the classification accuracy. Regarding the subset size, no single choice of this parameter was demonstrably better than the other choices, $M = 20$ has a slight edge over the other two values but the three curves behave very similarly. This is reassuring as it suggests that the splitting approach (with the chosen evaluation criterion) is not adversely sensitive to the choice of M.

Finally, to aid computational complexity analyses, Table 27 shows the time in seconds taken by the selection part of the code. It is worth mentioning how splitting considerably reduces the amount of time needed by SVM.

To sum up, splitting ranked better than the other methods the top of the list of features with highest starting points in the accuracy curves. The results suggest that repeatedly splitting the feature set and evaluating the subsets is accurate and computationally inexpensive. It was faster and moreover it ranked best among the competing methods on the overall accuracy.

5.7 Summary

In this section, we have presented a new method for scaling up feature selection algorithms, which consists of performing several rounds of weak feature selection on subsets of the original dataset and combined into a single subset of relevant features or a ranking. As feature selection algorithms, we used two of the most well-known and highly recommended feature selection algorithms: SVM-RFE and a genetic algorithm.

Our main objective was to design a method that would be able to successfully scale up feature selection algorithms, meaning that the running time as well as the storage reduction would be considerably reduced and that the accuracy would not drop to inadmissible values.

The experiments showed that our method is able to shorten the execution time impressively compared to the standard feature selection algorithms. Furthermore, *pseudoensembles* achieved a similar performance to the original feature selection algorithms. In terms of storage requirements and testing error, our approach is able to match and in some cases even improve the standard results applied to the non-partitioned datasets. The results demonstrate that *pseudoensembles* are able to scale up to millions of instances and thousands of features, with especially interesting results in one of the most widely used feature selection algorithms, SVM-RFE.

The proposed approach is applicable to any feature selection method without any modifications. Moreover, it is naturally parallelizable, as the application of the feature selection algorithm to each small subset of instances or features is independent of all the remaining subsets and can therefore be processed at the same time. In fact, because the size of the subset is chosen by the researcher, we can apply the method to any problem regardless of its size, whereas most other methods would be inapplicable in many cases. To prove this fact, in Section 5.6 we have described the application of our *pseudoensembles* scaling methodology, designed in collaboration with Dr. Kuncheva. Due to the good efficiency of this application, we are able to reduce a set of 3000 features to 200 features in real world medical problems, such as the complex field of fMRI, where other algorithms are overloaded for computational reasons.

6 Conclusions

Advances in digital and computer technology have led to massive amounts of information and collections of data to be processed, containing many gigabytes (even terabytes) of data. Knowledge discovery in databases (KDD) and data mining can help dealing with this problem because they aim to turn raw data into nuggets and create special edges. KDD practitioners would like to be able to apply data mining learning algorithms to these large data sets in order to discover useful knowledge.

However, the increase of the database's size is a staple problem, which is known as scaling up problem. The question of scalability asks whether the algorithm can process large data sets efficiently, while building the best possible models from them. As raw data are rarely used directly and manual analysis simply cannot keep up with the fast growth of data, with the growing size of the datasets the need is also growing to scale up data mining algorithms in all the fields of application of machine learning. Scientific research, ranging from astronomy to human natural genome, text mining or security among others, faces the same problem of how to deal with vast amounts of information.

Moreover, just selecting a representative subset of the original dataset is not an option, due to several reasons: if the dataset is huge, subsampling a percentage can itself pose a challenge and, above all, increasing the size of the training set often increases the accuracy of classifiers, avoiding overfitting.

The scaling problem produces excessive storage requirement, affects generalization accuracy and increases time complexity being very slow in learning a model or classifier and consequently cannot be used to address relevant problems due to scalability issues. Very large data sets present a challenge for both humans and machine learning algorithms. As a consequence, many useful algorithms are inundated by the flood of data and become impracticable.

In this thesis we have proposed a general methodology, called *democratization*, to scale up effectively any data mining algorithm without severe modifications, it can be specifically classified into the data partitioning scaling methods (Provost and Kolluri, 1999). Our scaling methodology is based on the ensembles of classifiers approach, which combines several weak classifiers to produce an efficient result. In our approach we do not have weak classifiers but weak data mining algorithms, each one applied to an independent and small-sized subset of the original dataset.

Regarding other scaling approaches, the main difference introduced by our scaling methodology is that it uses the whole dataset for the data mining algorithms to learn, not restricting the learning process to a subset of the

original dataset. Using just a subset of the whole dataset, as many scaling algorithms do, leads to less accurate results as not all the available information is used in the learning stage. Consequently, the proposed methodology allows a dramatic decrease of the running time as well as a considerable storage reduction without dropping the accuracy to inadmissible values.

