Information Sciences 228 (2013) 150–174

Contents lists available at SciVerse ScienceDirect

Information Sciences

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

A scalable approach to simultaneous evolutionary instance and feature selection

Nicolás García-Pedrajas *, Aida de Haro-García, Javier Pérez-Rodríguez

Department of Computing and Numerical Analysis, Edificio Einstein, 3ª Planta, University of Córdoba, Campus de Rabanales, 14071 Córdoba, Spain

ARTICLE INFO

Article history:
Received 23 March 2012
Received in revised form 25 August 2012
Accepted 14 October 2012
Available online 25 October 2012

Keywords:
Simultaneous instance and feature selection
Instance selection
Feature selection
Instance-based learning
Very large problems

ABSTRACT

An enormous amount of information is continually being produced in current research, which poses a challenge for data mining algorithms. Many of the problems in extremely active research areas, such as bioinformatics, security and intrusion detection and text mining, involve large or enormous datasets. These datasets pose serious problems for many data mining algorithms.

One method to address very large datasets is data reduction. Among the most useful data reduction methods is simultaneous instance and feature selection. This method achieves a considerable reduction in the training data while maintaining, or even improving, the performance of the data-mining algorithm. However, it suffers from a high degree of scalability problems, even for medium-sized datasets. In this paper, we propose a new evolutionary simultaneous instance and feature selection algorithm that is scalable to millions of instances and thousands of features.

This proposal is based on the divide-and-conquer principle combined with bookkeeping. The divide-and-conquer principle allows the execution of the algorithm in linear time. Furthermore, the proposed method is easy to implement using a parallel environment and can work without loading the entire dataset into memory.

Using 50 medium-sized datasets, we will demonstrate our method's ability to match the results of state-of-the-art instance and feature selection methods while significantly reducing the time requirements. Using 13 very large datasets, we will demonstrate the scalability of our proposal to millions of instances and thousands of features.

1. Introduction

The overwhelming amount of data that is currently available in any field of research poses new problems for data mining and knowledge discovery methods. This enormous amount of data makes most of the existing algorithms inapplicable to many real-world problems. Two approaches have been utilized to face this problem: scaling up data mining algorithms [53] and data reduction. Nevertheless, scaling up a certain algorithm is not always feasible. Data reduction consists of removing missing, redundant, information-poor data and/or erroneous data from the dataset to obtain a tractable problem size. Data reduction techniques use different approaches; among the most common are feature selection [44], feature-value discretization [34] and instance selection [5].

Instance selection [45] consists of choosing a subset of the total available data to achieve the original purpose of the data-mining application as though all the data were being used. Different variants on instance selection exist. We can distinguish two main models [10]: instance selection as a method for prototype selection for algorithms based on prototypes (such as

* Corresponding author.
E-mail addresses: npedrajas@uco.es (N. García-Pedrajas), adeharo@uco.es (A. de Haro-García), javier.perez@uco.es (J. Pérez-Rodríguez).

0020-0255/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2012.10.006

k-Nearest Neighbors) and instance selection for obtaining the training set for a learning algorithm that uses this training set (such as classification trees or neural networks).

The problem of instance selection for instance-based learning can be defined as [7] "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".

It has been shown that different groups of learning algorithms need different instance selectors in order to suit their learning/search biases [9]. This may render many instance selection algorithms useless if their philosophy of design is not suitable for the problem at hand. Wrapper approaches do not assume any structure of the data or behavior of the classifier, adapting the instance selection to the performance of the classifier. Therefore, they are usually the best-performing methods. However, wrapper approaches are most computationally expensive and may over-fit.

Brighton and Mellish [7] argued that the structure of the classes formed by the instances can be very different and therefore, an instance selection algorithm can have a good performance in one problem and be very inefficient in another. They stated 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 usually unavailable or very difficult to acquire, especially in real-world problems with many variables and complex boundaries between classes. In such a situation, an approach based on evolutionary computation (EC) may be helpful. Approaches based on EC do not assume any particular form of the space, the classes or the boundaries between the classes; they are only guided by each solution's ability 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 classes.

Evolutionary computation has been shown [10,25] to be the most efficient method of instance selection. However, it suffers from scalability problems. The computational cost, even for moderately large datasets [27], is a serious handicap for this approach. EC-based methods are not applicable to very large datasets [17].

Feature selection is one of the most important and frequently used techniques in data preprocessing for data mining [46,14]. In contrast to other dimensionality reduction techniques, feature selection preserves the original semantics of the variables, offering the advantage of interpretability by a domain expert [58].

Feature selection has been a fertile field of research and development since the 1970s in statistical pattern recognition [50,48], machine learning [5,36] and data mining [15]. It has been widely applied to many fields, such as text categorization [42], image retrieval [56] customer relationship management [52], intrusion detection [41] and genomic analysis [64].

Feature selection can be defined as the selection of a subset of M' features from a set of M features, $M' < M$ such that the value of a criterion function is optimized over all the subsets of size M' [51]. The objectives of feature selection are manifold. The most important objectives include the following [58]:

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

As shown above, many algorithms have been developed to tackle either instance or feature selection. However, few papers have aimed at simultaneous instance and feature selection [61]. One of the problems with combining both approaches is the lack of theoretical basis for developing algorithms with strong theoretical backgrounds. In the absence of this theory, we must resort to heuristics methods. An initial straightforward approach is to perform one process after the other. However, this approach does not benefit from the combination of both processes.

Although most proposed methods for both instance and feature selection address one of the problems or the other, but not both, feature and instance selection are closely related. Depending on the subset of instances considered, the relevant features might change. Conversely, different subsets of features might yield different subsets of relevant instances. Fig. 1

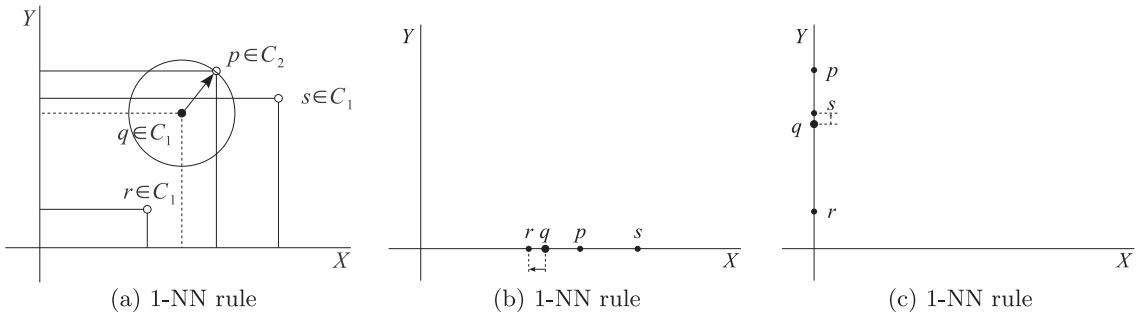

Fig. 1. Relationship between instance and feature selection. We have a test instance, q, and three training instances, p, s and r, belonging to classes 2, 1 and 1, respectively. Using a 1-NN rule, to classify q correctly, we need to select different instances depending on the features selected. If we select feature x, we need to select instance r, and if we select feature y, we need to select instance s. If both features are selected, then q is misclassified.

provides an illustration. The instances or features selected to classify query instance q depend on each other because different instances will be the nearest to the query instance in different subspaces. Thus, jointly searching for a subset of relevant instances and features may benefit the overall accuracy of the obtained subset. Furthermore, because the algorithm can remove instances as well as features, its data reduction capability increases.

Thus, a better approach would be to combine both searches. Evolutionary computation offers a very suitable tool for approaching the simultaneous selection of both instances and features. Using a chromosome that encodes instances and features, any evolutionary computation method can be used. This approach has been used in several studies [38,60,13,54]. Derrac et al. [18] used a cooperative co-evolutive approach with three populations: one performing instance selection, another performing feature selection and a third performing both instance and feature selection.

The problem with all of these previous approaches is scalability. This is a particularly serious problem for simultaneous instance and feature selection because most of the methods are iterative heuristic methods, which are computationally very expensive. Furthermore, instance selection can be made faster using bookkeeping [53]. For instance, for k nearest neighbors rule we can obtain a matrix of the nearest neighbors of every instance before running the algorithm, for support vector machines we can precompute the kernel values of all pairs of instances, and so on. Although these methods are not valid for large datasets due to memory requirements, they can speed up the algorithm in small to medium-sized datasets. However, bookkeeping cannot be used for feature selection, either alone or in combination with instance selection, because the neighbors of an instance or the precomputed kernel values are different in each subspace considered.

In this paper, we propose a new method for simultaneous instance and feature selection with the main goal of developing a scalable approach. Thus, we do not attempt to improve on previous methods, only to match the performance of previous evolutionary approaches in a method that can address very large datasets. As an evolutionary algorithm, our method is a wrapper approach and can, therefore, be used with any classification method. However, to provide the necessary focus, we will also use along the paper nearest neighbor (1-NN) as the classification method. It is important to note that the known methods for making the nearest neighbor search faster, such as k-d trees [4] or bucketing [62], are hardly efficient because the instances are tested in different subspaces for each individual of the population.

The approach is general enough to be used with other classifiers. The only requirement is a bookkeeping method applicable to the classifier which can make the method faster. In the following sections we will detail the application of our framework for nearest neighbor classifier. However, other classifiers may be used as well. As stated, using support vector machines we can precompute the kernel values [47], making the algorithm significantly faster. In fact, this method can be used to speed up the application of string kernels to large and very large datasets, such as are common in bioinformatics [3]. For decision trees bookkeeping has also been used with excellent results [2,53].

Our method is based on two main tools that provide scalability. First, the model's divide-and-conquer philosophy addresses the problem of scalability without compromising its performance. The method is based on applying a selection algorithm to subsets of the whole training set and combining the results obtained from those subsets by means of a voting scheme. Secondly, bookkeeping is used to speed up the selection process on each subset. Without bookkeeping, using even a small subset would be too computationally expensive. The proposal is named *scalable simultaneous instance and feature selection method* (SSIFSM).

This paper is organized as follows: Section 2 discusses some related work; Section 3 presents our proposal; Section 4 describes our experimental setup; Section 5 shows the experimental results and Section 6 provides the conclusions of our work.

2. Related work

The problem of scalability is one of the most relevant in current research. Many of the widely used methods in data mining were developed when the common dataset size was on the order of hundreds or thousands of instances. Now, it is not uncommon to have to deal with datasets with hundreds of thousands or millions of instances and tens of thousands of features. When facing these problems, data mining algorithms become less effective unless they are efficiently scaled up. Very large datasets present a challenge for both humans and machine learning algorithms. As Leavitt notes [40]:

"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 to produce results accurately and quickly. Efficiency and scalability are, indeed, the key issues when designing data mining systems for very large datasets [12]. In data mining, most scalability issues are linked with large datasets, with millions of instances and/or thousands of features. In the context of inductive algorithms, Provost and Kolluri [53] identified three fundamental approaches for scaling up learning methods: (i) designing fast algorithms, (ii) partitioning the data and (iii) using a relational representation. These three approaches are independent and can thus be combined into any data mining algorithm. A refinement of this taxonomy was proposed by García-Pedrajas and de Haro-García [30].

Several methods have been proposed for scaling either instance [32,28] or feature selection [59,57,6]. However, none of these methods is applicable to simultaneous instance and feature selection.

A very efficient approach is the stratification strategy [11]. The stratification strategy splits the training data into disjoint strata with equal class distribution.[1] The training data, T, are divided into t disjoint datasets, D_j, of approximately equal size:

$$T = \bigcup_{j=1}^{t} D_j. \tag{1}$$

Then, the data mining algorithm is applied to each subset separately and the results of all of the subsets are combined for the final solution. If we have an algorithm of quadratic complexity, $O(N^2)$, for a number of instances, N, and we employ M strata, we will have a time complexity $O(N/M)^2$. Because we have to apply the method to the M strata, the resulting complexity is $O(N^2/M)$, which is M times faster than the original algorithm. If we are able to run the algorithm in parallel in all the strata, our complexity will be $O(N^2/M^2)$ with a speedup of M^2. Of course, the drawback of this approach is the likely decrease in the performance of the algorithm. Derrac et al. [19] used stratification to scale up a steady-state memetic algorithm for instance selection. However, stratification is not feasible when feature selection is used, as no combination step is used.

García et al. [24] proposed a memetic algorithm where scaling-up was achieved with and *ad hoc* local search procedure. The method is efficient but not scalable to large and very large datasets.

Brill et al. [8] used sampling to speed up feature selection. Each individual was evaluated using its nearest neighbor error, an operation which is of $O(N^2M)$ time complexity, for N instances and M features. To avoid this complexity, a small random sample of instances was chosen to evaluate the individuals of each generation. They found that the results were competitive with the use of the whole dataset when tested on an artificial problem of 30 variables.

Similar ideas were used by de Haro-García and García-Pedrajas [32] for instance selection. The problem with the above methods is that the performance is degraded for some problems. For the case of instance selection using a genetic algorithm, the authors found that the evolutionary algorithm was too conservative when applied to subsets of the datasets. Thus, many useless instances where retained [32]. This problem was addressed with a recursive application of the stratified approach discussed above. After a first application of the stratification and the evolutionary algorithm to the different strata, a new round of stratification was applied with the selected values. This recursive approach was applied until a certain criterion was met. Although this method was very successful for instance selection, its generalization to other mining algorithms is troublesome.

One step forward in sampling techniques was developed for instance and feature selection by García-Osorio et al. [27,31]. This method combines the divide-and-conquer approach of sampling with the combination principle of the ensembles of classifiers. The method is composed of three basic steps: (i) divide the problem data into small disjoint subsets, (ii) apply the learning algorithm of interest to every subset separately and (iii) combine the results of the different applications. The first two steps are repeated several times. As the learning method is always applied to small datasets, the speed of the methodology is impressive. As the method combines results of the application of the same learning method to different subsets of the available dataset, it is called *democratization* of algorithms.

Democratization scaling method 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 of these subsets and a result $C_{i,j}$ is produced from each subset of the data. After all of the n_s rounds are applied (which can be done in parallel, as all rounds are independent from each other), the combination method constructs the final result C as the output of the data mining process.

However, none of these previous approaches is applicable to simultaneous instance and feature selection.

3. Scalable simultaneous instance and feature selection method (SSIFSM)

Consider a problem involving K classes and N training instances with M features whose class membership is known. Let $T = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}$ be the set of N training samples, where each instance $\mathbf{x}_i$ belongs to a domain X. Each label is an integer from the set $Y = \{1, \ldots, K\}$. The task is to find a subset of $N' \ll N$ instances and $M' < M$ features that is able to perform the classification task with as much accuracy as T. As previously mentioned, the problem with traditional approaches to simultaneous instance and feature selection is that they have serious scalability problems, especially for methods based on evolutionary computation that have been shown to achieve the best performance [26].

Bookkeeping [53,2] is an efficient method for scaling learning algorithms. Its main insight is that matching hypotheses against all of the data is not necessary. For most of the processing, the information from which the results of the matching can be inferred is sufficient. Separating the generation of the information from its use in the evaluation of hypotheses allows each to be treated separately—first using the data to populate the data structure and then operating only on the data structure, which affords both the optimized use of memory and an improved run-time complexity.

