Applied Soft Computing 11 (2011) 733–743

Contents lists available at ScienceDirect

Applied Soft Computing

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

Efficient Distributed Genetic Algorithm for Rule extraction

Miguel Rodríguez*, Diego M. Escalante, Antonio Peregrín

Dept. of Information Technologies, University of Huelva, 21819 Huelva, Spain

ARTICLE INFO

Article history:
Received 15 March 2009
Received in revised form
18 December 2009
Accepted 29 December 2009
Available online 13 January 2010

Keywords:
Classification rules
Rule induction
Distributed computing
Coarse-grained implementation
Parallel genetic algorithms

ABSTRACT

This paper presents an Efficient Distributed Genetic Algorithm for classification Rule extraction in data mining (EDGAR), which promotes a new method of data distribution in computer networks. This is done by spatial partitioning of the population into several semi-isolated nodes, each evolving in parallel and possibly exploring different regions of the search space. The presented algorithm shows some advantages when compared with other distributed algorithms proposed in the specific literature. In this way, some results are presented showing significant learning rate speedup without compromising the accuracy.

1. Introduction

Nowadays the size of datasets is growing quickly due to the widespread use of automatic processes in commercial and public domains and the lower cost of massive storage. Mining large datasets to obtain classification models with prediction accuracy can be a very difficult task because the size of the dataset can make data mining algorithms inefficacy and inefficient.

There are three main approaches to tackling the scaling problem:

- Use as much as possible a priori knowledge to search in subspaces small enough to be explored.
- Perform data reduction.
- Algorithm scalability.

The third approach, algorithm scalability, promotes the use of computation capacity in order to handle the full dataset. The use of computer grids to achieve a greater amount of computational resources has become more popular over the past few years because they are much more cost-effective than single computers of comparable speed. The main challenge when using distributed computing is the need for new algorithms that take the architecture into account. Genetic algorithms are especially well suited for this task because of their implicit parallelism. As a typical population algorithm, there is a direct way of distributing the algorithm by the use of several smaller populations that interchange individuals occasionally. This kind of distributed GA achieves a significant speedup when used on a network of computers and prevents an early convergence by keeping diversity in several populations.

There have been several efforts to make use of models based on distributed genetic algorithms (GA) in data mining emphasising aspects like scalability and efficiency. REGAL [10] and NOW G-Net [1] are well known references of this approach. Both of them increase the computational resources via the use of data distribution on a network of loosely coupled workstations.

In this paper we present an Efficient Distributed Genetic Algorithm for classification Rule extraction (EDGAR) with dynamic data partitioning that shows advantages in scalability for exploring high complexity search spaces with comparable classification accuracy.

The outline of the contribution is as follows: in Section 2 we review the distributed genetic models in rule induction. Section 3 is devoted to analysing the proposed algorithm and the strategies followed to keep the algorithm scalable. The experimental study developed is shown in Section 4 and finally, we reach some conclusions in Section 5. Appendix A is included containing a detailed table of results obtained in our study.

2. Machine learning and parallel genetic algorithms

This section reviews the main streams found in literature about parallel genetic algorithms in rule induction. The first subsection describes the approaches commonly used in the area to achieve machine learning. The second subsection is focused on the

* Corresponding author. Tel.: +34 959217372.
E-mail addresses: miguel.rodriguez@dti.uhu.es (M. Rodríguez), diego.escalante@dti.uhu.es (D.M. Escalante), peregrin@dti.uhu.es (A. Peregrín).

1568-4946/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2009.12.035

main strategies for parallelising genetic algorithms in data mining related tasks.

2.1. Genetic algorithms in data mining

Genetic algorithms are search algorithms based on natural genetics that provide robust search capabilities in complex spaces, and thereby offer a valid approach to problems requiring an effective search process [15].

GA has achieved a reputation for robustness in rule induction, in common problems associated with real world mining (noise, outliers, incomplete data, etc.). GA can be dedicated to machine learning algorithms [3]. For example, the search space can be seen as the entire possible hypothesis rule base that covers the data and the goodness can be formulated as a coverage function over a number of learning examples.

A key point in GA implementation is the selected representation; the proposals in the specialist literature follow two approaches in order to encode rules within a population of individuals:

The "Chromosome = Set of rules", also called the Pittsburgh approach, in which each individual represents a rule set [23]. The chromosome evolves a complete rule set and they compete among themselves throughout the evolutionary process. GABIL [6] and GA-MINER [9] are proposals that follow this approach.

The "Chromosome = Rule" approach, in which each individual codifies a single rule, and the whole rule set is provided by combining several individuals in a population (rule cooperation) or via different evolutionary runs (rule competition).

In turn, within the "Chromosome = Rule" approach, there are three generic proposals:

- The Michigan approach, in which each individual encodes a single rule. These kinds of systems, usually called learning classifier systems [14], are rule-based, message-passing systems that employ reinforcement learning and a GA to learn rules that guide their performance in a given environment. The GA is used to detect new rules that replace the bad ones via the competition between the chromosomes in the evolutionary process.
- The IRL (Iterative Rule Learning) approach uses several GA executions to obtain the rule set. Chromosomes compete in every GA run, choosing the best rule per run. The global solution is formed by the best rules obtained when the algorithm is run multiple times. SIA [26] is a proposal that follows this approach.
- The GCCL (genetic cooperative–competitive learning) approach encodes the rule set as the complete population or a subset of it. The chromosomes compete and cooperate simultaneously. This strategy requires the conservation of species in the same population to avoid a final solution consisting of clones of the best individual. COGIN [13], REGAL [10] and NOW G-Net [1] are examples using this representation.

2.2. Parallel genetic algorithms in data mining

The definition of scalable in computing could be something like "able to support the required quality of service as the system load increases". Applied to data mining and more precisely to supervised classification, the system load is provided by the complexity and the size of the dataset (in attributes or training examples), and the quality of service is relative to the processing time for producing a similar classifier mainly in terms of accuracy and interpretability.

As complexity of the dataset increases, GAs exhibit high computational cost and degradation of the quality of the solutions. Efforts towards solving these shortcomings have been made in several directions, and parallel GAs is one of the most significant. We stress four approaches that represent the main parallelisation strategies:

- Global parallelisation [8]. Only the evaluation of individuals' fitness values is parallelised by assigning a fraction of the population to each processor to be evaluated. This is an equivalent algorithm that will produce the same results as the sequential one.
- Coarse-grained [8] and fine-grained parallelisation [20]. In the former, the entire population is partitioned into subpopulations (demes). A GA is run on each subpopulation, and exchange of information between demes (migration) takes place occasionally [8] in analogy with the natural evolution of spatially distributed populations such as the island model (Fig. 1). Fine-grained course has just one individual per processor and rules to perform crossover in the closest neighbourhood defined by a topology.
- Supervised data distribution [11]. A master process uses a group of processors (slaves) by sending them partial tasks and a smaller data partition. Each node has a complete GA or a part of it. The master process uses the partial results to reassign data and tasks [10,1] to the processors until some condition is met.
- Not supervised data distribution [18]. The full dataset is shared out in several processors and moved to the next processor in the topology after a pre-specified number of generations without removing the existing population. The individuals will try to cover the newly arrived training data.

The proposal presented in this work follows a GCCL approach and as a parallelisation strategy uses a coarse-grained implementation and a master process to build up the final classifier on the basis of partial results.

3. Genetic Learning Proposal: EDGAR algorithm

This section describes the characteristics of an *Efficient Distributed Genetic Algorithm for classification Rules extraction*, from now on designated EDGAR. The proposed algorithm distributes population (rules in a GCCL approach) and training data in a coarse-grained model to achieve scalability.

We start by explaining the distributed model in Section 3.1. Sections 3.2–3.5 describe the components of the genetic algorithm: representation, genetic operators, genetic search and data reduction. Finally, Section 3.6 is devoted to the strategy used to determine the best set of rules that will make up the classifier from the redundant population of rules generated by the GCCL algorithm.

3.1. Distributed model

This subsection explains the main properties of the distributed framework. First, in Section 3.1.1 we describe the use of the coarse-grained implementation with data partition.