It is worth noting that the computational complexity of our scaling methodology is really competitive. The linear complexity of our method has been demonstrated both theoretically and experimentally. Moreover our scaling methodology is not very sensitive to the values of its parameters and has the additional advantage of allowing the user to adapt the size of the subproblems to the available resources. Along with the fact that our method only requires each subset to be in memory while it is processed but not during the processing of the remaining subsets, the *democratization* approach is applicable to any problem regardless of its size. This circumstance has been experimentally proven with the positive results obtained in the application of *democratization* to huge problems with millions of instances and thousands of features.

As mentioned, the scaling approach presented in this thesis is a simple 3-step methodology applicable to any data mining algorithm without modifications. Throughout this thesis we have proven the generality of the democratization methodology, applying it to scale up two important data mining problems: instance selection and feature selection.

As regards instance selection, we have successfully applied our *democratization* methodology to scale up a representative set of relevant instance selection algorithms, including a genetic algorithm selector whose computational complexity requirements are well-known to be high, giving us a good benchmark to test our methodology.

The application of our scaling framework to instance selection is called DEMOIS.. It allows a performance close to the application of the algorithm to the whole dataset, while retaining the advantages of a smaller subset.

Additionally, our method has been also 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 it is the case of instance selection algorithms for k-nearest neighbors method.

Our experimental results proved that in terms of reduction of storage requirements and testing error, our approach matches and in some cases is even better than the use of the original instance selection algorithms over the whole dataset. In terms of execution time, the behavior is excellent for the tested instance selection algorithms, being this fact the most important advantage of our method. The reported experiments showed a large difference when using

standard widely used instance selection algorithms.

Additionally, our method is straightforwardly paralellizable without significant modifications. To test the parallelization behavior of our methodology we parallelized DEMOIS., this approach was named FEDERAL instance selection. This parallel version of DEMOIS. was tested with huge datasets (the largest set of 50 million instances and 800 features) and provided an impressive reduction of the execution time when compared with other relevant and widely used instance selection algorithms.

Regarding the scaling up of feature selection algorithms, we presented an application of the *democratization* methodology to scale up feature selection methods. This scaling methodology applied to feature selection is called *pseudoensembles* and provides the output in the same format as the original feature selection algorithm, being possible to produce either a subset of relevant features or a ranked set of features ordered by their relevance.

Our extensive experiments included a comparison with well-known and highly recommended feature selection algorithms. Our results proved that *pseudoensembles* are especially efficient when we use feature selection algorithms that suffer from high computational cost or when we have to deal with very large datasets. Furthermore, *pseudoensembles* are able to shorten the execution time impressively compared to the standard feature selection algorithms and achieved a similar performance to the original feature selection algorithms.

Moreover, due to the same reasons as DEMOIS., *pseudoensembles* are also naturally parallelizable. As the size of the subset is chosen by the researcher, we can apply the method to any problem regardless of its size. In contrast, the majority of methods are inapplicable when dealing with large datasets. To prove this fact, we described the application of our *pseudoensembles* scaling methodology to the complex field of fMRI, during my collaboration with Dr. Kuncheva. Due to the good efficiency of *pseudoensembles* we were able to use them to reduce a set of 3000 features to 200 features in the designed feasibility pyramid.

To sum up, the main contribution of this thesis is the novel design of a simple and generic methodology that scales up any data mining algorithms. Another relevant contribution of this thesis is the application of our *democratization* methodology to scale up state of the art instance and feature selection algorithms. Both applications of the *democratization* methodology produced very competitive results when compared with the standard algorithms. Our scaling methodology has also been successfully applied to real-world problems.

As our main line of continuing research, we are not only working on selecting instances but also modifying prototypes. Considering the fact that our methodology is subtle to be applied to any data mining algorithm, we think

it is a challenging task to scale up new algorithms that combine feature and instance selection at the same time.

Moreover we are working on the development of better methods of partitioning the original dataset that we believe will have a relevant influence on the performance of the method. Specifically, we expect that partitioning by instances and by features as well as the development of data–dependent methods to partition the original datasets will provide very competitive results.

Additionally, we plan to apply this scaling framework to different data mining fields, such as clustering or ensembles of classifiers. We think that our approach would be capable of dealing with huge datasets in these areas of research, improving their performance. As an example we plan to scale up TIS recognition and gene identification algorithms that deal with huge bioinformatic datasets.

References

Agrawal, R., T. Imielinski, and A. Swami: 1993, 'Database mining: A performance perspective'. *IEEE Transactions on Knowledge and Data Engineering* **5**(6), 914–925.

Aha, D. W., D. Kibler, and M. K. Albert: 1991, 'Instance-based learning algorithms'. *Machine Learning* **6**, 37–66.