More specifically, for the evaluation of the fitness of an individual representing a certain selection of instances and features is enough if we know the neighbors of every instance. The actual values of the features of the instances or the actual distances among them are not needed. In this way, we can construct a matrix storing the ordered list of neighbors of each instance and use that matrix to obtain the fitness. This approach has the problem of scalability; it can be applied for instance selection in small and medium datasets, but for large datasets the size of the matrix, N^2, is prohibitive. For instance and feature selection the situation is even worse because the neighbor matrix must be available for every subspace. The amount of memory needed would be $N^2 2^{M-1}$, which makes the use of bookkeeping unfeasible.

[1] When dealing with class-imbalance problems the distribution of classes in each strata may be modified with respect to the whole training set to obtain a less skewed distribution.

On the other hand, a divide-and-conquer philosophy based on applying the selection process to subsets of the entire dataset has been proven useful for instance selection [11,24,27]. This philosophy may be extended to simultaneous instance and feature selection. Therefore, our methodology is also based on the divide-and-conquer approach. Instead of applying the selection method to the whole dataset, we perform a random partition of the instances and features and apply the selection to each of the subsets obtained. However, this method introduces significant randomness each time the selection process is applied as the algorithm sees a very partial view of the dataset. To avoid the negative effect of this partial view, this partition is repeated for several rounds and the results are combined through a voting process. This process shares the same philosophy of ensembles of classifiers [39] of combining many weak learners into a strong one. The whole process is illustrated in Fig. 2 for a dataset of 30 instances and nine features divided into subsets of 10 instances and three features.

Furthermore, this divide-and-conquer strategy allows resorting to bookkeeping as a scaling-up method. With our approach, the selection process is always applied to small datasets. The results of our experiments will show that values between 100 and 1000 instances are enough to produce good results. If we want to construct a cache of neighbors to be used in the evolutionary algorithm, we need to obtain the list of neighbors for each instance and each possible subspace. For a subset of n instances and m features, considering that we can store the list of neighbors using a short integer of two bytes, the total amount of memory, S, needed to store the cache would be as follows:

$$S = 2 \cdot n^2 2^{m-1} \text{ bytes}. \tag{2}$$

Fig. 3 shows the memory size needed to store such a cache of neighbors for different dataset sizes. The figure demonstrates that the memory size grows very quickly, which means that we must use small subsets. For a maximum memory size of approximately 300–400 MB, small enough to be dealt with by any computer, we have a range from 100 instances and 13 features to 1000 instances and seven features. We have explored the values in this range and the results presented in this paper show that these values achieved a very good performance.

Our process is as follows: First, the training data, T, is randomly divided into t disjoint subsets, D_j, of approximately equal size. Each has n instances, as follows:

$$T = \bigcup_{j=1}^{t} D_j. \tag{3}$$

Each subset is further divided into s random subsets, D_{ji}, with m features, as follows:

$$D_j = \bigcup_{i=1}^{s} D_{ji}. \tag{4}$$

In this way, each instance is present in s subsets and each feature is present in t subsets. We then perform an evolutionary algorithm for simultaneously selecting instances and features on each D_{ji} subset. Bookkeeping is introduced in this stage. The only information the instance selection algorithm needs to perform its task is the information about the nearest neighbors of each instance. Thus, instead of working with the instances we build a matrix of neighbors for each instance and for every subspace. This matrix is of size $n \times n$, where for each instance we store the list of neighbors from the nearest to the furthest. Each time we need the nearest neighbor of an instance we just need to get the corresponding value from the nearest neighbor matrix. As the size of this matrix is $n \times n$ it can only be used with the partition scheme we have developed. Such a matrix for the original whole dataset would not fit in memory even for medium size problems. Fig. 2 shows the partition of the dataset, Fig. 4 shows how bookkeeping method is applied to each dataset.

We have chosen a CHC algorithm as the evolutionary algorithm [21]. The CHC algorithm was used because Cano et al. [10] compared several evolutionary algorithms for the purpose of instance selection and CHC was found to be the best alternative. Because simultaneous instance and feature selection is a similar problem to that examined by Cano et al., we believe CHC is a reasonable choice. However, our proposal does not depend on this decision and another method could be used. CHC stands for *Cross-generational elitist selection, Heterogeneous recombination and Cataclysmic mutation*. The CHC genetic algorithm differs from traditional GAs in a number of ways:

1. To obtain the next generation for a population of size P, the parents and the offspring are put together and the P best individuals are selected.
2. To avoid premature convergence, only different individuals separated by a threshold Hamming distance—in our implementation, the length of the chromosome divided by four—are allowed to mate.
3. During crossover, two parents exchange exactly half of their nonmatching bits. This operator is referred to as *Half Uniform Crossover* (HUX) [21].
4. Mutation is not used during the regular evolution. To avoid premature convergence or stagnation of the search, the population is reinitialized when the individuals are not diverse. In such a case, only the best individual is retained in the new population.

The implementation of the genetic algorithm uses the most natural representation for each individual. The chromosome of each individual has as many bits as instances plus features. A bit with a value of 1 means that the corresponding instance or feature is selected and a value of 0 means that the corresponding instance is not selected. The fitness measure of an individual $x, f(x)$, is provided as follows:

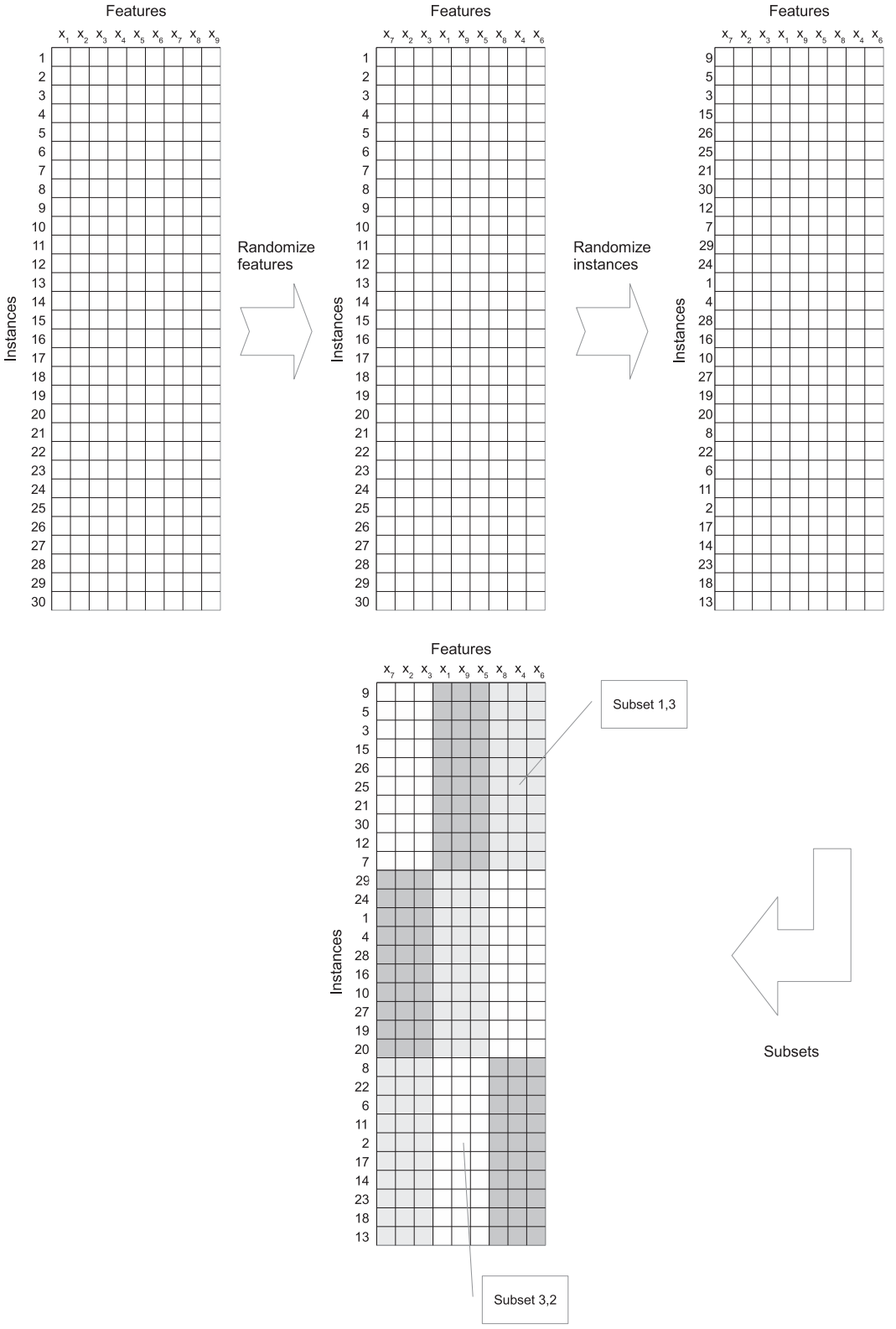

Fig. 2. Example of the partition of a dataset with 30 instances and nine features divided into subsets of 10 instances and three features.

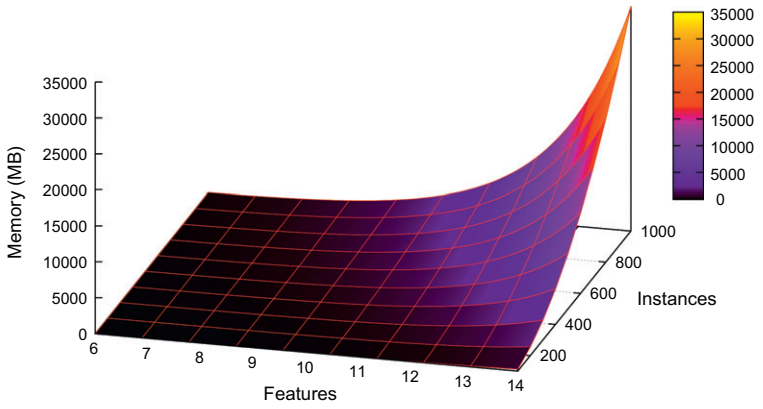

Fig. 3. Size, in megabytes, of the cache for the neighbors for different sizes of the subsets.

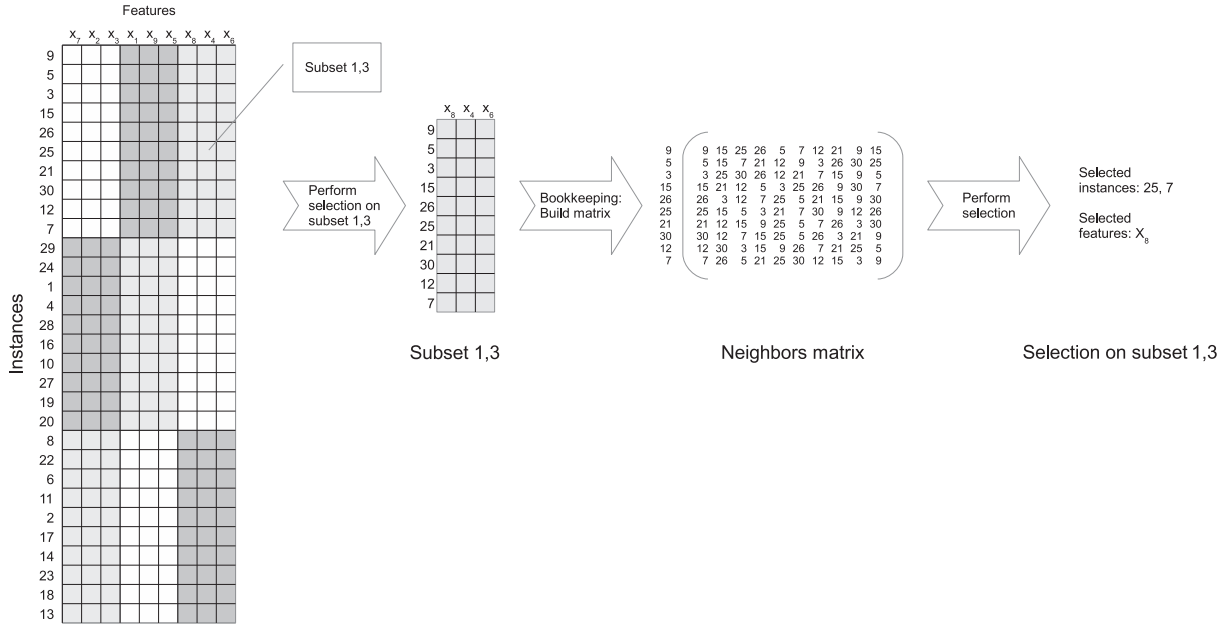

Fig. 4. Example of the bookkeeping process for a subset. Neighbor matrix is obtained calculating for each instance the distance to all the remaining instances and sorting these distances using a `quicksort` algorithm.

$$f(x) = \alpha acc(x) + (1-\alpha) red(x), \tag{5}$$

where $acc(x)$ is the accuracy and $red(x)$ the reduction. The accuracy of an individual is measured using a 1-NN classifier and the instances and features selected by the individual. The reduction for an individual that selects N' instances and M' features is $1 - \frac{N'}{N}\frac{M'}{M}$. α Is a weighting parameter, which is fixed in our experiments to $\alpha = 0.5$, as it is common in the literature [10,24]. The results of the selection performed on every subset are recorded. After applying the instance and feature selection algorithm, we record the number of times that each instance and feature has been selected to be kept. We call this record the number of votes due to its similarity to the combination of classifiers in an ensemble by voting [39]. This process is repeated for r rounds with different random partitions of the dataset. In some cases, especially in class-imbalanced datasets, all the instances of a certain subset belong to the same class. In such a case, that subset is ignored and the next one is used.

The final step is the combination of the different rounds. To obtain a meaningful combination, we use the philosophy of ensembles of classifiers. In an ensemble, several *weak* learners are combined to form a strong classifier; in our method, several *weak* (in the sense that they are applied to subsets of the training data) instance and feature selection procedures are combined to produce a strong and fast selection method. Each round can be considered 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 [29].

Once we have performed the r rounds, we have recorded the number of votes received by each instance and feature. The number of votes received by an instance is in the interval $[0, r \cdot s]$, as an instance is in s subsets in every round. Each feature is in t subsets each round. Thus, the number of votes is in the interval $[0, r \cdot t]$. The final combination of votes must set a

threshold to decide whether an instance or a feature must be selected as the final output of the process. A first natural choice would be majority voting: instances and features are kept if they receive at least half of the possible votes. However, the performance of this fixed threshold depends heavily on the problem. Therefore, we developed a method of automatically selecting an optimum vote threshold.