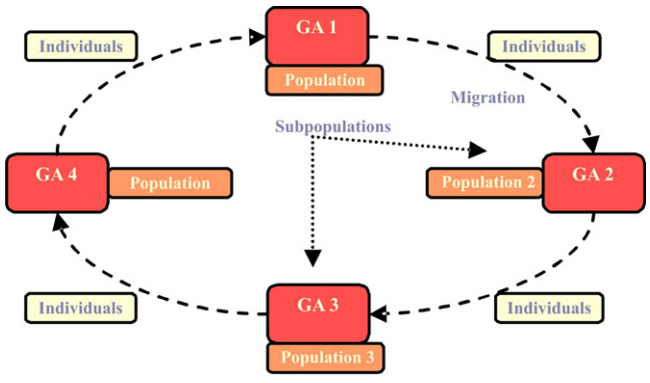

Fig. 1. Island model.

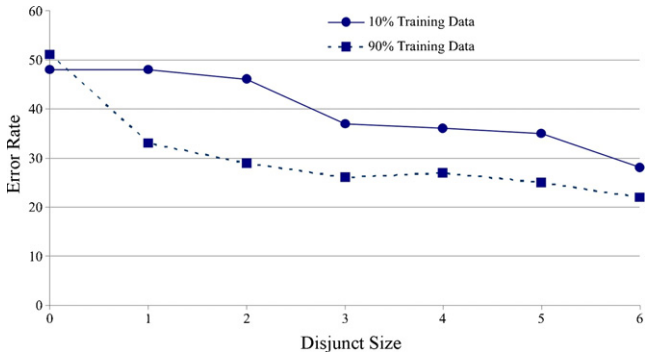

Fig. 2. Effect of training size on disjunct error rate.

3.1.1. Parallelisation strategy

The coarse-grained parallelisation [8] model is a simple way to improve the search capability of a GA. Nevertheless, it has some disadvantages when dealing with large datasets because each node will deal with a complete copy of the dataset, making it less efficient.

To achieve a better speedup on this model, we propose to assign different partitions of the learning data to each node. The GA in each node will try to cover the local data proposing a concept description, but the main characteristics of the coarse-grained model will remain. On one hand, the migration of the best individuals between subpopulations will enforce those individuals (rules) that perform properly in more than one node and, on the other hand, we propose a ring topology that will prevent an early convergence (see Fig. 3).

3.1.2. Data Learning Flow technique

The data partition in a coarse-grained implementation is able to produce similar quality classifiers to a single GA [20] if the data does not have small disjuncts [12]. In these cases, a few data examples representing a concept description (one rule) may be split into several nodes in the initial partition, preventing the local GA from inducing the rule.

Fig. 2 [27], shows the relation between error rates and training set size. This result further suggests that the presence of small disjuncts in a small training dataset decreases the accuracy and should take training set size into account.

In our proposal the training data assigned to a local GA will be a smaller percentage in size relative to the original dataset. For instance, a dataset using 50% partition for training and testing and 10 nodes as configuration for data distribution will handle in each node just 5% of training data compared with the full dataset.

We propose a novel technique called Data Learning Flow (DLF) to join the examples of small disjuncts together. It is based on the idea that a training example not properly covered in one node may be better covered in another data partition. Training examples covered by low fitness rules are copied to the next node in the neighbourhood. The maximum migration rate of learning examples was set to 1% of the local dataset to keep the size of local dataset small enough.

3.1.3. Elite pool

The standard coarse-grained model will converge with the same population in all nodes, but our algorithm may produce a different classifier in each node due to the data partition. Moreover, a rule with a high accuracy in a local data partition may have a poor coverage when applied against the entire dataset. For this reason, the local node cannot decide whether it has a good set of rules. We propose an elite pool that holds the full dataset, to:

- Validate whether a rule is good enough to have been kept.
- Stop the algorithm when a stall criterion is reached.
- Extract the final solution.

The process is as follows: the nodes send their best rules to the elite pool. Then, the pool selects the rules with better coverage (see Section 3.7) and fitness. The process ends when the classification ratio in the pool does not improve after a number of iteration of this process (Central Stall Parameter or C_{SP}).

The number of rules accumulated can become a bottleneck. In order to keep it as small as possible, we have the following policies:

- Only the rules in the last proposed classifier are kept in the pool.
- Only new discovered rules are included; the nodes keep track of the sent rules and the pool checks against the received rules, preventing evaluating them again. This check is run in a reasonable time frame by the use of native Java hash table implementation based on the chromosome string representation to detect the duplicated rules.

3.2. Genetic algorithm

EDGAR uses a GA in each node in a ring communications topology with the neighbourhood as in the aforementioned island model and some training data for examples poorly covered by the local classifier.

Each node will work on a partition of the full dataset generated by random samples. The initial population is created constructing rules that cover some of the examples in the local dataset (seeding). Universal Suffrage (US) operator selects a set of individuals (g) for crossover and mutation in each generation. Each offspring will replace a randomly selected individual in the current population.

After a number of generations (Local Number of Generations L_{NG}), some operations are performed (see Fig. 4):

- Using a greedy algorithm (see Section 3.7) to extract the set of rules that better classifies the local data and copy them to the next node in the ring and the pool.
- Randomly replace selected individuals in the current population with the individuals received from the previous node in the ring.
- Copy the learning examples not covered or covered by low fitness rules to the next node in the ring.
- Perform data training set reduction if the best individual does not change (see Section 3.6).

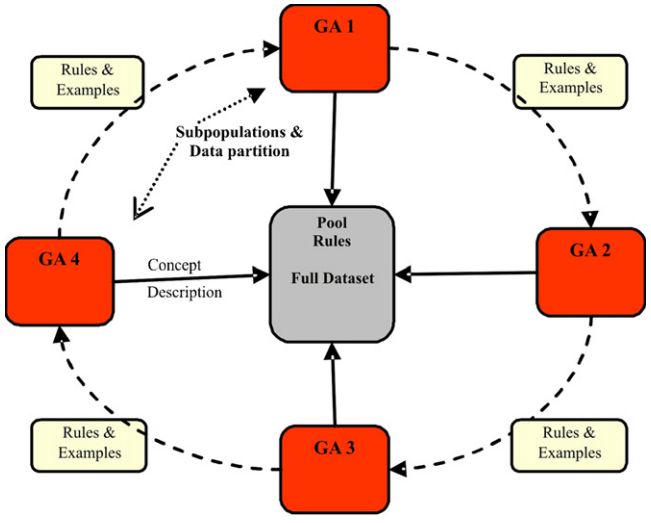

Fig. 3. Distributed model.

```
Generate initial population using seeding
While (exist training data or external stop)
  For a number of generations LNG
    Select g individuals by US
      For each individual
          Perform crossover
          Perform mutation
       End
      Replace g individual from population
      Exchange individuals
      Exchange training examples
    End
  End
  Extract set of rules by greedy algorithm
  Send set of rules to Central Pool
  Data training set reduction
End
```

Fig. 4. Genetic algorithm.

The node will stop searching when there is no data in the local dataset or because the pool node orders it.

3.3. Representation

EDGAR uses a fixed length bit string representation to code a disjunctive rule. The use of fixed length chromosomes allows simpler genetic operators.

Each different possible value of an attribute in a rule is represented as a single bit. A bit set to 1 signifies the presence of the value in the rule and a bit set to 0 means the absence of this value in the rule. One advantage of this representation is that mutation on a bit of the genotype only leads to a small change on the phenotype.

A rule is composed of characteristics $C = c_1, c_2, \ldots, c_j$, where each one can take only one value in each instance of the data mined but the rule may have more than one value for this characteristic.

For example, Fig. 5 shows a rule with three antecedents $c_1(v_1, v_2, v_3)$, $c_2(v_4, v_5)$, $c_3(v_6, v_7, v_8)$ and the consequent $class(v_9, v_{10})$. As seen in this figure, when all the bits corresponding to the same attribute are set to 0, it means that it does not affect the rule.

