A Scalable Memetic Algorithm for Simultaneous Instance and Feature Selection

Nicolás García-Pedrajas npedrajas@uco.es
Department of Computing and Numerical Analysis, University of Cordoba, Córdoba, 14014, Spain

Aida de Haro-García adeharo@uco.es
Department of Computing and Numerical Analysis, University of Cordoba, Córdoba, 14014, Spain

Javier Pérez-Rodríguez javier.perez@uco.es
Department of Computing and Numerical Analysis, University of Cordoba, Córdoba, 14014, Spain

doi:10.1162/EVCO_a_00102

Abstract

Instance selection is becoming increasingly relevant due to the huge amount of data that is constantly produced in many fields of research. At the same time, most of the recent pattern recognition problems involve highly complex datasets with a large number of possible explanatory variables. For many reasons, this abundance of variables significantly harms classification or recognition tasks. There are efficiency issues, too, because the speed of many classification algorithms is largely improved when the complexity of the data is reduced. One of the approaches to address problems that have too many features or instances is feature or instance selection, respectively. Although most methods address instance and feature selection separately, both problems are interwoven, and benefits are expected from facing these two tasks jointly. This paper proposes a new memetic algorithm for dealing with many instances and many features simultaneously by performing joint instance and feature selection. The proposed method performs four different local search procedures with the aim of obtaining the most relevant subsets of instances and features to perform an accurate classification. A new fitness function is also proposed that enforces instance selection but avoids putting too much pressure on removing features. We prove experimentally that this fitness function improves the results in terms of testing error. Regarding the scalability of the method, an extension of the stratification approach is developed for simultaneous instance and feature selection. This extension allows the application of the proposed algorithm to large datasets. An extensive comparison using 55 medium to large datasets from the UCI Machine Learning Repository shows the usefulness of our method. Additionally, the method is applied to 30 large problems, with very good results. The accuracy of the method for class-imbalanced problems in a set of 40 datasets is shown. The usefulness of the method is also tested using decision trees and support vector machines as classification methods.

Keywords

Memetic algorithms, instance selection, feature selection, scaling-up.

1 Introduction

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

Manuscript received: July 7, 2011; revised: January 24, 2012, August 17, 2013, and March 1, 2013; accepted: March 13, 2013.

of data makes most existing algorithms inapplicable to many real-world problems. Two approaches have been used to face this problem: scaling up data mining algorithms (Provost and Kolluri, 1999) and data reduction. However, scaling up a certain algorithm is not always feasible. Data reduction consists of removing missing data, redundant data, information-poor data and/or erroneous data to obtain a tractable problem size.

Instance selection (Liu and Motoda, 2002) consists of choosing a subset of the total available data to achieve the original purpose of the data mining application as successfully as the purpose would have been achieved with the whole dataset. Different variants of instance selection exist.

We can distinguish two main models (Cano et al., 2003): instance selection as a method of prototype selection for algorithms based on prototypes (such as 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 support vector machines).

At the same time, in real-world situations, relevant features are often unknown a priori. Therefore, many candidate features are introduced to better represent the domain. Unfortunately, many of these features are either partially or completely irrelevant or redundant to the target concept.

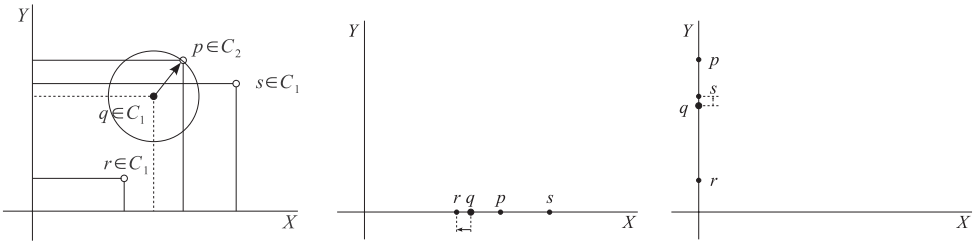
Data mining (Agrawal et al., 1993), as a multidisciplinary joint effort from databases, machine learning, and statistics, is turning mountains of data into nuggets. To use data mining tools effectively, data preprocessing is essential. Feature selection is one of the most important and frequently used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

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 subsets of size m' (Narendra and Fukunaga, 1977). The objectives of feature selection are manifold, with the most important ones being as follows (Saeys et al., 2007):

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

Although most proposed methods for instance selection and for feature selection address either one of the problems, 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 to different subsets of relevant instances. Figure 1 is an illustration. The instances or features selected to classify query instance q depend on each other, as different instances will be the nearest to the query instance in different subspaces. Thus, searching for a subset of relevant instances and features jointly may benefit the overall accuracy of the obtained subset. Furthermore, because the algorithm can remove instances as well as features, its capability for data reduction increases.

Memetic algorithms (Chen et al., 2011) are one of the recent growing areas of research in evolutionary computation. The term memetic algorithm is usually used to refer to an algorithm that combines an evolutionary method, or any population-based



(a) 1-NN rule using inputs x and y.

(b) 1-NN rule using input x.

(c) 1-NN rule using input y.

Figure 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.

approach, together with one, or many, local search procedures that perform an optimization on some, or all, of the individuals of the population. Other terms have been used for these algorithms, such as hybrid genetic algorithms or cultural genetic algorithms. However, as the term memetic algorithm is becoming the most common one, we will use that term throughout the text.

In this paper, we address the simultaneous selection of instances and features by means of a memetic algorithm. The algorithm searches for the optimal combination of a subset of instances and features using the standard operators of selection, mutation, and crossover. Additionally, the algorithm introduces four different local search procedures that locally optimize the individuals of the population searching for better subsets of instances, features, or both. The use of a memetic algorithm, instead of any other genetic algorithm, is based on the well-known problem of evolutionary methods for fine-tuning solutions when no local search is used.

Our preliminary experiments showed that using the standard fitness function for instance and feature selection, which treats the reduction of instances and features on equal terms, caused the elimination of too many features, with a subsequent loss of accuracy. Thus, we also propose a new fitness function that corrects this undesired effect. The fitness function enforces the suppression of the worst features, but makes removing more features increasingly difficult, unless their elimination contributes positively to the accuracy.

As a third relevant aspect of our method, we have considered its scalability. As it will be shown, the proposed memetic algorithm achieves very good performance. However, as in almost all evolutionary algorithms, its computational cost is high, making its application to large problems difficult. To deal with this scalability problem, Cano et al. (2005) developed a method called stratification for evolutionary instance selection. In this work, we extend this concept of stratification to the problem of simultaneous instance and feature selection. The experimental results will show the efficiency of the method in scaling up the memetic algorithm and in keeping the overall performance.

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

2 Related Work

The problem of instance selection for instance-based learning can be defined as "the isolation of the smallest set of instances that enable us to predict the class of a query instance with the same (or higher) accuracy than the original set" (Brighton and Mellish, 2002).

It has been shown that different groups of learning algorithms need different instance selectors to suit their learning/search bias (Brodley, 1995). This may render many instance selection algorithms useless, if their philosophy of design is not suitable for the problem at hand. Our algorithm does not assume any structure of the data or any behavior of the classifier; instead, our algorithm adapts the instance and feature selection to the performance of the classifier.

Brighton and Mellish (2002) argued that the structure of the classes formed by the instances can be very different; thus, an instance selection algorithm can have a good performance in one problem and be very inefficient in another. They state that the instance selection algorithm must gain some insight into the structure of the classes to perform an efficient instance selection. However, this insight is usually not available or is very difficult to acquire, especially in real-world problems with many variables and with complex boundaries between the classes. In such a situation, an approach based on evolutionary computation may be of help. The approaches based on evolutionary computation do not assume any special form of the space, classes, or boundaries between the classes; they are only guided by the ability of each solution to solve the task. In this way, the algorithm learns the relevant instances from the data without imposing any constraints on the form of the classes or the boundaries between them. Feature selection is a different task from feature construction (Neshatian et al., 2012) where new features are obtained.

Feature selection has been a fertile field of research and development since the 1970s, appearing in statistical pattern recognition (Jain and Zongker, 1997; Mitra et al., 2002), machine learning (Blum and Langley, 1997; Kohavi and John, 1997; Neshatian and Zhang, 2011), and data mining (Dash et al., 2002; Kim et al., 2000), and widely applied to many fields such as classification (Xue et al., 2012, 2013), text categorization (Leopold and Kindermann, 2002; Nigam et al., 1999), image retrieval (Rui and Huang, 1999; Swets and Weng, 1995), customer relationship management (Ng and Liu, 1999), intrusion detection (Lee et al., 2000), visual learning (Krawiec and Bhanu, 2007), and genomic analysis (Xing et al., 2001).

Although the literature of instance selection (Liu and Motoda, 2002) and feature selection (Liu and Motoda, 1998) is vast, attempts to perform both tasks jointly are far less common. One of the first works was carried out by Kuncheva and Jain (1999), who performed the selection of instances and features at the same time using a genetic algorithm. A similar idea was used by Ishibuchi and Nakashima (2000) for a nearest neighbor classifier, and by Ishibuchi et al. (2001) for an artificial neural network.

The intelligent multiobjective evolutionary algorithm (IMOEA; Chen et al., 2005) method was proposed for performing instance and feature selection jointly. This method is a multi-objective evolutionary algorithm that considers not only instance selection but also feature selection. The algorithm has three objectives: maximization of training accuracy, minimization of the number of instances selected, and minimization of the features selected. The multi-objective algorithm used is based on Pareto dominance, as is common in multi-objective algorithms (Zitzler et al., 2003). 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 (Leung and Wang, 2001) to estimate the contribution of each gene to the fitness of the individuals.

Teixeira de Souza et al. (2008) proposed the use of simulated annealing for this task. More recently, Derrac et al. (2010a) proposed a cooperative approach for the simultaneous selection of instances and features. In their model, three populations coevolve together, one in which instances are selected, another in which features are selected, and a third one in which both instances and features are selected. There have also been other works focused on the simultaneous selection of instances and the weighting of features (Yu et al., 2003).

In a recent paper, Zhang et al. (2012) proposed a unified approach for both instance and feature selection. In that paper, a greedy approach was proposed to solve the combined problem of selecting the most informative instances and features. However, Zhang et al.'s approach is more a filter method, whereas our approach considers the classifier we will use once the selection process is finished.

To the best of our knowledge, no other memetic algorithm has been proposed for the simultaneous selection of instances and features.

3 Memetic Algorithm for Instance and Feature Selection

The codification of the individuals is straightforward. The selection of instances and features will be represented by a binary individual with length $l = n + m$, where n is the number of instances and m the number of features. This coding is used in almost all applications of genetic algorithms to instance or feature selection.

The first decision that we made in the design of the memetic algorithm was to select which basic genetic algorithm approach to use. Most of the many different variants of evolutionary methods have been applied to instance and feature selection, considering this task to be a search problem. The application is easy and straightforward. Each individual is a binary vector that codes a certain sample of the training set and the feature set. The evaluation is usually made by considering both the data reduction and the classification accuracy. Examples of applications of genetic algorithms to instance selection can be found in Kuncheva (1995), Ishibuchi and Nakashima (2000), and Reeves and Bush (2001).

Cano et al. (2003) performed a comprehensive comparison of the performance of different evolutionary algorithms for instance selection. They compared a generational genetic algorithm (Goldberg, 1989), a steady-state genetic algorithm (Whitley, 1989), a CHC genetic algorithm (Eshelman, 1990; where CHC stands for *cross generational elitist selection, heterogeneous recombination, and cataclysmic mutation*), and a population-based incremental learning algorithm (Baluja, 1994). They found that evolutionary-based methods outperformed classical algorithms in both classification accuracy and data reduction. Among the evolutionary algorithms, CHC achieved the best overall performance. Furthermore, our own previous experience (de Haro-García and Pedrajas, 2009; García-Osorio et al., 2010; García-Pedrajas et al., 2010) has also shown that the CHC algorithm was the best option. As a result, we based our memetic algorithm on a CHC method. The non-traditional CHC genetic algorithm differs from traditional genetic algorithms in several ways, as follows (Louis and Li, 1997).

1. To obtain the next generation for a population of size N, the parents and the offspring are considered together and the N best individuals are selected.

2. To avoid premature convergence, only different individuals, separated by a threshold Hamming distance, $d_{\min}$, are allowed to mate. The threshold distance is set to $d_{\min} = l/4$, where $l = m + n$ is the length of the individual. If no individuals with a Hamming distance above $d_{\min}$ are found in a generation, and hence no matings are performed, then the threshold is decreased by 1.
3. Crossover operator HUX (for half uniform crossover) is used (Eshelman, 1990). HUX crossover mates two parents to obtain two offspring. Both offspring inherit the matching bits of the two parents. The non-matching bits are inherited by the offspring in turn.
4. Mutation is not used during regular evolution. To avoid premature convergence or stagnation of the search, the population is reinitialized when the individuals are not sufficiently diverse. In such a case only the best individual is kept in the new population. The population is considered not diverse enough when $d_{\min} = 1$.

The second decision is the selection of the local search procedures to apply. Our algorithm faces two different tasks at the same time: instance and feature selection. Thus, we designed the algorithm using three different local search methods: a method that selects features, a method that selects instances, and a method that selects both instances and features.[1] All three methods are designed with one common constraint, that the local search procedure may remove instances or features, but cannot add them. With this constraint, the result of a local search procedure is always an individual with lower or equal storage requirements. The three local search procedures are feature selection local search (FSLS), instance selection local search (ISLS), and instance and feature selection local search (IFSLS).

1. *Feature selection local search*. This procedure begins the search with the selection of instances and features given by a certain individual. Then, in turn, each feature is removed and the fitness of the individual is reevaluated. If the fitness is equal to or better than the fitness with the feature selected, the feature is removed permanently; otherwise, the feature is kept. This search can only produce individuals that have a higher fitness than the initial point of the search. To avoid any effect derived from the order of the features, the features are removed in random order.
2. *Instance selection local search*. For an instance selection local search, we tried different known classical instance selection methods, as they are fast and have good performance. We tried Drop3 (Wilson and Martinez, 2000), ICF (Brighton and Mellish, 2002), RNN (Gates, 1972), MSS (Barandela et al., 2005), and IB3 (Aha et al., 1991). To allow a faster execution, all of them were applied considering only the selected instances. ICF, Drop3, and IB3 achieved the best results. We chose IB3 because it was the fastest of the three with the best results. In contrast to the other two local search methods, IB3 can obtain individuals with a worse fitness. In such a case, the individual is not added to the population, and the original individual is kept.
3. *Instance and feature selection local search*. This procedure begins the search with the selection of instances and features given by a certain individual. Then, in turn, each instance is removed and the fitness of the individual is reevaluated. If the

[1]Many other local search methods were tried that were less efficient and are not reported here.

fitness is equal to or better than the fitness with the instance selected, then the instance is removed permanently; otherwise, the instance is kept. This search can only produce individuals that have a higher fitness than the initial point of the search. To avoid any effect derived from the order of the instances, the instances are removed in random order. Once the search in the space of instances is finished, the same procedure is repeated for the features. This method has an important advantage from the point of view of the efficiency of the algorithm. When an instance is removed, only the instances that have this instance as one of their nearest neighbors must be reclassified. In most cases, there are only a few instances to reclassify and the local search is performed very efficiently.