To this end, we define an evaluation function, J, that measures the goodness of a certain subset of instances and features, $T(\theta_i, \theta_f)$. $T(\theta_i, \theta_f)$ is the subset of the training set T obtained selecting the instances, in which the number of votes is above or equal to θ_i and the features which number of votes is above or equal to θ_f. $J(T(\theta_i, \theta_f))$ is calculated as follows:

$$J(T(\theta_i, \theta_f)) = \beta acc(T(\theta_i, \theta_f)) + (1 - \beta) red(T(\theta_i, \theta_f)), \tag{6}$$

where $red(T(\theta_i, \theta_f))$ is the reduction achieved using threshold θ_i and θ_f to select $T(\theta_i, \theta_f)$ and $acc(T(\theta_i, \theta_f))$ is the accuracy achieved with this selection using a 1-NN classifier.[2] The values of accuracy and reduction can be measured as in Eq. (5), or using other method, because $f(x)$ and $J(T(\theta_i, \theta_f))$ are used in different stages of the algorithm. In fact, as we will see, reduction is measured differently in both equations.

To obtain the best pair of thresholds, all the possible values are evaluated and the optimum is chosen. β is used to balance the weight of accuracy and reduction in the evaluation of the thresholds of votes. The votes are obtained using an evolutionary instance and feature selection algorithm in every subset. Because of the efficiency of this algorithm in removing instances and features, the votes are biased to remove too much information. In this way, β must be chosen closer to 1 to counteract this effect. Otherwise, the algorithm tends to remove too many instances and features and obtains poor accuracy. Thus, we used $\beta = 0.75$ to avoid a large reduction at the expense of poor accuracy.

In the experiments we discovered a flaw in the evaluation function. If we measure the reduction as the actual reduction in the dataset, we have for s_i selected instances and s_f selected features a reduction of $1 - (s_i/N)(s_f/M)$. That means that reducing a feature usually has a greater impact on the reduction value because $N > M$ or even $N \gg M$. The observed behavior is that too many features are removed and the performance of our method is poor in terms of accuracy. To avoid this effect, we modified the way the reduction is calculated. For s_i selected instances and s_f selected features, the reduction in Eq. (6) is measured as $1 - (s_i + s_f)/(N + M)$.

For every pair of thresholds, we evaluate $J(T(\theta_i, \theta_f))$ and select the pair of thresholds with the highest J. This means that we must evaluate all possible pairs of instance and feature thresholds: $[r \cdot s] \times [r \cdot t]$. The evaluation of this number of thresholds might exclude the scalability achieved by the divide-and-conquer approach. To avoid this negative effect, the evaluation of a pair of thresholds is also approached using a divide-and-conquer method.[3] Instead of evaluating the accuracy of $T(\theta_i, \theta_f)$ with the whole dataset, which is of the complexity $O(N^2M)$, we apply the same partition philosophy used in the previous step. The training set is divided into random disjoint subsets and the accuracy is estimated separately in each subset using the average evaluation of all the subsets for the fitness of each pair of thresholds. For the evaluation, the partition is only performed by instances, as partition by features would make no sense. The whole SSIFSM procedure is shown in Algorithm 1.

Algorithm 1. Scalable simultaneous instance and feature selection method (SSIFSM).

Data : A training set $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, number of instance subsets t, number of feature subsets s and number of rounds r.
Result: The set of selected instances and features $T_S \subset T$.
for $i = 1$ **to** r **do**
1 Divide instances into t disjoint subsets $D_i : \bigcup_i D_i = T$ of approximately the same size n
for $j = 1$ **to** t **do**
2 Divide D_j into s disjoint subsets D_{jk} of features with approximately the same number of features m
for $k = 1$ **to** s **do**
3 Apply instance selection algorithm to D_{jk}
4 Store votes of selected instances from D_{jk}
end
end
end
5 Obtain thresholds of votes to keep an instance, θ_i, and a feature, θ_f
6 $T_S = \{x_i \in T | (\text{votes}(x_i) \geq \theta_i \text{ and which features } j | \text{votes}(j) \geq \theta_f\}$
7 **return** T_S

[2] Any other classifier can be used in this function to obtain the accuracy of the subset selected.
[3] This method is only applied to accuracy in Eq. (6) because is where we have the scalability problem. Eq. (5) is the standard fitness measure in evolutionary instance and feature selection and it is only used for the evolution of small subsets.

The process for evaluating a pair of thresholds using the divide-and-conquer procedure is shown in Algorithm 2.

Algorithm 2. Divide-and-conquer method for evaluating pairs of thresholds.

Data : A training set $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, number of instance subsets t and pair of thresholds θ_i and θ_f for instances and features, respectively.

Result: The evaluation of the goodness of the pair of thresholds.

1 Calculate the reduction, $red(T(\theta_i, \theta_f))$, obtained using thresholds θ_i and θ_f to select instances and features

2 Divide instances into t disjoint subsets $D_i : \bigcup_i D_i = T$ of approximately the same size

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

  Obtain selection, S_j, in D_j, $S_j \subset D_j$, using thresholds θ_i and θ_f

  Obtain, acc_j, the accuracy in D_j using S_j

end

3 $acc(T(\theta_i, \theta_f)) = 1/t \sum_{j=1}^{t} acc_j$

4 **return** $J(T(\theta_i, \theta_f)) = \beta acc(T(\theta_i, \theta_f)) + (1 - \beta) red(T(\theta_i, \theta_f))$

The scalability of the method is assured by the following features:

1. Application of the method to small datasets. Due to the small size of the subsets in which the selection process is applied, the selection process will always be fast, regardless of the complexity of the instance and feature selection method used in the subsets.
2. Only small datasets must be kept in memory. This allows the application of instance and feature selection when datasets do not fit into memory.
3. Bookkeeping is applied in the evolution for every subset.

3.1. Complexity of our methodology

The aim of this work is to obtain an instance and feature selection methodology that is able to scale up to large and even huge problems. Thus, an analysis of the complexity of the method is necessary. 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 n, which are subsequently divided into subsets of m features. The process of random partition is of $O(NM)$ time complexity. The instance and feature selection algorithm is always applied to a subset of fixed size, n, which is independent from the actual size of the dataset. The complexity of this application of the algorithm depends on the complexity of the CHC selection algorithm that we are using, but it will always be small because the size n is always small. Let K be the number of operations required by the selection algorithm to perform its task in a dataset of size n. For a dataset of N instances and M features, we must perform this selection process once for each subset, $\frac{N}{n}\frac{M}{m}$ times, expending a time proportional to $(\frac{N}{n}\frac{M}{m})K$. The total time needed by the algorithm to perform r rounds will be proportional to $r(\frac{N}{n}\frac{M}{m})K$, which is linear in the number of instances, as K is a constant value. The same study applies to the evaluation of the thresholds.

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

3.2. Parallel implementation

In the experimental section, we will also show the execution time results using a parallel implementation of our algorithm. The parallel implementation is based on the master/slave architecture. The master performs the partition of the dataset and sends the subsets to each slave. Each slave performs the selection algorithm using only the instances and features of its subset and then returns the selected instances and features to the master. The master stores the votes for each kept instance and feature. The general architecture of the system is shown in Fig. 5. This method has the advantage of being applicable to very large datasets, as only a small part of the dataset must be kept in memory.

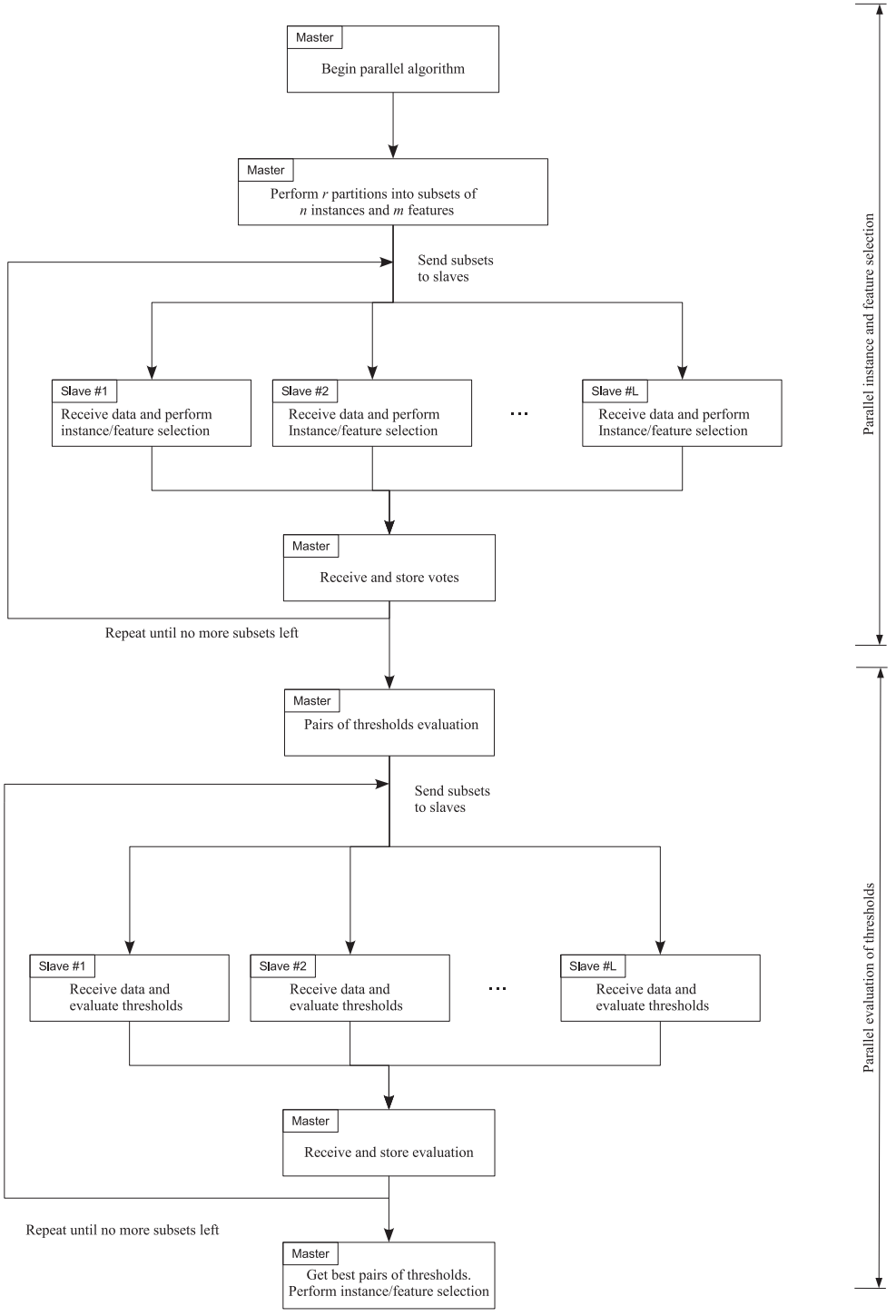

Fig. 5. Parallel implementation SSIFSM.

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

1. Before each slave initiates its selection process, it must receive the subset of data to perform that process. This amount of information is always small because the method is based on each slave taking care of only a small part of the whole dataset. Furthermore, if the slaves can access the disk, they can read the necessary data directly from it.
2. Once the selection process is finished, the slaves send the selection performed to the master. This selection consists of a list of the selected instances and features, 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. 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 pair of thresholds, it must receive the subset of data to perform that task. This amount of information is always small because the method is based on each slave taking care of only a small part of the whole dataset.
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 pair of thresholds is used, which is a real number.

In both cases, the amount of data exchanged between master and slaves is not large, avoiding a bottleneck in the algorithm. First, the algorithm sends a subset to every slave. It then waits for the first slave to finish, obtains the results and sends the finished slave a new subset until all the subsets have been processed. The same procedure is performed to evaluate the best pair of thresholds.

4. Experimental setup

We used a set of 50 problems to test the performance of the proposed method, which is shown in Table 1 in the Appendix. The datasets `barleyblumeria` and `cottonmeloidogyne` represent problems from the field of Bioinformatics; the remaining datasets are from the UCI Machine Learning Repository [22]. To estimate the reduction and accuracy, we used a 10-fold cross-validation method and a 1-nearest neighbor (1-NN) classifier. This first set of problems represents datasets from small to medium sizes. In Section 5.2, we address very large problems.

4.1. Algorithms for the comparison

To allow a fair assessment of the validity of our proposal, we must compare our approach with the state-of-the-art methods for instance selection, feature selection and simultaneous instance and feature selection. We have chosen the following methods with the name used in the text and the tables shown between parentheses:

1. Nearest neighbor error using all instances and features is chosen as a baseline measure (1-NN). Any method of performing selection of features or instances must at least match the performance of the 1-NN algorithm or improve its accuracy if possible.
2. As stated, Cano et al. [10] performed a comprehensive comparison of the performances of different evolutionary algorithms for instance selection. They compared a generational genetic algorithm, a steady-state genetic algorithm, a CHC genetic algorithm and a population-based incremental learning algorithm. Among these methods, CHC achieved the best overall performance (ISCHC). Therefore, we have included the CHC algorithm in our comparison.
3. Feature selection using a genetic algorithm (FSCHC). Following the same idea used for instance selection, we used a CHC algorithm for feature selection with the same characteristics of the algorithm used for instance selection.
4. Instance and feature selection using a genetic algorithm (IS + FSCHC). Combining the two previous methods, we also used a CHC algorithm that simultaneously evolved the instances and features.
5. Intelligent Multiobjective Evolutionary Algorithm (IMOEA) [13] method. This method is a multi-objective evolutionary algorithm, which considers both instance and feature selection. The algorithm has two objectives, maximization of training accuracy and minimization of the number of instances and features selected. The multi-objective algorithm used is based on Pareto dominance because the approach is common in multi-objective algorithms [65]. The fitness of each individual is the difference between the individuals it dominates and the individuals that dominate it. The algorithm also includes a new crossover operator called *intelligent crossover*, which incorporates the systematic reasoning ability of orthogonal experimental design [43] to estimate the contribution of each gene to the fitness of the individuals.
6. The major aim of our method is scalability. However, if scalability can be achieved with a simple random sampling method, it may be argued that our method is superfluous. Thus, our last method for comparison is a CHC algorithm for instance and feature selection, which uses a random sampling of instances (SAMPLING). The method is applied using a random 10% of all the instances in the dataset.

The source code used for all methods is in *C* and is licensed under the GNU General Public License. The code and the partitions of the datasets are freely available upon request from the authors.

Table 1
Summary of datasets. The attributes of each dataset can be *C* (continuous), *B* (binary) or *N* (nominal). Features represent the number of variables once nominal attributes have been converted into integer values using a 1-out-of-N codification.