3.4. Fitness function

The fitness function is based on the following measurements:

- Complexity: considered as the number of conditions in a rule. In a bit string representation, the more zeros that are present in the formula, the fewer conditions in the rule.
- Accuracy: is inversely the number of misclassifications. This means examples covered with a different assigned class.

if c_1 in (v_1,v_3) and c_3 in (v_6) then class is v_{10}

c_1			c_2		c_3			Class	
v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
1	0	1	0	0	1	0	0	0	1

Fig. 5. Rule representation example.

The fitness function is:

$$f(r) = \left(1 + \frac{zeros(r)}{length(r)}\right)^{-1*FP}$$

where *FP* means the number of covered examples predicted has false positives (different consequent as the rule). *Zeros(r)* is the number of zeroes in the bit string representation of the rule *r* and *length* is the chromosome length in bits. This function will give very low values to rules covering just a few numbers of false positives. As soon as the rule improves, the length will take on more relevance.

3.5. Genetic operators

The species formation is of great importance in a GCCL algorithm. For this purpose, EDGAR uses the US selection operator first used in [10]. This mechanism creates coverage niches that do not compete with each other (co-evolution) through a voting process: each generation a set of learning examples is selected. The process searches the set of rules in the population that better covers each example (fitness) and perform a weighted roulette based on the fitness and the number of positive cases. In the event that no rule covers an example, a new rule is created generalising the learning example.

Crossover and mutation operators are based on standard bit string representation and are applied on the selected parents based on a given probability. The crossover operator used is the two-point crossover where the offspring are evaluated to cover at least one example before being inserted in the population. The mutation operator changes one random bit in the chromosome depending on the individual fitness. The mutation operator behaves in a different manner depending on the fitness of the selected individual. In the early stage of the process, some of the rules cover only a few examples. In this case, the mutation is driven to generalise the rule, increasing the possibilities of covering new examples (i.e. generalises the rule). If the offspring has a higher fitness than its parents, it will be inserted in the population; otherwise the parent will be used instead.

3.6. Data training set reduction under evolution and covering

US depends on a proper population–dataset ratio for a good coverage. When training examples representing a concept are in a smaller amount than other concepts, the rules for the some concepts may disappear under the attraction of the rules with more voters. For instance, in a local dataset of 1000 instances of training data examples and 10 individuals in the population, US will use 10 examples randomly each time to select the 10 best rules that represent them. The probability of a particular instance of being selected will be less than 1%. Once selected, there will be at least one rule representing it, but this one will disappear in the next 100 generations with a probability of 99%.

EDGAR deletes the examples already learned to focus on those less represented examples. The process is as follows: when the algorithm detects that the proposed rule set does not change in a number of consecutive times (Local Stall Parameter L_{SP}), the best rule and its covered data are removed from the node. Therefore, the rest of the examples will receive more computational effort, making it possible to induct rules on them.

This strategy makes the algorithm less dependent on the ratio between learning examples and population because it guarantees that all the examples will be selected either in the standard phase with the initial dataset, or later, once all the examples covered by the already learned rules have been removed from the local dataset.

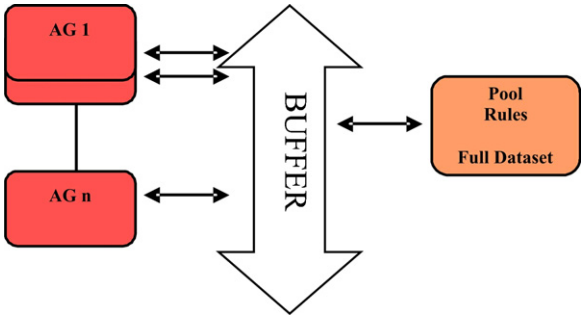

Fig. 6. Communication architecture.

3.7. Generation of a classifier from a redundant rule set

The population in any node is a redundant set of rules that does not specify how they perform the classification. Our aim is to generate the shortest and fastest classifier that covers the training examples. The proposed classifier is an ordered set of rules in which the first applicable rule will return the assigned class. Nevertheless, as the complete set of rules covering the data may be high, the rule position in the classifier is relative to its correct classified examples (True Positive cases or TP). This organisation has some advantages; on one hand, the first rules express a rough idea of the main concept descriptions and on the other hand it gives a better response time for classification in the event of a high number of rules.

The order criteria (Π) ensures that length is taken into account when more than one rule competes for a similar coverage and classification ratio by using accuracy and complexity in a derived expression of the fitness rule:

$$\Pi : \left(1 + \frac{zeros(r)}{length(r)}\right)^{-1*FP^-} * TP$$

Once the rule is selected, all its positive cases are removed and the remaining candidate rules are newly ordered. The process finalises when all examples are covered or there are no more rules in the rule set.

3.8. Architecture scalability

Scalability in parallel implementations is described in the literature [17] as the relation between number of processors and execution time. The following paragraphs describe the policy for keeping execution time scalable considering the network speed and synchronisation in distributed processes.

The components of execution time in distributed systems are [16]: time of each processor (T_{pi}), number of communications (c), average time for communication (T_c), idle time waiting for synchronisation (T_i) and probability of idle status for a node (p).

$$T = \sum_{1}^{n} T_{pi} + c * (T_c + T_i * p)$$

If the term $c^*(T_c + T_i^*p)$ is lower than the computational time expended by each local genetic algorithm, the algorithm will present a speedup independent of network speed and synchronisation issues. To minimise these terms, the following directives were followed:

- Avoid synchronous calls to other processes and process synchronisation. The communication of proposed rules and flow of learning data is performed through buffers (see Fig. 6) avoiding the producer–consumer synchronisation. *Idle time waiting for synchronisation* (T_i) and *probability of idle status* (p) for a node and the pool becomes then close to zero.
- *Communication time* (c^*T_c) depends on size and frequency of communication of the best individuals and training data (DLF technique) to the neighbourhood and to the pool. A system parameter (Local Number of Generations L_{NG}) allows adjusting the communication frequency (number of generations between communications) to the convergence of the local model and the network speed. If this parameter is too low, the local model will over learn the assigned dataset. If the parameter is too high, the newly arrived individuals will slow down or even prevent the convergence with the local data [8] and the time expended in communication handling will increase. When this time is less than the time used in sending learning data and rule through the network (T_c), the algorithm execution time will be independent of the network speed.

Table 1
Compared percentage accuracy.

	C4.5	EDGAR	REGAL
Monk-1	100	100.0	100.0
Monk-2	67.0	96.0	95.0
Monk-3	100.0	99.6	100.0
Tic-tac-toe	92.9	98.9	98.7
Credit	86.0	85.0	84.0
Breast	94.1	94.0	94.1
Vote	96.4	97.1	96.2

4. Experimental study

In this section, we describe the experimental study carried out on a variety of datasets, ranging from standard benchmark to test the accuracy against standard algorithms in rule induction to specific comparison with more complex problems. The experimental study was carried out in a cluster of 8 workstations with 2 CPU Intel Xeon 3 GHz each. In order to run more than 16 nodes at a time, each node was implemented as a thread in a Java VM and communications time was simulated, adding a time equivalent to an Ethernet 100 mbs in a configuration of 8 processors. As an example, usually the communication unit in Ethernet is 512 bytes per package; if a rule is 30 bytes in length, the pack will have 25 records, which means a delay of 0.0512/25 = 0.00015 s per rule.

Section 4.1 compares EDGAR with standard benchmarks. Section 4.2 analyses the effect of data distribution on accuracy and speedup in a study case. Section 4.3 develops a statistical analysis over a set of commonly used datasets.

4.1. Comparison with standard benchmarks

This section is devoted to testing in a first run whether EDGAR is able to obtain similar accuracy to distributed and non-distributed learners in a variety of datasets chosen from University of California at Irvine repository [25]. The selected problems are well known in the literature so we will simply describe the main characteristics of each. Monk-1, Monk-2 and Monk-3 are artificial classification problems whose aim is to test specific abilities of learning systems. Tic-tac-toe consists in classifying the states of the homonymous strategy game as "winning" or "losing". Credit, Breast and Vote are prediction problems related to the reliability of applicants for credit cards, the prognosis of breast cancer, and the prediction of the vote given by congressmen on the basis of their political previous choices. All dataset testing was performed with 10-fold cross.