The application of these three procedures always obtains individuals with fewer instances/features selected. In many datasets, the reduction produced very good individuals in terms of storage requirements, but the testing error was damaged. Thus, we designed a fourth local search method, which acts in the opposite way of the three described methods.

4. *Instance and feature addition local search (IFALS).* This procedure begins the search with the selection of instances and features given by a certain individual. Then, in turn, each instance not selected is added and the fitness of the individual is reevaluated. If the fitness is better than the fitness with the instance removed, the instance is added permanently; otherwise, the instance is removed again. This search can only produce individuals that have a higher fitness than the initial point of the search. To avoid any effect derived from the order of the instances, the instances are added in random order. Once the search in the space of instances is finished, the same procedure is repeated for the features. As adding an instance or feature has a negative effect on the reduction part of the fitness function, that negative effect must be counteracted by an improvement in the accuracy. Thus, only instances or features that contribute positively to the training error will be added permanently.

These local search operators are similar to sequential search, which has been used in selection problems such as feature selection (Pudil et al., 1994) and instance selection (Olvera-López et al., (2010). In sequential search, a discarded instance cannot be reconsidered. This drawback can be solved when the selection is done in the forward and backward directions. Our operators allow for reconsidering of the deletion of addition of one instance/feature just after it is removed or added. The deletion/addition of new instances/features cannot affect the instances/features previously considered. This simplification makes the local search procedures faster. However, it is a suboptimal solution. This drawback is compensated for by the evolutionary algorithm which will reconsider instances/features wrongly removed or added during the evolution of the individuals. In such a way, our local search operators would not be useful as standalone procedures. Our memetic operators have similarities with forward selection and backward elimination methods (Mao, 2004). Thus, they may suffer the same drawbacks as these methods, such as selecting redundant instances or features or being trapped in local optima. However, their use as local search operators within a memetic algorithm framework alleviates these problems and makes them useful, as is shown in the experimental results. Furthermore, the inherent parallel nature of the search performed by the memetic algorithms reduces the risk of being trapped in local optima, which is higher in forward selection and backward elimination methods.

Some of the drawbacks of forward selection and backward elimination methods can be ameliorated with more complex search procedures such as the methods of *plus-l-take-away-r* of floating search (Pudil et al., 1994). However, these methods severely increment the computational cost, making their scalability a relevant problem.

One of the most important issues in memetic algorithms is when to apply a local search and to which individuals. It has been shown that by applying a local search to a small percentage of individuals, important improvements might be achieved. Different methods have been proposed (Lozano et al., 2004) that are based on applying a local search to certain members of the population. Our approach is that of applying a local search in the same way that mutation is applied. Some members of the population are selected and a local search is applied. Each one of the four previous procedures is applied with a 10% probability to any member of the population. As the different local search algorithms are performed sequentially, the same individual may be subject to a local search more than once in the same generation. This method is similar to the idea of the local search chain proposed by Molina et al. (2010).

3.1 Fitness Function

The usual fitness functions for instance selection (Cano et al., 2003) and feature selection (Guyon and Elisseeff, 2003), or for simultaneous feature and instance selection (Derrac et al., 2010a), are based on two terms: training error and reduction. In the few papers that have addressed the simultaneous selection of features and instances (Chen et al., 2005; Derrac et al., 2010a), the reduction of instances and features was treated on equal terms. However, our preliminary experiments showed that the impact of removing an instance or a feature is very different, and must be considered differently.

In standard practice, we have a fitness function based on reduction, r_i, and training error, e_i, where the reduction is considered regardless of whether or not we are removing instances or features:

$$\text{fitness}_i = \alpha(1 - e_i) + (1 - \alpha)r_i, \tag{1}$$

where $0 < \alpha < 1$. The most common case is having a number of instances that is far larger than the number of features. A negative effect appears from that situation. Removing a feature has a major impact on r_i, which means that unless the training error is clearly worse, the fitness of the individual with the removed feature is better than the individual with the feature retained. The result is that features are removed frequently and very poor individuals are obtained. The only way to avoid that result is to increase the value of α, but then we have the problem of harming the reduction performance of the memetic algorithm.

To avoid this issue, we have opted for a different approach. First, we consider reduction in terms of instances and features separately. However, this alone does not solve the problem, as the reduction of features still has too much influence on the fitness.[2] Second, we aim at different targets when optimizing the reduction of instances and features. For the reduction of instances, we use the standard method of considering the fraction of instances removed as a measure of reduction.

However, for feature selection, we use a different approach. If we study the results of feature selection algorithms, we often see that removing a subset consisting of the worst features has a small impact on the performance of the classifier, and usually even improves the accuracy. However, once the first few worst features have been removed,

[2]Reducing the weight of the term of feature reduction is not a solution either.

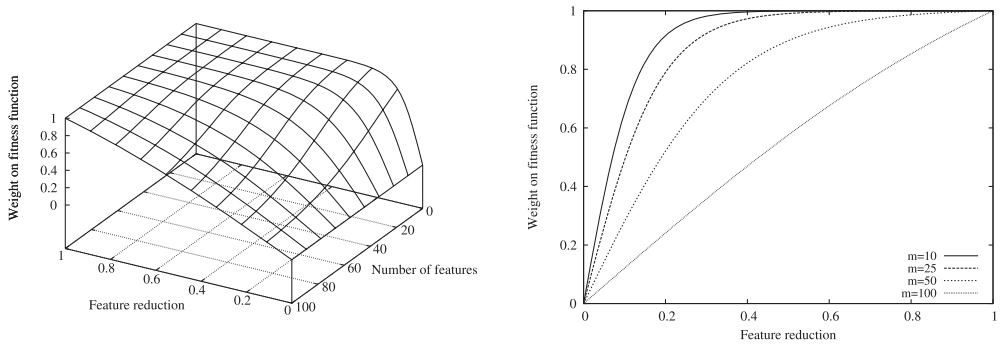


Figure 2: Weight of the feature selection factor in the fitness function, as a function of the fraction of selected instances, f_f, and the number of features, m, from 0 to 100, (a), and for values of $m = \{10, 25, 50, 100\}$, (b).

it is increasingly difficult to remove more features without damaging the performance. This effect is more marked when the number of features is small. Thus, we want a term in the fitness function for measuring feature reduction that allows removing the worst instances, but that makes it increasingly hard for the algorithm to remove more features unless the training error is not damaged.

We use a fitness value for an individual i, fitness_i, given by:

$$\text{fitness}_i = \alpha_e(1 - e_i) + \alpha_{f_i}(1 - f_i) + \alpha_{f_f} f(f_f, m), \tag{2}$$

where f_i is the fraction of selected instances, f_f is the fraction of selected features, and m is the number of features, $\alpha_e + \alpha_{f_i} + \alpha_{f_f} = 1$. In all of the reported experiments, we have set these values to: $\alpha_e = 0.5, \alpha_{f_i} = \alpha_{f_f} = 0.25$, thus weighting accuracy and reduction equally. $f(\cdot)$ is the smoothing function that must work as explained. Experimentally we chose the function:

$$f(x, y) = \frac{\tanh(10 \exp(-0.25y)x)}{M}, \tag{3}$$

where $M = \tanh(10 \exp(-0.25y))$ is a scaling factor, to ensure that f is in the interval $[0, 1]$. A plot of this function is shown in Figure 2. The constants have been manually set to obtain the desired smoothing function. How the fitness term of feature selection works is better shown in Figure 2(b). We show the behavior of the feature reduction for four different numbers of features. If the number of features is small, then we can see how removing the first few features increases the fitness. However, removing new features has a very limited effect on the fitness function. The new features will only be removed when they do not negatively affect the accuracy. As the number of features in the problem grows, the behavior is smoothed. When we have many features, we can expect to be able to remove more features, as more redundancy might be expected.

As we have three different terms in the fitness function, a multi-objective optimization approach may also be used (Deb, 2001). However, an algorithm focused on searching for Pareto individuals would not be appropriate, because each individual must achieve some goals in each objective to be useful. For example, a nondominated individual with large reduction but low accuracy would not be a useful solution. In such a case, we would need to resort to a multi-objective algorithm with goals and priority

Algorithm 1 Memetic algorithm for simultaneous instance and feature selection

Data : A training set $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, population size, s, and number of evaluations, n_e.
Result: The set of selected instances and features $T_S \subset T$.
Randomly initialize the population
Evaluate fitness of all individuals
while *evaluations* $< n_e$ **do**
  Apply HUX crossover to obtain s new individuals, creating a population of $2s$ individuals
  Evaluate fitness of new individuals
  Apply FSLS with probability P_{FSLS} to the $2s$ individuals
  Apply ISLS with probability P_{ISLS} to the $2s$ individuals
  Apply IFSLS with probability P_{IFSLS} to the $2s$ individuals
  Apply IFALS with probability P_{IFALS} to the $2s$ individuals
  Evaluate new individuals
  Select best s individuals from the population of size $2s$.
end
return T_S of the best individual
NOTE: After each local search or evaluation of an individual, if the number of evaluations exceeds the limit, then the algorithms is finished.

specifications (Tan et al., 2003). The use of that kind of algorithm is an interesting future line of research.

To evaluate e_i, we use a 1-NN classifier. The training error, for instance $\mathbf{x}$, using an individual that selects a subset of instances S is evaluated using as prototype set $S \backslash \{\mathbf{x}\}$ and the subspace selected by the individual. Algorithm 1 shows a pseudocode of the complete memetic algorithm.

3.2 Scaling Up the Method

One of the problems with the proposed memetic algorithm is its computational cost. Although the algorithm performs comparatively better compared to other methods in terms of runtime, it will be difficult to address problems with hundreds of thousands of instances. Cano et al. (2005) proposed a stratified approach to deal with huge datasets. This stratification strategy splits the training data into disjoint strata with equal class distribution.[3] The training data, $\mathbb{T}$, is divided into t disjoint datasets, D_j, of approximately equal size:

$$\mathbb{T} = \bigcup_{j=1}^{t} D_j. \tag{4}$$

Next, the evolution is applied to each subset separately and the results of all such applications are combined for the final solution. If we have an algorithm of quadratic complexity, $O(n^2)$, in the number of instances, n, and we employ T strata, we will have a time complexity $O(n/T)^2$. Because we have to apply the method to the T strata, the resulting complexity is $O(n^2/S)$, with runs S times faster than the original algorithm.

[3]When dealing with class-imbalanced problems, the distribution of classes in each stratum may be modified with respect to the whole training set, to obtain a less skewed distribution.

If we are able to run the algorithm in parallel in the strata, our complexity will be $O(n^2/T^2)$, with a speedup of T^2. Of course, the drawback of this approach is likely to be a loss in the performance of the algorithm. Derrac et al. (2010b) used stratification to scale up a steady-state memetic algorithm for instance selection. These authors found that although stratification decreased the accuracy of standard instance selection methods, namely DROP3 and FCNN, the approach based on a memetic algorithm and stratification was able to beat the performance of DROP3 (Wilson and Martinez, 2000) and FCNN (Angiulli, 2007) on the whole dataset. These results support the fact that an evolutionary algorithm applied in a suboptimal way to allow the scaling up of the method can still improve the results of classical methods applied to the whole dataset. Similar principles can be applied to other evolutionary methods and problems (García et al., 2008). One of the problems with stratification is the size of the strata. Derrac et al. (2010b) found experimentally that, as expected, a larger stratum obtained a better performance. However, the differences in performance obtained with different stratum size were not marked.

We have extended this approach to simultaneous instance and feature selection. The previous approach considered each stratum separately, and the subset of selected instances is the aggregation of the selected instances from each stratum. However, when features are also selected, the situation is different, as all features are present in all strata.[4] Thus, a certain form of combination of results must be devised. To maintain the advantages of stratification, that would be precluded if a complex method of combination were applied, we considered the three simplest ways of combining the results. After applying the memetic algorithm to T strata, a certain feature, i, has been selected n_i times. The first method is based on an OR combination, which means that any feature selected at least once is selected for the final result of the algorithm; that is, i is selected if $n_i \geq 1$. The problem with this approach is that almost all features are selected. A second method is an AND combination, which means that any feature selected in every stratum is selected for the final result of the algorithm; that is, i is selected if $n_i = T$. The problem with this approach is that almost no features are selected. A compromise solution is a "majority vote," in which a feature is selected for the final result of the algorithm if the feature is selected in at least half of the strata; that is, i is selected if $n_i \geq T/2$. This method is the one used in the experiments, and we will show that very good results were achieved.

4 Experimental Setup

To make a fair comparison between the standard algorithms and our proposal, we selected 55 problems from the UCI Machine Learning Repository (Bache and Lichman, 2013). A summary of these datasets is shown in Table 1. This first set of problems was intended to evaluate our proposal in small to medium sized problems. For estimating the storage reduction and generalization error, we used a 10-fold cross-validation method. In some plots throughout the paper, we use the number of order of each dataset shown in the table, instead of its name, to reduce the size needed by the graphs. Other datasets will be presented for testing the scalability of our proposal and its performance for class-imbalanced datasets.

It is difficult to establish a framework for comparing all of the methods under the same conditions. It is common practice to compare methods for the same number of

[4]If the strata are formed by subsets of features as well as subsets of instances, the achieved performance is poor.

Table 1: Summary of datasets.

			Variables				
	Dataset	Instances	Continuous	Binary	Nominal	Classes	Features
1	anneal	898	6	14	18	5	59
2	arrhythmia	452	279	—	—	13	279
3	audiology	226	—	61	8	24	93
4	autos	205	15	4	6	6	72
5	breast-cancer	286	—	3	6	2	15
6	cancer	699	9	—	—	2	9
7	car	1,728	—	—	6	4	16
8	card	690	6	4	5	2	51
9	dermatology	366	1	1	32	6	34
10	ecoli	336	7	—	—	8	7
11	euthyroid	3,163	7	18	—	2	44
12	german	1,000	6	3	11	2	61
13	glass	214	9	—	—	6	9
14	glass-g2	163	9	—	—	2	9
15	heart	270	13	—	—	2	13
16	heart-c	302	6	3	4	2	22
17	hepatitis	155	6	13	—	2	19
18	horse	364	7	2	13	3	58
19	hypothyroid	3,772	7	20	2	4	29
20	ionosphere	351	33	1	—	2	34
21	kr vs. kp	3,196	—	34	2	2	38
22	led24	200	—	24	—	10	24
23	liver	345	6	—	—	2	6
24	lrs	531	101	—	—	10	101
25	lymphography	148	3	9	6	4	38
26	mammography	961	2	—	3	2	18
27	mfeat-fac	2,000	216	—	—	10	216
28	mfeat-fou	2,000	76	—	—	10	76
29	mfeat-kar	2,000	64	—	—	10	64
30	mfeat-mor	2,000	6	—	—	10	6
31	mfeat-pix	2,000	240	—	—	10	240
32	mfeat-zer	2,000	47	—	—	10	47
33	new-thyroid	215	5	—	—	3	5
34	optdigits	5,620	64	—	—	10	64
35	ozone1hr	2,536	72	—	—	2	72
36	ozone8hr	2,534	72	—	—	2	72
37	page-blocks	5,473	10	—	—	5	10
38	phoneme	5,404	5	—	—	2	5
39	pima	768	8	—	—	2	8
40	primary-tumor	339	—	14	3	22	23
41	promoters	106	—	—	57	2	114
42	satimage	6,435	36	—	—	6	36
43	segment	2,310	19	—	—	7	19
44	sick	3,772	7	20	2	2	33
45	soybean	683	—	16	19	19	82
46	texture	5,500	40	—	—	11	40
47	tic-tac-toe	958	—	—	9	2	9

Table 1: Continued.