Anderson, T. W.: 1984, *An introduction to multivariate statistical analysis*, Wiley Series in Probability and Mathematical Statistics. New York: John Wiley & Sons, 2nd edition.

Andrews, D. F.: 1972, 'Plots of High Dimensional Data'. *Biometrics* **28**, 125–136.

Asimov, D.: 1985, 'The Grand Tour: a Tool for Viewing Multidimensional Data'. *SIAM Journal on Scientific and Statistical Computing* **6**(1), 128–143.

Baluja, S.: 1994, 'Population-based incremental learning'. Technical Report CMU-CS-94-163, Carnegie Mellon University, Pittsburgh.

Banfield, R. E., L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer: 2005, 'Ensembles of Classifiers from Spatially Disjoint Data'. In: *Lecture Notes in Computer Science*, Vol. 3541. pp. 196–205.

Barandela, R., F. J. Ferri, and J. S. Sánchez: 2005, 'Decision boundary preserving prototype selection for nearest neighbor classification'. *International Journal of Pattern Recognition and Artificial Intelligence* **19**(6), 787–806.

Bauer, E. and R. Kohavi: 1999, 'An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants'. *Machine Learning* **36**(1/2), 105–142.

Berman, H.-M., J. Westbrook, Z. Feng, G. Gilliland, T.-N. Bhat, H. Weissig, I. Shindyalov, and P.-E. Bourne: 2000, 'The Protein Data Bank'. *Nucleic Acids Res* **28**, 235–242.

Bezdek, J. and L.-I. Kuncheva: 2000, 'Some Notes on Twenty One 21 Nearest Prototype Classifiers'. In: F. Ferri, J. Iñesta, A. Amin, and P. Pudil (eds.): *Advances in Pattern Recognition*, Vol. 1876 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, pp. 1–16.

Bins, J. and B. A. Draper: 2001, 'Feature selection from huge feature sets'. In: *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, Vol. 2. pp. 159–165.

Blum, A. and P. Langley: 1997, 'Selection of relevant features and examples in machine learning'. *Artificial Intelligence* **97**, 245–271.

Blum, A. L. and R. L. Rivest: 1992, 'Training a 3-node neural network is NP-complete'. *Neural Networks* **5**, 117–127.

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

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

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

Breiman, L.: 1998, 'Arcing classifiers'. *Annals of Statistics* **26**, 801–824.

Breiman, L.: 1999, 'Pasting Small Votes for Classification in Large Databases and On-Line'. *Machine Learning* **36**, 85–103.

Brighton, H. and C. Mellish: 2002, 'Advances in Instance Selection for Instance-Based Learning Algorithms'. *Data Mining and Knowledge Discovery* **6**, 153–172.

Brodley, C. E. and M. A. Friedl: 1999, 'Identifying mislabeled training data'. *Journal of Artificial Intelligence Research* **11**, 131–167.

Buja, A. and D. Asimov: 1986, 'Grand Tour Methods: An Outline'. In: D. Allen (ed.): *Computer Science and Statistics: Proceedings of the Seventeenth Symposium on the Interface*. Amsterdam: North Holland, pp. 63–67.

Buja, A., D. Cook, D. Asimov, and C. Hurley: 2005, *Computational Methods for High-Dimensional Rotations in Data Visualization*, Chapt. 14, pp. 391–414. North-Holland Publishing Co.

Cano, J. R., F. Herrera, and M. Lozano: 2003, 'Using Evolutionary Algorithms as Instance Selection for Data Reduction in KDD: An Experimental Study'. *IEEE Transactions on Evolutionary Computation* **7**(6), 561–575.

Cano, J. R., F. Herrera, and M. Lozano: 2005, 'Stratification for scaling up evolutionary prototype selection'. *Pattern Recognition Letters* **26**(7), 953–963.

Cano, J. R., F. Herrera, and M. Lozano: 2007, 'Evolutionary Stratified Training Set Selection for Extracting Classification Rules with trade off Precision-Interpretability'. *Data & Knowledge Engineering* **60**(1), 90–108.

Caruana, R. and D. Freitag: 1994, 'Greedy Attribute Selection'. In: *The 11th International Conference on Machine Learning*. pp. 28–36.

Chang, C.-C. and C.-J. Lin: 2001, 'LIBSVM: a library for support vector machines'. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Chaudhuri, S., R. Motwani, and V. Narasayya: 1998, 'Random sampling for histogram construction: How much is enough?'. In: L. Haas and A. Tiwary (eds.): *Proceedings of ACM SIGMOD, International Conference on Management of Data*. New York, USA, pp. 436–447.