Table 1 reports results on this first group of problems. The systems used for the comparison are C4.5 and REGAL. C4.5 [19] is a classical propositional learner, whose results are used as a baseline. Performance of C4.5 is reported in [2]. REGAL was executed in the

Table 2
Execution parameters.

	REGAL	EDGAR
Stopping criteria	500 gen.	$C_{SP} = 5$, $L_{NG} = 20$
Mutation percentage	0.01%	1%
Crossover percentage	60%	90%
Selection percentage *g*	10%	10%
Communication ratio	10%	10% max
Training dataset reduction	–	$L_{SP} = 5$
Training dataset communication ratio	–	1% max

same hardware as EDGAR with the parameter shown in Table 2: 16 nodes and a global population of 400 individuals. For a more detailed execution over a different number of node configurations on different datasets, see Section 4.3.

Monk-1 and Monk-3 are easily handled by most of the algorithms with accuracies of nearly 100%. Monk-2 and Tic-tac-toe are handled with better accuracy in the tree distributed algorithms than in C4.5. Credit is a little bit better in EDGAR than in REGAL, but in this case does not behave better than C4.5. Finally, EDGAR gets better results than the other learners in Vote. Note that algorithm parameters in all runs were not modified and the results show average on 100 runs with a regular behaviour, being able to find good solutions if the population in the nodes is large enough to cover the assigned data.

4.2. Data distribution, speedup and accuracy

This section analyses the effect of data distribution on the accuracy and speedup. For this study, we run REGAL [10] and EDGAR using parameters shown in Table 2. As mentioned previously, REGAL is a distributed genetic algorithm that also uses US.

For these experimental studies, two well known problems were chosen from UCI [25]: Nursery and Mushroom. Mushroom is simple (just two classes) and large enough for testing accuracy. Nursery is also a medium size dataset with six characteristics and five non-balanced classes, three of them representing more than 97% of the dataset.

We evaluated both algorithms using Dietterich [4] 5x2 cross-validation with 50% training and testing and 5 different seeds.

The comparison will measure the accuracy of the classifier and speedup achieved relative to the number of processors. The former is given by the number of rules and classification ratio. The speedup is measured through the total execution time having the same parameters in each execution.

The comparison was carried out with 1600 individuals as the sum of the whole local node population. This population proved to be large enough for both algorithms to obtain the maximum accuracy and lowest number of rules for the datasets.

The experiments were executed with configurations from 4 to 64 nodes to study the impact of the distribution on the referenced variables. All parameters in REGAL execution are from their original paper [10] except for the stopping criteria, calculated experimentally using the same conditions as with the population. The differences in parameter configuration in both algorithms are the reason for the different learning approach. EDGAR needs a higher mutation ratio, as most of the offspring mutated are not used because they are worse than their parents.

Tables 3–6 show averages on 150 executions (5 for each partition, 6 different seeds and 5 node configurations). First column is the number of nodes. Second is the number of communications from a node (either one rule or one learning example). Third column shows execution time in minutes. Finally, the fourth, fifth and sixth columns show the classification results for test and training datasets.

Table 3
Results of Mushroom – REGAL.

Nodes	Comm.	Time	Rules	%Test.	%Tra.
4	318,092	0.79	16	99.93	99.98
8	558,501	0.68	14	99.95	99.99
16	681,782	0.57	15	99.95	99.99
32	1,085,929	0.54	15	99.94	99.99
64	1,267,774	0.43	16	99.95	99.99

Table 4
Results of Mushroom – EDGAR.

Nodes	Comm.	Time	Rules	%Test	%Tra.
4	2129	1.20	14	99.92	99.94
8	3907	0.55	13	99.94	99.96
16	8232	0.35	15	99.96	99.99
32	17,60	0.23	16	99.95	99.90
64	20,11	0.18	16	99.93	99.95

Table 5
Results of Nursery – REGAL.

Nodes	Comm.	Time	Rules	%Test	%Tra.
4	784,870	1.78	290	98.6	98.9
8	2,319,559	1.97	251	99.0	99.3
16	6,304,130	2.32	250	98.9	99.2
32	18,595,490	2.81	268	98.5	98.7
64	50,664,658	3.08	316	97.9	98.0

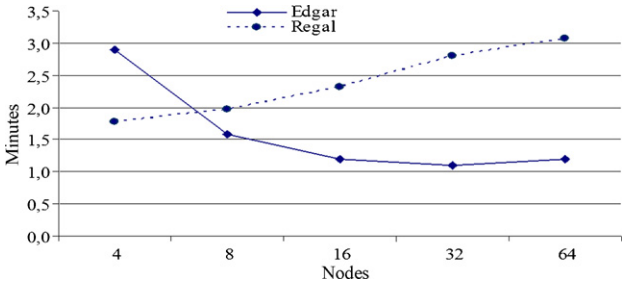

Fig. 7. Compared time Nursery execution.

Analysing Tables 2–4, we can point out some conclusions in each measured property. Regarding the execution time (see Fig. 7), we observe a considerable speedup and a better behaviour than the compared algorithm when the number of processors increases.

Nevertheless, the configuration with four nodes gives better time results in REGAL than in EDGAR. This effect is a consequence of the recalculation in central pool concept every few iterations. On the other hand, the experimentation shows that there is an equilibrium point that depends on each dataset regarding the number of possible partition. Having a large number of partitions may increase the processing time. The local learning examples are not properly covered, because of small disjuncts split into more than one partition or due a local dataset that does not have enough learning examples. DLF will reorganise the learning data in order to have a local dataset large enough to generalise better rules.

Classification accuracy is similar in both algorithms and does not follow any tendency in terms of the number of processors. EDGAR

Table 6
Results of Nursery – EDGAR.

Nodes	Comm.	Time	Rules	%Test	%Tra.
4	6818	2.89	173	99.4	99.7
8	3356	1.59	209	98.5	98.8
16	4836	1.20	206	98.9	99.2
32	8309	1.11	231	98.3	98.8
64	77,859	1.21	199	98.5	98.6

Table 7
EDGAR, covered positives and negative cases for Mushroom execution.

Rule order	Positive cases	Negative cases
1	1985	0
2	1943	0
3	1786	0
4	1532	0
5	1098	0
6	11	0
7	9	0
8	5	0
9	5	0
10	2	0
11	25	22
12	1855	1833
13	1909	1513

is able to find a good solution if the ratio between population and data is enough to generalise a rule. As the ratio of individual and data remains the same in all experiments (1600 individuals as a sum of all local populations) in the different node configuration, the accuracy does not decrease with the number of nodes.

Regarding complexity, the greedy algorithm is able to generate between 60% and 80% less rules in Nursery than REGAL. Mushroom does not show this behaviour, having a similar number of rules.

Table 7 shows the covered cases generated by EDGAR for Mushroom. The rules in the classifier are ordered by number of positive cases*fitness. This order guarantees that the rules with less false positives will be applicable first. This classifier is able to have an accuracy of 99% with only the first five rules. This represents the average behaviour on the classifiers generated by EDGAR. We stress the meaning of the rules with less than 11 examples. All of them represent small disjuncts in the problem. For instance, in a dataset of 4062 instances, these rules represent less than 0.2% of the learning examples. The probability of having these 11 examples in one partition on a 64 node distribution is lower than 0.01%. Thanks to DLF, the algorithm brings together the examples belonging to small disjuncts in some of the nodes, otherwise it will generate one rule per example, increasing the number of rules to two or three times the size of the current classifier.