	Dataset	Instances	Variables			Classes	Features
			Continuous	Binary	Nominal		
48	titanic	2,201	—	—	3	2	8
49	vehicle	846	18	—	—	4	18
50	vote	435	—	16	—	2	16
51	vowel	990	10	—	—	11	10
52	waveform	5,000	40	—	—	3	40
53	wine	178	13	—	—	3	13
54	yeast	1,484	8	—	—	10	8
55	zoo	101	1	15	—	7	16

evaluations of the fitness function. However, when a local search is applied, it does not need to evaluate the fitness function in many cases, so any memetic algorithm would benefit from such a comparison. To account for all of the differences of the methods, we have to consider full and partial evaluations of the fitness function. When a local search method is applied, any evaluation of a subset of instances is considered and counted as a partial evaluation of the fitness function. In that way, the computational cost of the algorithms compared is as close as possible.

A second issue is the number of evaluations the algorithm performs. A small number of evaluations will end in a low performance that can preclude the conclusions of the comparison. To avoid that effect, we have stopped the evolution after 100,000 fitness evaluations, which is even above the standard limit for evolutionary algorithms for instance selection (Cano et al., 2003), and large enough to allow the algorithms to converge. A population size of 100 individuals was used. The local search methods were applied to 10% of the individuals of the population, which were randomly selected. For the initial population, instances and features were randomly selected with a probability of 50%. All of the common parameters for the different algorithms were always set to the same values.

We have used as our main comparing algorithm a CHC genetic algorithm without local search, as in previous works (Cano et al., 2003) this algorithm has proven to be the best performing method when compared to other evolutionary algorithms and classical instance selection methods. In this first set of experiments, we have also used IMOEA, which has also shown very good performance in terms of both testing error and reduction (Chen et al., 2005; García-Pedrajas et al., 2010).

Statistical tests are a very important issue in any experimental setup. We have used the Wilcoxon test as the main statistical test for comparing pairs of algorithms. This test was chosen because it assumes limited commensurability and is safer than parametric tests, as it does not assume normal distributions or homogeneity of variance. Thus, this test can be applied to error ratios and storage requirements. Furthermore, empirical results (Demšar, 2006) show that this test is also stronger than other tests.

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

$d_i = 0$ are split evenly among the sums:

$$R^+ = \sum_{d_i>0} \text{rank}(d_i) + \frac{1}{2}\sum_{d_i=0} \text{rank}(d_i), \tag{5}$$

and,

$$R^- = \sum_{d_i<0} \text{rank}(d_i) + \frac{1}{2}\sum_{d_i=0} \text{rank}(d_i). \tag{6}$$

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

$$z = \frac{T - \frac{1}{4}N(N+1)}{\sqrt{\frac{1}{24}N(N+1)(2N+1)}} \tag{7}$$

is distributed approximately according to $N(0, 1)$. In the result tables we show the values of R^+ and R^- together with the p-value of the test.

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

5 Experimental Results

Our first set of experiments was aimed at studying the performance of our memetic algorithm in terms of the testing error and the storage reduction. Thus, we ran our method for the 50 datasets of our benchmark set. The results are shown in Tables 2 and 3 for the standard method and our memetic approach, respectively, together with the results of a standard CHC algorithm. As an initial comparison, a good baseline measure for evaluating the performance of the proposed model is the testing error of a nearest neighbor classifier, 1-NN, using all of the instances and features. The win/loss record[5] of such a comparison is 35/19, in favor of our memetic algorithm. The differences are significant according to the Wilcoxon test, with a p-value of .0101. This result means that our approach achieved a significant improvement of the testing error of the 1-NN classifier, while at the same time obtained a reduction that on average was almost 98% of the data. This result proved the efficiency of the proposed method.

Once we validated the method against 1-NN, we compared it to other similar approaches. The results using a standard CHC and our proposal shown in Tables 2 and 3 are plotted in Figure 3(a). The figure shows the results for the testing error and the storage requirements. Throughout the paper, we will use this graphic representation, based on the κ-error relative movement diagrams (Maudes-Raedo et al., 2008). However, here, instead of the κ difference value, we will use the storage difference. The idea of these diagrams is to represent with an arrow the results of two methods applied to the same dataset. The arrow starts at the coordinate origin, and the coordinates of the tip of the arrow are given by the difference between the errors and the storage requirements of our method and those of the standard instance selection algorithm. The numbers indicate the dataset according to Table 1. These graphs are a convenient way to summarize the results. A positive value in either the storage or the testing error means that our method performed better. Thus, arrows pointing up and to the right represent datasets for which our method outperformed the standard algorithm in both

[5]Draws are not shown as they are implicit in the win/loss record.

Table 2: Summary of the performance of standard CHC instance and feature selection method in terms of testing error and storage reduction. Additionally, the κ measure, the number of selected instances and features, and the execution time are shown.

	Standard CHC IS+FS						
Dataset	Training error	Storage	Testing error	κ	Instances selected	Features selected	Time (s)
anneal	0.0075	0.0242	0.0135	0.9791	189.6	6.1	4,459.5
arrhythmia	0.1681	0.0428	0.3045	0.4520	122.8	39.9	6,178.5
audiology	0.1152	0.0505	0.2864	0.7086	64.4	14.8	565.2
autos	0.0843	0.0361	0.1950	0.7967	84.9	5.7	379.2
breast-cancer	0.2361	0.0170	0.2643	0.1879	32.8	2.0	108.7
cancer	0.0364	0.0366	0.0536	0.8503	103.9	2.0	479.9
car	0.1704	0.0593	0.2581	0.1445	362.0	4.0	5,909.7
card	0.1374	0.0071	0.1551	0.6764	118.2	1.7	1,702.0
dermatology	0.0161	0.0317	0.0278	0.9160	51.5	6.9	457.2
ecoli	0.1396	0.0648	0.2606	0.5909	68.7	2.0	128.1
euthyroid	0.0199	0.0171	0.0285	0.7670	1,070.7	2.0	46,704.7
german	0.2391	0.0228	0.2920	0.1953	287.0	4.3	5,691.1
glass	0.1311	0.0840	0.3381	0.5705	55.4	2.6	59.3
glass-g2	0.0728	0.0514	0.1875	0.5013	30.9	2.2	29.8
heart	0.1399	0.0343	0.1555	0.4618	36.1	3.0	103.0
heart-c	0.1449	0.0179	0.1567	0.5399	35.7	3.0	176.4
hepatitis	0.1243	0.0196	0.2267	0.1259	20.3	2.4	33.9
horse	0.1747	0.0360	0.3778	0.2710	122.8	5.6	918.6
hypothyroid	0.0259	0.0136	0.0387	0.6878	1,342.0	1.0	43,644.6
ionosphere	0.0383	0.0211	0.1086	0.6932	75.0	3.0	405.8
kr vs. kp	0.0588	0.0335	0.0574	0.7243	850.7	4.3	43,518.6
led24	0.2544	0.0438	0.3150	0.5573	37.0	5.1	88.4
liver	0.2791	0.0573	0.4294	0.0373	77.1	1.4	109.1
lrs	0.0525	0.0210	0.1283	0.7951	178.4	5.7	2,858.6
lymphography	0.0866	0.0350	0.2786	0.4861	23.5	7.6	85.2
mammography	0.1704	0.0153	0.2010	0.4473	157.6	1.5	1,377.7
mfeat-fac	0.0162	0.0323	0.0480	0.9526	805.7	15.6	110,028.5
mfeat-fou	0.0775	0.0535	0.1740	0.8150	779.0	9.4	30,632.1
mfeat-kar	0.0188	0.0603	0.0540	0.9420	694.6	10.0	24,215.8
mfeat-mor	0.2453	0.1174	0.4045	0.4844	633.9	2.0	4,538.4
mfeat-pix	0.0227	0.0510	0.0525	0.9481	805.5	27.4	147,955.8
mfeat-zer	0.1125	0.0853	0.2220	0.7404	708.4	10.2	28,596.6
new-thyroid	0.0655	0.0243	0.0762	0.7570	23.6	1.0	30.4
optdigits	0.0234	0.0949	0.0537	0.9426	2,477.0	12.4	366,791.4
ozone1hr	0.0288	0.0049	0.0289	0.0353	802.9	1.0	39,693.0
ozone8hr	0.0547	0.0079	0.0648	0.1028	992.1	1.3	44,038.2
page-blocks	0.0557	0.0350	0.0629	0.5580	1,721.3	1.0	37,852.7
phoneme	0.1419	0.0886	0.2743	0.3224	2,154.9	1.0	32,530.4
pima	0.2230	0.0383	0.2895	0.2272	212.2	1.0	666.7
primary-tumor	0.5317	0.0395	0.5940	0.2346	54.1	5.1	277.1
promoters	0.0052	0.0318	0.2300	0.5829	15.9	22.9	116.7
satimage	0.0848	0.0553	0.1530	0.8068	2,812.0	4.1	221,031.8
segment	0.0130	0.0527	0.0338	0.9650	694.2	3.0	14,693.4
sick	0.0336	0.0101	0.0371	0.5061	1,131.8	1.0	44,622.2

Table 2: Continued.

	Standard CHC IS+FS						
Dataset	Training error	Storage	Testing error	κ	Instances selected	Features selected	Time (s)
soybean	0.0447	0.0405	0.0647	0.9234	177.5	11.5	4,182.8
texture	0.0151	0.0698	0.0351	0.9701	2,341.6	5.9	188,695.8
tic-tac-toe	0.3007	0.0183	0.3000	0.1165	142.0	1.0	806.7
titanic	0.3037	0.0642	0.3099	0.0000	557.3	1.8	5,615.6
vehicle	0.1568	0.0745	0.3298	0.5539	292.1	3.5	1,935.6
vote	0.0436	0.0076	0.0442	0.8112	47.8	1.0	226.8
vowel	0.0392	0.1514	0.4222	0.5444	337.2	4.0	1,710.3
waveform	0.1094	0.0826	0.2300	0.6294	2,066.6	7.2	127,755.2
wine	0.0130	0.0270	0.0706	0.8355	21.3	2.7	44.0
yeast	0.4548	0.0788	0.5831	0.1559	456.4	1.8	2,590.6
zoo	0.0330	0.0444	0.0800	0.8342	15.4	4.2	18.5

the error and the storage. Arrows pointing up and to the left indicate that our algorithm improved the storage but had a worse testing error, while arrows pointing down and to the right indicate that our algorithm improved the testing error but had a worse storage reduction. Arrows pointing down and to the left indicate that our algorithm was worse in both the testing error and the storage reduction.

Because any partial or full evaluation made by the local search is included in the count of the number of evaluations of the fitness functions, the computational cost of the three algorithms, CHC IS+FS, IMOEA, and our memetic approach, is approximately the same, and the comparison is a fair one. Table 4 shows a comparison of our memetic algorithm and the two reference methods, CHC and IMOEA. Table 4 shows a clear advantage of our proposal in both the testing error and the reduction. The Wilcoxon test finds significant differences for both terms, the testing error and the storage requirements, and the two algorithms.

An interesting result is shown in Tables 2 and 3, which show the different behaviors or our memetic algorithm and the standard CHC. Our memetic algorithm achieved a better reduction, with a win/loss record of 48/7. However, this improvement is not reached by a similar improvement in instances and features reduction. In fact, our algorithm removed fewer features but retained 10 times fewer instances. We think that this is one of the sources of its good results in terms of the testing error.

A second important issue is the runtime of the algorithms. Although the number of evaluations is the same, there are other parts of the implementation that can have an impact on the time used by the algorithm. For example, an individual is evaluated faster if it selects fewer instances or features. Figure 4 shows the difference between the execution time, on a logarithmic scale, of our method and CHC IS+FS. A filled bar means that our algorithm is faster, and an empty bar means that our algorithm is slower. In the plot, we can only see filled bars, meaning that our algorithm is faster for all datasets. Furthermore, the gain in runtime is very significant for the most costly datasets. The explanation is in the use of the new fitness function and the local search. The evaluation of an individual has a time complexity quadratic in the number of instances. Fewer instances mean a faster evaluation of the fitness function. Our memetic algorithm is

Table 3: Summary of the performance of memetic instance and feature selection method in terms of testing error and storage reduction. Additionally, the κ measure, the number of selected instances and features, and the execution time are shown.

	Memetic IS+FS						
Dataset	Training error	Storage	Testing error	κ	Instances selected	Features selected	Time (s)
anneal	0.0161	0.0027	0.0214	0.9362	22.1	5.7	1,157.2
arrhythmia	0.2224	0.0035	0.2867	0.3564	24.4	16.5	1,581.2
audiology	0.2201	0.0084	0.2773	0.5575	21.2	7.5	175.9
autos	0.1551	0.0232	0.3150	0.7864	39.6	7.8	174.0
breast-cancer	0.2217	0.0090	0.3214	0.0927	6.4	5.0	42.6
cancer	0.0235	0.0050	0.0420	0.9180	5.9	4.8	143.7
car	0.0591	0.0282	0.2157	0.7478	81.2	8.6	1,897.9
card	0.1221	0.0017	0.1464	0.6039	10.7	5.1	327.9
dermatology	0.0121	0.0187	0.0361	0.9392	16.4	12.8	164.9
ecoli	0.1036	0.0465	0.1667	0.7157	20.4	4.8	43.9
euthyroid	0.0243	0.0003	0.0247	0.7675	9.9	3.5	7,262.1
german	0.2123	0.0047	0.2700	0.2026	25.1	9.6	931.3
glass	0.1596	0.0707	0.3048	0.6005	25.4	4.6	33.1
glass-g2	0.0592	0.0618	0.1688	0.6506	18.5	4.4	17.6
heart	0.1070	0.0253	0.1926	0.5393	10.7	7.4	36.5
heart-c	0.1110	0.0161	0.1633	0.4855	9.8	9.6	58.8
hepatitis	0.0971	0.0164	0.2267	0.2755	5.4	8.0	18.1
horse	0.2561	0.0075	0.3500	0.2203	15.1	9.5	179.6
hypothyroid	0.0157	0.0007	0.0183	0.8165	14.7	4.4	8,264.1
ionosphere	0.0500	0.0138	0.1143	0.7776	17.6	8.2	145.7
kr vs. kp	0.0345	0.0036	0.0360	0.8816	41.8	9.5	6,159.5
led24	0.2167	0.0457	0.2650	0.5969	23.7	8.3	57.4
liver	0.2322	0.0322	0.3382	0.2488	17.4	3.3	38.7
lrs	0.0866	0.0043	0.1283	0.7800	28.2	7.1	782.7
lymphography	0.0664	0.0376	0.1643	0.5671	12.6	15.2	37.2
mammography	0.1542	0.0046	0.1813	0.3927	12.5	5.6	363.6
mfeat-fac	0.0478	0.0098	0.0790	0.9521	135.0	28.7	34,557.9
mfeat-fou	0.1498	0.0103	0.1975	0.8111	93.1	14.9	11,415.7
mfeat-kar	0.0544	0.0134	0.0755	0.9571	84.8	18.3	11,084.4
mfeat-mor	0.2323	0.0288	0.2935	0.5754	77.6	4.0	1,315.8
mfeat-pix	0.0511	0.0141	0.0830	0.9632	142.3	42.6	46,305.1
mfeat-zer	0.1434	0.0279	0.2035	0.7778	128.4	18.4	9,636.1
new-thyroid	0.0150	0.0213	0.0571	0.9220	7.5	2.8	15.5
optdigits	0.0431	0.0194	0.0653	0.9715	327.6	19.6	89,697.2
ozone1hr	0.0288	0.0000	0.0289	0.0177	1.0	1.0	8,670.9
ozone8hr	0.0604	0.0000	0.0640	0.0878	4.9	1.2	10,115.4
page-blocks	0.0300	0.0070	0.0397	0.7936	69.5	5.0	8,992.8
phoneme	0.1307	0.0196	0.1617	0.6671	158.8	3.0	5,660.2
pima	0.1954	0.0137	0.2579	0.2655	21.2	3.6	167.8
primary-tumor	0.4742	0.0449	0.5758	0.2547	27.5	11.1	130.3
promoters	0.0458	0.0072	0.2300	0.3314	6.6	11.7	35.2
satimage	0.1185	0.0078	0.1288	0.8654	109.0	14.9	45,624.7
segment	0.0235	0.0184	0.0446	0.9651	104.2	7.0	3,040.3
sick	0.0185	0.0005	0.0199	0.7457	12.1	4.9	9,086.5

Table 3: Continued.