	Dataset	Cases	Attributes			Features	Classes
			C	*B*	*N*		
1	Abalone	4177	7	–	1	10	29
2	Ads	3279	–	–	–	1558	2
3	Adult	48,842	6	1	7	105	2
4	Anneal	898	6	14	18	59	5
5	Arrhythmia	452	279	–	–	279	13
6	Barleyblumeria	4340	64	–	–	64	2
7	Basehock	1993	4862	–	–	4862	2
8	Car	1728	–	–	6	16	4
9	Card	690	6	4	5	51	2
10	Cottonmeloidogyne	4156	64	–	–	64	2
11	Dexter	600	20,000	–	–	20,000	2
12	Euthyroid	3163	7	18	–	44	2
13	Gene	3175	–	–	60	120	3
14	German	1000	6	3	11	61	2
15	Gina	3468	970	–	–	970	2
16	Gisette	7000	5000	–	–	5000	2
17	Hiva	4229	1617	–	–	1617	2
18	Hypothyroid	3772	7	20	2	29	4
19	Isolet	7797	617	–	–	617	26
20	Krvskp	3196	–	34	2	38	2
21	Letter	20,000	16	–	–	16	26
22	Lrs	531	101	–	–	101	10
23	Madelon	2,600	500	–	–	500	2
24	Magic04	19,020	10	–	–	10	2
25	Mfeat-fac	2000	216	–	–	216	10
26	Mfeat-fou	2000	76	–	–	76	10
27	Mfeat-kar	2000	64	–	–	64	10
28	Mfeat-mor	2000	6	–	–	6	10
29	Mfeat-pix	2000	240	–	–	240	10
30	Mfeat-zer	2000	47	–	–	47	10
31	Mushroom	8124	–	6	16	117	2
32	Musk	6598	166	–	–	166	2
33	Nursery	12,960	–	1	7	23	5
34	Optdigits	5620	64	–	–	64	10
35	Ozone1hr	2536	72	–	–	72	2
36	Ozone8hr	2534	72	–	–	72	2
37	Page-blocks	5473	10	–	–	10	5
38	Pcmac	1943	–	–	–	3,289	2
39	Pendigits	10,992	16	–	–	16	10
40	Phoneme	5404	5	–	–	5	2
41	Satimage	6435	36	–	–	36	6
42	Segment	2310	19	–	–	19	7
43	Shuttle	58,000	9	–	–	9	7
44	Sick	3772	7	20	2	33	2
45	Soybean	683	–	16	19	82	19
46	Texture	5500	40	–	–	40	11
47	Titanic	2201	–	–	3	8	2
48	Waveform	5000	40	–	–	40	3
49	Yeast	1484	8	–	–	8	10
50	Zip	9298	256	–	–	256	10

All the experiments were carried out in a cluster of 32 blades. Each blade is a biprocessor DELL Power Edge M600 with four cores per processor. Thus, we count 256 cores. The blades are interconnected with a master node and between them with a 1 GB network. The processors run at 2.5 GHz and each blade has 16 GB of memory.

4.2. Statistical tests

For comparing groups of methods, we first carry out an Iman-Davenport test to ascertain whether there are significant differences among the methods. The Iman-Davenport test is based on the χ_F^2 Friedman test [23], which compares the average ranks of k algorithms, but it is more powerful than the Friedman test. Iman and Davenport [35] found Friedman's statistic to be too conservative and developed a better one, $F_F = \frac{(N-1)\chi_F^2}{N(k-1)-\chi_F^2}$, which is distributed following a F distribution with $k-1$ and

$(k-1)(N-1)$ degrees of freedom. After carrying out the Iman-Davenport test, we can perform pairwise comparisons such as the Wilcoxon test.

The Wilcoxon [63] test is a statistical test for comparing pairs of algorithms. Empirical results [16] show that it is stronger than other tests. However, when all algorithms are compared with a control method, it is not advisable to perform many Wilcoxon tests against the control method. Alternatively, we can use one of the general procedures for controlling the family-wise error in multiple hypothesis testing.

The test statistic for comparing the ith and jth classifier using these methods is $z = \frac{(R_i - R_j)}{\sqrt{k(k+1)/6N}}$. The z value is used to find the corresponding probability from the table of normal distribution, which is then compared with the appropriate α. The tests differ in the ways that they adjust the value of α to compensate for multiple comparisons.

The Bonferroni–Dunn test [20] controls the family-wise error rate by dividing α by the number of comparisons made ($k-1$, in our case). In contrast to the single-step Bonferroni–Dunn procedure, step-up and step-down procedures sequentially test the hypotheses ordered by their significance. We will denote the ordered p values by $p_1, p_2, \ldots$, so that $p_1 \leqslant p_2 \leqslant \ldots \leqslant p_{k-1}$. One of the simplest such methods was developed by Holm [33]. It compares each p_i with $\alpha/(k-i)$. Holm's step-down procedure starts with the most significant p value. If p_1 is below $\alpha/(k-1)$, the corresponding hypothesis is rejected and we are allowed to compare p_2 with $\alpha/(k-2)$. If the second hypothesis is rejected, the test proceeds with the third and so on. As soon as a certain null hypothesis cannot be rejected, all of the remaining hypotheses are retained as well. Holm's procedure is more powerful than Bonferroni–Dunn's and makes no additional assumptions about the hypotheses tested.

4.3. Evaluation measures

We will use accuracy and reduction as the main comparison measures. The accuracy is measured as the percentage of instances classified correctly and the reduction as the percentage of total data removed during the evolution. However, some of the large datasets used (see Table 2) are class-imbalanced problems. Accuracy is not a useful measure for imbalanced data. Given the number of true positives (TPs), false positives (FPs), true negatives (TNs) and false negatives (FNs), we can define several measures. Perhaps the most common are the sensitivity (Sn), $\mathrm{Sn} = \frac{\mathrm{TP}}{\mathrm{TP+FN}}$, and the specificity (Sp), $\mathrm{Sp} = \frac{TN}{\mathrm{TN+FP}}$.

From these basic measures, others have been proposed, if concern exists about the performance on both negative and positive classes, the $G-mean$ measure [37] can be used: $G\text{-mean} = \sqrt{Sp \cdot Sn}$. This measure is used as the accuracy measure in all of our experiments when class-imbalanced datasets are involved. Therefore, in the tables reporting our results for very large class-imbalanced datasets, we will show the G-mean as the accuracy measure. For an algorithm selecting n instances and m features, the reduction is measured as $1 - \frac{n}{N}\frac{m}{M}$.

5. Experimental results

Our first experiment aimed to study the behavior of SSIFSM regarding the size of the subsets used. We consider three different subset sizes: 1000 instances and seven features, 500 instances and nine features and 100 instances and 13 features. We represent these sizes using (1000,7), (500,9) and (100,13), respectively. These sizes were chosen from among the possible choices for which the cache of neighbors described above would be approximately 300–400 MB, which is a reasonable value for any computer. Larger subsets would very likely produce memory thrashing.

Table 3, shows the results of SSIFSM with these three configurations. For the representative configuration (500,9), sequential implementation run time is also shown. A comparison of these three configurations is shown in Table 4. The

Table 2
Summary of very large datasets. The attributes of each dataset can be C (continuous), B (binary) or N (nominal). Features represent the number of variables once nominal attributes have been converted into integer values using a 1-out-of-N codification. For the class-imbalanced dataset we also show the imbalance ratio (IR).

	Dataset	Instances	Attributes			Features	Classes	Memory (GB)	IR
			C	B	N				
1	Arabidopsis	33,971	–	–	1004	4016	2	0.51	1:123
2	Ccds	364,495	–	–	1004	4016	2	5.45	1:25
3	Census	299,285	7	–	30	409	2	0.46	1:15
4	Chrom19	1,700,517	5	–	–	5	2	0.03	1:1061
5	Chrom21	1,267,701	–	–	403	1,612	2	7.61	1:4912
6	Covtype	581,012	54	–	–	54	7	0.12	–
7	Dna	50,000,000	–	–	200	800	2	149.01	1:344
8	Kddcup98	95,412	–	–	–	23,549	2	8.37	1:19
9	Kddcup99	4,898,431	33	4	3	122	23	2.23	–
10	Poker	1,025,103	5	–	5	25	10	0.10	–
11	Reuters21578	8293	18,933	–	–	18,933	65	0.58	–
12	Rcv1	534,135	47,236	–	–	47,236	53	93.99	–
13	Ustilago	614,211	–	–	1003	4012	2	9.18	1:93

Table 3
Results of SSIFSM for three different sizes of the subsets of instances and features. For the representative configuration (500,9), sequential implementation run time is also shown.

Dataset	(100,13)			(500,9)				(1000,7)		
	Reduct.	Acc.	Time (s)	Reduct.	Acc.	Time (s)	Time seq. (s)	Reduct.	Acc.	Time (s)
Abalone	0.9949	0.2537	32.8	0.9891	0.2552	11.1	204.3	0.9966	0.2516	10.5
Ads	0.9973	0.9536	309.2	0.9882	0.9573	409.0	86237.6	0.9808	0.9579	419.1
Adult	0.9977	0.8208	1572.0	0.9979	0.8264	709.0	92012.0	0.9986	0.8047	491.0
Anneal	0.9629	0.9348	95.1	0.9498	0.9708	10.9	770.3	0.9490	0.9506	34.3
Arrhythmia	0.9708	0.6689	18.7	0.9828	0.6822	24.4	2447.5	0.9846	0.6956	13.4
Barley	0.9716	0.8383	352.9	0.9714	0.7975	52.5	4748.2	0.9839	0.8170	105.1
Basehock	0.9968	0.8570	400.5	0.9922	0.8043	627.0	153217.5	0.9898	0.7968	760.2
Car	0.9738	0.7616	7.4	0.9542	0.7685	8.3	370.0	0.9574	0.7923	32.5
Card	0.9987	0.8519	544.9	0.9975	0.8522	5.3	400.2	0.9919	0.8363	16.3
Cotton	0.9968	0.6477	142.9	0.9810	0.6107	28.1	4017.5	0.9855	0.6304	56.9
Dexter	0.9993	0.8383	266.7	0.9986	0.7984	515.7	114488.2	0.9924	0.6134	545.8
Euthyroid	0.9982	0.9532	72.2	0.9943	0.9639	36.5	2740.2	0.9980	0.9478	70.0
Gene	0.9887	0.8382	94.7	0.9928	0.8133	46.5	7424.7	0.9966	0.7713	70.6
German	0.9933	0.6900	39.0	0.9917	0.7040	14.1	1115.7	0.9952	0.7110	42.4
Gina	0.9884	0.8645	237.4	0.9818	0.8878	331.0	85110.2	0.9757	0.9086	292.0
Gisette	0.9939	0.9350	3357.8	0.9950	0.9300	3355.5	361722.9	0.9937	0.9341	3098.9
Hiva	0.9998	0.9624	275.0	0.9991	0.9629	700.2	150792.6	0.9998	0.9641	760.8
Hypothyroid	0.9971	0.9666	122.2	0.9989	0.9568	30.9	1648.2	0.9989	0.9618	36.9
Isolet	0.8720	0.8501	448.0	0.8742	0.8488	540.3	102332.8	0.8423	0.8488	466.8
Krvskp	0.9911	0.9351	64.6	0.9805	0.9220	27.6	1489.1	0.9793	0.9072	57.0
Letter	0.7608	0.8220	424.7	0.8130	0.8428	151.4	5978.9	0.8139	0.8761	186.2
Lrs	0.9819	0.8528	8.8	0.9646	0.8774	16.8	1002.6	0.9739	0.8623	14.4
Madelon	0.9943	0.8200	91.0	0.9927	0.8339	142.7	28707.1	0.9954	0.7866	131.2
Magic04	0.9197	0.7944	302.1	0.9454	0.8150	79.0	1008.7	0.9316	0.8276	46.3
Mfeat-fac	0.9558	0.9260	40.8	0.9258	0.9435	54.7	9813.3	0.9399	0.9485	87.3
Mfeat-fou	0.9703	0.7855	20.7	0.9573	0.7945	24.2	2724.6	0.9584	0.7940	53.3
Mfeat-kar	0.9317	0.9190	20.1	0.9339	0.9205	16.1	2047.4	0.9385	0.9175	48.2
Mfeat-mor	0.9133	0.6850	6.3	0.9212	0.6885	2.2	42.8	0.8716	0.6955	3.0
Mfeat-pix	0.9384	0.9190	40.3	0.9384	0.9245	57.1	10484.6	0.9243	0.9315	81.1
Mfeat-zer	0.9194	0.7675	21.1	0.8973	0.7845	16.7	1443.2	0.9090	0.8055	57.3
Mushroom	0.9771	0.9664	221.3	0.9840	0.9665	95.7	16790.5	0.9912	0.9802	133.5
Musk	0.9710	0.9385	83.9	0.9862	0.9391	124.9	24368.4	0.9806	0.9198	182.0
Nursery	0.9722	0.7687	263.5	0.9653	0.7881	76.3	3576.8	0.9530	0.7958	90.5
Optdigits	0.8881	0.9381	50.0	0.9379	0.9365	46.5	5762.1	0.9358	0.9409	79.6
Ozone1hr	0.9931	0.9664	16.0	0.9966	0.9652	29.8	3809.1	0.9971	0.9640	44.3
Ozone8hr	0.9873	0.9170	20.6	0.9858	0.9115	36.9	3944.6	0.9972	0.9099	53.2
Page-blocks	0.8990	0.9554	57.1	0.9270	0.9545	22.9	429.4	0.8841	0.9618	30.1
Pcmac	0.9991	0.5826	236.0	0.9988	0.5564	492.5	102648.0	1.0000	0.5836	504.0
Pendigits	0.8772	0.9355	213.2	0.8224	0.9777	66.0	3026.4	0.7815	0.9844	96.6
Phoneme	0.8236	0.8369	41.0	0.8004	0.8359	12.9	131.7	0.8126	0.8346	10.6
Satimage	0.9043	0.8680	92.3	0.9344	0.8826	61.5	5615.3	0.9261	0.8876	79.2
Segment	0.9296	0.9377	19.8	0.8824	0.9489	9.7	369.9	0.9191	0.9606	61.1
Shuttle	0.9511	0.9957	1108.7	0.9690	0.9971	275.7	2852.6	0.9809	0.9972	178.3
Sick	0.9988	0.9650	41.9	0.9978	0.9597	30.0	2295.9	0.9987	0.9578	85.5
Soybean	0.8595	0.8574	4.0	0.8522	0.8971	7.1	571.2	0.8048	0.9221	16.9
Texture	0.9457	0.9218	43.4	0.8367	0.9635	47.8	4004.1	0.8460	0.9718	118.7
Titanic	0.9900	0.7751	9.5	0.9847	0.7660	5.5	93.7	0.9748	0.7665	11.7
Waveform	0.9837	0.8260	83.6	0.9821	0.8106	38.0	3325.6	0.9773	0.8162	64.6
Yeast	0.9231	0.5392	4.9	0.9321	0.5000	2.3	44.7	0.9345	0.5095	2.9
Zip	0.9990	0.5163	205.6	0.9996	0.5052	272.3	54277.0	0.9991	0.4932	307.2
Average	0.9568	0.8344	244.9	0.9535	0.8360	196.0	29489.60	0.9508	0.8341	204.8

Iman–Davenport test showed no significant differences among the three configurations for both accuracy and reduction. The results in terms of reduction and accuracy are very similar for the three configurations and all the datasets. Regarding the execution time, although the average values are very similar, there are remarkable differences, depending on the datasets. For datasets with few instances and many inputs, the configuration (100,13) is faster. On the other hand, for datasets with many instances and few features (1000,7) is faster. For the remaining comparisons, we chose the configuration (500,9) as representative of our method.

The next set of experiments aimed to compare SSIFSM with the methods described in Section 4.1. The detailed results of these methods are shown in Table 5. A comparison among the methods is shown in Table 6. As in the previous table, we present the results and comparison of accuracy, reduction and execution time.