4.3. Statistical analysis

The aim of this section is to test the behaviour of the proposed algorithm statistically in a representative number of datasets (see Table 8) chosen from the UCI database [25]. The splice dataset provided by Towell et al. [24] was chosen since it was originally used by REGAL [10] as an example case. The number of datasets is determined by the chosen non-parametric test, Wilcoxon signed-ranks test [5]. This test has maximum confidence when the number of paired data (results on dataset) is at least 25. The continuous attributes in the datasets were discretised using 10 equal frequency values because both algorithms work only with nominal attributes. The accuracy of the classifier produced using a discretised dataset may be lower than in an originally discrete dataset, but we think the comparison between two nominal algorithms is still valid because both of them will be affected in a similar way, and allows having a number of standard datasets instead of synthetic datasets or not so common nominal ones. The datasets were chosen taking size criteria into account, with datasets ranging from hundreds to some thousand instances and from two to some ten features. The comparison was carried out with 1000 individuals as the sum of the whole local node population, stopping when no improvement was achieved, and the remaining parameters were the same as in the previous subsection. This configuration will not achieve the best results for a specific dataset, but allows evaluation of the algorithm flexibility in a variety of input conditions.

Table 8
Dataset characteristics.

Dataset	Instances	Features	Classes
Car	1727	6	4
Cleveland	297	13	5
Credit	1000	20	2
Ecoli	336	7	2
Glass	214	9	7
Haberman	305	3	2
House Votes	432	16	2
H Hypothyroid	1920	29	4
Iris	150	4	3
Krvskp	3198	37	2
Monks	432	6	2
Mushrooms	8124	22	2
New-Thyroid	215	5	3
Nursery	12,960	6	2
Pima	768	8	2
Segment	2308	19	7
Soybean	307	35	19
Splice	3190	60	3
Tic-tac-toe	958	9	2
Vehicle	846	18	4
Waveform	5000	41	3
Wine	178	13	3
Wisconsin	683	9	2
Vote	435	16	2
Thyroid	7200	21	3
Zoo	100	16	7

To compare the obtained results non-parametric tests were used, following the recommendations made in [7,21,22]. They are safer than parametric tests since they do not assume normal distribution or homogeneity of variance. As such, these non-parametric tests can be applied to classification accuracies, error ratios or any other measure for evaluation of classifiers, even including model sizes and computation times. Empirical results suggest that they are also stronger than the parametric test. Demšar recommends a set of simple, safe and robust non-parametric tests for statistical comparisons of classifiers. Given that the evaluation of only the mean classification accuracy over all the datasets would hide important information and that each dataset represents a different classification problem with different degrees of difficulty, we have included a second table that shows the average and standard deviation. As mentioned, [7,21,22] recommend a set of simple, safe and robust non-parametric tests for statistical comparisons of classifiers, one of which is the Wilcoxon signed-ranks test.

This is analogous to the paired t-test in non-parametrical statistical procedures; therefore, it is a pairwise test that aims to detect significant differences in the behaviour of two algorithms. In our study, we show the level of significance (p) at which the rejection of null Hypothesis (H_0) can be rejected. A p-value of 0.05 means that H_0 can be rejected with a confidence level of 95%.

The Wilcoxon signed-ranks test works as follows. Let d_i be the difference between the performance scores of the two classifiers in ith out of N_{ds} datasets. The differences are ranked according to their absolute values; average ranks are assigned in case of ties. Let R^+ be the sum of ranks for the datasets in which the second algorithm outperformed the first, and R^- the sum of ranks for the opposite. Ranks of $d_i = 10$ are split evenly among the sums; if there is an odd number of them, one is ignored:

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

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

Table 9
Average results.

	EDGAR			REGAL		
	Rules	Test	Time	Rules	Test	Time
Car						
Mean	61	97%	0.10	66	98%	63.4
sd	6.06	0.006	0.07	8.13	0.006	71.6
Cleveland						
Mean	73	91%	0.07	62	96%	100.5
sd	14.30	0.070	0.07	8.33	0.016	91.6
Credit						
Mean	69	90%	0.05	75	94%	69.4
sd	10.62	0.015	0.03	6.31	0.028	63.3
Ecoli						
Mean	61	90%	0.04	47	96%	95.5
sd	9.89	0.036	0.03	7.84	0.022	86.7
Glass						
Mean	61	92%	0.07	45	96%	106.2
sd	14.24	0.105	0.03	9.57	0.025	98.1
Haberman						
Mean	29	82%	0.04	52	92%	125.4
sd	4.84	0.038	0.03	4.83	0.018	115.5
House Votes						
Mean	26	98%	0.02	26	97%	0.7
sd	5.79	0.026	0.01	5.79	0.025	9.5
Hypothyroid						
Mean	200	95%	1.00	178	95%	35.6
sd	28.08	0.038	0.94	20.42	0.029	27.1
Iris						
Mean	14	96%	0.01	11	99%	36.2
sd	5.55	0.038	0.01	2.55	0.017	9.6
Krvskp						
Mean	62	98%	0.10	54	86%	22.5
sd	8.78	0.014	0.06	10.40	0.092	11.1
Monk						
Mean	46	99%	0.03	58	77%	61.7
sd	7.25	0.043	0.03	6.51	0.055	41.3
Mushroom						
Mean	13	100%	0.14	16	100%	4.9
sd	3.23	0.012	0.08	4.19	0.013	5.0
New-Thyroid						
Mean	14	99%	0.81	14	99%	51.9
sd	2.97	0.027	0.42	2.97	0.027	49.7
Nursery						
Mean	270	99%	1.63	309	98%	91.0
sd	28.41	0.027	1.50	22.41	0.013	77.2
Pima						
Mean	128	94%	0.19	127	94%	65.5
sd	9.34	0.017	0.22	9.49	0.017	68.5
Segment						
Mean	132	99%	0.47	137	99%	30.5
sd	13.51	0.012	0.35	22.00	0.013	29.5
Soybean						
Mean	72	98%	0.41	81	97%	21.8
sd	6.97	0.026	0.36	11.94	0.031	17.6
Splice						
Mean	77	100%	9.71	72	95%	29.1
sd	10.91	0.027	4.53	12.41	0.026	4.5
Tic-tac-toe						
Mean	87	99%	0.06	50	91%	61.4
sd	14.45	0.028	0.04	13.12	0.055	51.8
Vehicle						
Mean	181	95%	0.64	161	95%	39.5
sd	14.11	0.019	1.42	18.32	0.028	46.2
Vote						
Mean	24	97%	0.02	8	96%	11.8
sd	7.13	0.021	0.01	9.36	0.442	
	11.7					
Waveform						
Mean	1026	95%	15.15	1070	93%	15.1
sd	77.79	0.044	8.81	77.79	0.044	8.8
Wine						
Mean	57	97%	0.03	60	97%	1.2
sd	13.51	0.031	0.01	14.52	0.031	0.3
Wisconsin						
Mean	25	98%	0.02	25	100%	14.7
sd	4.10	0.056	0.01	3.63	0.026	5.4
Zoo						
Mean	9	69%	0.02	6	65%	9.1
sd	5.035	0.314	0.01	4.097	0.311	14.5

Let T be the smaller of the sums, $T = min(R^+;R^-)$. If T is less than or equal to the value of the distribution of Wilcoxon for N_{ds} degrees of freedom [28], the null hypothesis of equality of means is rejected.

The Wilcoxon signed-ranks test is more sensitive than the t-test. It assumes commensurability of differences, but only qualitatively: greater differences count still more, which is probably to be desired, but absolute magnitudes are ignored.

From the statistical point of view, the test is safer since it does not assume normal distribution. Moreover, outliers (exceptionally good/bad performances on a few datasets) have less effect on Wilcoxon than on the t-test. Wilcoxon's test assumes continuous differences d_i, which therefore should not be rounded to, say, one or two decimals, since this would decrease the power of the test due to a high number of ties.

Table 9 shows average results and standard deviation on the compared execution over the selected datasets. The first column has the dataset name. The second, the number of rules. The third, four and fifth columns are training, test and processing time respectively for EDGAR. Columns 6–10 are rules, training, test and processing time respectively for REGAL. Table 10 summarises Wilcoxon test results as follows: first column has the number of nodes. Second column express the condition measured in the test. Third, four and fifth columns are the number of wins, loss and ties for the condition. The last column is the p-value.

Table 10
Wilcoxon signed-ranks test.