	Memetic IS+FS						
Dataset	Training error	Storage	Testing error	κ	Instances selected	Features selected	Time (s)
soybean	0.0563	0.0166	0.0779	0.9284	56.2	15.0	1,650.5
texture	0.0217	0.0178	0.0358	0.9888	259.9	13.6	37,701.2
tic-tac-toe	0.0302	0.0948	0.0800	0.7595	105.2	7.0	358.6
titanic	0.2095	0.0017	0.2094	0.0006	9.1	3.0	1,292.3
vehicle	0.1757	0.0631	0.3012	0.6146	87.6	9.9	560.8
vote	0.0253	0.0068	0.0395	0.8219	7.4	5.3	84.1
vowel	0.0283	0.1777	0.3858	0.5744	228.9	6.9	1,295.1
waveform	0.1656	0.0032	0.1812	0.6716	37.2	14.6	21,160.9
wine	0.0006	0.0227	0.0412	0.9452	7.4	6.4	18.5
yeast	0.3344	0.0391	0.4324	0.3701	83.6	5.0	791.2
zoo	0.0121	0.0692	0.0700	0.9200	14.0	7.2	12.8

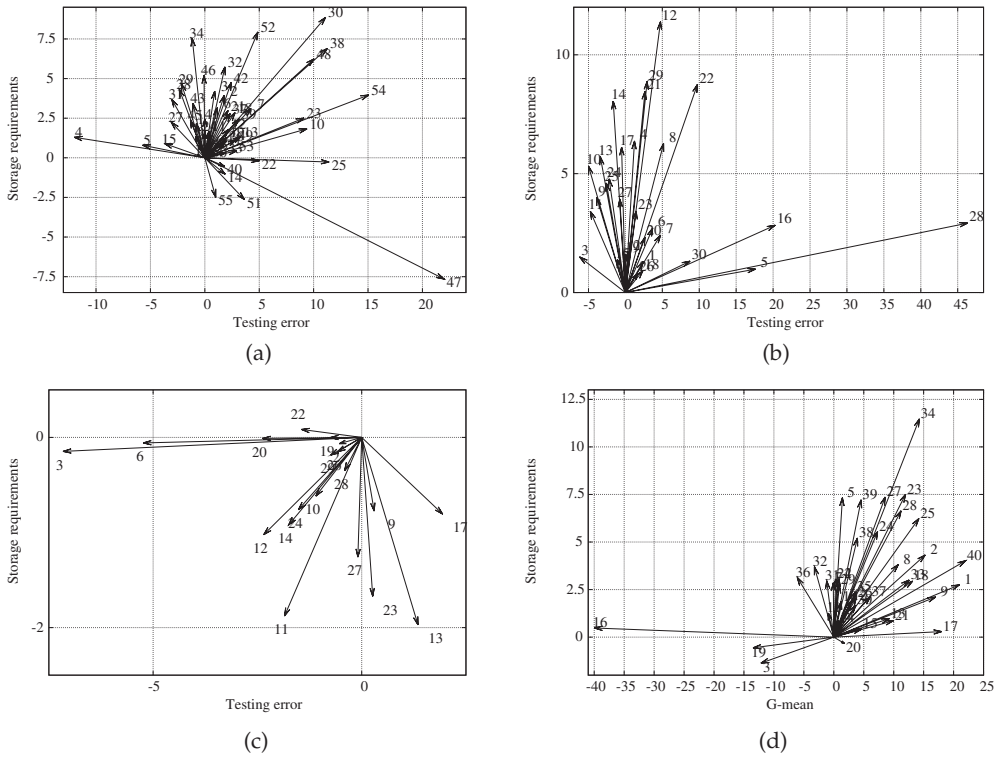


Figure 3: Testing error, or *G*-mean measure, and storage reduction obtained by using our memetic algorithm and: (a) standard CHC method, (b) standard CHC method with stratification, (c) our memetic algorithm with stratification, and (d) standard CHC method for class-imbalanced datasets.

Table 4: Comparison of the performances of instance and feature selection methods in terms of the testing error and the storage requirements. The table shows the win/loss record of each algorithm in the column against the algorithm in the row, the p-value of a two-tailed sign test on the win/loss record, p_s, and the R^+ and R^- values and p-value of the Wilcoxon test, p_w. Significant differences at a confidence level of 95% are marked with a tick ($\surd$).

Compared Method		IMOEA	Memetic IS+FS
Testing error			
CHC IS+FS	win/loss	23/31	37/15
	p_s	0.3409	0.0032 $\surd$
	R^+/R^-	1,034.5/505.5	380.5/1,159.5
	p_w	0.0267 $\surd$	0.0011 $\surd$
IMOEA	win/loss		45/7
	p_s		0.0000 $\surd$
	R^+/R^-		150.5/1,389.5
	p_w		0.0000 $\surd$
Storage requirements			
CHC IS+FS	win/loss	28/27	48/7
	p_s	1.0000	0.0000 $\surd$
	R^+/R^-	829.0/711.0	152.0/1,388.0
	p_w	0.6211	0.0000 $\surd$
IMOEA	win/loss		38/17
	p_s		0.0065 $\surd$
	R^+/R^-		342.5/1,197.5
	p_w		0.0003 $\surd$

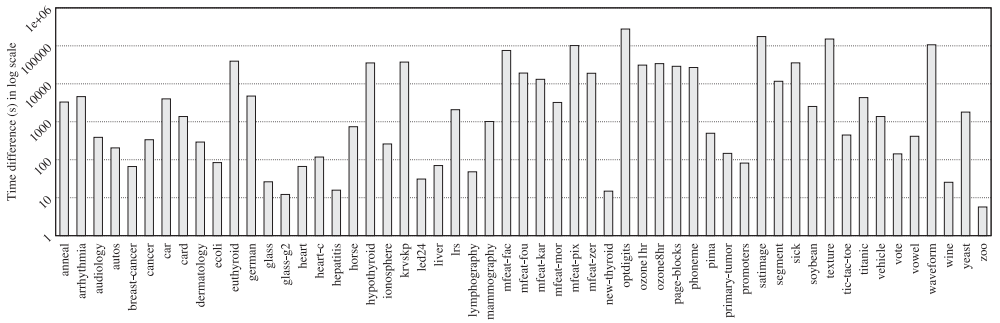


Figure 4: Difference between the execution time, on a logarithmic scale, of our method and CHC IS+FS. A filled bar means that our algorithm is faster, and an empty bar means that our algorithm is slower. (In this case, our algorithm is faster for all datasets.)

able to improve the reduction in the number of instances from the beginning of the evolution, which means that the evaluation of the individuals is made faster, and large reductions in runtime are shown in the experiments. The cumulative effect is a very significant reduction in runtime, as is shown in Figure 4.

Table 5: Comparison of the performance of instance and feature selection methods in terms of the κ measure. The table shows the win/loss record of each algorithm in the column against the algorithm in the row, the p-value of a two-tailed sign test on the win/loss record, p_s, and the p-value of the Wilcoxon test, p_w. Significant differences at a confidence level of 95% are marked with a tick ($\surd$).

Compared Method		Memetic IS+FS
CHC IS+FS	win/loss	39/15
	p_s	0.0015 $\surd$
	R^+/R^-	365.0/1,175.0
	p_w	0.0007 $\surd$

Recent works have shown that misclassification rates may be biased because they contain a lot of randomness (Ben-David, 2007). Thus, as an additional measure, and as an alternative to the misclassification rate, we have also used Cohen's κ measure, which is a method that compensates for random hits. The original purpose of this measure was to measure the degree of agreement. However, κ can also be adapted to measure classification accuracy, and its use is recommended because it takes random successes into consideration (Ben-David, 2007). κ can be computed from the confusion matrix in a classification task as follows:

$$\kappa = \frac{n \sum_{i=1}^{C} x_{ii} - \sum_{i=1}^{C} x_{i.} x_{.i}}{n^2 - \sum_{i=1}^{C} x_{i.} x_{.i}}, \tag{8}$$

where x_{ii} is the cell count on the main diagonal, n is the number of examples, C is the number of classes, and $x_{i.}$ and $x_{.i}$ are the column and row total counts, respectively. The value of κ ranges from –1 (total disagreement) to 1 (perfect agreement). For multiclass problems, κ is a very useful, yet simple, metric for measuring the accuracy of the classifier while compensating for random successes. Classification rates score all of the successes over all classes, whereas κ scores the successes independently for each class, and aggregates them. The second way of scoring is less sensitive to randomness caused by a different number of examples in each class, which causes a bias in the learner toward obtaining data-dependent models.

Tables 2 and 3 show the κ measure for CHC and our memetic approach, respectively, and Table 5 shows the comparison of the two methods. Table 5 shows that the differences with this measure are even more marked. These differences are statistically significant at a confidence level of 99%, using both the sign test and the Wilcoxon test. As this measure removes the effect of random hits, the advantage of our proposal is greatly demonstrated.

In the previous experiments, we arbitrarily set the number of fitness function evaluations to 100,000. It may be argued that this number is too large for some of the problems. In such problems, overfitting would be a possible outcome of the experiments. To test this effect, we carried out an additional experiment. We used early stopping and cross-validation to get the optimal number of fitness function evaluations. First, the training set was divided into a dataset for performing the evolution with 75% of the instances and a validation set with the remaining 25% of the instances. The evolution was carried out and at each generation the validation accuracy was obtained. The evolution was

Table 6: Comparison of the performance of the memetic algorithm using 100,000 fitness function evaluations and early stopping. The table shows the win/loss record of each algorithm in the column against the algorithm in the row, the p-value of a two-tailed sign test on the win/loss record, p_s, and the p-value of the Wilcoxon test, p_w. Significant differences at a confidence level of 95% are marked with a tick ($\surd$).

Compared Method		Early stopping
Testing error		
100,000	win/loss	35/18
	p_s	0.0270 $\surd$
	R^+/R^-	376.0/1,164.0
	p_w	0.0010 $\surd$
Storage requirements		
100,000	win/loss	17/33
	p_s	0.0328 $\surd$
	R^+/R^-	1,125.5/414.5
	p_w	0.0029 $\surd$

stopped after two consecutive generations for which the validation accuracy worsened. The number of evaluations until this happened was recorded and the evolutionary process was repeated with the whole training set using that number of evaluations as the new limit. A comparison of the results of this experiment and using the limit of 100,000 evaluations is shown in Table 6.

The results show that a large number of evaluations is not causing overfitting. As the fitness function is a combination of accuracy and reduction, the reduction term is acting as a regularization parameter that prevents overfitting. Furthermore, the early stopping approach has the effect of improving reduction but with a worse testing error. The behavior may be explained due to the fact that improving reduction is easier than improving accuracy, thus when the number of evaluations is reduced, it is easier to achieve reduction than accuracy.

In current relevant research fields, such as bioinformatics, it is common to have datasets with many features but very few instances. It is interesting to test whether our approach can deal with this kind of dataset. To test its efficiency in these problems, we selected a set of 18 problems with many features and few samples. The characteristics of these problems are shown in Table 7.

We carried out experiments using our proposal and the standard methods CHC IS+FS, and IMOEA. The accuracy was also compared with 1-NN with no selection. The results of all the methods are shown in Table 8. The comparison of the performance of these methods is shown in Table 9.

The results show that our proposal is also useful when the datasets have many features and few instances. Our memetic algorithm matched the performance of CHC IS+FS and IMOEA in terms of accuracy but with a significant improvement in terms of storage reduction. In fact, our approach achieved better reduction than the other two algorithms for all datasets, with a worse result of 3.66%. Our method proved to be especially efficient in removing features.

Table 7: Summary of datasets with many features and few instances.

	Dataset	Instances	Variables			Classes	Features
			Continuous	Binary	Nominal		
1	all-aml	72	7,129	—	—	2	7,129
2	breast-cancer-ma	97	24,481	—	—	2	24,481
3	central-nervous-system	60	7,129	—	—	2	7,129
4	colon-tumor	62	2,000	—	—	2	2,000
5	dbworld	64	7,402	—	—	2	7,402
6	dexter	600	20,000	—	—	2	20,000
7	dlbcl-harvard-outcome	58	7,129	—	—	2	7,129
8	dlbcl-harvard-tumor	77	7,129	—	—	2	7,129
9	dlbcl-nih	240	7,399	—	—	2	7,399
10	dlbcl-stanford	47	4,026	—	—	2	4,026
11	leukemia	72	12,582	—	—	3	12,582
12	lung-cancer-harvard1	203	12,600	—	—	5	12,600
13	lung-cancer-harvard2	181	12,533	—	—	2	12,533
14	lung-cancer-michigan	96	7,129	—	—	2	7,129
15	lung-cancer-ontario	39	2,880	—	—	2	2,880
16	ovarian-cancer	253	15,154	—	—	2	15,154
17	pediatric-leukemia	327	12,558	—	—	7	12,558
18	prostate	136	12,600	—	—	2	12,600

5.0.1 Control Experiments

The proposed approach has different components and different design decisions, which have been explained in the description of the model. Although the performance of the whole method is very good, as shown above, to ascertain that all of its components are useful we need to evaluate them separately. In this section, we show a series of control experiments that were carried out to test whether all of the parts of our memetic algorithm are needed. To evaluate a certain part of the model, we will proceed by removing that part and evaluating the performance of the method without it.

Our first important design decision was the fitness function. As explained, feature reduction is not considered equally with instance reduction, as our preliminary experiments showed a major negative impact on performance when too many features were removed. To test whether this decision was correct, we carried out experiments using the same weight for feature and instance reduction. Instead of the fitness function shown in Equation (2), we used:

$$\text{fitness}_i = \alpha_e(1 - e_i) + \alpha_{f_i}(1 - f_i) + \alpha_{f_f}(1 - f_f). \tag{9}$$

Furthermore, we have proposed a model with four different local search procedures: ISLS, FSLS, IFSLS, and IFALS. We also performed experiments removing, in turn, each one of these four local search methods. Table 10 shows the comparison of the memetic algorithm and the five control experiments, in terms of error and storage requirements. Regarding the fitness function, the effect is clear: the standard fitness function puts too much pressure on feature reduction. The achieved reduction is significantly better, but at the cost of a significantly worse testing error. However, that function can still be used if we are mainly focused on reduction, as the testing error is still competitive with respect to the CHC algorithm.