Fig. 6 shows the average Friedman rankings for these experiments. These ranks are, by themselves, a good measure of the relative performance of a group of methods. The Iman–Davenport test with a p-value of 0.0000 for both accuracy and reduction showed significant differences between the methods. The rankings showed that the best-performing methods in terms

Table 4
Comparison of accuracy, reduction and execution time for the three configurations of our proposal. The table shows the win/loss record of the method in the column against the method in the column, row labeled with s and the p-value of the Wilcoxon test, row p_w.

		(100,13)	(500,9)	(1000,7)
Accuracy				
Average		0.8344	0.8360	0.8341
(100,13)	s		30/20	26/24
	p_s		0.2861	0.4486
(500,9)	s			30/19
	p_s			0.2507
Reduction				
Average		0.9568	0.9535	0.9508
(100,13)	s		20/30	21/29
	p_s		0.2370	0.1059
(500,9)	s			28/22
	p_s			0.8925
Execution time				
Average		244.9	196.0	204.8
(100,13)	s		30/20	24/26
	p_s		0.1397	0.9116
(500,9)	s			11/39
	p_s			0.0001

of reduction were SSIFSM and IS + FSCHC. In terms of accuracy, all the methods show similar rankings, with the exception of FSCHC and SAMPLING, which are clearly worse. For that reason, SAMPLING was not considered for reduction, as its performance in accuracy was inferior to the other methods.

As stated in Section 4.2, the use of many pairwise comparisons is not the most effective way of comparing a number of algorithms. To further assure the comparative results of SSIFSM, we used Holm's procedure. SSIFSM was used as the control method. Fig. 7 shows the results of Holm's test for accuracy and reduction. The test showed that in terms of accuracy, SSIFSM is no worse than ISCHC, IS + FSCHC, IMOEA and 1-NN, and it improves the results of FSCHC and SAMPLING. Furthermore, it obtained a significantly better reduction with respect to all the methods, with the exception of IS + FSCHC. These results fulfilled the major aim of our work, which was to obtain a scalable method with a performance comparable with the best state-of-the-art algorithms. Sections 5.1 and 5.2 will show that this performance is obtained along with a significant reduction in the execution time.

The results of the best four methods against SSIFSM are illustrated in Fig. 8. The figure shows the results for accuracy and reduction. This graphic representation is based on the κ-error relative movement diagrams [49]. However, here we use the reduction difference instead of the κ difference value. These diagrams use an arrow to represent 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 represent the difference between the accuracy and reduction of our method and those of the standard undersampling algorithm. These graphs are a convenient way of summarizing the results. A positive value in either reduction or accuracy means that our method performed better. Thus, arrows pointing up-right represent datasets for which our method outperformed the standard algorithm in both accuracy and reduction. Arrows pointing up-left indicate that our algorithm improved reduction but had worse accuracy, whereas arrows pointing down-right indicate that our algorithm improved accuracy but had a worse reduction. Arrows pointing down-left indicate that our algorithm performed worse in both accuracy and reduction.

If we inspect the plots, we can see that for IMOEA, ISCHC and FSCHC, most of the arrows are in the top right part of the coordinates, indicating a general advantage of SSIFSM in accuracy, reduction or both. We can also see a remarkable improvement in some problems. For IS + FSCHC, the distribution is more homogeneous, showing that the performances of both methods are similar.

One of the relevant parameters of our method is the number of rounds. We chose 10 rounds, using the results of other approaches combining difference methods, such as ensembles of classifiers, which have shown that most of the performance gain is obtained by the first few classifiers added. To test the validity of this value, we performed experiments with 1, 5, 10, 15 and 25 rounds. Fig. 9 shows the average accuracy, reduction and execution time of the experiments conducted on each number of rounds.

As in the previous experiment, we performed an Iman-Davenport test for accuracy and reduction. For accuracy the test obtained a p-value of 0.0002 and for reduction the p-value was 0.5758, indicating that there were significant differences for accuracy but for not reduction. Fig. 10 shows Friedman's ranks for the four different numbers of rounds. Regarding time, the graph shows an approximately linear behavior when more rounds are used. This is an expected result because as shown in Section 3.1, each round has linear complexity.

For accuracy, we also performed Holm test using 15 rounds as a control experiment as it achieved the best rank. This test was not performed for reduction because the Iman–Davenport test did not show significant differences. The results of the Holm test are shown in Fig. 11. The test and the ranks show that below ten rounds the accuracy was worse but that

Table 5
Results for the six standard methods described in Section 4.1.

Dataset	1-NN	IS + FSCHC			Sampling			ISCHC			FSCHC			IMOEA		
	Acc.	Reduct.	Acc.	Time (s)	Reduct.	Acc.	Time (s)	Reduct.	Acc.	Time (s)	Reduct.	Acc.	Time (s)	Reduct.	Acc.	Time (s)
Abalone	0.1966	0.9710	0.2053	1293.6	0.9968	0.2151	8.6	0.8336	0.2113	1132.0	0.9000	0.1923	30830.2	0.9469	0.2501	10041.0
Ads	0.9561	0.8837	0.9497	4647050.0	0.9986	0.9341	2296.4	0.7830	0.9500	1200803.3	0.5321	0.9591	3251210.0	0.7684	0.9540	503703.7
Adult	0.7998	0.8891	0.8210	6700940.0	0.9953	0.8395	359937.8	0.7542	0.7974	3953276.8	0.6076	0.8059	2407837.0	0.7521	0.8343	5410266.7
Anneal	0.9865	0.9864	0.9922	199.4	0.9988	0.9202	2.7	0.9099	0.9584	94.7	0.9153	0.9719	2591.0	0.9845	0.9663	10002.0
Arrhythmia	0.5578	0.9595	0.6822	510.1	0.9990	0.5556	3.0	0.9297	0.6667	90.6	0.9054	0.6400	3013.9	0.9822	0.6489	10005.0
Barley	0.8147	0.9666	0.8334	66561.4	0.9971	0.8353	81.2	0.8421	0.8477	2874.0	0.8266	0.7700	945925.0	0.8655	0.8551	32136.5
Basehock	0.8269	0.8764	0.7421	5105150.0	0.9968	0.6699	4429.3	0.7645	0.7241	1958791.7	0.5104	0.8455	3814070.0	0.7544	0.7090	879400.0
Car	0.9479	0.9143	0.9346	325.1	0.9981	0.7876	1.8	0.8403	0.8108	367.0	0.9375	0.7002	7013.8	0.8419	0.8850	10006.0
Card	0.7826	0.9954	0.8348	79.0	0.9980	0.8203	5.0	0.8968	0.7710	69.7	0.9804	0.8551	19183.4	0.9941	0.8464	10002.0
Cotton	0.6757	0.9641	0.6944	18703.1	0.9953	0.6764	362.2	0.8291	0.6923	2745.1	0.8156	0.6143	872141.7	0.8478	0.7115	29142.5
Dexter	0.5367	0.8889	0.6017	1407330.0	0.9970	0.6500	2679.3	0.8356	0.6317	509431.9	0.4988	0.6183	1415600.0	0.7686	0.5333	273897.4
Euthyroid	0.9383	0.9912	0.9715	2240.2	0.9997	0.9639	14.6	0.8633	0.9269	1233.6	0.9545	0.9627	40912.0	0.9792	0.9661	10004.0
Gene	0.7353	0.9739	0.8969	6072.2	0.9955	0.7394	100.1	0.8214	0.6824	3326.0	0.7058	0.7227	1085042.9	0.8704	0.8315	23018.4
German	0.6880	0.9729	0.7240	419.6	0.9974	0.6740	3.8	0.8972	0.7020	131.3	0.9770	0.6920	7731.4	0.9890	0.7180	10001.0
Gina	0.7941	0.8817	0.7817	2803000.0	0.9950	0.7999	1991.2	0.7888	0.7494	670435.7	0.5309	0.7970	2251550.0	0.7651	0.8123	436370.4
Gisette	0.9327	0.8778	0.9039	6371560.0	0.9891	0.8569	558949.2	0.7583	0.9169	3451690.0	0.5170	0.9161	2650497.0	0.7519	0.9394	6881200.0
Hiva	0.9522	0.8838	0.9584	992100.0	0.9995	0.9607	1425.1	0.7683	0.9558	3467416.7	0.5328	0.9525	3885100.0	0.7543	0.9652	1019200.0
Hypothyroid	0.9308	0.9877	0.9809	936.4	0.9998	0.9706	15.9	0.8587	0.9350	1315.9	0.9655	0.9544	77114.6	0.9706	0.9793	10011.0
Isolet	0.8963	0.8799	0.8293	979050.0	0.9923	0.7906	8330.4	0.7598	0.8302	4318233.3	0.5525	0.8856	4153790.0	0.7606	0.8706	927600.0
Krvskp	0.9172	0.9692	0.9605	1860.5	0.9994	0.9323	29.9	0.8384	0.8561	1115.4	0.8868	0.9320	43561.1	0.9303	0.9549	10014.0
Letter	0.9553	0.8829	0.8944	649105.3	0.9887	0.7662	1746.1	0.7722	0.8997	487995.6	0.6313	0.8541	2097350.0	0.8010	0.7764	92810.8
Lrs	0.8113	0.9788	0.8661	168.7	0.9973	0.7491	4.8	0.9264	0.8434	115.5	0.9535	0.8623	664.2	0.9955	0.8396	10001.0
Madelon	0.5315	0.8989	0.5392	565648.6	0.9948	0.5512	657.6	0.8462	0.5473	4651.4	0.5782	0.5173	2028375.0	0.7920	0.5492	116379.3
Magic04	0.8187	0.9037	0.8151	393226.8	0.9947	0.8081	558.3	0.7777	0.8017	326238.6	0.9000	0.6519	250469.7	0.8299	0.8338	59836.3
Mfeat-fac	0.9650	0.9734	0.9535	5999.1	0.9950	0.8785	177.0	0.8855	0.9380	1052.8	0.8616	0.9490	343481.8	0.8685	0.9550	17402.4
Mfeat-fou	0.7920	0.9572	0.8310	1655.4	0.9939	0.7265	49.1	0.8485	0.7750	721.5	0.9066	0.8020	8579.1	0.9527	0.7995	10013.0
Mfeat-kar	0.9565	0.9608	0.9380	1618.8	0.9932	0.8495	109.0	0.8629	0.9165	528.7	0.8812	0.9145	9384.8	0.9424	0.9155	10018.0
Mfeat-mor	0.6605	0.8950	0.7060	255.9	0.9935	0.6340	5.4	0.8713	0.7060	152.8	0.8333	0.3930	9104.0	0.8909	0.5930	10003.0
Mfeat-pix	0.9730	0.9613	0.9570	8302.0	0.9937	0.8740	237.2	0.8795	0.9540	1595.2	0.8033	0.9565	411766.7	0.8495	0.9595	19883.6
Mfeat-zer	0.7885	0.9419	0.7950	1576.8	0.9925	0.7040	50.9	0.8568	0.7745	409.2	0.8255	0.7390	7757.1	0.9186	0.7885	10027.0
Mushroom	1.0000	0.9385	0.9982	925506.5	0.9992	0.9961	2301.3	0.8260	0.9987	160808.0	0.6359	0.9941	4824550.0	0.8167	1.0000	155946.7
Musk	0.9461	0.9155	0.9364	1326133.3	0.9974	0.9233	916.4	0.8234	0.9553	131270.3	0.6127	0.9467	4770650.0	0.7862	0.9535	187507.9
Nursery	0.7742	0.8716	0.8555	329571.9	0.9969	0.7810	396.4	0.7881	0.7660	228700.3	0.7391	0.8222	1192523.8	0.8186	0.7357	39217.2
Optdigits	0.9744	0.9469	0.9509	192729.4	0.9939	0.9068	493.8	0.8495	0.9527	3447.7	0.7859	0.9100	788564.4	0.8441	0.9498	39470.6
Ozone1hr	0.9553	0.9977	0.9711	2044.4	0.9998	0.9711	24.7	0.9004	0.9707	620.2	0.9861	0.9636	14019.0	0.9584	0.9703	10067.0
Ozone8hr	0.9166	0.9947	0.9320	2081.2	0.9997	0.9312	27.9	0.8818	0.9316	749.9	0.9847	0.9225	12579.5	0.9569	0.9364	10032.0
Page-blocks	0.9638	0.9812	0.9382	1555.4	0.9998	0.9304	22.2	0.8488	0.9581	1429.1	0.9000	0.9212	20097.2	0.8794	0.9552	10026.0
Pcmac	0.5657	0.8856	0.5584	2880133.3	0.9991	0.5780	688.4	0.8120	0.5698	733729.2	0.5149	0.5460	1227300.0	0.7625	0.5594	349333.3
Pendigits	0.9934	0.9060	0.9591	55332.2	0.9924	0.9159	458.0	0.8286	0.9854	12215.8	0.7438	0.8774	411681.7	0.7698	0.9289	30856.3
Phoneme	0.9048	0.8853	0.8413	1294.0	0.9970	0.7254	32.7	0.8306	0.8535	1542.6	0.8000	0.7228	10927.6	0.7434	0.8247	10031.0
Satimage	0.9061	0.9641	0.8709	12117.4	0.9968	0.8474	204.0	0.8349	0.8902	4941.1	0.9056	0.8160	163659.9	0.8447	0.8897	25716.8
Segment	0.9649	0.9672	0.9611	653.1	0.9961	0.8935	17.3	0.8695	0.9333	383.7	0.8421	0.9719	15570.9	0.9573	0.8701	10014.0
Shuttle	0.9990	0.9447	0.9978	4270850.0	0.9963	0.9966	2229.8	0.7551	0.9983	4246600.0	0.8111	0.9653	1119340.0	0.8178	0.9737	406564.1
Sick	0.9570	0.9899	0.9695	2548.3	0.9998	0.9687	31.1	0.8641	0.9539	1484.1	0.9697	0.9520	37000.6	0.9648	0.9788	10046.0
Soybean	0.9221	0.9625	0.9339	429.7	0.9938	0.6559	12.7	0.8790	0.9059	259.6	0.8817	0.9309	1796.1	0.9692	0.9030	10003.0
Texture	0.9895	0.9608	0.9742	7749.2	0.9959	0.9137	215.8	0.8456	0.9706	2861.6	0.8750	0.9460	159430.6	0.8567	0.9818	18524.9
Titanic	0.3435	0.9656	0.7529	405.1	0.9996	0.7756	5.6	0.8873	0.7906	281.2	0.8750	0.3230	3719.9	0.9725	0.7833	10002.0
Waveform	0.7140	0.9422	0.7974	5453.2	0.9950	0.7960	213.4	0.8259	0.7366	3358.3	0.8875	0.7064	40614.7	0.8966	0.8226	15573.0
Yeast	0.5311	0.9109	0.5149	141.9	0.9945	0.4345	3.6	0.8626	0.5662	272.4	0.8750	0.3216	894.4	0.8344	0.5568	10002.0
Zip	0.4745	0.8903	0.4565	3820300.0	0.9957	0.4933	2618.2	0.7695	0.4439	931949.4	0.5824	0.4708	2402730.0	0.7681	0.5215	538481.5
Average	0.8183	0.9378	0.8353	891399.35	0.9963	0.7913	19103.52	0.8376	0.8181	536698.8	0.7887	0.7892	986985.95	0.8667	0.8267	375196.2

Table 6
Comparison of accuracy, reduction and execution time for the six standard methods described in 4.1 and our proposal. The table shows the win/loss record of the method in the column against the method in the column, row labeled with s and the p-value of the Wilcoxon test, row p_w.

