Evol. Intel. (2008) 1:209–232
DOI 10.1007/s12065-008-0013-9

REVIEW ARTICLE

Genetic-based machine learning systems are competitive for pattern recognition

Albert Orriols-Puig · Jorge Casillas · Ester Bernadó-Mansilla

Received: 13 February 2008 / Revised: 3 June 2008 / Accepted: 2 July 2008 / Published online: 25 July 2008

Abstract During the last decade, research on *Genetic-Based Machine Learning* has resulted in several proposals of supervised learning methodologies that use *evolutionary algorithms* to evolve rule-based classification models. Usually, these new GBML approaches are accompanied by little experimentation and there is a lack of comparisons among different proposals. Besides, the competitiveness of GBML systems with respect to non-evolutionary, highly-used machine learning techniques has only been partially studied. This paper reviews the state of the art in GBML, selects some of the best representatives of different families, and compares the accuracy and the interpretability of their models. The paper also analyzes the behavior of the GBML approaches with respect to some of the most influential machine learning techniques that belong to different learning paradigms such as decision trees, support vector machines, instance-based classifiers, and probabilistic classifiers. The experimental observations emphasize the suitability of GBML systems for performing classification tasks. Moreover, the analysis points out the strengths of the different systems, which can be used as recommendation guidelines on which systems should be employed depending on whether the user prefers to maximize the accuracy or the interpretability of the models.

Keywords Pattern recognition · Supervised learning · Fuzzy logics · Genetic-based machine learning · Learning classifier systems · Learning fuzzy-classifier systems

1 Introduction

Pattern recognition [83] is concerned with the design of algorithms that are able to extract novel, useful, and hidden patterns from repositories of data. In this context, a competent supervised learning technique is required to be able to (i) identify patterns hidden between a set of descriptive attributes and a dependent variable, i.e., the output or the class, (ii) represent these patterns in some legible structure, and (iii) generalize over the input patterns to produce compact representations. During the last decades, several approaches have been designed to totally or partially fulfill the aforementioned requirements such as decision trees [73], support vector machines [84], instance-based algorithms [4], and probabilistic classifiers [57].

Recently, Genetic-Based Machine Learning (GBML) [51] has appeared as an appealing alternative to traditional learning systems for pattern recognition tasks. GBML systems are machine learning techniques that use *Evolutionary Algorithms* (EAs) [31, 43, 50, 63] to search efficiently over complex search spaces. Initially focused on the simulation of animal behavior, research on GBML has been historically conducted from two perspectives: Michigan-style GBML [51, 52] and Pittsburgh-style GBML [22, 80]. The increased understanding of how EAs work [44] has widened the application of EAs as the heart of different

A. Orriols-Puig (✉) · E. Bernadó-Mansilla
Grup de Recerca en Sistemes Intel·ligents,
Enginyeria i Arquitectura La Salle, Universitat Ramon Llull,
Quatre Camins 2, 08022 Barcelona, Spain
e-mail: aorriols@salle.url.edu

E. Bernadó-Mansilla
e-mail: esterb@salle.url.edu

J. Casillas
Department of Computer Science and Artificial Intelligence,
University of Granada, 18071 Granada, Spain
e-mail: casillas@decsai.ugr.es

types of learning algorithms. Some of these approaches lay between the definitions of Pittsburgh- and Michigan-style GBML, such as the application of EAs mixed with the *Iterative Rule Learning* approach (IRL) [85] and the *Genetic Cooperative-Competitive Learning* approach (GCCL) [41, 47]. Other methodologies propose to include EAs as robust search mechanisms to assist the building of statistical classifiers [25, 71].

There have also been some advances in the rule representation used by GBML systems. First GBML systems used a binary or ternary rule representation [80, 87]. Later on, several new representations were introduced to tackle real-world problems with different types of attributes. Two of the most prominent rule representations are the interval-based representation and the fuzzy representation. In an interval-based representation, each rule variable takes an interval of values to which the rule is applicable [19, 81, 89]. In a fuzzy rule representation, each variable takes a fuzzy set or a disjunction of fuzzy sets. The application of GBML to evolve fuzzy rules has lead to the so-called *Genetic Fuzzy Rule-Based Systems* (GFRBSs) [20], which have received special attention during the last decades since they provide highly legible models (in which the inference process is similar to human reasoning) and they are naturally adapted to deal with uncertainty. Research on GFRBSs has mainly focused on the use of EAs to learn/tune different components of a fuzzy-rule based system, such as the fuzzy sets or the fuzzy rules. The reader is referred to [48] for an update, sound review on GFRBSs.