Table 8: Summary of the performance of instance and feature selection methods in terms of testing error and storage reduction. The number of selected instances and features is also shown.

	1-NN	Memetic IS+FS				CHC IS+FS				IMOEA			
Dataset	Testing error	Testing error	Storage	Instances selected	Features selected	Testing error	Storage	Instances selected	Features selected	Testing error	Storage	Instances selected	Features selected
all-aml	0.1405	0.1637	0.0009	3.8	57.3	0.0827	0.0421	7.1	2,744.1	0.1524	0.1900	25.7	3,414.3
breast-cancer-ma	0.4443	0.4128	0.0050	40.7	256.1	0.4481	0.0589	10.9	11,537.3	0.4221	0.2307	40.6	12,152.9
central-nervous-system	0.4538	0.2786	0.0004	3.7	32.6	0.3495	0.0479	6.4	2,887.8	0.3948	0.1302	14.9	3,367.5
colon-tumor	0.2714	0.1976	0.0003	3.2	9.9	0.1929	0.0395	7.7	575.4	0.2238	0.0808	11.0	820.2
dbworld	0.4233	0.2262	0.0009	6.3	46.6	0.2543	0.0262	3.4	2,084.9	0.2986	0.1439	17.8	2,188.6
dexter	0.4633	0.4083	0.0206	273.2	815.6	0.4217	0.2421	262.2	9,972.7	0.4417	0.2484	270.0	9,934.4
dlbcl-harvard-outcome	0.4733	0.6000	0.0003	3.9	23.5	0.4400	0.0926	12.2	2,819.8	0.4067	0.1462	16.0	3,403.4
dlbcl-harvard-tumor	0.1696	0.1571	0.0004	3.3	46.3	0.1429	0.0482	8.6	2,778.0	0.2196	0.1784	25.7	3,431.5
dlbcl-nih	0.4418	0.4503	0.0355	103.2	545.0	0.4456	0.0549	24.6	3,564.8	0.4107	0.2309	100.8	3,662.5
dlbcl-stanford	0.2383	0.1383	0.0002	2.5	11.4	0.2383	0.0366	5.2	1,191.8	0.4050	0.0632	6.1	1,763.5
leukemia	0.1583	0.1810	0.0010	7.9	81.8	0.1851	0.0513	8.2	5,099.3	0.1738	0.1992	26.3	6,173.7
lung-cancer-harvard1	0.1146	0.1307	0.0055	85.5	146.5	0.0940	0.0421	16.4	5,906.6	0.1498	0.2429	88.8	6,293.9
lung-cancer-harvard2	0.0667	0.0611	0.0048	73.8	130.5	0.0111	0.0117	4.3	5,543.1	0.0602	0.2394	78.2	6,252.0
lung-cancer-michigan	0.0100	0.0311	0.0002	4.1	29.6	0.0200	0.0174	4.0	2,683.5	0.0000	0.1812	32.1	3,479.9
lung-cancer-ontario	0.3183	0.3133	0.0003	3.5	7.3	0.2800	0.0372	5.2	719.5	0.1900	0.0311	2.9	1,080.6
ovarian-cancer	0.0634	0.0711	0.0068	110.2	215.9	0.1428	0.0685	31.9	7,406.2	0.1189	0.2523	115.7	7,524.8
pediatric-leukemia	0.2289	0.2293	0.0366	148.9	905.6	0.1715	0.1447	86.6	6,176.7	0.3057	0.2432	144.3	6,227.5
prostate	0.1758	0.2055	0.0111	54.1	313.2	0.1665	0.0635	16.6	5,903.2	0.2357	0.2287	56.7	6,220.2

Table 9: Comparison of the performance of instance and feature selection methods in terms of the testing error and the storage requirements for datasets with many features and few instances. The table shows the win/loss record of each algorithm in the column against the algorithm in the row, the p-value of a two-tailed sign test on the win/loss record, p_s, and the R^+ and R^- values and p-value of the Wilcoxon test, p_w. Significant differences at a confidence level of 95% are marked with a tick ($\surd$).

Compared Method		Memetic IS+FS
Testing error		
1-NN	win/loss	9/9
	p_s	1.0000
	R^+/R^-	71.0/100.0
	p_w	0.5277
CHC IS+FS	win/loss	11/7
	p_s	0.4807
	R^+/R^-	58.0/113.0
	p_w	0.2311
IMOEA	win/loss	7/11
	p_s	0.4807
	R^+/R^-	103.0/68.0
	p_w	0.4460
Storage requirements		
CHC IS+FS	win/loss	18/0
	p_s	0.0000 $\surd$
	R^+/R^-	0.0/171.0
	p_w	0.0002 $\surd$
IMOEA	win/loss	18/0
	p_s	0.0065 $\surd$
	R^+/R^-	0.0/171.0
	p_w	0.0002 $\surd$

Regarding the four different local search algorithms, their impact on the performance of the model is varied. The two operators that modify the selected instances and features, IFSLS and IFALS, have a major effect. On the other hand, the two that modify instance selection, ISLS, and feature selection, FSLS, separately, have a lesser impact. IFSLS is very efficient in reducing storage, and when removed, the memetic algorithm obtains a worse reduction on 48 datasets. Furthermore, IFSLS achieves that reduction without harming the accuracy of the classifier. When removed, the performance of the memetic algorithm is not improved in terms of testing error. As a result, IFSLS is the most relevant operator from our proposal.

IFALS is also useful for the model. In terms of reduction, its effect is to increment storage for 54 of the 55 datasets. That increase is an expected result, as this operator can only add instances or features, but cannot remove them. However, this small increment has the beneficial effect of incrementing the accuracy of the algorithm. When removed, the memetic algorithm performed worse in terms of testing error in 41 datasets. Thus, as was the case for IFSLS, this operator is an important part of our memetic algorithm. The behavior of ISLS and FSLS searches are somewhat different. The overall effect of both is positive, as the results in terms of testing error and storage requirements are

Table 10: Comparison of the performance of the memetic algorithm and the control experiments in terms of testing error and storage requirements. The table shows the win/loss record of each algorithm in the column against the algorithm in the row, the p-value of a two-tailed sign test on the win/loss record, p_s, and the p-value of the Wilcoxon test, p_w. Significant differences at a confidence level of 95% are marked with a tick ($\surd$).

		Memetic algorithm	
Compared Method		Error	Storage
Fitness Equation (9)	win/loss	44/10	0/54
	p_s	0.0000$\surd$	0.0000$\surd$
	R^+/R^-	181.0/1,359.0	1,539.5/0.5
	p_w	0.0000$\surd$	0.0000$\surd$
FSLS	win/loss	30/22	30/24
	p_s	0.3317	0.4966
	R^+/R^-	583.5/956.5	549.5/990.5
	p_w	0.1181	0.0647
ISLS	win/loss	26/23	29/25
	p_s	0.7754	0.6835
	R^+/R^-	677.5/862.5	672.5/867.5
	p_w	0.4383	0.4140
IFSLS	win/loss	27/26	46/8
	p_s	1.0000	0.0000$\surd$
	R^+/R^-	681.5/858.5	123.0/1,417.0
	p_w	0.4584	0.0000$\surd$
IFALS	win/loss	41/12	1/53
	p_s	0.0001 $\surd$	0.0000$\surd$
	R^+/R^-	277.5/1,262.5	72.5/1,467.5
	p_w	0.0000 $\surd$	0.0000$\surd$

improved by their presence. However, the improvement is less marked, and they might be removed without dramatic harm to the model.

To further ensure the utility of the proposed local searches, we carried out another experiment with the aim of testing whether any of the four local searches alone were able to match the performance of the memetic algorithm with all of them. Thus, we performed experiments using each one of the local searches separately. The results are shown in Table 11.

In terms of accuracy, the results show that the memetic algorithm with all the four local search procedures beat the same algorithm using any of them alone. The differences are statistically significant for all the procedures. In terms of reduction, the results are less homogeneous. IFALS was worse in reduction than the memetic algorithm. That is an expected result because IFALS does not remove any instances. FSLS was also less efficient in reducing storage requirements, achieving a worse storage but with no significant differences. ISLS and IFSLS achieved a better reduction than the memetic algorithm but at the cost of accuracy. As a summary, we can conclude that none of the four local search procedures used separately matched the performance of the memetic algorithm using all of them.

Table 11: Comparison of the performance of the memetic algorithm and the algorithm using only one of the four local search procedures in terms of testing error and storage requirements. The table shows the win/loss record of each algorithm in the column against the algorithm in the row, the p-value of a two-tailed sign test on the win/loss record, p_s, and the p-value of the Wilcoxon test, p_w. Significant differences at a confidence level of 95% are marked with a tick (√).

		Memetic algorithm	
Compared Method		Error	Storage
FSLS	win/loss	40/12	30/24
	p_s	0.0001√	0.4966
	R^+/R^-	222.5/1,317.5	842.0/698.0
	p_w	0.0000√	0.5463
ISLS	win/loss	34/18	13/41
	p_s	0.0365√	0.0002√
	R^+/R^-	446.5/1,093.5	1,238.0/302.0
	p_w	0.0067√	0.0001√
IFSLS	win/loss	39/12	0/53
	p_s	0.0002√	0.0000√
	R^+/R^-	356.5/1,183.5	1,538.0/2.0
	p_w	0.0005√	0.0000√
IFALS	win/loss	37/17	43/12
	p_s	0.0091√	0.0000√
	R^+/R^-	432.0/1,108.0	209.0/1,331.0
	p_w	0.0046√	0.0000√

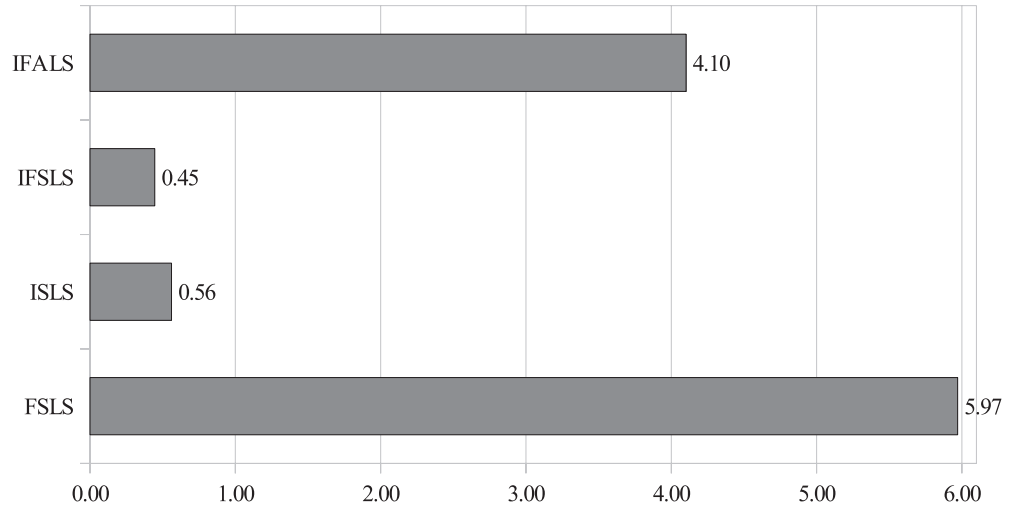


Figure 5: Time ratio, with respect to the memetic algorithm using the four local search procedures, needed for the evolution using each one of the four local searches separately.

An interesting aspect to study is the relative time needed for each local search procedure when used alone compared with the time needed by the memetic algorithm to perform its task. Figure 5 shows the time used by the evolutionary algorithm when using the four local search procedures separately. The time is shown as the ratio with respect to the memetic algorithm with the four local search procedures. When ISLS and IFSLS were used alone, the time needed was about half the time needed when all four searches were used. This is an expected result, because three local searches were not carried out. However, when IFALS or FSLS were used alone, the time needed by the algorithm was greatly increased, by four and six times, respectively. The reason is that IFALS and FSLS were less efficient in removing instances. Thus, the individual evaluated had more instances than when the four procedures were used. Although the number of evaluations of the fitness function were the same, each evaluation was more costly, as the individual had more instances. This highlights the important result that IFALS and FSLS are efficient if they are considered as part of the memetic algorithm together with ISLS and FFSLS, but they are counterproductive if they are used alone.

An additional interesting question is whether simultaneous instance and feature selection, however efficient, is better than instance and feature selection separately. Thus, we run a set of experiments for testing the performance of instance selection and feature selection against our memetic approach. Because our previous experiments and other works (Cano et al., 2003) have shown that CHC is the overall best performing method for instance selection, we used CHC for instance selection. For feature selection, we compared the CHC, LVF (Liu and Setiono, 1996), and FOCUS (Almuallim and Dietterich, 1991) algorithms, and found that the best results are also with CHC. Table 12 shows a comparison of the performance of our memetic approach and CHC for instance and feature selection.

First, it is interesting to compare the performance of CHC IS and CHC FS. We found that CHC IS achieved a better testing error, although the reduction is clearly worse. On the other hand, CHC FS reduction is better but with a significantly worse error. This result is due to the effect of feature reduction in the fitness function, explained above. The large impact that removing a feature has on the fitness function was caused by the tendency of CHC FS to remove too many features. Regarding the comparison with our memetic approach, the results are clearly in favor of the proposal. The memetic algorithm is significantly better than both instance selection and feature selection, when performed separately, in terms of testing error, and especially in terms of storage reduction where the differences are large.

An interesting question to answer is why the method is performing better when all the local search operators are used than when any of them is used alone. However, it is difficult to obtain a simple explanation due to the complexity of the whole evolutionary process. The effect of IFALS is avoiding the loss of interesting instances or features that are removed during the evolutionary process. This effect was clearly shown in the experiments above; when IFALS was removed the number of instances and features selected decreased, but the accuracy of the selection also decreased. The net effect of the other three procedures is less straightforward to explain, as their effect is not independent. However, the experiments showed that only the three local search procedures together achieved the best reduction.

5.1 Stratified Approach

As explained in Section 3.2, we have also proposed an extension of the stratification method used for instance selection to simultaneously perform instance and feature

Table 12: Comparison of the performance of instance and feature selection separately against our memetic method, in terms of the testing error and the storage requirements. The table shows the win/loss record of each algorithm in the column against the algorithm in the row. The row labeled p_s is the result of a two-tailed sign test on the win/loss record and the row labeled p_w shows the result of a Wilcoxon test. Significant differences at a confidence level of 95% are marked with a tick ($\surd$).

Compared Method		CHC FS	Memetic IS+FS
Testing error			
CHC IS	win/loss	16/38	37/18
	p_s	0.0038 $\surd$	0.0145 $\surd$
	R^+/R^-	1,150.5/389.5	443.0/1,097.0
	p_w	0.0014 $\surd$	0.0061 $\surd$
CHC FS	win/loss		46/8
	p_s		0.0000 $\surd$
	R^+/R^-		122.5/1,417.5
	p_w		0.0000 $\surd$
Storage requirements			
CHC IS	win/loss	48/7	55/0
	p_s	0.0000 $\surd$	0.0000 $\surd$
	R^+/R^-	94.0/1,446.0	0.0/1,540.0
	p_w	0.0000 $\surd$	0.0000 $\surd$
CHC FS	win/loss		55/0
	p_s		0.0065 $\surd$
	R^+/R^-		0.0/1,540.0
	p_w		0.0003 $\surd$

selection. In this section, we study the performance of our method in this case. We compare its performance with the application of stratification to the standard CHC algorithm. Table 13 shows a summary of the datasets used for this comparison. These datasets can be considered of large size. We have included the largest datasets used in the previous experiments, to test the stratified approach against the use of the whole dataset.