		(500,9)	1-NN	FS + IS	Sampling	IS	FS	IMOEA
Accuracy								
Average		0.8360	0.8183	0.8353	0.7913	0.8181	0.7892	0.8267
	s		27/23	30/19	12/37	21/29	17/33	32/18
(500,9)	p_s		0.8811	0.2078	0.0000	0.1436	0.0010	0.4400
Reduction								
		(500,9)		FS + IS	Sampling	IS	FS	IMOEA
Average		0.9535		0.9378	0.9963	0.8376	0.7887	0.8667
	s			18/32	46/4	4/46	2/48	7/43
(500,9)	p_s			0.0798	0.0000	0.0000	0.0000	0.0000
Execution time								
		(500,9) Parallel	(500,9) Sequential	FS + IS	Sampling	IS	FS	IMOEA
Average		196.0	29489.6	891399.4	19103.5	536698.8	986986.0	375196.2
(500,9)	s			0/50	12/38	0/50	0/50	0/50
Parallel	p_s			0.0000	0.0000	0.0000	0.0000	0.0000
(500,9)	s			16/34	48/2	25/25	1/49	0/50
Sequential	p_s			0.0002	0.0000	0.1517	0.0000	0.0000

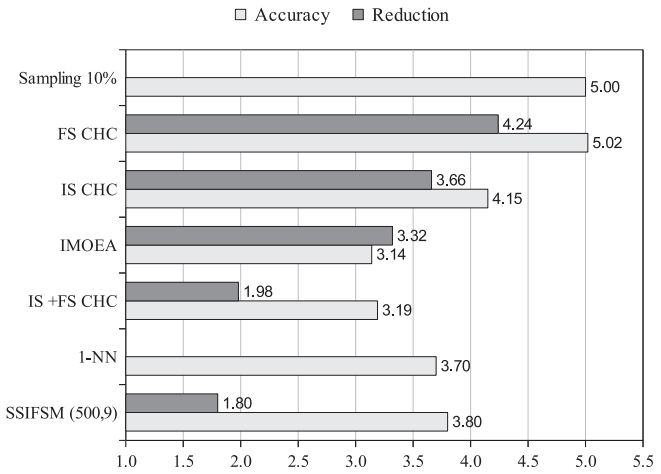

Fig. 6. Average Friedman's ranks for SSIFSM and the six algorithms described in 4.1.

performing more than 10 rounds did not improve the SSIFSM's results. In this way, using 15 rounds was better than using 1 and 5 rounds but did not improve the results when using 10 rounds.

5.1. Execution time

In the description of our method, we claimed that applying the philosophy of divide-and-conquer together with bookkeeping would produce a method that was far more scalable than the previous approaches. To verify this claim, Fig. 12 shows the average time of each method together with the time required for the longest execution. The execution time shown in the tables is the wall-clock time spent by each algorithm. We measure the time elapsed from the beginning of the algorithm until it completes its final output. The figure shows that SSIFSM is significantly faster than the other methods. The difference is especially remarkable for the largest problems and the parallel implementation of our method. We also compare a parallel implementation of SSIFSM against the other methods because natural parallelization is an inherent feature of SSIFSM. Furthermore, the sequential implementation also shows its remarkably good scalability.

To obtain a clearer idea of the improvement in terms of the running time of our approach, Figs. 13 and 14 show the speed-up of SSIFSM with respect to the standard IS + FSCHC for the parallel and sequential implementations respectively. For the sequential version, SSIFSM achieved an average speed-up of 53.6. However, in the largest datasets, the improvement is more significant. `shuttle` is an example of this situation, with a speed-up of almost 1,500 times. For the parallel implementation, the average speed-up is almost 2555, with the most extreme example of 15,490 times faster for the `shuttle` dataset.

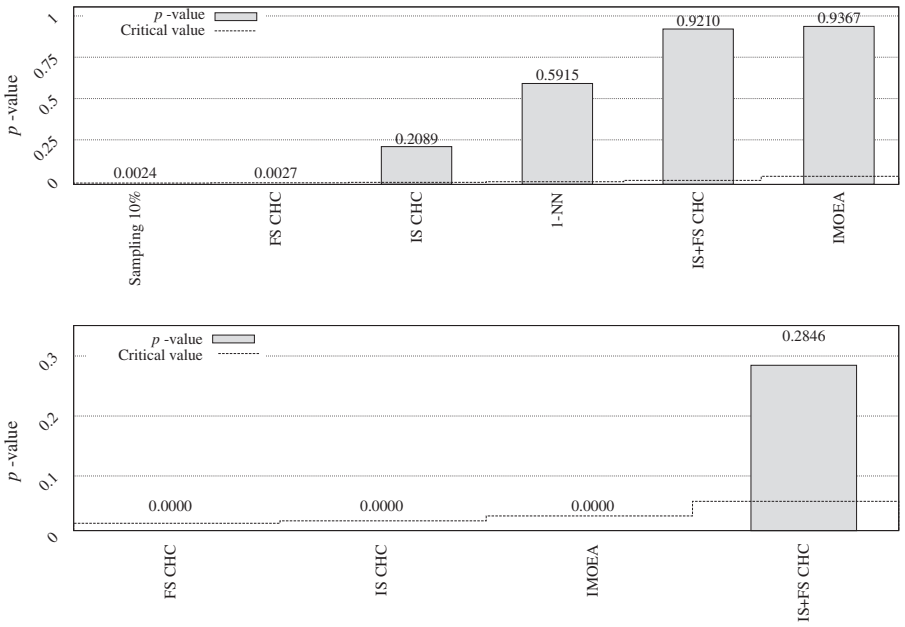

Fig. 7. Results of the Holm test for the six different methods and SSIFSM as the control method for accuracy (top) and reduction (bottom). *p*-Values are shown numerically.

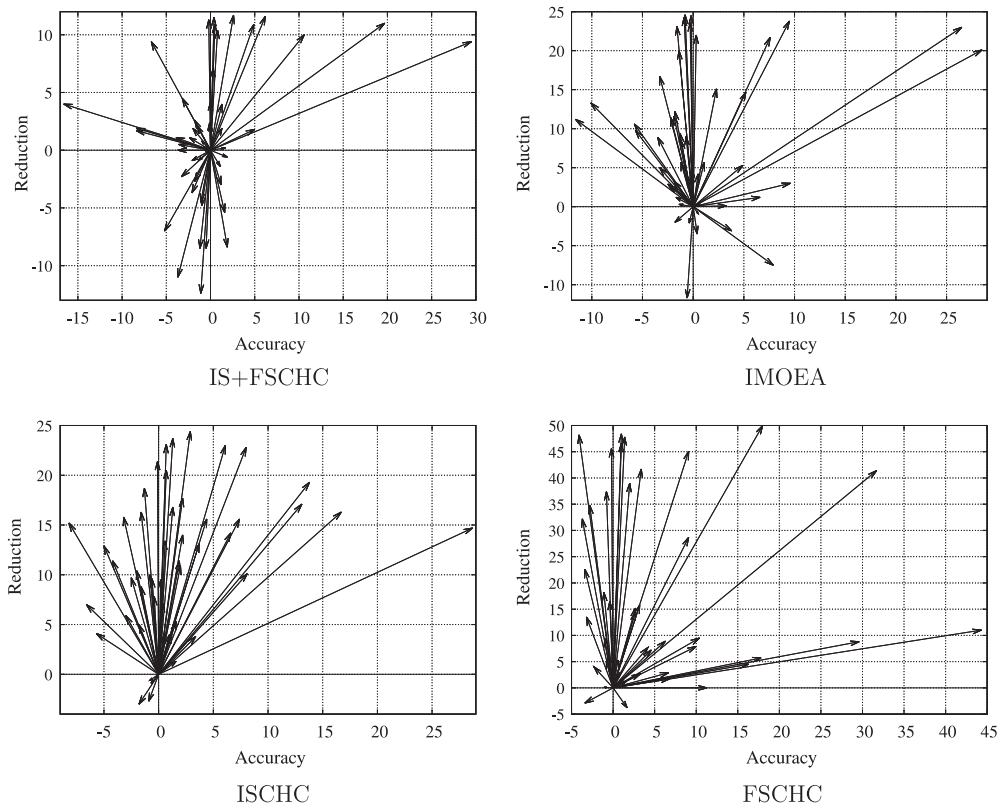

Fig. 8. Reduction/accuracy using relative movement diagrams for SSIFSM and the best four standard methods. Positive values on both axes show a better performance by SSIFSM.

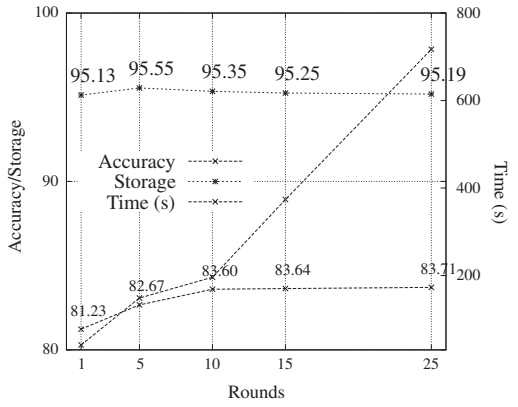

Fig. 9. Accuracy, reduction and running time for different numbers of rounds for SSIFSM.

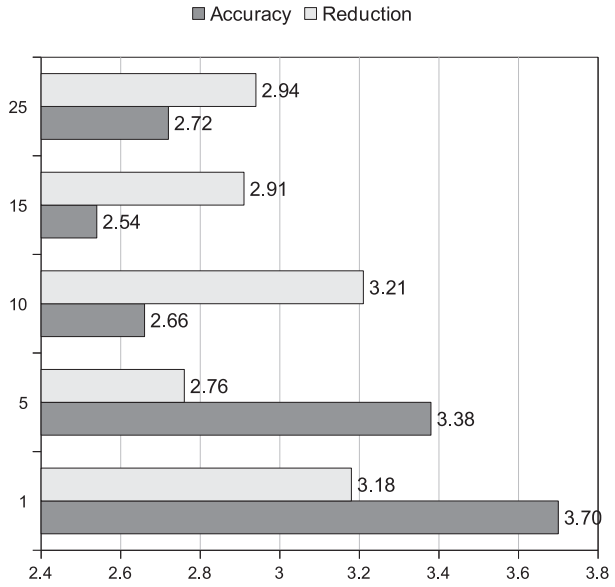

Fig. 10. Average Friedman's ranks for the five numbers of rounds for SSIFSM.

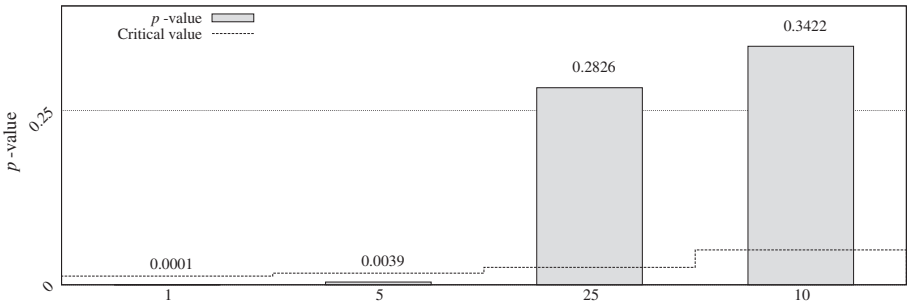

Fig. 11. Results of the Holm test for accuracy and the five numbers of rounds and 15 rounds as the control method. *p*-Values are shown numerically.

To further assure the superior performance of our approach we carried out a final set of three control experiments. The first experiment consisted on a random selection method. For each problem, we fixed a selection rate for instance and

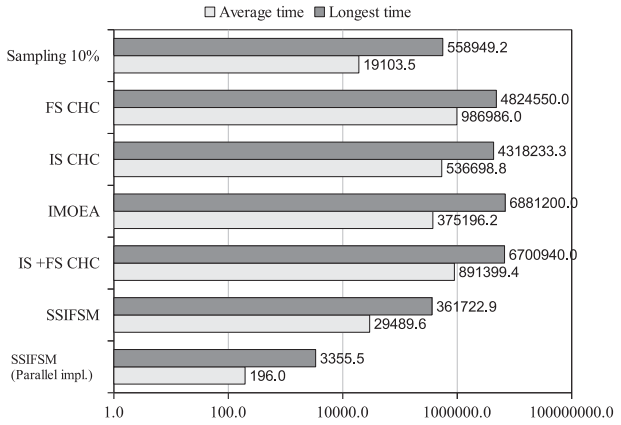

Fig. 12. Average time and longest time, both in seconds, in logarithmic scale for the five standard instance and/or feature selection algorithms and SSIFSM.

features randomly obtained in the interval [5%,15%] and [15%,25%] respectively.[4] Then we randomly selected instances and features with the obtained rates and evaluated the accuracy of the selected subsample. This process was repeated as many times as it can be done in the runtime required by the proposed methodology sequential implementation for the same problem. Finally the best selection was used as the final selection. The second experiment was designed to test whether two fast instance and feature selection methods run one after the other could outperform our approach. We used two of the fastest methods for instance and feature selection among the algorithms with competitive performance. We chose IB3 [1] algorithm for instance selection and ReliefF [55] method for feature selection. We tried both alternatives, running first IB3 and then ReliefF and vice versa. A comparison of SSIFSM against these three methods is shown in Table 7. Detailed results are shown in Table 8.

For the combination of IB3 and ReliefF, in both cases, we found that the performance in terms of reduction was similar to our approach, but with a very significant worsening of the accuracy. The random selection had two main problems. Firstly, evaluating each selection is a $O(N^2)$ process, thus the procedure did not scale up well. Secondly, the random process was more prone to obtain large reduction than good accuracy, as finding a random selection with good accuracy is a rarer event. Due to these two reasons, the process obtained very poor accuracy. In fact, although the reduction was good, the accuracy was far worse than the accuracy of SSIFSM, with a p-value of the Wilcoxon test of 0.0000, and results worse than SSIFSM for 48 out of 50 datasets.

5.2. Scalability to very large datasets

In previous experiments, we demonstrated the performance of SSIFSM in datasets of medium to large sizes. However, the major aim of this work is the scalability of the presented method to very large datasets. Thus, we applied our algorithm to 13 very large problems. These datasets are shown in Table 2. We have chosen problems with many instances, many features and both many instances and features. With the largest set containing 50 million instances and 800 features, these datasets represent a serious challenge for any method. Furthermore, `chroml9`, `chrom2l` and `dna` also show a large imbalance ratio, which makes them more difficult problems. To highlight the scalability problem of the datasets, we show the memory size of these datasets using a standard float point precision stored with four bytes.