Chawla, N. W., L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer: 2004, 'Learning Ensembles from Bites: A Scalable and Accurate Approach'. *Journal of Machine Learning Research* **5**, 421–451.

Chen, J. H., H. M. Chen, and S. Y. Ho: 2005, 'Design of nearest neighbor classifiers: multi-objective approach'. *International Journal of Approximate Reasoning* **40**(1-2), 3–22.

Cochran, W.: 1977, *Sampling Techniques*. New York, USA: John Wiley & Sons.

Cover, T. M. and P. E. Hart: 1967, 'Nearest neighbor pattern classification'. *IEEE Transactions on Information Theory* **IT-13**, 21–27.

Cover, T. M. and J. A. Thomas: 1991, *Elements of Information Theory*, Wiley series in telecommunication. John Wiley & Sons, Inc.

Das, S.: 2001, 'Filters, Wrappers and a Boosting-Based Hybrid for Feature Se-

lection'. In: *The 18th International Conference on Machine Leaning (ICML-03)*. San Francisco, CA, USA, pp. 74–81.

Dash, M., K. Choi, P. Scheuermann, and H. Liu: 2002, 'Feature Selection for Clustering - A Filter Solution'. In: *Proceedings of the Second International Conference on Data Mining*. pp. 115–122.

Dash, M. and H. Liu: 1997, 'Feature Selection for Classification'. *Intelligent Data Analysis* **1**, 131–156.

de Haro-García, A., J.-A. R. del Castillo, and N. G. Pedrajas: 2010, 'Large scale instance selection by means of a parallel algorithm'. In: *Proceedings of the 11th international conference on Intelligent data engineering and automated learning*. Berlin, Heidelberg, pp. 1–12.

de Haro-García, A., L. I. Kuncheva, and N. García-Pedrajas: 2011, 'Random splitting for cascade feature selection'. Technical report, Computer Sciences Department, University of Cordoba.

de Haro-García, A. and N. G. Pedrajas: 2009, 'A divide-and-conquer recursive approach for scaling up instance selection algorithms'. *Data Mining and Knowledge Discovery* **18**(3), 392–418.

de Haro-García, A. and N. G. Pedrajas: 2010, 'Scaling up feature selection by means of democratization'. In: *Proceedings of the 23rd international conference on Industrial engineering and other applications of applied intelligent systems - Volume Part II*. Berlin, Heidelberg, pp. 662–672.

de Oca, M., D. Aydin, and T. Stutzle: 2010, 'An incremental particle swarm for large-scale continuous optimization problems: an example of tuning-in-the-loop (re)design of optimization algorithms'. *Soft Computing. A Fusion of Foundations, Methodologies and Applications* pp. 1–23.

Demšar, J.: 2006, 'Statistical Comparisons of Classifiers over Multiple Data Sets'. *Journal of Machine Learning Research* **7**, 1–30.

Derrac, J., S. García, and F. Herrera: 2010, 'Stratified prototype selection based on a steady-state memetic algorithm: a study of scalability'. *Memetic Computing* **2**, 183–189.

Dietterich, T. G.: 1998, 'Approximate statistical tests for comparing supervised classification learning algorithms'. *Neural Computation* **10**(7), 1895–1923.

Dietterich, T. G.: 2000a, 'An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, Boosting, and Randomization'. *Machine Learning* **40**, 139–157.

Dietterich, T. G.: 2000b, 'Ensemble Methods in Machine Learning'. In: J. Kittler and F. Roli (eds.): *Proceedings of the First International Workshop on Multiple Classifier Systems*, Vol. 1857 of *Lecture Notes in Computer Science*. pp. 1–15.

Domingos, P. and G. Hulten: 2001a, 'A general method for scaling up machine learning algorithms and its application to clustering'. In: *Proceedings of the Eighteenth International Conference on Machine Learning*. pp. 106–113.

Domingos, P. and G. Hulten: 2001b, 'Learning from Infinite Data in Finite Time'. In: *Proceedings of Advances in Neural Information Systems 14*. Van-

couver, Canada, pp. 673–680.

Dy, J.-G. and C.-E. Brodley: 2000, 'Feature Subset Selection and Order Identification for Unsupervised Learning'. In: *The 17th International Conference on Machine Learning*. pp. 247–254.

Dzeroski, S. and B. Zenko: 2004, 'Is combining classifiers with stacking better than selecting the best one?'. *Machine Learning* **54**, 255–273.

Eshelman, L. J.: 1990, *The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination*. San Mateo, CA: Morgan Kauffman.

Fern, A. and R. Givan: 2003, 'Online ensemble learning: An empirical study'. *Machine Learning* **53**, 71–109.