We have used a stratum size of 1,000 instances. Although larger sizes might obtain better results, the advantage of stratification would be lost if we use subsets that are too large. In previous experiments, we used 100,000 evaluations of the fitness function. Because we have to perform the algorithm in many strata for the larger datasets, that number of evaluations would be too high. Thus, we reduced the number of evaluations to 10,000 for all datasets and methods.

The results in terms of testing error and overall reduction are shown in Table 14, and are plotted in Figure 3(b). The comparison of the results is shown in Table 15. In terms of testing error, our memetic method is performing better than CHC, although the differences are not statistically significant. However, Figure 3(b) shows large differences for some problems, such as krkopt, magic04, mfeat-mor, nursery, phoneme, titanic, and zip. In terms of storage, the differences are very marked, with our method performing better in all datasets.

The comparison using the κ measure is also shown in Table 15. As in the previous experiments, κ shows a large advantage of our method over the standard approach, with a significant difference at a confidence level of 99%.

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

			Variables				
	Dataset	Instances	C	B	N	Classes	Features
1	adult	48,842	6	1	7	2	105
2	euthyroid	3,163	7	18	—	2	44
3	hypothyroid	3,772	7	20	2	4	29
4	isolet	7,797	34	—	—	26	617
5	krkopt	28,056	6	—	—	18	6
6	kr vs. kp	3,196	—	34	2	2	38
7	letter	20,000	16	—	—	26	16
8	magic04	19,020	10	—	—	2	10
9	mfeat-fac	2,000	216	—	—	10	216
10	mfeat-fou	2,000	76	—	—	10	76
11	mfeat-kar	2,000	64	—	—	10	64
12	mfeat-mor	2,000	6	—	—	10	6
13	mfeat-pix	2,000	240	—	—	10	240
14	mfeat-zer	2,000	47	—	—	10	47
15	mushroom	8,124	—	6	16	2	117
16	nursery	12,960	—	1	7	5	23
17	optdigits	5,620	64	—	—	10	64
18	ozone1hr	2,536	72	—	—	2	72
19	ozone8hr	2,534	72	—	—	2	72
20	page-blocks	5,473	10	—	—	5	10
21	pendigits	10,992	16	—	—	10	16
22	phoneme	5,404	5	—	—	2	5
23	satimage	6,435	36	—	—	6	36
24	segment	2,310	19	—	—	7	19
25	shuttle	58,000	9	—	—	7	9
26	sick	3,772	7	20	2	2	33
27	texture	5,500	40	—	—	11	40
28	titanic	2,201	—	—	3	2	8
29	waveform	5,000	40	—	—	3	40
30	zip	9,298	256	—	—	10	256

The runtime comparison using stratification is shown in Figure 6. The plot shows that our algorithm is faster than standard CHC on most of the datasets, with the exception of mfeat-fac, mfeat-fou, mfeat-kar, mfeat-pix, optidigits, and zip. All of these six datasets have a characteristic in common, which is that they have many features. The fitness function we used in our memetic algorithm (see Equation (2)) yielded a lesser reduction of features, which we have identified as one of the sources of the success of the proposed algorithm. However, when we have many features, our method can be slower than standard CHC. If we also have many instances, the efficiency of removing instances compensates for this effect and our approach is still faster than standard CHC. However, in the stratified approach, the subset size is limited to 1,000 instances, and our method was slower on those five datasets.

Table 14: Summary of the performance of instance selection methods in terms of testing error and storage reduction for large datasets. Additionally, the κ measure, the number of selected instances and features, and the execution time are shown.

	Standard CHC IS+FS							Memetic IS+FS						
Dataset	Training error	Storage	Testing error	κ	Instances selected	Features selected	Time (s)	Training error	Storage	Testing error	κ	Instances selected	Features selected	Time (s)
adult	0.2060	0.0133	0.2090	0.4376	16,609.1	3.7	65,207.2	0.1824	0.0003	0.1850	0.3936	489.5	3	63,582.9
euthyroid	0.0235	0.0180	0.0335	0.7732	935	2.4	2,402.4	0.0293	0.0009	0.0301	0.7470	20.7	5.5	1,068.4
hypo	0.0216	0.0171	0.0279	0.7599	1,006.5	1.7	1,380.4	0.0837	0.0022	0.0899	0.7989	29.4	7.3	840.5
isolet	0.1629	0.1199	0.2511	0.7485	2,725.4	10.5	4,515.5	0.2149	0.0562	0.2388	0.8299	638	21	4,478.4
krkopt	0.8335	0.0416	0.8331	0.0329	6,294.1	1	3,419.4	0.6460	0.0317	0.6569	0.2028	1,265.5	3.8	1,940.6
krvskp	0.1224	0.0313	0.1254	0.7307	743.7	4.6	2,304.2	0.0823	0.0042	0.0884	0.8714	40.9	10.2	889.3
letter	0.0999	0.1470	0.1419	0.8736	6,831.4	6.2	5,921.1	0.0786	0.1229	0.0943	0.9521	3,539	10	5,748.6
magic04	0.1540	0.0730	0.2207	0.4510	6,244.2	2	3,786	0.1663	0.0103	0.1690	0.5915	418.4	4.2	1,242.9
mfeat-fac	0.0197	0.0575	0.0370	0.9632	736.1	30.4	5,529.1	0.0621	0.0175	0.0760	0.9559	130	52.2	6,306.9
mfeat-fou	0.0964	0.0696	0.1585	0.8111	755.7	12.6	1,591.9	0.2028	0.0164	0.2085	0.8061	96.5	23.2	1,920.9
mfeat-kar	0.0296	0.0664	0.0465	0.9493	670.7	11.4	1,303.8	0.0721	0.0322	0.0940	0.9565	157	24.2	1,663.3
mfeat-mor	0.2397	0.1529	0.3645	0.5346	590.1	2.8	243.6	0.2742	0.0390	0.3170	0.6019	78.4	5.4	107.5
mfeat-pix	0.0171	0.0909	0.0355	0.9660	700.7	56.1	6,770.7	0.0493	0.0338	0.0695	0.9671	173.2	84.5	8,306.5
mfeat-zer	0.1217	0.1176	0.2045	0.7610	682.2	14.6	1,526.1	0.1935	0.0371	0.2210	0.7867	115.5	27.2	1,128
mushroom	0.0533	0.0141	0.0149	0.9522	2,369.7	5.1	12,121.8	0.0536	0.0003	0.0257	0.9537	71.1	3.8	9,812.5

Table 14: Continued.

	Standard CHC IS+FS							Memetic IS+FS						
Dataset	Training error	Storage	Testing error	κ	Instances selected	Features selected	Time (s)	Training error	Storage	Testing error	κ	Instances selected	Features selected	Time (s)
nursery	0.4559	0.0405	0.4432	0.4037	2,849.9	3.8	4,587.8	0.1825	0.0122	0.2404	0.5216	384.3	8.4	2,471.2
optdigits	0.0213	0.0886	0.0408	0.9640	1,815.2	15.8	6,181.8	0.0362	0.0275	0.0459	0.9769	308	28.9	7,602.8
ozone1hr	0.0407	0.0092	0.0526	0.0211	755.9	2	2,144	0.0288	0.0000	0.0289	0.0008	2.3	2	1,446.5
ozone8hr	0.0528	0.0162	0.0715	0.1756	912.1	2.9	2,254.7	0.0707	0.0000	0.0715	0.0500	3.7	2.4	1,511.7
page-blocks	0.0841	0.0304	0.0900	0.5503	1,356	1.1	885.5	0.0622	0.0071	0.0634	0.7760	63.1	5.5	412.3
pendigits	0.0349	0.1431	0.0512	0.9478	3,097.3	7.3	3,353	0.0206	0.0585	0.0247	0.9894	651.9	14.2	1,627.5
phoneme	0.1714	0.1063	0.2737	0.3469	1,851.4	1.4	888	0.1716	0.0188	0.1761	0.6671	151.8	3	271.5
satimage	0.0936	0.0588	0.1408	0.8029	2,397.9	5.1	3,973.8	0.1178	0.0245	0.1261	0.8765	220.1	23.2	2,527.5
segment	0.0202	0.0738	0.0377	0.9666	693.1	4.2	769.6	0.0472	0.0260	0.0597	0.9630	105.5	9.7	295.8
shuttle	0.0021	0.0482	0.0021	0.9903	10,766	2.1	8,727.9	0.0276	0.0021	0.0279	0.9904	495.3	2	3,235.4
sick	0.0420	0.0102	0.0425	0.5828	949.3	1.2	1,497.7	0.0255	0.0021	0.0257	0.7512	31.2	7.3	1,066.4
texture	0.0176	0.0697	0.0291	0.9760	1,944.4	7.1	4,398.6	0.0279	0.0304	0.0367	0.9886	347.7	17.2	3,380.2
titanic	0.6769	0.0344	0.6770	0.0000	389	1.4	247.8	0.2113	0.0052	0.2135	0.0085	17	4.8	109.5
waveform	0.1333	0.0939	0.2180	0.6563	1,859.2	9.1	3,174	0.1861	0.0051	0.1886	0.6782	62.6	14.6	1,633.2
zip	0.2692	0.0200	0.4352	0.3777	3,511.1	12.2	39,168.5	0.2466	0.0068	0.3477	0.4587	540.2	16.2	123,283.5

Table 15: Comparison of the performance of instance and feature selection methods in terms of the testing error, the storage requirements and the κ measure for large datasets using stratification. The table shows the win/loss record of each algorithm in the column against the algorithm in the row, the p-value of a two-tailed sign test on the win/loss record, p_s, and the p-value of the Wilcoxon test, p_w. Significant differences at a confidence level of 95% are marked with a tick ($\surd$).

Compared Method		Memetic IS+FS
Testing error		
CHC IS+FS	win/loss	18/11
	p_s	0.2649
	R^+/R^-	148.5/316.5
	p_w	0.0840
Storage requirements		
CHC IS+FS	win/loss	30/0
	p_s	0.0000 $\surd$
	R^+/R^-	0.0/45.0
	p_w	0.0000 $\surd$
κ		
CHC IS+FS	win/loss	23/7
	p_s	0.0052 $\surd$
	R^+/R^-	82.0/383.0
	p_w	0.0020 $\surd$

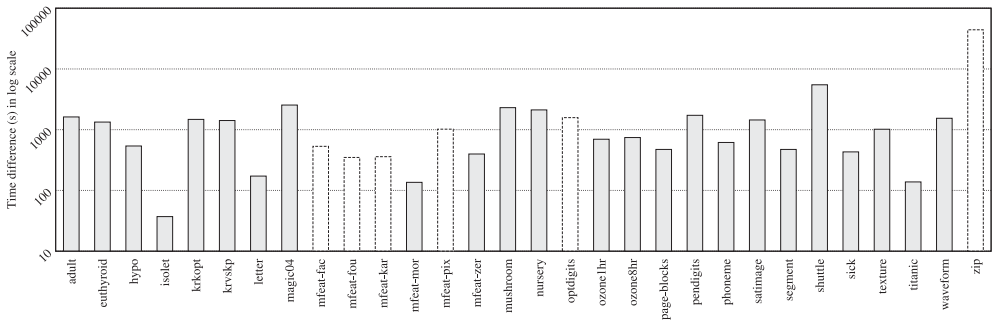


Figure 6: The difference between the execution time, on a logarithmic scale, of our method and CHC IS+FS. A filled bar means that our algorithm is faster, and an empty bar means that our algorithm is slower.

Figure 3(c) shows a comparison with no stratification for the datasets that were used for both experiments. The plot shows the usefulness of the approach. In fact, for the testing error, the Wilcoxon test finds significant differences in favor of the stratified approach, with a p-value of 0.0212, and, for storage reduction, with a p-value of 0.0025. Thus, the stratification is proven useful even with a small subset of 1,000 instances.

5.2 Class-Imbalance Problems

In previous sections, we showed that our proposal improved the performance of the standard CHC algorithm in terms of testing error, with a significant reduction in storage

and execution time. In this section, we study its behavior in class-imbalanced problems. The relevance of this kind of problem (Orriols-Puig and Bernadó-Mansilla, 2009) can hardly be overestimated because of the high number of interesting real-world problems that are class imbalanced (Chawla et al., 2002).

We must bear in mind that we are not proposing a method for informative undersampling for the class-imbalanced problem (García et al., 2009). Our aim is to study whether our proposal is able to improve the results of the standard CHC algorithm in class-imbalanced problems. A positive answer will open a new research line of developing a memetic algorithm for the simultaneous selection of instances and features toward a class-imbalanced aware selection.

To test the ability of our memetic algorithm on class-imbalanced datasets, we have used the problems shown in Table 16. All problems are two-class problems, with an imbalance ratio of the minority class to the majority class from 1:2 to 1:130. The datasets breast-cancer, cancer, euthyroid, german, haberman, hepatitis, ionosphere, ozone1hr, ozone8hr, phoneme, pima, sick, tic-tac-toe, and titanic are from the UCI Machine Learning Repository. The remaining datasets were artificially created following García and Herrera (2009).

All of the details of the evolutionary process are the same as those for the previous datasets, with an exception. In the previous experiments, for the fitness function of both our memetic algorithm and the CHC IS+FS algorithm, we used training accuracy. However, accuracy is not a useful measure for imbalanced data, especially when the number of instances of the minority class is very small compared to the majority class. If we have a ratio of 1:100, a classifier that assigns all instances to the majority class will have a 99% accuracy. Several measures (Sun et al., 2007) have been developed to take into account the imbalanced nature of these problems. Given the ratio of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN), we can define several measures. Perhaps the most common is the true positive rate ($\mathrm{TP}_{\mathrm{rate}}$), or recall ($R$), or sensitivity ($Sn$):

$$\mathrm{TP}_{\mathrm{rate}} = R = Sn = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \tag{10}$$

which is relevant if we are only interested in the performance on the positive class; and the true negative rate ($\mathrm{TN}_{\mathrm{rate}}$) or specificity ($Sp$):

$$\mathrm{TN}_{\mathrm{rate}} = Sp = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}}. \tag{11}$$

From these basic measures, others have been proposed, such as the F-measure (Lewis and Gale, 1998) or if we are concerned about the performance on both negative and positive classes the G-mean measure (Kubat et al., 1998):

$$G\text{-mean} = \sqrt{Sp \cdot Sn}. \tag{12}$$

This measure is a good compromise between specificity and sensitivity, so it is the measure used for the fitness function. The G-mean is also the measure used in the plots. The remaining parameters of the evolution are the same as in previous experiments.

Table 17 shows the results for the memetic algorithm and CHC for class-imbalanced datasets. Figure 3(d) shows a plot of the comparison between our approach and CHC. Both methods are compared in Table 18. As was the case for the previous datasets, our memetic approach beat CHC in both testing error, measured using G-mean, and reduction. The differences in both terms are statistically significant at a confidence level of 99%.

Table 16: Summary of class-imbalanced datasets. All of these datasets are two-class problems. The variables of each dataset can be C (continuous), B (binary), or N (nominal). The features column shows the number of features, as this number depends not only on the number of variables but also on their type.