The experimental setup used is the same as that described in the previous sections. Class-imbalanced datasets are treated in the same way as the other datasets. The only exception is the use of the G-mean as the accuracy measure (see Section 4) for both the evolution of the subsets and the evaluation of the thresholds. Regarding subset size, we used the configuration (100, 13) for all the datasets, with the exception of `chroml9`, `covtype`, `kddcup99` and `poker`, because most of the datasets have both many instances and many features. For `chroml9`, `covtype`, `kddcup99` and `poker` we used the configuration (1000, 7) because these four datasets have many instances but not many features.

As the control experiment, we used 1-NN accuracy because none of the standard methods is scalable to datasets of this size. Table 9 shows the results. The first interesting result is the good scalability of our proposal. For the largest dataset of 50 million instances and 800 features, SSIFSM took 140 h to complete the process. If we estimate the time that the standard IS + FSCHC would require to process such a dataset, considering that it needed more than 77 days for `adult` dataset, which has 48,842 instances and 105 features, we can conclude that SSIFSM reduces the time by several orders of magnitude.

Furthermore, the performance of SSIFSM is very competitive in terms of accuracy improvement. We compared for each dataset the accuracy achieved using SSIFSM with the accuracy usign 1-NN with all the instances and features. This comparison was performed using a resampled t-test. Table 9 marks significant differences at a significance level of 0.05 with a ✓. We can see that SSIFSM improved the accuracy of 1-NN for 11 of the 13 datasets, and matched the accuracy of 1-NN in the

[4] We chose these rates because these were the typical values obtained in our experiments with the other methods.

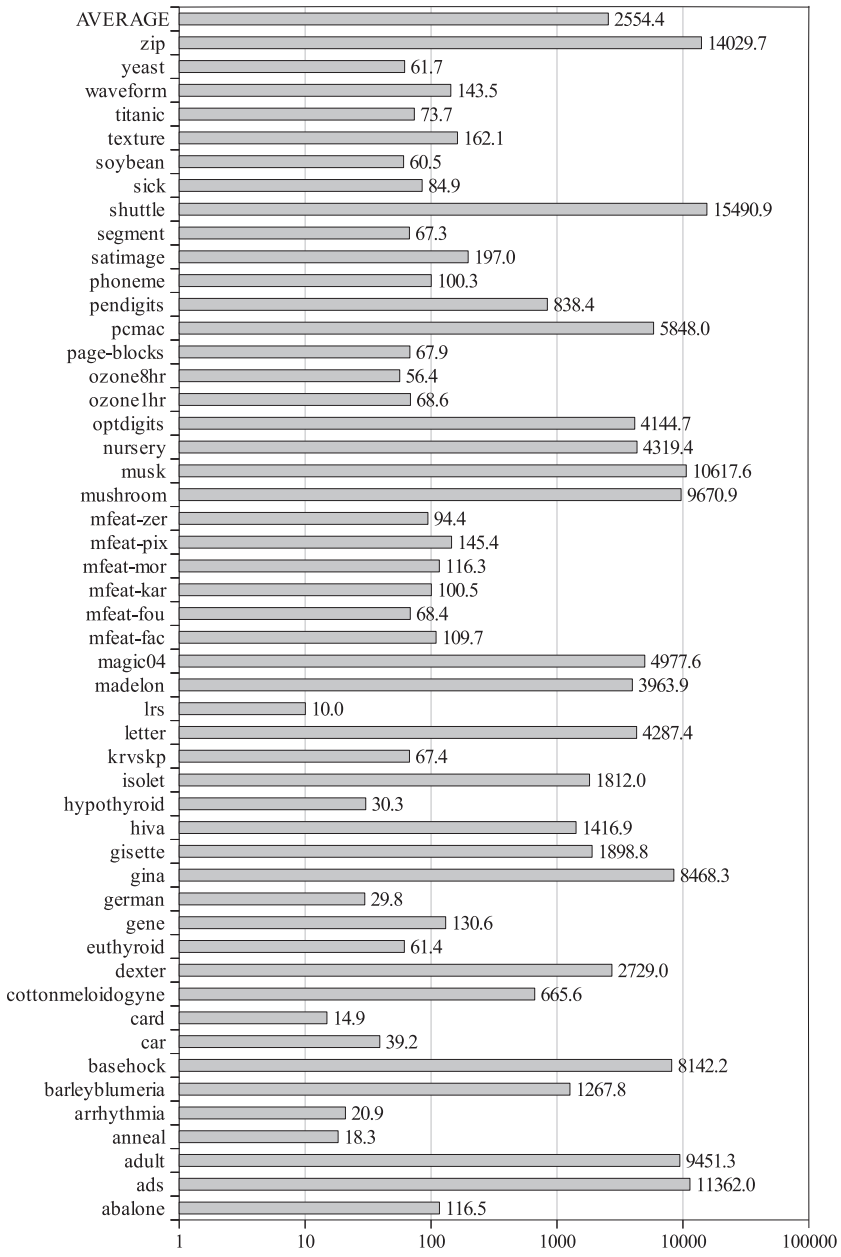

Fig. 13. Speed-up for the parallel implementation.

remaining two datasets. Furthermore, the differences for class-imbalanced datasets are remarkably large. These improvements are achieved with an average reduction of 99.72% of the instances and features.

The speed of the execution for large problems such as `chrom21` is also remarkable. One of the reasons for this fast execution is the class-imbalanced nature of the problems. As stated above, our method first divides the whole dataset into small subsets and then performs the process of instance and feature selection in each subset separately. When a subset contains instances of only one class, nothing is done with it, as a selection process with just one class is meaningless. In heavily class-imbalanced datasets, this situation is common as many subsets do not contain instances from the minority class, which makes the algorithm faster. That makes our approach very suitable for class-imbalanced problems.

6. Conclusions and future work

In this paper, we presented a new method for simultaneous instance and feature selection that is applicable to very large datasets. The method consists of concurrently applying instance and feature selection to small subsets of the original dataset

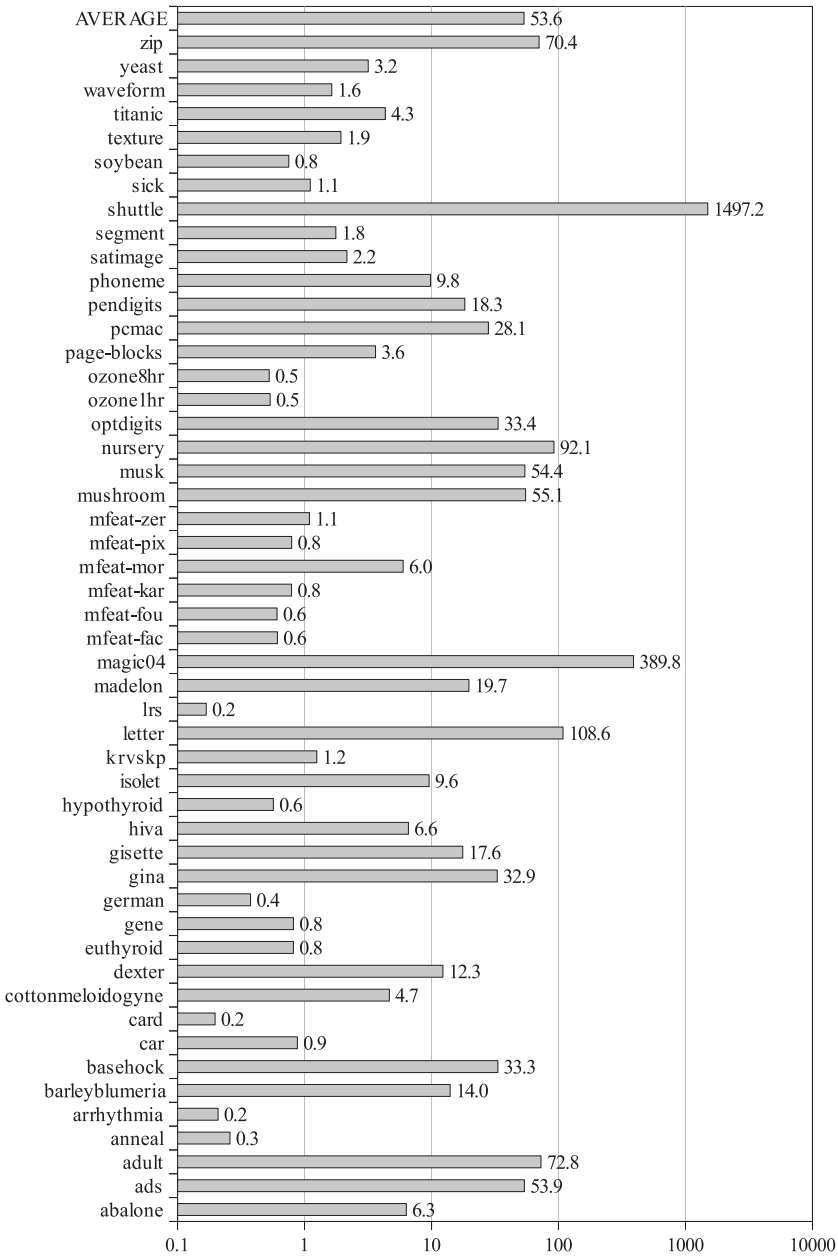

Fig. 14. Speed-up for the sequential implementation.

Table 7
Comparison of accuracy and reduction results for the three control experiments and our proposal. The table shows the win/loss record of the method in the column against the method in the column, row labeled with *s*, and the *p*-value of the Wilcoxon test, row p_w.

		(500,9)	Random	IB3 + ReliefF	ReliefF + IB3
Accuracy					
Average		0.8360	0.6782	0.7150	0.7406
(500,9)	*s*		1/48	11/39	11/39
	p_s		0.0000	0.0000	0.0000
Reduction					
Average		0.9535	0.9633	0.9291	0.9137
(500,9)	*s*		24/26	26/24	22/28
	p_s		0.8507	0.6605	0.0749

Table 8
Results of accuracy, reduction and time for the three control experiments.

Dataset	Random			IB3 + ReliefF			ReliefF + IB3		
	Reduct.	Acc.	Time (s)	Reduct.	Acc.	Time (s)	Reduct.	Acc.	Time (s)
Abalone	0.9618	0.1698	204.1	0.2562	0.1962	1.9	0.2562	0.1962	1.4
Ads	0.9630	0.9399	86604.0	0.9982	0.6071	3.1	0.9708	0.9079	38.8
Adult	0.9567	0.7284	93653.4	0.9973	0.7373	148.8	0.9976	0.7440	570.3
Anneal	0.9750	0.7236	770.0	0.9632	0.9562	0.3	0.9293	0.9764	0.0
Arrhythmia	0.9537	0.5333	2447.1	0.8828	0.5067	0.0	0.9529	0.4756	0.1
Barley.	0.9562	0.7449	4748.0	1.0000	0.6382	0.3	1.0000	0.3618	2.6
Basehock	0.9499	0.6995	153601.5	0.9987	0.5605	5.9	0.9784	0.7557	46.0
Car	0.9823	0.6887	370.0	0.9206	0.8432	0.0	0.8589	0.9074	0.2
Card	0.9495	0.6478	400.0	0.9829	0.7942	0.1	0.9742	0.8217	0.2
Cotton	0.9561	0.5681	4017.0	1.0000	0.4949	0.5	1.0000	0.5051	2.2
Dexter	0.9648	0.5133	114600.4	0.9997	0.5700	4.8	0.9999	0.5067	23.1
Euthyroid	0.9901	0.8674	2740.0	0.9799	0.8301	0.1	0.9904	0.3535	1.3
Gene	0.9534	0.5347	7424.0	0.9990	0.3729	0.7	0.9949	0.8098	1.9
German	0.9573	0.6060	1115.0	0.9863	0.5590	0.0	0.9989	0.4670	0.2
Gina	0.9554	0.7330	85601.4	0.9998	0.4983	2.9	1.0000	0.5147	28.2
Gisette	0.9593	0.8341	363666.0	0.9996	0.6929	104.1	0.9987	0.9130	485.19
Hiva	0.9878	0.9622	153606.5	0.9994	0.3521	13.9	0.9999	0.9553	96.8
Hypothyroid	0.9796	0.9159	1648.1	0.9689	0.9164	0.5	0.9855	0.9398	0.7
Isolet	0.9560	0.7502	103604.7	0.7793	0.8470	19.4	0.7735	0.8426	153.4
Krvskp	0.9611	0.6567	1489.1	0.9785	0.8787	0.2	0.9582	0.9433	0.6
Letter	0.9506	0.3054	5979.2	0.8578	0.9132	7.8	0.8578	0.9132	17.9
Lrs	0.9588	0.8057	1002.0	0.7949	0.8528	0.0	0.8094	0.8264	0.2
Madelon	0.9598	0.5027	28707.0	1.0000	0.5000	1.9	1.0000	0.5000	13.1
Magic04	0.9512	0.6720	1008.1	1.0000	0.6484	5.0	1.0000	0.3516	26.5
Mfeat-fac	0.9636	0.9220	9813.0	0.8952	0.9345	0.4	0.8912	0.9320	1.5
Mfeat-fou	0.9539	0.5095	2724.0	0.8638	0.7995	0.4	0.6966	0.7430	0.6
Mfeat-kar	0.9621	0.4765	2047.2	0.9287	0.9320	0.2	0.8549	0.9130	0.4
Mfeat-mor	0.9566	0.3345	42.0	0.6465	0.6315	0.0	0.6465	0.6315	0.3
Mfeat-pix	0.9574	0.8845	10484.0	0.9211	0.9470	0.3	0.9039	0.9475	1.6
Mfeat-zer	0.9490	0.5810	1443.1	0.7914	0.7610	0.3	0.7134	0.7420	0.6
Mushroom	0.9884	0.9042	16790.4	0.9969	0.9947	0.5	0.9924	0.9998	22.5
Musk	0.9616	0.9224	24368.4	1.0000	0.7765	1.5	0.9878	0.8651	22.1
Nursery	0.9837	0.4608	3576.9	0.9615	0.6829	2.2	0.9169	0.7519	8.0
Optdigits	0.9592	0.6185	5762.1	0.9422	0.9491	0.7	0.9212	0.9349	3.2
Ozone1hr	0.9908	0.9652	3809.1	0.9993	0.4605	0.4	1.0000	0.5012	0.9
Ozone8hr	0.9885	0.9312	3944.0	0.9995	0.7285	0.4	1.0000	0.4091	1.0
Page-blocks	0.9774	0.9269	429.0	0.9705	0.9185	0.1	0.9631	0.9239	0.9
Pcmac	0.9596	0.5121	103600.8	1.0000	0.4488	4.0	1.0000	0.5831	44.8
Pendigits	0.9556	0.6010	3026.4	0.9654	0.9783	0.8	0.9654	0.9783	4.1
Phoneme	0.9517	0.6883	131.1	1.0000	0.4991	0.5	0.9247	0.8300	0.9
Satimage	0.9565	0.8254	5615.1	0.8570	0.8712	0.9	0.8570	0.8712	3.1
Segment	0.9550	0.7411	369.0	0.9373	0.9394	0.2	0.9059	0.9338	0.3
Shuttle	0.9829	0.8740	2861.3	0.9975	0.9987	7.1	0.9971	0.9983	88.1
Sick	0.9777	0.9032	2295.2	0.9907	0.7058	0.0	0.9951	0.7690	1.0
Soybean	0.9525	0.6279	571.0	0.8519	0.9015	0.1	0.8366	0.8941	0.2
Texture	0.9556	0.8558	4004.1	0.9316	0.9653	0.3	0.9316	0.9653	2.2
Titanic	0.9873	0.4619	93.0	0.9850	0.6833	0.1	0.9749	0.6901	0.3
Waveform	0.9528	0.5566	3325.0	0.9786	0.6046	1.1	0.9722	0.7150	2.0
Yeast	0.9391	0.3189	44.0	0.7515	0.3730	0.2	0.5859	0.4811	0.2
Zip	0.9559	0.4045	54602.5	0.9469	0.5014	26.8	0.9640	0.5352	76.5
Average	0.9633	0.6782	29695.57	0.9291	0.7150	7.43	0.9137	0.7406	35.96

where a bookkeeping mechanism can be used. The results of the subsets are combined by means of a voting method, setting different thresholds for instances and features. Using a CHC algorithm as the base algorithm, we have shown that our method matches the performance of state-of-the-art instance and/or feature selection methods. The method has also proven its usefulness for class-imbalanced data in both accuracy and reduction. Furthermore, the method has proven its usefulness using as base instance selection a time consuming CHC algorithm. Faster execution can be obtained using other faster selection methods.