Fountain, T., H. Almuallim, and T. Dietterich: 1991, 'Learning With Many Irrelevant Features'. In: *In Proceedings of the Ninth National Conference on Artificial Intelligence*. pp. 547–552.

Freund, Y. and R. Schapire: 1996, 'Experiments with a new boosting algorithm'. In: *Proc. of the Thirteenth International Conference on Machine Learning*. Bari, Italy, pp. 148–156.

Friedman, J. H., J. L. Bentley, and R. A. Finkel: 1977, 'An Algorithm for Finding Best Matches in Logarithmic Expected Time'. *ACM Trans. Math. Softw.* **3**, 209–226.

Gadat, S. and L. Younes: 2007, 'A Stochastic Algorithm for Feature Selection in Pattern Recognition'. *Journal of Machine Learning Research* **8**, 509–547.

Garcia, S., J. Cano, and F. Herrera: 2008, 'A memetic algorithm for evolutionary prototype selection: A scaling up approach'. *Pattern Recognition* **41**, 2693–2709.

García-Osorio, C., A. de Haro-García, and N. García-Pedrajas: 2010, '*Democratic* Instance Selection: a linear complexity instance selection algorithm based on classifier ensemble concepts'. *Artificial Intelligence* **174**(5-6), 410–441.

García-Osorio, C. and C. Fyfe: 2005, 'Regaining Sparsity in Kernel Principal Components'. *Neurocomputing* **67**, 398–402.

García-Pedrajas, N., J. A. R. del Castillo, and D. Ortiz-Boyer: 2010, 'A cooperative coevolutionary algorithm for instance selection for instance-based learning'. *Machine Learning* **78**, 381–420.

García-Pedrajas, N., C. García-Osorio, and C. Fyfe: 2007, 'Nonlinear Boosting Projections for Ensemble Construction'. *Journal of Machine Learning Research* **8**, 1–33.

García-Pedrajas, N., C. Hervás-Martínez, and D. Ortiz-Boyer: 2005, 'Cooperative Coevolution of Artificial Neural Network Ensembles for Pattern Classification'. *IEEE Transactions on Evolutionary Computation* **9**(3), 271–302.

García-Pedrajas, N. and D. Ortiz-Boyer: 2007, 'A cooperative constructive method for neural networks for pattern recognition'. *Pattern Recognition* **40**(1), 80–99.

García-Pedrajas, N., D. Ortiz-Boyer, and C. Hervás-Martínez: 2004, 'Cooperative coevolution of generalized multi-layer perceptrons'. *Neurocomputing*

56C, 257–283.

Gates, G. W.: 1972, 'The reduced nearest neighbor rule'. *IEEE Transactions on Information Theory* **18**(3), 431–433.

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

Golub, T. R., D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander: 1999, 'Molecular classification of cancer: class discovery and class prediction by gene expression monitoring.'. *Science (New York, N.Y.)* **286**(5439), 531–537.

Guyon, I., A. B. Hur, S. Gunn, and G. Dror: 2004, 'Result analysis of the NIPS 2003 feature selection challenge'. In: *Advances in Neural Information Processing Systems 17*. pp. 545–552.

Guyon, I., J. Weston, S. Barnhill, and V. Vapnik: 2002, 'Gene Selection for Cancer Classification using Support Vector Machines'. *Machine Learning* **46**, 389–422.

Hart, P. E.: 1968, 'The condensed nearest neighbor rule'. *IEEE Transactions on Information Theory* **14**(3), 515–516.

Haxby, J. V., M. I. Gobbini, M. L. Furey, A. Ishai, J. L. Schouten, and P. Pietrini: 2001, 'Distributed and overlapping representations of faces and objects in ventral temporal cortex.'. *Science (New York, N.Y.)* **293**(5539), 2425–2430.

Hettich, S., C. Blake, and C. Merz: 1998, 'UCI Repository of machine learning databases'. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

Holland, J. H.: 1975, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: The University of Michigan Press.

Holte, R., L. Acker, and B. Porter: 1989, 'Concept Learning and the Problem of Small Disjuncts'. Technical report, Queen's University, Austin, TX, USA.

Howffding, W.: 1963, 'Probability inequalities for sums of bounded random variables'. *Journal of the American Statistical Association* **58**, 13–30.

Hulten, G. and P. Domingos: 2002, 'Mining Complex Models from Arbitrarily Large Databases in Constant Time'. In: *Proceedings of the International Conference on Knowledge Discovery and Data Mining*. Edmonton, Canada, pp. 525–531.

Ishibuchi, H. and T. Nakashima: 2000, 'Pattern and Feature Selection by Genetic Algorithms in Nearest Neighbor Classification'. *Journal of Advanced Computational Intelligence and Intelligent Informatics* **4**(2), 138–145.