			Variables				
	Dataset	Instances	C	B	N	Ratio	Features
1	abalone19	4,177	7	—	1	1:130	10
2	abalone9-18	731	7	—	1	1:17	10
3	breast-cancer	286	—	3	6	1:3	15
4	cancer	699	9	—	—	1:2	9
5	carG	1,728	—	—	6	1:25	16
6	ecoliCP-IM	220	7	—	—	1:2	7
7	ecoliM	336	7	—	—	1:4	7
8	ecoliMU	336	7	—	—	1:9	7
9	ecoliOM	336	7	—	—	1:16	7
10	euthyroid	3,163	7	18	—	1:10	44
11	german	1,000	6	3	11	1:3	61
12	glassBWFP	214	9	—	—	1:3	9
13	glassBWNFP	214	9	—	—	1:2	9
14	glassContainers	214	9	—	—	1:16	9
15	glassNW	214	9	—	—	1:4	9
16	glassTableware	214	9	—	—	1:23	9
17	glassVWFP	214	9	—	—	1:12	9
18	haberman	306	3	—	—	1:3	3
19	hepatitis	155	6	13	—	1:4	19
20	ionosphere	351	33	1	—	1:2	34
21	new-thyroidT	215	5	—	—	1:6	5
22	optdigitsZ	5,620	64	—	—	1:10	64
23	ozone1hr	2,536	72	—	—	1:34	72
24	ozone8hr	2,534	72	—	—	1:15	72
25	phoneme	5,404	5	—	—	1:3	5
26	pima	768	8	—	—	1:2	8
27	satimageF	6,435	36	—	—	1:8	36
28	satimageT	6,435	36	—	—	1:10	36
29	segmentO	2,310	19	—	—	1:7	19
30	sick	3,772	7	20	2	1:16	33
31	splice-EI	3,175	—	—	60	1:4	120
32	Splice-IE	3,175	—	—	60	1:4	120
33	tic-tac-toe	958	—	—	9	1:2	9
34	titanic	2,201	—	—	3	1:3	8
35	vehicleVAN	846	18	—	—	1:4	18
36	vowelZ	990	10	—	—	1:11	10
37	yeastCYT-POX	483	8	—	—	1:24	8
38	yeastEXC	1,484	8	—	—	1:42	8
39	yeastME1	1,484	8	—	—	1:33	8
40	yeastME2	1,484	8	—	—	1:29	8

Table 17: Summary of the performance of instance selection methods in terms of *G*-mean and storage reduction. Additionally, specificity, sensitivity, the number of selected instances and features, and the execution time are shown.

	Standard CHC IS+FS							Memetic IS+FS						
Dataset	Storage	Specificity	Sensitivity	*G*-mean	Instances selected	Features selected	Time (s)	Storage	Specificity	Sensitivity	*G*-mean	Instances selected	Features selected	Time (s)
abalone19	0.0788	0.6031	0.5333	0.4453	1,273.5	0.9	18,711.3	0.0513	0.8521	0.5917	0.6551	205.5	4.0	4,531.9
abalone9-18	0.0522	0.9535	0.3850	0.5641	171.5	2.0	725.3	0.0090	0.8608	0.6800	0.7167	13.7	4.1	185.1
breast-cancer	0.0182	0.7210	0.5896	0.6429	35.2	2.0	105.7	0.0319	0.6415	0.4488	0.5212	13.8	8.5	48.6
cancer	0.0357	0.9496	0.9481	0.9484	101.1	2.0	489.7	0.0049	0.9494	0.9656	0.9571	5.7	4.9	158.8
carG	0.0814	0.9259	1.0000	0.9622	337.6	6.0	5,334.8	0.0081	0.9536	1.0000	0.9765	25.4	7.9	1,185.0
ecoliCP-IM	0.0127	1.0000	0.9520	0.9752	17.6	1.0	35.2	0.0038	0.9941	0.9722	0.9826	4.2	1.2	18.1
ecoliM	0.0417	0.8835	0.9050	0.8929	50.0	1.7	110.5	0.0131	0.9053	0.8831	0.8920	11.7	2.3	38.0
ecoliMU	0.0515	0.9186	0.7300	0.7703	54.6	2.0	132.4	0.0133	0.8918	0.8717	0.8782	8.3	3.4	42.5
ecoliOM	0.0313	0.9744	0.7500	0.7626	36.8	1.8	100.1	0.0102	0.9810	0.9000	0.9328	5.7	3.8	43.1
euthyroid	0.0194	0.9836	0.8737	0.9268	1,099.4	2.2	48,478.5	0.0006	0.9690	0.9113	0.9393	17.4	4.0	4,769.9
german	0.0221	0.6714	0.6823	0.6748	257.2	4.7	5,619.1	0.0095	0.6898	0.6517	0.6658	38.1	12.9	866.1
glassBWFP	0.0447	0.8788	0.7915	0.8282	38.8	2.0	49.9	0.0258	0.8788	0.8450	0.8568	14.4	3.1	25.4
glassBWNFP	0.0569	0.7696	0.7307	0.7404	46.9	2.1	56.3	0.0471	0.8275	0.8515	0.8334	20.3	3.9	28.9
glassContainers	0.0299	0.9652	0.9000	0.8849	25.9	2.0	49.1	0.0224	0.9850	0.9000	0.8924	8.1	4.8	25.9
glassNW	0.0207	0.9073	0.8742	0.8873	24.3	1.5	44.8	0.0165	0.9478	0.9239	0.9339	7.3	3.8	21.6
glassTableware	0.0228	0.9802	1.0000	0.9899	19.8	2.0	43.4	0.0181	0.9757	0.6000	0.5901	7.7	4.0	30.6
glassVWFP	0.0637	0.8566	0.3600	0.3925	35.8	3.1	49.6	0.0608	0.8619	0.5600	0.5728	22.3	4.7	32.7
haberman	0.0539	0.5731	0.4969	0.5146	44.6	1.0	60.0	0.0244	0.6707	0.6951	0.6463	10.1	2.0	29.9
hepatitis	0.0328	0.8186	0.7005	0.7067	22.3	3.9	51.3	0.0384	0.7872	0.5562	0.5719	9.8	10.5	22.0
ionosphere	0.0161	0.8849	0.8696	0.8741	71.7	2.4	379.9	0.0195	0.9157	0.8755	0.8921	21.2	9.7	148.7

Table 17: Continued.

Dataset	Standard CHC IS+FS							Memetic IS+FS						
	Storage	Specificity	Sensitivity	*G*-mean	Instances selected	Features selected	Time (s)	Storage	Specificity	Sensitivity	*G*-mean	Instances selected	Features selected	Time (s)
new-thyroidT	0.0241	0.9281	0.8500	0.8787	23.4	1.0	30.0	0.0155	0.9836	0.9750	0.9783	5.0	3.0	16.8
optdigitsZ	0.0309	0.9966	0.9782	0.9874	2,500.9	4.0	303,467.3	0.0003	0.9957	0.9855	0.9906	18.7	6.2	19,748.1
ozone1hr	0.0767	0.9625	0.4198	0.6293	1,017.7	12.4	73,768.2	0.0016	0.8795	0.6468	0.7491	28.5	8.8	4,075.4
ozone8hr	0.0591	0.9326	0.5412	0.7060	1,045.7	9.3	66,213.3	0.0036	0.8270	0.7374	0.7791	50.0	10.8	4,369.1
phoneme	0.0900	0.7636	0.6175	0.6859	2,189.7	1.0	32,666.7	0.0277	0.8084	0.8485	0.8280	224.1	3.0	6,230.8
pima	0.0408	0.7265	0.6561	0.6863	225.8	1.0	682.8	0.0202	0.7504	0.7020	0.7239	26.9	4.1	179.0
satimageF	0.0844	0.9377	0.6455	0.7776	2,842.5	6.2	248,620.3	0.0108	0.8676	0.8596	0.8634	154.9	14.5	23,639.9
satimageT	0.0806	0.9429	0.6005	0.7517	2,850.3	5.9	248,727.6	0.0142	0.8717	0.8571	0.8641	184.2	16.1	23,442.0
segmentO	0.0295	0.9959	0.9700	0.9828	582.1	2.0	11,327.9	0.0022	0.9970	0.9876	0.9922	15.4	5.4	2,044.5
sick	0.0228	0.9739	0.8488	0.9074	1,349.9	1.8	54,264.8	0.0060	0.9618	0.9278	0.9442	74.3	8.8	5,966.7
splice-EI	0.0297	0.9565	0.9647	0.9606	1,272.7	8.0	144,123.1	0.0001	0.9104	0.9886	0.9487	9.2	4.2	20,152.1
Splice-IE	0.0372	0.9224	0.9419	0.9317	1,356.1	9.4	168,459.7	0.0003	0.8281	0.9787	0.8993	19.0	4.4	21,556.8
tic-tac-toe	0.1243	0.8029	0.7872	0.7906	187.3	5.1	917.6	0.0943	0.9145	0.9202	0.9165	104.6	7.0	366.8
titanic	0.1162	0.6403	0.6360	0.5551	580.0	2.0	5,399.3	0.0014	0.8148	0.5991	0.6980	5.6	4.0	1,414.4
vehicleVAN	0.0394	0.9448	0.8906	0.9165	200.5	2.7	1,632.0	0.0156	0.9449	0.9617	0.9527	28.6	7.4	376.5
vowelZ	0.0414	0.9789	0.8333	0.8881	184.6	2.0	1,221.1	0.0101	0.9678	0.7333	0.8270	20.9	4.1	307.2
yeastCYT-POX	0.0483	0.9460	0.5500	0.5912	65.8	2.5	221.5	0.0270	0.9222	0.6000	0.6522	22.5	4.3	93.7
yeastEXC	0.0596	0.9696	0.5583	0.7193	356.1	1.8	2,715.6	0.0075	0.9515	0.6417	0.7589	17.8	4.5	583.2
yeastME1	0.0777	0.9777	0.7583	0.8550	321.4	2.6	2,495.4	0.0055	0.9749	0.8417	0.9007	15.3	4.0	544.7
yeastME2	0.0510	0.9447	0.4259	0.5467	425.2	1.3	2,455.0	0.0108	0.8596	0.7090	0.7671	27.2	4.2	556.9

Table 18: Comparison of the performance of instance and feature selection methods in terms of *G*-mean and storage requirements for class-imbalanced datasets. The table shows the win/loss record of each algorithm in the column against the algorithm in the row, the *p*-value of a two-tailed sign test on the win/loss record, p_s, and the *p*-value of the Wilcoxon test, p_w. Significant differences at a confidence level of 95% are marked with a tick (√).

Compared Method		Memetic IS+FS
G-mean		
CHC IS+FS	win/loss	32/8
	p_s	0.0002 √
	R^+/R^-	154.0/666.0
	p_w	0.0005 √
Storage requirements		
CHC IS+FS	win/loss	37/3
	p_s	0.0000 √
	R^+/R^-	18.0/802.0
	p_w	0.0000 √

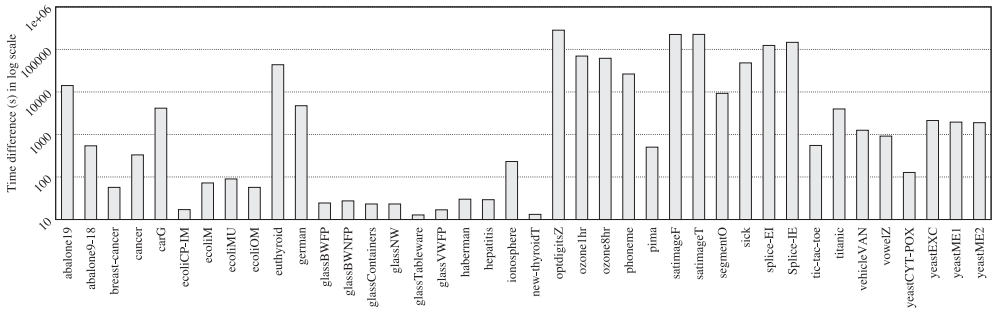


Figure 7: Difference between the execution time, in logarithmic scale, of our method and CHC IS+FS. A filled bar means that our algorithm is faster, and an empty bar means that our algorithm is slower. (In this case, our algorithm is faster for all datasets.)

The runtime comparison is shown in Figure 7. This plot shows that our algorithm is faster than the standard CHC on all of the datasets.

5.3 Using Other Classifiers

A natural question than can be raised about the utility of our approach is whether the obtained results using 1-NN rule are competitive when compared with other widely used classifiers, such as decision trees or support vector machines (SVMs). Although the achieved reduction is remarkably large, the results would not be useful if the performance of the resulting classifier is poor. To test this issue, in this section we compare the performance of our memetic algorithm against a decision tree using the C4.5 learning algorithm (Quinlan, 1993) and an SVM. We chose these two classifiers because they are among the most widely used in any machine learning application. Decision trees and SVMs are sensitive to parameters, so we performed our experiments

Table 19: Comparison in terms of testing error of our memetic algorithm and a decision tree and an SVM. The table shows the win/loss record of each algorithm in the column against the algorithm in the row, the p-value of a two-tailed sign test on the win/loss record, p_s, and the p-value of the Wilcoxon test, p_w. Significant differences at a confidence level of 95% are marked with a tick ($\surd$).

Compared Method		Decision tree	SVM
Memetic	win/loss	18/37	24/29
	p_s	0.0145 $\surd$	0.5831
	R^+/R^-	1,104.0/436.0	879.0/661.0
	p_w	0.0051 $\surd$	0.3611

using cross-validation for setting the values of the parameters. For each of the classifiers used, we obtained the best parameters from a set of different values. For SVM we tried a linear kernel with $C \in \{0.1, 1, 10\}$, and a Gaussian kernel with $C \in \{0.1, 1, 10\}$ and $\gamma \in \{0.0001, 0.001, 0.01, 0.1, 1, 10\}$, testing all 18 possible combinations. For C4.5, we tested one and 10 trials and softening of thresholds trying all four possible combinations. Although this method does not ensure an optimum set of parameters, at least a good set of parameters is obtained in a reasonable time. The SVM learning algorithm was programmed using functions from the LIBSVM library (Chang and Lin, 2001). For C4.5 we used the source code provided by the author of the algorithm.

Table 19 shows the comparison of our memetic algorithm against these two methods. For C4.5, the table shows that our memetic algorithm performed significantly better than C4.5. For an SVM, the memetic algorithm matched the performance of SVMs with a simpler model.

Furthermore, the principles of our approach can also be applied to other classifiers as well. Other learners can benefit from the instance and feature selection algorithm. Classifiers whose complexity is related to the size of the training set, such as decision trees and SVMs, can benefit from instance and feature selection because the constructed classifier would be simpler (Cano et al., 2007; Sebban et al., 2000). When the instances selected are used as a training set for an instance-based learner, the terms prototype selection or training set selection (García-Pedrajas, 2011) are used more often than instance selection.

Our method can be used without any significant modification with any of these classifiers. The only difference is that instead of using a 1-NN rule to evaluate an individual, we train a certain classifier and obtain its accuracy. In this section we present experiments showing the applicability of our method when using decision trees and SVMs as classifiers. We chose these two classifiers because their complexity depends on the quality of the training set (Sebban et al., 2000).