Nodes	REGAL/EDGAR	R^+	R^-	Ties	p-Value
4	Rules < Rules	16	6	3	0.13
	Acc. < Acc.	17	8	0	0.11
	Time < Time	0	24	1	0.00
8	Rules < Rules	12	9	4	0.49
	Acc. < Acc.	17	8	0	0.31
	Time < Time	1	24	0	0.00
16	Rules < Rules	12	10	3	0.40
	Acc. < Acc.	18	7	0	0.09
	Time < Time	0	25	0	0.00
32	Rules < Rules	10	12	3	0.21
	Acc. < Acc.	17	8	0	0.32
	Time < Time	0	25	0	0.00
All	Rules < Rules	12	10	3	0.73
	Acc. < Acc.	20	4	1	0.01
	Time < Time	0	24	1	0.00

Observing Tables 9–11 (see Appendix A), we can make the following analysis:

- The number of nodes in EDGAR does not follow any trend regarding the accuracy. Even in some of the datasets the results are better with 32 nodes than with 16 or less.
- Processing time decreases in EDGAR with the number of nodes, but it does not achieve a linear speedup.
- Table 10, shows that in average (nodes = All), EDGAR wins in 20 out of 25 of the datasets in accuracy with a confidence of 99%. The other configuration shows a variety of results that does not allow rejection of the null hypothesis. Processing time is better in EDGAR with any configuration in 100% of the cases.
- The number of rules cannot reject the null hypothesis because the confidence is only 27% for the average case and less than 90% in the rest of the node configurations.

5. Conclusions

This work presents a distributed genetic algorithm for classification rules extraction based on the island model and enhanced for scalability with data training partitioning. To be able to generate an accurate classifier with data partition, two techniques were proposed: an elitist pool for rule selection and a novel technique of data distribution (DLF) that uses heuristics based on the local data to dynamically redistribute the training data in the node neighbourhood.

In this study, EDGAR shows a considerable speedup and, moreover, this improvement does not compromise the accuracy and complexity of the classifier.

The complementarities of the proposed techniques allow having low dependency on parameter setting. Proportion of individuals per learning example is compensated by the data training set reduction that will handle the removal of the already learned rules, redirecting computations efforts to the more difficult cases. Seeding operator reintroduces rules preventing loss of diversity. The elite pool also ensures that already discovered rules will be kept in the final classifier even if they are removed from the nodes.

Finally, we would like to point out the absence of a master process to guide the search. This architecture suggests a better scalability by avoiding idle time due to synchronisation issues or network bottlenecks typically associated with master–slave synchronous relation.

Acknowledgements

This work was supported by the Spanish Ministry of Education and Science under Grant No. TIN2008-06681-C06-06, and the Andalusian Government under Grant Nos. P05-TIC-00531 and P07-TIC-03179.

Appendix A. Detailed results

This section shows the obtained results for the different node configuration as extension of Section 4.3. First column has the dataset name. Second the number of rules. Third and fourth and fifth are training, test and processing time respectively for EDGAR. Columns 5–6 have number of rules, test accuracy and processing time respectively for REGAL. Each dataset was executed following four node configuration: 4, 8, 16 and 32 nodes. For each configuration, the compared algorithms were executed using Dietterich 5x2 cv [4] and 5 different seeds (Table 11).

Table 11
Detailed results.

		Edgar			Regal		
		Rules	Test	Time	Rules	Test	Time
Car							
Mean	4	57	96%	0.09	58	96%	15.8
sd		4.34	0.005	0.01	3.52	0.007	7.4
Mean	8	61	95%	0.05	61	95%	28.6
sd		6.85	0.005	0.00	5.99	0.006	20.4
Mean	16	61	94%	0.08	67	94%	75.8
sd		6.57	0.003	0.01	4.17	0.005	45.5
Mean	32	56	94%	0.22	73	94%	148.6
sd		4.72	0.004	0.08	7.03	0.006	87.6
Mean	All	61	97%	0.10	66	98%	63.4
sd		6.06	0.006	0.07	8.13	0.006	71.6
Cleveland							
Mean	4	69	93%	0.09	54	96%	16.6
sd		7.90	0.054	0.10	4.04	0.053	2.6
Mean	8	65	84%	0.06	55	92%	42.6
sd		22.06	0.116	0.09	4.66	0.007	7.5
Mean	16	70	87%	0.05	62	91%	89.6
sd		7.24	0.029	0.02	4.24	0.010	17.6
Mean	32	78	89%	0.05	68	90%	234.9
sd		7.48	0.026	0.01	6.41	0.013	35.0
Mean	All	73	91%	0.07	62	96%	100.5
sd		14.30	0.070	0.07	8.33	0.016	91.6
Credit							
Mean	4	62	91%	0.08	73	94%	15.2
sd		6.39	0.027	0.03	5.56	0.059	2.7
Mean	8	63	88%	0.04	71	89%	32.5
sd		7.36	0.010	0.01	5.88	0.022	11.5
Mean	16	68	87%	0.03	73	90%	62.3
sd		7.51	0.014	0.01	7.53	0.025	27.8
Mean	32	79	88%	0.04	73	89%	147.4
sd		9.62	0.013	0.03	6.75	0.028	56.3
Mean	All	69	90%	0.05	75	94%	69.4
sd		10.62	0.015	0.03	6.31	0.028	63.3
Ecoli							
Mean	4	52	92%	0.04	41	95%	21.2
sd		4.88	0.053	0.03	5.23	0.056	5.4
Mean	8	57	84%	0.04	41	91%	48.2
sd		6.19	0.035	0.04	4.01	0.016	16.2
Mean	16	55	89%	0.03	43	91%	89.3
sd		7.15	0.016	0.01	4.94	0.019	53.9
Mean	32	70	84%	0.06	53	94%	158.0
sd		6.87	0.029	0.03	6.52	0.019	97.8
Mean	All	61	90%	0.04	47	96%	95.5
sd		9.89	0.036	0.03	7.84	0.022	86.7
Glass							
Mean	4	55	96%	0.11	35	97%	18.1
sd		10.22	0.115	0.09	3.68	0.042	3.4
Mean	8	53	85%	0.04	40	93%	42.0
sd		6.21	0.019	0.02	3.64	0.024	5.9
Mean	16	51	77%	0.04	45	93%	100.8
sd		23.40	0.216	0.02	4.14	0.020	16.2
Mean	32	65	85%	0.05	56	93%	250.9
sd		9.63	0.013	0.03	7.96	0.024	53.0
Mean	All	61	92%	0.07	45	96%	106.2
sd		14.24	0.105	0.03	9.57	0.025	98.1
Haberman							
Mean	4	31	83%	0.04	53	91%	15.6
sd		5.73	0.110	0.01	4.50	0.036	8.7
Mean	8	25	74%	0.03	50	88%	37.3
sd		5.33	0.030	0.01	4.91	0.014	21.6
Mean	16	24	73%	0.03	51	89%	122.9
sd		5.14	0.023	0.04	4.22	0.016	52.4
Mean	32	21	54%	0.05	49	91%	228.6
sd		3.61	0.009	0.09	5.14	0.015	106.9
Mean	All	29	82%	0.04	52	92%	125.4
sd		4.84	0.038	0.03	4.83	0.018	115.5
House Votes							
Mean	4	24	98%	0.04	24	96%	12.5
sd		4.21	0.115	0.01	4.21	0.112	0.6
Mean	8	21	89%	0.02	21	86%	8.0
sd		3.11	0.006	0.00	3.11	0.006	0.3

Table 11 (*Continued*)