Jain, A. and D. Zongker: 1997, 'Feature selection: Evaluation, application, and small sample performance'. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **19**, 153–158.

John, G.-H., R. Kohavi, and K. Pfleger: 1994, 'Irrelevant Features and the Subset Selection Problem'. In: *The 11th International Conference on Machine Learning*. pp. 121–129.

Kim, S.-W. and B. J. Oommen: 2004, '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), 1384–1397.

Kim, Y., W.-N. Street, and F. Menczer: 2000, 'Feature Selection in Unsupervised Learning via Evolutionary Search'. In: *The 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. pp. 365–369.

Kira, K. and L.-A. Rendell: 1992, 'A practical approach to feature selection'. In: *The 9th international workshop on Machine learning*. San Francisco, CA, USA, pp. 249–256.

Kivinen, J. and H. Mannila: 1994, 'The power of sampling in knowledge discovery'. In: *Proceedings of the thirteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems* . Minneapolis, Minnesota, USA, pp. 77–85.

Kohavi, R. and G.-H. John: 1997, 'Wrappers for feature subset selection'. *Artificial Intelligence* **97**(1-2), 273–324.

Kohavi, R. and C. Kunz: 1997, 'Option decision trees with majority voting'. In: *Proceedings of the Fourteenth International Conference on Machine Learning*. San Francisco, CA, USA, pp. 161–169.

Kuncheva, L.: 1995, 'Editing for the k-Nearest Neighbors rule by a genetic algorithm'. *Pattern Recognition Letters* **16**, 809–814.

Kuncheva, L. and C. J. Whitaker: 2003, 'Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy'. *Machine Learning* **51**(2), 181–207.

Kuncheva, L. I.: 2001, 'Combining classifiers: Soft computing solutions'. In: S. K. Pal and A. Pal (eds.): *Pattern Recognition: From Classical to Modern Approaches*. World Scientific, pp. 427–451.

Kuncheva, L.-I. and J.-C. Bezdek: 1998, 'Nearest prototype classification: clustering, genetic algorithms, or random search?'. *IEEE Transactions on Systems, Man and Cybernetics, Part C* **28**(1), 160–164.

Kuncheva, L. I., J. J. Rodriguez, C. O. Plumpton, D. E. Linden, and S. J. Johnston: 2010, 'Random subspace ensembles for FMRI classification.'. *IEEE transactions on medical imaging* **29**(2), 531–542.

Lai, C., M. Reinders, J. Marcel, and L. Wessels: 2006, 'Random subspace method for multivariate feature selection'. *Pattern Recogn. Lett.* **27**, 1067–1076.

Lazarevic, A. and Z. Obradovic: 2002, 'Boosting Algorithms for Parallel and Distributed Learning'. *Distrib. Parallel Databases* **11**, 203–229.

Leavitt, N.: 2002, 'Data Mining for the Corporate Masses?'. *Computer* **35**, 22–24.

Lee, W., S.-J. Stolfo, and K.-W. Mok: 2000, 'Adaptive Intrusion Detection: a Data Mining Approach'. *Artificial Intelligence Review* **14**, 533–567.

Leopold, E. and J. Kindermann: 2002, 'Text Categorization with Support Vector Machines. How to Represent Texts in Input Space?'. *Machine Learning* **46**(1-3), 423–444.

Li, J., M. T. Manry, C. Yu, and D. R. Wilson: 2005, 'Prototype classifier

design with pruning'. *International Journal of Artificial Intelligence Tools* **14**(1-2), 261–280.

Li, L., C.-R. Weinberg, T.-A. Darden, and L.-G. Pedersen: 2001, 'Gene selection for sample classification based on gene expression data: study of sensitivity to choice of parameters of the GA/KNN method'. *Bioinformatics* **17**(12), 1131–1142.

Liu, D., T.-S. Chang, and Y. Zhang: 2002, 'A constructive algorithm for feedforward neural networks with incremental training'. *IEEE Transactions on Circuits and Systems — I: Fundamental Theory and Applications* **49**(12), 1876–1879.

Liu, H. and H. Motoda: 2001, *Feature Extraction, Construction and Selection: A Mining Perspective*. Kluwer Academic Publishers.

Liu, H. and H. Motoda: 2002, 'On Issues of Instance Selection'. *Data Mining and Knowledge Discovery* **6**, 115–130.

Liu, H. and R. Setiono: 1996, 'A Probabilistic Approach to Feature Selection - A Filter Solution'. In: *The 13th International Conference on Machine Learning (ICML'96)*. pp. 319–327.

Liu, H. and L. Yu: 2005, 'Toward Integrating Feature Selection Algorithms for Classification and Clustering'. *IEEE Trans. on Knowl. and Data Eng.* **17**, 491–502.