The parameters of the memetic algorithm were the same used in the previous experiments. The only modification is in the classifier used to evaluate the accuracy of every individual. Table 20 shows the comparison. In terms of accuracy, our memetic algorithm matched the performance of the classifiers for both methods, decision trees and SVMs. Furthermore, the method significantly reduced the size of the classifiers obtained in terms of either tree nodes or support vectors. These results support the usefulness of our approach in other classification methods.

In the previous experiments, we also tested the performance of our memetic algorithm using 1-NN algorithm for class-imbalanced datasets. Thus, we also tested the

Table 20: Comparison in terms of testing error of our memetic algorithm using decision trees and SVMs. The table shows the win/loss record of each algorithm in the column against the algorithm in the row, the p-value of a two-tailed sign test on the win/loss record, p_s, and the p-value of the Wilcoxon test, p_w. Significant differences at a confidence level of 95% are marked with a tick ($\surd$).

Compared Method		Memetic (C4.5)	Compared Method		Memetic (SVM)
Testing error					
C4.5	win/loss	25/27	SVM	win/loss	21/30
	p_s	0.8899		p_s	0.2624
	R^+/R^-	767.5/772.5		R^+/R^-	717.0/823.0
	p_w	0.9833		p_w	0.6570
Classifiers size					
C4.5	win/loss	51/4	SVM	win/loss	55/0
	p_s	0.0000$\surd$		p_s	0.0000$\surd$
	R^+/R^-	113.0/1,427.0		R^+/R^-	0.0/1,540.0
	p_w	0.0000$\surd$		p_w	0.0000$\surd$

Table 21: Comparison in terms of G-mean of our memetic algorithm and a decision tree and an SVM for class-imbalanced datasets. The table shows the win/loss record of each algorithm in the column against the algorithm in the row, the p-value of a two-tailed sign test on the win/loss record, p_s, and the p-value of the Wilcoxon test, p_w. Significant differences at a confidence level of 95% are marked with a tick ($\surd$).

Compared Method		Decision tree	SVM
Memetic	win/loss	5/35	7/33
	p_s	0.0000$\surd$	0.0000$\surd$
	R^+/R^-	90.0/730.0	93.0/727.0
	p_w	0.0000$\surd$	0.0000$\surd$

behavior of our proposal using C4.5 and SVMs for class-imbalanced datasets. First, we compared the results of the memetic algorithm using 1-NN against the performance of a decision tree and a support vector machine. The results are shown in Table 21. The table shows a significantly better performance of our method with differences that are even larger than for the previous case.

As the next experiment, as in the previous comparison, we applied the memetic algorithm using as evaluator a decision tree and an SVM. The results are shown in Table 22. The table shows that the memetic approach achieved better and simpler classifiers than the corresponding classification method using all the dataset. In summary, these results support the usefulness of our memetic algorithm when its performance is compared with two of the best current classification methods.

6 Conclusions

In this paper, we have presented a new memetic algorithm for simultaneously performing instance and feature selection for instance-based learning. The proposed method combines a CHC genetic algorithm with four different local search procedures. A new fitness function was introduced to avoid the undesired effect of removing too many

Table 22: Comparison in terms of *G*-mean and reduction of our memetic algorithm using a decision trees and SVMs and standard decision trees and SVMs. The table shows the win/loss record of each algorithm in the column against the algorithm in the row, the *p*-value of a two-tailed sign test on the win/loss record, p_s, and the *p*-value of the Wilcoxon test, p_w. Significant differences at a confidence level of 95% are marked with a tick ($\surd$).

Compared Method		Memetic (C4.5)	Compared Method		Memetic (SVM)
Testing error					
C4.5	win/loss	31/8	SVM	win/loss	31/8
	p_s	0.0003$\surd$		p_s	0.0003$\surd$
	R^+/R^-	61.5/758.5		R^+/R^-	101.5/718.5
	p_w	0.0000$\surd$		p_w	0.0000$\surd$
Classifiers size					
C4.5	win/loss	39/0	SVM	win/loss	40/0
	p_s	0.0000$\surd$		p_s	0.0000$\surd$
	R^+/R^-	0.5/819.5		R^+/R^-	0.0/820.0
	p_w	0.0000$\surd$		p_w	0.0000$\surd$

features that have been observed in preliminary experiments using the CHC algorithm. This new fitness function modifies the way feature reduction affects the fitness of an individual.

Additionally, we have proposed an extension of the stratification method to allow its application to instance and feature selection. The stratification has proven to be very efficient for scaling up the proposed memetic algorithm, while maintaining, and even improving, the performance of the algorithm in many datasets.

The experiments have been carried out considering different situations. We have compared our method with the best thus far, a CHC algorithm, in small to medium sized datasets, in large datasets using stratification, and in class-imbalanced datasets. In all of the three situations, our memetic approach achieved better performance than CHC in terms of storage requirements and testing error. The differences are statistically significant, as shown by the Wilcoxon test. The good performance of the method using decision trees and SVMs has also been shown.

We have also shown that our method is faster than CHC in most cases. The evaluation of an individual has a time complexity quadratic in the number of instances. Our memetic algorithm is able to improve the reduction in the number of instances from the beginning of the evolution, which means that the evaluation of the individuals is made faster and large reductions in runtime are shown in the experiments.

Acknowledgments

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

References

Agrawal, R., Imielinski, T., and Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 207–216.

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

Almuallim, H., and Dietterich, T. G. (1991). Learning with many irrelevant features. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pp. 547–552.

Angiulli, F. (2007). Fast nearest neighbor condensation for large data sets classification. *IEEE Transactions on Knowledge and Data Engineering*, 19:1450–1464.

Bache, K., and Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/MI]. Irvine, CA: University of California, School of Information and Computer Science.

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

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

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

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

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

Brodley, C. E. (1995). Recursive automatic bias selection for classifier construction. *Machine Learning*, 20(1–2):63–94.

Cano, J. R., Herrera, F., and Lozano, M. (2003). Using evolutionary algorithms as instance selection for data reduction in KDD: An experimental study. *IEEE Transactions on Evolutionary Computation*, 7(6):561–575.

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

Cano, J. R., Herrera, F., and Lozano, M. (2007). Evolutionary stratified training set selection for extracting classification rules with trade off precision-interpretability. *Data & Knowledge Engineering*, 60(1):90–108.

Chang, C.-C., and Lin, C.-J. (2001). *LIBSVM: A Library for Support Vector Machines*. http://www.csie.ntu.edu.tw/~cjlin/libsvm/. Accessed May 20, 2013.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16:321–357.

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

Chen, X., Ong, Y.-S., Lim, M.-H., and Tan, K. C. (2011). A multi-facet survey on memetic computation. *IEEE Transactions on Evolutionary Computation*, 15:591–607.

Dash, M., Choi, K., Scheuermann, P., and Liu, H. (2002). Feature selection for clustering—A filter solution. In *Proceedings of the Second International Conference on Data Mining*, pp. 115–122.

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

de Souza, J. T., do Carmo, R. A. F., and de Campos, G. A. L. (2008). A novel approach for integrating feature and instance selection. In *Proceedings of the 7th International Conference on Machine Learning and Cybernetics*, pp. 374–379.

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*. New York: Wiley.

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

Derrac, J., García, S., and Herrera, F. (2010a). Ifs-coco: Instance and feature selection based on cooperative coevolution with nearest neighbor rule. *Pattern Recognition*, 43:2082–2105.

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

Eshelman, L. J. (1990). *The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination*. San Mateo, CA: Morgan Kauffmann.

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

García, S., Fernández, A., and Herrera, F. (2009). Enhancing the effectiveness and interpretability of decision tree and rule induction classifiers with evolutionary training set selection over imbalanced problems. *Applied Soft Computing*, 9:1304–1314.

García, S., and Herrera, F. (2009). Evolutionary undersampling for classification with imbalanced datasets: Proposals and taxonomy. *Evolutionary Computation*, 17(3):275–306.

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

García-Pedrajas, N. (2011). Evolutionary computation for training set selection. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(6):512–523.

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

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

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Reading, MA: Addison-Wesley.

Guyon, I., and Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182.

Ishibuchi, H., and Nakashima, T. (2000). Pattern and feature selection by genetic algorithms in nearest neighbor classification. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 4(2):138–145.

Ishibuchi, H., Nakashima, T., and Nii, M. (2001). Learning of neural networks with GA-based instance selection. In *Proceedings of the IFSA World Congress and 20th NAFIPS International Conference*, Vol. 4, pp. 2102–2107.

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

Kim, Y., Street, W.-N., and Menczer, F. (2000). Feature selection in unsupervised learning via evolutionary search. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 365–369.

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

Krawiec, K., and Bhanu, B. (2007). Visual learning by evolutionary and coevolutionary feature synthesis. *IEEE Transactions on Evolutionary Computation*, 11:635–650.

Kubat, M., Holte, R., and Matwin, S. (1998). Machine learning for the detection of oil spills in satellite radar images. *Machine Learning*, 30:195–215.

Kuncheva, L. (1995). Editing for the *k*-nearest neighbors rule by a genetic algorithm. *Pattern Recognition Letters*, 16:809–814.

Kuncheva, L., and Jain, L. C. (1999). Nearest neighbor classifier: Simultaneous editing and descriptor selection. *Pattern Recognition Letters*, 20:1149–1156.

Lee, W., Stolfo, S.-J., and Mok, K.-W. (2000). Adaptive intrusion detection: A data mining approach. *Artificial Intelligence Review*, 14:533–567.

Leopold, E., and Kindermann, J. (2002). Text categorization with support vector machines. How to represent texts in input space? *Machine Learning*, 46(1–3):423–444.

Leung, Y. W., and Wang, Y. P. (2001). An orthogonal genetic algorithm with quantization for global numerical optimization. *IEEE Transactions on Evolutionary Computation*, 5(1):41–53.

Lewis, D., and Gale, W. (1998). Training text classifiers by uncertainty sampling. In *Proceedings of the Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information*, pp. 73–79.

Liu, H., and Motoda, H. (1998). *Feature selection for knowledge discovery and data mining*. Dordrecht, The Netherlands: Kluwer.

Liu, H., and Motoda, H. (2002). On issues of instance selection. *Data Mining and Knowledge Discovery*, 6:115–130.

Liu, H., and Setiono, R. (1996). A probabilistic approach to feature selection—A filter solution. In *Proceedings of the 13th International Conference on Machine Learning (ICML'96)*, pp. 319–327.

Liu, Y., Yao, X., Zhao, Q., and Higuchi, T. (2001). Evolving a cooperative population of neural networks by minimizing mutual information. In *Proceedings of the 2001 IEEE Congress on Evolutionary Computation*, pp. 384–389.

Louis, S. J., and Li, G. (1997). Combining robot control strategies using genetic algorithms with memory. *Evolutionary Programming VI Lecture Notes in Computer Science*, Vol. 1213 (pp. 431–442).

Lozano, M., Herrera, F., Krasnogor, N., and Molina, D. (2004). Real-coded memetic algorithms with crossover hill-climbing. *Evolutionary Computation*, 12:273–302.

Mao, K. Z. (2004). Orthogonal forward selection and backward elimination algorithms for feature subset selection. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 34:629–634.

Maudes-Raedo, J., Rodríguez-Díez, J. J., and García-Osorio, C. (2008). Disturbing neighbors diversity for decision forest. In G. Valentini and O. Okun (Eds.), *Workshop on Supervised and Unsupervised Ensemble Methods and Their Applications (SUEMA 2008)*, pp. 67–71.

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

Molina, D., Lozano, M., García-Martínez, C., and Herrera, F. (2010). Memetic algorithms for continuous optimisation based on local search chains. *Evolutionary Computation*, 18:27–63.

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

Neshatian, K., and Zhang, M. (2011). Using genetic programming for context-sensitive feature scoring in classification problems. *Connection Science*, 23:183–207.

Neshatian, K., Zhang, M., and Andreae, P. (2012). A filter approach to multiple feature construction for symbolic learning classifiers using genetic programming. *IEEE Transactions on Evolutionary Computation*, 16(5):645–661.

Ng, K., and Liu, H. (1999). Customer retention via data mining. *AI Review*, 14:590.

Nigam, K., Mccallum, A.-K., Thrun, S., and Mitchell, T. (1999). Text classification from labeled and unlabeled documents using em. In *Proceedings of Machine Learning*, pp. 103–134.

Olvera-López, J. A., Carrasco-Ochoa, J. A., Martínez-Trinidad, J. F., and Kittler, J. (2010). A review of instance selection methods. *Artificial Intelligence Reviews*, 34:133–143.

Orriols-Puig, A., and Bernadó-Mansilla, E. (2009). Evolutionary rule-based systems for imbalanced data sets. *Soft Computing*, 13:213–225.

Provost, F. J., and Kolluri, V. (1999). A survey of methods for scaling up inductive learning algorithms. *Data Mining and Knowledge Discovery*, 2:131–169.

Pudil, P., Novovičová, J., and Kittler, J. (1994). Floating search methods in feature selection. *Pattern Recognition Letters*, 15:1119–1125.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Mateo, CA: Morgan Kaufmann.

Reeves, C. R., and Bush, D. R. (2001). Using genetic algorithms for training data selection in RBF networks. In H. Liu and H. Motoda (Eds.), *Instances selection and construction for data mining* (pp. 339–356). Dordrecht, The Netherlands: Kluwer.

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

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

Sebban, M., Nock, R., Chauchat, J. H., and Rakotomalala, R. (2000). Impact of learning set quality and size on decision tree performances. *International Journal of Computers, Systems and Signals*, 1(1):85–105.

Sun, Y., Kamel, M. S., Wong, A. K., and Wang, Y. (2007). Cost-sensitive boosting for classification of imbalanced data. *Pattern Recognition*, 40:3358–3378.

Swets, D.-L., and Weng, J.-J. (1995). Efficient content-based image retrieval using automatic feature selection. In *Proceedings of the IEEE International Symposium on Computer Vision*, pp. 85–90.

Tan, K. C., Khor, E. F., and Lee, T. H. (2003). An evolutionary algorithm with advanced goal and priority specification for multi-objective optimization. *Journal of Artificial Intelligence Research*, 18:183–215.

Whitley, D. (1989). The GENITOR algorithm and selective pressure. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pp. 116–121.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics*, 1:80–83.

Wilson, D. R., and Martinez, T. R. (2000). Reduction techniques for instance-based learning algorithms. *Machine Learning*, 38:257–286.

Xing, E.-P., Jordan, M.-I., and Karp, R.-M. (2001). Feature selection for high-dimensional genomic microarray data. In *Proceedings of the 18th International Conference on Machine Learning*, pp. 601–608.

Xue, B., Cervante, L., Shang, L., Browne, W. N., and Zhang, M. (2012). A multi-objective particle swarm optimization for filter-based feature selection in classification problems. *Connection Science*, 24:91–116.

Xue, B., Zhang, M., and Browne, W. N. (2013). Particle swarm optimization for feature selection in classification: A multi-objective approach. *IEEE Transactions on Cybernetics*. In press.

Yu, K., Xu, X., Ester, M., and Kriegel, H.-P. (2003). Feature weighting and instance selection for collaborative filtering: An information-theoretic approach. *Knowledge and Information Systems*, 5:201–224.

Zhang, L., Chen, C., Bu, J., and He, X. (2012). A unified feature and instance selection framework using optimum experimental design. *IEEE Transactions on Image Processing*, 21:2379–2388.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., and Grunert, V. (2003). Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132.