		Edgar			Regal		
		Rules	Test	Time	Rules	Test	Time
Mean	16	23	89%	0.01	23	87%	9.6
sd		3.92	0.006	0.00	3.92	0.006	0.4
Mean	32	30	89%	0.01	30	87%	9.3
sd		4.83	0.005	0.00	4.83	0.005	0.7
Mean	All	26	98%	0.02	26	97%	0.7
sd		5.79	0.026	0.01	5.79	0.025	9.5
Hypothyroid							
Mean	4	170	95%	0.52	158	95%	9.6
sd		28.72	0.146	0.08	26.63	0.147	4.0
Mean	8	173	85%	0.40	147	85%	13.6
sd		11.98	0.002	0.07	17.50	0.005	2.8
Mean	16	197	85%	0.99	150	71%	21.9
sd		13.67	0.003	0.06	14.11	0.001	6.4
Mean	32	180	92%	2.57	180	90%	56.5
sd		6.40	0.001	0.36	10.71	0.004	20.5
Mean	All	200	95%	1.00	178	95%	35.6
sd		28.08	0.038	0.94	20.42	0.029	27.1
Iris							
Mean	4	13	97%	0.02	10	98%	11.8
sd		4.06	0.117	0.00	2.32	0.035	5.2
Mean	8	14	88%	0.01	10	95%	30.2
sd		2.77	0.024	0.00	2.34	0.013	11.4
Mean	16	12	90%	0.01	12	96%	66.1
sd		4.87	0.030	0.01	2.74	0.008	29.0
Mean	32	15	99%	0.01	14	97%	28.8
sd		3.47	0.030	0.01	3.23	0.064	12.5
Mean	All	14	96%	0.01	11	99%	36.2
sd		5.55	0.038	0.01	2.55	0.017	9.6
Krvskp							
Mean	4	56	98%	0.15	56	90%	13.3
sd		8.14	0.035	0.07	8.99	0.004	4.8
Mean	8	60	93%	0.07	58	90%	33.9
sd		8.42	0.005	0.05	6.68	0.003	11.3
Mean	16	63	93%	0.08	47	66%	23.0
sd		6.59	0.006	0.03	1.41	0.004	1.6
Mean	32	61	88%	0.07	59	65%	29.0
sd		7.99	0.005	0.03	1.41	0.004	2.7
Mean	All	62	98%	0.10	54	86%	22.5
sd		8.78	0.014	0.06	10.40	0.092	11.1
Monk							
Mean	4	52	99%	0.05	51	82%	5.6
sd		6.40	0.000	0.02	0.00	0.068	0.0
Mean	8	43	88%	0.02	48	70%	39.6
sd		3.90	0.007	0.01	2.73	0.024	27.1
Mean	16	32	88%	0.03	51	61%	43.1
sd		6.42	0.005	0.05	6.59	0.018	19.4
Mean	32	36	88%	0.02	56	64%	88.2
sd		3.12	0.009	0.01	5.93	0.021	48.7
Mean	All	46	99%	0.03	58	77%	61.7
sd		7.25	0.043	0.03	6.51	0.055	41.3
Mushroom							
Mean	4	14	100%	0.24	13	100%	3.2
sd		3.39	0.035	0.07	2.80	0.054	2.4
Mean	8	14	98%	0.07	15	95%	4.1
sd		3.36	0.000	0.01	4.54	0.000	2.2
Mean	16	11	95%	0.07	16	95%	4.1
sd		1.78	0.000	0.02	2.97	0.000	1.0
Mean	32	11	95%	0.08	16	95%	7.4
sd		3.05	0.000	0.03	5.25	0.000	8.8
Mean	All	13	100%	0.14	16	100%	4.9
sd		3.23	0.012	0.08	4.19	0.013	5.0
New-Thyroid							
Mean	4	12	98%	0.72	12	99%	12.3
sd		2.50	0.128	0.12	2.50	0.117	6.7
Mean	8	12	90%	0.66	12	90%	25.0
sd		2.13	0.008	0.56	2.13	0.008	13.8
Mean	16	14	90%	0.80	14	90%	49.5
sd		2.56	0.009	0.62	2.56	0.008	23.4
Mean	32	15	90%	0.58	15	90%	103.3
sd		2.52	0.007	0.73	2.52	0.007	55.9

Table 11 (*Continued*)

		Edgar			Regal		
		Rules	Test	Time	Rules	Test	Time
Mean	All	14	99%	0.81	14	99%	51.9
sd		2.97	0.027	0.42	2.97	0.027	49.7
Nursery							
Mean	4	259	97%	0.80	290	98%	13.7
sd		26.59	0.116	0.54	25.11	0.053	8.0
Mean	8	261	88%	0.75	292	94%	40.8
sd		14.27	0.006	1.23	23.64	0.003	20.9
Mean	16	262	92%	0.98	297	94%	105.3
sd		24.03	0.002	0.09	11.86	0.003	30.9
Mean	32	224	91%	3.56	311	94%	187.5
sd		34.43	0.001	0.34	15.18	0.004	49.3
Mean	All	270	99%	1.63	309	98%	91.0
sd		28.41	0.027	1.50	22.41	0.013	77.2
Pima							
Mean	4	125	94%	0.23	120	90%	12.5
sd		11.27	0.051	0.31	10.49	0.052	6.2
Mean	8	126	90%	0.15	125	83%	27.1
sd		9.27	0.011	0.27	8.74	0.022	12.6
Mean	16	132	90%	0.14	122	82%	51.7
sd		8.06	0.013	0.21	8.43	0.014	23.0
Mean	32	132	90%	0.18	124	89%	159.0
sd		11.53	0.013	0.26	8.22	0.028	57.8
Mean	All	128	94%	0.19	127	94%	65.5
sd		9.34	0.017	0.22	9.49	0.017	68.5
Segment							
Mean	4	132	98%	0.26	124	99%	8.0
sd		13.86	0.034	0.20	12.46	0.054	4.2
Mean	8	130	93%	0.17	117	94%	15.6
sd		13.55	0.006	0.02	11.91	0.004	6.9
Mean	16	131	95%	0.37	128	94%	27.9
sd		11.89	0.003	0.02	13.99	0.005	12.0
Mean	32	119	96%	0.95	160	94%	65.2
sd		11.54	0.003	0.16	12.84	0.005	33.6
Mean	All	132	99%	0.47	137	99%	30.5
sd		13.51	0.012	0.35	22.00	0.013	29.5
Soybean							
Mean	4	66	98%	0.17	72	98%	5.2
sd		7.32	0.116	0.02	8.73	0.118	1.3
Mean	8	65	89%	0.12	66	89%	9.6
sd		7.62	0.003	0.01	3.89	0.015	1.0
Mean	16	67	89%	0.33	78	89%	19.4
sd		4.08	0.003	0.04	7.10	0.017	2.7
Mean	32	71	90%	0.90	87	88%	45.3
sd		6.93	0.003	0.14	9.03	0.022	3.8
Mean	All	72	98%	0.41	81	97%	21.8
sd		6.97	0.026	0.36	11.94	0.031	17.6
Splice							
Mean	4	84	100%	11.90	83	100%	24.4
sd		10.26	0.095	5.81	9.37	0.088	5.5
Mean	8	73	91%	6.52	86	83%	36.5
sd		7.97	0.001	2.74	8.24	0.001	2.5
Mean	16	66	91%	10.22	68	86%	39.7
sd		6.91	0.001	4.28	6.42	0.001	4.0
Mean	32	53	86%	5.99	65	84%	25.6
sd		8.59	0.000	1.19	8.77	0.000	1.2
Mean	All	77	100%	9.71	72	95%	29.1
sd		10.91	0.027	4.53	12.41	0.026	4.5
Tic-tac-toe							
Mean	4	70	99%	0.12	63	97%	5.2
sd		10.38	0.119	0.03	11.28	0.113	2.8
Mean	8	74	90%	0.03	39	85%	42.2
sd		6.83	0.006	0.00	13.25	0.014	17.8
Mean	16	83	90%	0.04	44	82%	69.4
sd		4.68	0.007	0.01	11.31	0.011	56.9
Mean	32	95	91%	0.03	40	77%	106.8
sd		7.97	0.002	0.01	7.09	0.021	24.1
Mean	All	87	99%	0.06	50	91%	61.4
sd		14.45	0.028	0.04	13.12	0.055	51.8

Table 11 (*Continued*)