The results showed that our approach scaled up to problems of very large size. Three features of our method guarantee that scalability: first, instance and feature selection is always performed over small datasets, keeping the time spent by the process low; second, a bookkeeping approach is used where the list of neighbors of each instance for each possible subspace is calculated before running the evolutionary algorithm; and third, only small subsets must be kept in memory, removing any scalability constraint due to memory limits. This scalability has been experimentally proven, with the largest dataset containing 50 million instances and 800 features. No previous method of instance and feature selection has been applied so such a very large dataset.

Table 9
Summary of results for large datasets. Significant differences at a significance level of 0.05 using a resampled *t*-test are marked with a ✓ for the best of the two methods.

Dataset	1-NN	SSIFSM		
	Accuracy	Reduction	Accuracy	Time (s)
Arabidopsis	0.5021	0.9999	0.5810✓	3966.5
Ccds	0.4364	0.9989	0.6497✓	926.6
Census	0.5927	0.9928	0.8019✓	4659.4
Chrom19	0.9088	0.9947	0.9216	4287.0
Chrom21	0.0000	0.9999	0.4732✓	2590.1
Covtype	0.7264	0.9982	0.7742✓	4057.0
Dna	0.1580	0.9966	0.6005✓	505439.8
Kddcup98	0.2183	0.9985	0.5266✓	115673.2
Kddcup99	0.9995	0.9972	0.9984	59330.9
Poker	0.5017	0.9900	0.6362✓	1403.5
Reuters21578	0.6361	0.9978	0.7036✓	10346.0
Rcv1	0.4056	0.9995	0.4926✓	156949.6
Ustilago	0.3588	0.9999	0.6853✓	1163.8
Average	0.4957	0.9972	0.6804	66984.1

The natural extension of this work is the application of the same principles to other widely used classifiers, such as decision trees and support vector machines. As we have stated in the introduction, the bookkeeping principle may be used with many of the most common classifiers. Almost all of them, are based in a training stage that uses information from the datasets, but does not need to have the actual instances. Thus, our next step is studying the efficiency of SSIFSM with these classifiers.

Acknowledgement

This work was supported in part by Project TIN2011-22967 of the Spanish Ministry of Science and Innovation and Project P09-TIC-4623 of the Junta de Andalucía.

References

[1] D.W. Aha, D. Kibler, M.K. Albert, Instance-based learning algorithms, Machine Learning 6 (1991) 37–66.
[2] J. Aronis, F. Provost, Increasing the efficiency of data mining algorithms with breadth-first marker propagation, in: Proceedings of the Third International Conference on Knowledge Discovery and Data Mining, AAAI Press, 1997, pp. 119–122.
[3] A. Ben-Hur, C. Soon-Ong, S. Sonnenburg, B. Schölkopf, G. Rätsch, Support vector machines and kernels for computationalbiology, PLoS Comput Biology 4 (2008) e1000173.
[4] J.L. Bentley, Multidimensional binary search trees used for associative searching, Communications of the ACM 18 (1975) 509–517.
[5] A. Blum, P. Langley, Selection of relevant features and examples in machine learning, Artificial Intelligence 97 (1997) 245–271.
[6] M. Boullé, A parameter-free classification method for large scale learning, Journal of Machine Learning Research 10 (2009) 1367–1385.
[7] H. Brighton, C. Mellish, Advances in instance selection for instance-based learning algorithms, Data Mining and Knowledge Discovery 6 (2002) 153–172.
[8] F.Z. Brill, D.E. Brown, W.N. Martin, Fast genetic selection of features for neural networks classifiers, IEEE Transactions on Neural Networks 3 (1992) 324–334.
[9] C.E. Brodley, Recursive automatic bias selection for classifier construction, Machine Learning 20 (1995) 63–94.
[10] J.R. Cano, F. Herrera, M. Lozano, Using evolutionary algorithms as instance selection for data reduction in KDD: an experimental study, IEEE Transactions on Evolutionary Computation 7 (2003) 561–575.
[11] J.R. Cano, F. Herrera, M. Lozano, Stratification for scaling up evolutionary prototype selection, Pattern Recognition Letters 26 (2005) 953–963.
[12] N.W. Chawla, L.O. Hall, K.W. Bowyer, W.P. Kegelmeyer, Learning ensembles from bites: a scalable and accurate approach, Journal of Machine Learning Research 5 (2004) 421–451.
[13] J.H. Chen, H.M. Chen, S.Y. Ho, Design of nearest neighbor classifiers: multi-objective approach, International Journal of Approximate Reasoning 40 (2005) 3–22.
[14] T.F. Covões, E.R. Hruschka, Towards improving cluster-based feature selection with a simplified silhouette filter, Information Sciences 181 (2011) 3766–3782.
[15] M. Dash, K. Choi, P. Scheuermann, H. Liu, Feature selection for clustering – a filter solution, in: Proceedings of the Second International Conference on Data Mining, pp. 115–122.
[16] J. Demšar, Statistical comparisons of classifiers over multiple data sets, Journal of Machine Learning Research 7 (2006) 1–30.
[17] J. Derrac, C. Cornelis, S. García, F. Herrera, Enhancing evolutionary instance selection algorithms by means of fuzzy rough set based feature selection, Information Sciences 186 (2012) 73–92.
[18] J. Derrac, S. García, F. Herrera, Ifs-coco: instance and feature selection based on cooperative coevolution with nearest neighbor rule, Pattern Recognition 43 (2010) 2082–2105.
[19] J. Derrac, S. García, F. Herrera, Stratified prototype selection based on a steady-state memetic algorithm: a study of scalability, Memetic Computing 2 (2010) 183–189.
[20] O.J. Dunn, Multiple comparisons among means, Journal of the American Statistical Association 56 (1961) 52–64.
[21] L.J. Eshelman, The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination, Morgan Kauffman, San Mateo, CA, 1990.
[22] A. Frank, A. Asuncion, UCI Machine Learning Repository, 2010.
[23] M. Friedman, A comparison of alternative tests of significance for the problem of m rankings, Annals of Mathematicals Statistics 11 (1940) 86–92.

[24] S. García, J.R. Cano, F. Herrera, A memetic algorithm for evolutionary prototype selection: a scaling up approach, Pattern Recognition 41 (2008) 2693–2709.
[25] S. García, J. Derrac, J. Cano, F. Herrera, Prototype selection for nearest neighbor classification: taxonomy and empirical study, IEEE Transactions on Pattern Analysis and Machine Intelligence 34 (2012) 417–435.
[26] S. García, F. Herrera, Evolutionary undersampling for classification with imbalanced datasets: proposals and taxonomy, Evolutionary Computation 17 (2009) 275–306.
[27] C. García-Osorio, A. de Haro-García, N. García-Pedrajas, Democratic instance selection: a linear complexity instance selection algorithm based on classifier ensemble concepts, Artificial Intelligence 174 (2010) 410–441.
[28] N. García-Pedrajas, J.A.R. del Castillo, D. Ortiz-Boyer, A cooperative coevolutionary algorithm for instance selection for instance-based learning, Machine Learning 78 (2010) 381–420.
[29] N. García-Pedrajas, C. García-Osorio, C. Fyfe, Nonlinear boosting projections for ensemble construction, Journal of Machine Learning Research 8 (2007) 1–33.
[30] N. García-Pedrajas, A. de Haro-García, Scaling up data mining algorithms: review and taxonomy, Progress in Artificial Intelligence 1 (2012) 71–87.
[31] A. de Haro-García, N. García-Pedrajas, Scaling up feature selection by means of pseudoensembles of feature selectors, IEEE Transactions on Pattern Analysis and Machine Intelligence (submitted for publication).
[32] A. de Haro-García, N.G. Pedrajas, A divide-and-conquer recursive approach for scaling up instance selection algorithms, Data Mining and Knowledge Discovery 18 (2009) 392–418.
[33] S. Holm, A simple sequentially rejective multiple test procedure, Scandinavian Journal of Statistics 6 (1979) 65–70.
[34] F. Hussain, H. Liu, C. Tan, M. Dash, Discretization: An Enabling Technique, Technical Report TRC6/99, School of Computing, National University of Singapore, 1999.
[35] R.L. Iman, J.M. Davenport, Approximations of the critical regions of the friedman statistic, Communications in Statistics 6 (1980) 571–595.
[36] R. Kohavi, G.H. John, Wrappers for feature subset selection, Artificial Intelligence 97 (1997) 273–324.
[37] M. Kubat, R. Holte, S. Matwin, Machine learning for the detection of oil spills in satellite radar images, Machine Learning 30 (1998) 195–215.
[38] L. Kuncheva, L.C. Jain, Nearest neighbor classifier: simultaneous editing and descriptor selection, Pattern Recognition Letters 20 (1999) 1149–1156.
[39] L. Kuncheva, C.J. Whitaker, Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy, Machine Learning 51 (2003) 181–207.
[40] N. Leavitt, Data mining for the corporate masses?, Computer 35 (2002) 22–24
[41] W. Lee, S.J. Stolfo, K.W. Mok, Adaptive intrusion detection: a data mining approach, Artificial Intelligence Review 14 (2000) 533–567.
[42] E. Leopold, J. Kindermann, Text categorization with support vector machines. How to represent texts in input space?, Machine Learning 46 (2002) 423–444
[43] Y.W. Leung, Y.P. Wang, An orthogonal genetic algorithm with quantization for global numerical optimization, IEEE Transactions on Evolutionary Computation 5 (2001) 41–53.
[44] H. Liu, H. Motoda, Feature Selection for Knowledge Discovery and Data Mining, Kluwer, Norvell, USA, 1998.
[45] H. Liu, H. Motoda, On issues of instance selection, Data Mining and Knowledge Discovery 6 (2002) 115–130.
[46] Y. Liu, X. Yao, Q. Zhao, T. Higuchi, Evolving a cooperative population of neural networks by minimizing mutual information, in: Proceedings of the 2001 IEEE Congress on Evolutionary Computation, Seoul, Korea, pp. 384–389.
[47] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Christiani, C. Watkins, Text classification using string kernels, Journal of Machine Learning Research 2 (2002) 419–444.
[48] S. Maldonado, R. Weber, J. Basak, Simultaneous feature selection and classification using kernel-penalized support vector machines, Information Sciences 181 (2011) 115–128.
[49] J. Maudes-Raedo, J.J. Rodríguez-Díez, C. García-Osorio, Disturbing neighbors diversity for decision forest, in: G. Valentini, O. Okun (Eds.), Workshop on Supervised and Unsupervised Ensemble Methods and Their Applications (SUEMA 2008), Patras, Grecia, 2008, pp. 67–71.
[50] P. Mitra, C.A. Murthy, S.K. Pal, Unsupervised feature selection using feature similarity, IEEE Transactions on Pattern Analysis and Machine Intelligence 24 (2002) 301–312.
[51] P. Narendra, K. Fukunaga, Branch, and bound algorithm for feature subset selection, IEEE Transactions, Computer C-26 (1977) 917–922.
[52] K. Ng, H. Liu, Customer retention via data mining, AI Review 14 (1999) 590.
[53] F.J. Provost, V. Kolluri, A survey of methods for scaling up inductive learning algorithms, Data Mining and Knowledge Discovery 2 (1999) 131–169.
[54] J.F. Ramírez-Cruz, O. Fuentes, V. Alarcón-Aquino, L. García-Banuelos, Instance selection and feature weighting using evolutionary algorithms, in: 15th International Conference on Computing, CIC '06, 2006, pp. 73–79.
[55] M. Robnik-Šikonja, I. Kononenko, Theoretical and empirical analysis of relieff and rrelieff, Machine Learning Journal 53 (2003) 23–69.
[56] Y. Rui, T.S. Huang, Image retrieval: current techniques, promising directions and open issues, Journal of Visual Communication and Image Representation 10 (1999) 39–62.
[57] R. Ruiz, Incremental wrapper-based gene selection from microarray data for cancer classification, Pattern Recognition 39 (2006) 2383–2392.
[58] Y. Saeys, T. Abeel, S. Degroeve, Y.V. de Peer, Translation initiation site prediction on a genomic scale: beauty in simplicity, Bioinformatics 23 (2007) 418–423.
[59] M. Sebban, R. Nock, A hybrid filter/wrapper approach of feature selection using information theory, Pattern Recognition 35 (2002) 835–846.
[60] B. Sierra, E. Lazkano, I. Inza, M. Merino, P. Larrañaga, J. Quiroga, Prototype selection and feature subset selection by estimation of distribution algorithms. a case study in the survival of cirrhotic patients treated with TIPS, in: Proceedings of the 8th Conference on Artificial Intelligence in Medicine in Europe, Springer-Verlag, 2001, pp. 20–29.
[61] J.T. de Souza, R.A.F. do Carmo, G.A.L. de Campos, A novel approach for integrating feature and instance selection, in: Proceedings of the 7th International Conference on Machine Learning and Cybernetics, Springer-Verlag, Kunming, China, 2008, pp. 374–379.
[62] T. Welch, Bounds on the Information Retrieval Efficiency of Static File Structures, Technical Report 88, MIT, 1971.
[63] F. Wilcoxon, Individual comparisons by ranking methods, Biometrics 1 (1945) 80–83.
[64] E.P. Xing, M.I. Jordan, R.M. Karp, Feature selection for high-dimensional genomic microarray data, in: The 18th International Conference on Machine Learning, Morgan Kaufmann, 2001, pp. 601–608.
[65] E. Zitzler, L. Thiele, M. Laumanns, C.M. Fonseca, V. Grunert, Performance assessment of multiobjective optimizers: an analysis and review, IEEE Transactions on Evolutionary Computation 7 (2003) 117–132.