Maudes-Raedo, J., J. J. Rodríguez-Díez, and C. García-Osorio: 2008, 'Disturbing Neighbors Diversity for Decision Forest'. In: G. Valentini and O. Okun (eds.): *Workshop on Supervised and Unsupervised Ensemble Methods and Their Applications (SUEMA 2008)*. Patras, Grecia, pp. 67–71.

Merz, C. J.: 1999, 'Using Correspondence Analysis to Combine Classifiers'. *Machine Learning* **36**(1), 33–58.

Michalewicz, Z.: 1994, *Genetic Algorithms + Data Structures = Evolution Programs*. New York: Springer–Verlag.

Mitra, P., C.-A. Murthy, and S.-K. Pal: 2002, 'Unsupervised feature selection using feature similarity'. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **24**, 301–312.

Narayan, B.-L., C.-A. Murthy, and S.-K. Pal: 2006, 'Maxdiff kd-trees for data condensation'. *Pattern Recogn. Lett.* **27**, 187–200.

Narendra, P.-M. and K. Fukunaga: 1977, 'Branch, and bound algorithm for feature subset selection'. *IEEE Transactions Computer* **C-26**(9), 917–922.

Ng, K. and H. Liu: 1999, 'Customer Retention via Data Mining'. *AI Review* **14**, 590.

Nigam, K., A.-K. Mccallum, S. Thrun, and T. Mitchell: 1999, 'Text Classification from Labeled and Unlabeled Documents using EM'. In: *Machine Learning*. pp. 103–134.

Olvera-López, J., J. Carrasco-Ochoa, J. Martínez-Trinidad, and J. Kittler: 2010, 'A review of instance selection methods'. *Artificial Intelligence Review* **34**, 133–143.

Paredes, R. and E. Vidal: 2000, 'Weighting Prototypes. A New Editing Approach.'. In: *In XV International Conference on Pattern Recognition. 15th*

ICPR. pp. 25–28.

Pedrajas, N. G., A. de Haro-García, and J. A. R. del Castillo: 2011, 'Large scale instance selection by means of federal instance selection'. *Data & Knowledge Engineering"*. submitted.

Pereira, F., T. Mitchell, and M. Botvinick: 2009, 'Machine learning classifiers and fMRI: A tutorial overview'. *NeuroImage* **45**(1), S199–S209.

Provost, F.-J. and D.-N. Hennessy: 1996, 'Scaling Up: Distributed Machine Learning with Cooperation'. In: *In Proceedings of the Thirteenth National Conference on Artificial Intelligence*. pp. 74–79.

Provost, F. J. and V. Kolluri: 1999, 'A survey of methods for scaling up inductive learning algorithms'. *Data Mining and Knowledge Discovery* **2**, 131–169.

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

Reeves, C. R. and D. R. Bush: 2001, 'Using genetic algorithms for training data selection in RBF networks'. In: H. Liu and H. Motoda (eds.): *Instances Selection and Construction for Data Mining*. Norwell, Massachusetts, USA: Kluwer, pp. 339–356.

Rodríguez, J. J., C. García-Osorio, and J. Maudes: 2010, 'Forests of Nested Dichotomies'. *Pattern Recognition Letters* **31**(2), 125–132.

Rokach, L.: 2009, 'Taxonomy for characterizing ensemble methods in classification tasks: A review and annotated bibliography'. In: *Computational Statistics and Data Analysis*.

Rui, Y. and T.-S. Huang: 1999, 'Image retrieval: Current techniques, promising directions and open issues'. *Journal of Visual Communication and Image Representation* **10**, 39–62.

Saeys, Y., I. Inza, and P. Larrañaga: 2007, 'A Review of Feature Selection Techniques in Bioinformatics'. *Bioinformatics* **23**(19), 2507–2517.

Sane, S. S. and A. A. Ghatol: 2007, 'A novel Supervised Instance Selection algorithm'. *International Journal on Business Intelligence and Data Mining* **2**(4), 471–495.

Schapire, R. E., Y. Freund, P. L. Bartlett, and W. S. Lee: 1998, 'Boosting the margin: A new explanation for the effectiveness of voting methods'. *Annals of Statistics* **26**(5), 1651–1686.

Sebban, M. and R. Nock: 2000, 'Identifying and eliminating irrelevant instances using information theory'. In: H. Hamilton and Q. Yang (eds.): *13th Biennial Conference of the Canadian Society for Computational Studies of Intelligence, AI 2000 Montreal*, Vol. 1822 of *Lecture Notes in Artificial Intelligence*. Springer, pp. 90–101.