		Edgar			Regal		
		Rules	Test	Time	Rules	Test	Time
Vehicle							
Mean	4	190	92%	1.23	143	95%	6.9
sd		13.06	0.049	2.40	17.44	0.115	1.5
Mean	8	171	89%	0.46	139	86%	13.3
sd		10.66	0.010	0.92	8.89	0.013	3.2
Mean	16	162	90%	0.23	150	86%	23.1
sd		7.32	0.012	0.19	10.96	0.014	5.2
Mean	32	147	63%	0.13	168	86%	101.0
sd		5.00	0.018	0.02	10.98	0.011	36.1
Mean	All	181	95%	0.64	161	95%	39.5
sd		14.11	0.019	1.42	18.32	0.028	46.2
Vote							
Mean	4	18	97%	0.03	13	66%	12.8
sd		3.54	0.115	0.00	1.00	0.000	14.9
Mean	8	17	88%	0.01	13	66%	16.6
sd		4.53	0.006	0.00	2.83	0.007	0.4
Mean	16	19	89%	0.01	13	66%	12.6
sd		3.48	0.006	0.00	1.00	0.003	13.9
Mean	32	29	93%	0.02	14	67%	17.1
sd		5.56	0.008	0.01	1.41	0.004	0.1
Mean	All	24	97%	0.02	8	53%	11.8
sd		7.13	0.021	0.01	9.36	0.442	11.7
Waveform							
Mean	4	1064	93%	9.74	1032	92%	9.7
sd		96.97	0.075	4.58	96.97	0.074	4.6
Mean	8	530	47%	19.93	1040	93%	9.8
sd		54.27	0.075	4.77	97.84	0.073	4.6
Mean	16	880	88%	18.41	843	46%	19.9
sd		45.21	0.005	6.03	121.09	0.059	4.7
Mean	32	894	85%	17.52	1069	86%	18.4
sd		35.87	0.004	5.98	45.21	0.005	6.0
Mean	All	1026	95%	15.15	1070	93%	15.1
sd		77.79	0.044	8.81	77.79	0.044	8.8
Wine							
Mean	4	51	97%	0.04	60	94%	1.1
sd		9.97	0.055	0.02	11.30	0.054	0.5
Mean	8	52	88%	0.02	53	86%	0.7
sd		5.99	0.013	0.00	5.99	0.013	0.1
Mean	16	65	89%	0.02	63	88%	1.1
sd		15.12	0.016	0.01	14.14	0.016	0.3
Mean	32	68	89%	0.03	64	88%	1.2
sd		14.55	0.017	0.01	17.20	0.019	0.3
Mean	All	57	97%	0.03	60	97%	1.2
sd		13.51	0.031	0.01	14.52	0.031	0.3
Wisconsin							
Mean	4	24	98%	0.03	23	99%	4.9
sd		4.74	0.115	0.00	4.40	0.117	2.8
Mean	8	22	87%	0.01	22	90%	7.0
sd		3.93	0.007	0.00	2.56	0.004	4.4
Mean	16	15	48%	0.05	24	91%	16.0
sd		3.93	0.027	0.03	4.37	0.004	10.1
Mean	32	18	87%	0.01	25	91%	25.8
sd		4.29	0.047	0.00	2.65	0.004	8.2
Mean	All	25	98%	0.02	25	100%	14.7
sd		4.10	0.056	0.01	3.63	0.026	5.4
Zoo							
Mean	4	10	99%	0.04	9	98%	10.2
sd		2.21	0.125	0.01	1.50	0.120	3.3
Mean	8	9	90%	0.02	8	89%	15.6
sd		2.52	0.017	0.01	1.49	0.015	3.8
Mean	16	8	90%	0.01	9	82%	28.8
sd		4.80	0.016	0.00	1.70	0.034	3.7
Mean	32	12	66%	0.01	13	90%	32.9
sd		3.63	0.314	0.00	4.59	0.024	4.6
Mean	All	9	69%	0.02	6	65%	9.1
sd		5.04	0.314	0.01	4.10	0.311	14.5

References

[1] C. Anglano, M. Botta, NOW G-Net: learning classification programs on networks of workstations, IEEE Transactions on Evolutionary Computation (October) (2002) 463–480.
[2] C. Schaffer, When does over fitting decrease prediction accuracy in induced decision trees and rule sets? in: Proceedings of the European Working Session on Learning (EWSL-91), 1991, pp. 192–205.
[3] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, New York, 1989.
[4] T.G. Dietterich, "Approximate statistical tests for comparing supervised classification", learning algorithms Neural Computation 10 (7) (1999) 1895–1924.
[5] D.J. Sheskin, Handbook of Parametric and Nonparametric Statistical Procedures, CRC Press, Boca Raton, FL, 2003.
[6] K.A. De Jong, W.M. Spears, D.F. Gordon, Using genetic algorithms for concept learning, Machine Learning (1993) 161–188.
[7] J. Demšar, Statistical comparisons of classifiers over multiple data sets, Journal of Machine Learning Research 7 (2006) 1–30.
[8] E. Cantu-Paz, Efficient and Accurate Parallel Genetic Algorithms, Kluwer Academic Publishers, 2000.
[9] I.W. Flockhart, N.J. Radcliffe, GA-MINER: parallel data mining with hierarchical genetic algorithms—final report, EPCC-AIKMS-GA-MINER Report1.0, University of Edinburgh, UK, 1995.
[10] A. Giordana, F. Neri, Search-intensive concept induction, Evolutionary Computation (1995) 375–416.
[11] A.A. Freitas, S.H. Lavington, Mining Very Large Databases with Parallel Processing, Kluwer Academic Publishers, 1998.
[12] G.M. Weiss, Learning with rare cases and small disjuncts, in: Proceedings of the 12th International Conference on Machine Learning (ML-95), 1995, pp. 558–565.
[13] D.P. Greene, S.F. Smith, Competition-based induction of decision models from examples, Machine Learning (1993) 229–257.
[14] J.H. Holland, J.S. Reitman, in: D.A. Waterman, F. Hayes-Roth (Eds.), Cognition Systems Based on Adaptive Algorithms, in Pattern-Directed Inference Systems, Academic Press, New York, 1978.
[15] J.H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, 1975.
[16] S.T. Leutenegger, X. Sun, Limitations of cycle stealing for parallel processing on a network of homogeneous workstations, Journal of Parallel and Distributed Computing 43 (1997) 169–178.
[17] M.J. Quinn, Parallel Computing: Theory and Practice, McGraw-Hill, 1994.
[18] Y. Nojima, I. Kuwajima, H. Ishibuchi, Data set subdivision for parallel distribution implementation of genetic fuzzy rule selection, in: IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'07), London, 2007, pp. 2006–2011.
[19] J.R. Quinlan, C4.5, in: Programs for Machine Learning, Morgan Kaufmann Publishers, 1993.
[20] F. Provost, D. Hennessy, Distributed machine learning: scaling up with coarse-grained parallelism, in: Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology (ISMB-94), Stanford, 1994.
[21] S. García, F. Herrera, An extension on 'Statistical comparisons of classifiers over multiple data sets' for all pairwise comparisons, Journal of Machine Learning Research 9 (2008) 2677–2694.
[22] S. García, A. Fernández, J. Luengo, F. Herrera, A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability, Soft Computing- A Fusion of Foundations, Methodologies and Applications 13 (10) (2009) 959–977.
[23] S. Smith, A learning system based on genetic algorithms, PhD Thesis, University of Pittsburgh, Pittsburgh, 1980.
[24] G.G. Towell, J.W. Shavlik, Knowledge-based artificial neural networks, Artificial Intelligence 70 (1994) 119–165, Kluwer Academic Publishers.
[25] UCI, C.J. Merz, P.M. Murphy, UCI Repository of Machine Learning Databases, University of California Irvine, Department of Information and Computer Science, 1996, http://kdd.ics.uci.edu.
[26] G. Venturini, SIA: a supervised inductive algorithm with genetic search for learning attribute based concepts, in: Proceedings of European Conference on Machine Learning, Vienna, 1993, pp. 280–296.
[27] G.M. Weiss, H. Hirsh, A quantitative study of small disjuncts, in: Proceedings of the Seventeenth National Conference on Artificial Intelligence, AAAI Press, Menlo Park, CA, 2000, pp. 665–670.
[28] J.H. Zar, Biostatistical Analysis, Prentice Hall, 1999.