Sebban, M., R. Nock, J. H. Chauchat, and R. Rakotomalala: 2000, 'Impact of learning set quality and size on decision tree performances'. *International Journal of Computers, Systems and Signals* **1**(1), 85–105.

Siedlecki, W. and J. Sklansky: 1989, 'A note on genetic algorithms for large-scale feature selection'. *Pattern Recognition Lett.* **10**, 335–347.

Sikonja, M.-R.: 1998, 'Speeding up Relief Algorithm with K-d Trees'. In:

Electrotechnical and Computer Science Conference ERK'98. Slovenia, pp. 137–140.

Smith, P.: 1998, *Into Statistics*. Singapore: Springer-Verlag.

Smyth, B. and M. T. Keane: 1995, 'Remembering to forget'. In: C. S. Mellish (ed.): *Proceedings of the Fourteenth International Conference on Artificial Intelligence*, Vol. 1. pp. 377–382.

Son, S. H. and J. Y. Kim: 2006, 'Data reduction for instance-based learning using entropy-based partitioning'. In: *Proceedings of the International Conference on Computational Science and Its Applications - ICCSA 2006*, No. 3982 in Lecture Notes in Computer Science. Springer, pp. 590–599.

Sonnenburg, S., K. Rieck, F. F. Ida, and G. Rätsch: 2007, 'Large scale learning with string kernels'. In: *Large Scale Kernel Machines*. pp. 73–103.

Spillmann, B., M. Neuhaus, H. Bunke, E. Pekalska, and R. Duin: 2006, 'Transforming Strings to Vector Spaces Using Prototype Selection.'. In: *SSPR/SPR'06*. pp. 287–296.

Suganthan, P. N., N. Hansen, J.-J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari: 2005, 'Problem definitions and evaluation criteria for the CEC 2005 Special Session on Real Parameter Optimization'. Technical report, Nanyang Technological University.

Swets, D.-L. and J.-J. Weng: 1995, 'Efficient Content-Based Image Retrieval using Automatic Feature Selection'. In: *In IEEE International Symposium on Computer Vision*. pp. 85–90.

Webb, G. I.: 2000, 'MultiBoosting: A Technique for Combining Boosting and Wagging'. *Machine Learning* **40**(2), 159–196.

Wegman, E. J. and J. Shen: 1993, 'Three-Dimensional Andrews Plots and the Grand Tour'. *Computing Science and Statistics* **25**, 284–288.

Wegman, E. J. and J. L. Solka: 2002, 'On Some Mathematics for Visualising High Dimensional Data'. *Indian Journal of Statistics* **64**(Series A, Pt. 2), 429–452.

Whitley, D.: 1989, 'The GENITOR Algorithm and Selective Pressure'. In: M. K. Publishers (ed.): *Proc 3rd International Conf. on Genetic Algorithms*. Los Altos, CA, pp. 116–121.

Whitley, D. and J. Kauth: 1988, 'GENITOR: A Different Genetic Algorithm'. In: *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*. Denver, CO, pp. 118–130.

Wilcoxon, F.: 1945, 'Individual comparisons by ranking methods'. *Biometrics* **1**, 80–83.

Wilson, D. L.: 1972, 'Asymptotic properties of nearest neighbor rules using edited data'. *IEEE Transactions on Systems, Man, and Cybernetics* **2**(3), 408–421.

Wilson, D. R. and A. R. Martinez: 1997, 'Instance pruning techniques'. In: D. Fisher (ed.): *Proceedings of the Fourteenth International Conference on Machine Learning*. San Francisco, CA, USA, pp. 404–411.

Wilson, D. R. and T. R. Martinez: 2000, 'Reduction Techniques for Instance-Based Learning Algorithms'. *Machine Learning* **38**, 257–286.

Xing, E.-P., M.-I. Jordan, and R.-M. Karp: 2001, 'Feature selection for high-dimensional genomic microarray data'. In: *The 18th International Conference on Machine Learning*. pp. 601–608.

Yang, J. and V. Honavar: 1998, 'Feature Subset Selection Using a Genetic Algorithm'. *IEEE Intelligent Systems* **13**, 44–49.

Yang, Z., K. Tang, and X. Yao: 2008, 'Large scale evolutionary optimization using cooperative coevolution'. *Information Sciences* **178**, 2985–2999.

Yu, L. and H. Liu: 2003, 'Feature selection for high-dimensional data: A fast correlation-based filter solution'. In: *The 20th International Conference on Machine Leaning (ICML-03)*. pp. 856–863.

Zhu, X. and X. Wu: 2006, 'Scalable representative instance selection and ranking'. In: *Proceedings of the 18th International Conference on Patter Recognition (ICPR'06)*, Vol. 3. pp. 352 – 355.