The aim of this work is to review some of the most relevant GBML approaches for pattern recognition and to compare them with some of the most influential non-evolutionary supervised learning techniques. We include representatives of the different GBML families for both non-fuzzy (interval-based) and fuzzy rule representations. More specifically, we consider the following non-fuzzy GBML systems: (1) UCS [10], a Michigan-style GBML method derived from XCS [87] but specialized for supervised learning; (2) GAssist [8], one of the best representatives of Pittsburgh-style GBML systems; and (3) HIDER [2, 3], a hierarchical GBML that follows the iterative rule learning approach. We also include the following GFRBSs in our review: (4) HMOF, a Pittsburgh-style fuzzy GFRBSs that includes some ideas of Michigan-style GBML [54]; (5) SLAVE [18, 45], a fuzzy iterative rule learning approach; and (6) Fuzzy LogitBoost [71], a statistical classifier that introduces a fuzzy representation to LogitBoost [36]. We describe the six GBML systems and compare the accuracy and size of the models that they evolve, highlighting the differences among them.

Later on, we compare the six GBML systems with some of the most influential pattern recognition methods [94], including (1) the decision tree C4.5 [73], (2) the instance-based algorithm IBk [4], (3) the probabilistic classifier Naïve Bayes [57], (4) the rule-induction method PART [33], and (5) the support vector machine SMO [72]. In both analyses, the algorithms are compared on a collection of twenty real-world datasets extracted from the UCI repository [7] and local repositories. Along with this paper, the partitioned data sets used in the present analysis are made available at the authors[1] with the aim of letting researchers compare the results obtained with their own learning systems. The experimental results are statistically compared following the state of the art in multi-comparison tests [26].

The remainder of this paper is organized as follows. Section 2 overviews the different families of GBML systems. Section 3 briefly describes each one of the six GBML selected for the comparison. Section 4 provides details about the experimental methodology. Section 5 presents the results and performs the statistical analysis. Finally, Sect. 6 summarizes and concludes the work.

2 Genetic-based machine learning

Since Holland presented the first schemes of GBML systems [51], originally addressed as *Learning Classifier Systems* (LCS), the research on GBML has been conducted from two perspectives: the *Pittsburgh* approach [80] and the *Michigan* approach [52]. Recently, a third methodology has received an increasing amount of attention: the *Incremental Rule Learning* approach [85]. These three families are briefly introduced as follows.

Pittsburgh-style GBML systems resulted of directly extending *genetic algorithms* (GAs) [50] to supervised learning problems. This approach represents an individual as a rule set. The system maintains a population of candidate rule sets whose quality is evaluated with a fitness function that considers different aspects such as the prediction accuracy and the generality of the rule sets. The population is evolved by means of the typical genetic operators, i.e., selection, crossover, and mutation, which are adapted to deal with rule sets. At the end of the learning process, the best individual found during the evolutionary process is used to predict the class of unknown examples. The first successful developments of Pittsburgh-style GBML for supervised learning are GABIL [23] and GIL [6]. A new generation Pittsburgh-style GBML derived from GABIL can be found in GAssist [8]. One of the most recent fuzzy systems in this family is the Pittsburgh-style method with Michigan-style inspired local search presented by Ishibuchi and Nojima [54].

Michigan-style GBML methods, initially defined as *cognitive systems* [52], combine a credit-apportionment

[1] http://www.salle.url.edu/~aorriols/DataSets.tgz.

system with EAs to evolve populations of accurate rules. Each individual codifies a single rule; therefore, the whole population collaborates to predict new input examples. The rules are evaluated on-line by the credit-apportionment system. A steady-state genetic algorithm is periodically applied on the population (or on subpopulations) to discover new promising rules, which replace other low-fit rules. Some of the first developments of Michigan-style GBML are SCS [43] and NewBoole [13]. Although these systems were able to solve certain classification tasks, several drawbacks, mainly associated with the achievement of accurate generalizations, hindered their success. This led to further research that culminated in the design of XCS [87], by far the most influential Michigan-style GBML system. XCS was designed for reinforcement learning, although it can be used for pattern recognition by considering that a classification problem is a reinforcement problem in which maximum rewards are given to correct classifications and low rewards correspond to incorrect classifications. On the other hand, classification tasks can be solved in a more straightforward way using UCS [10], a system which inherits the main components from XCS but specializes them to supervised learning.

Iterative Rule Learning GBML systems inherit characteristics from both Pittsburgh- and Michigan-style GBMLs. It uses a separate-and-conquer methodology to create an ordered list of individuals [85]. Each individual is represented by a single rule, as in the Michigan approach. The system iteratively invokes an EA, which evaluates the individuals according to their accuracy and generality. The best individual returned by the EA is added to the end of a list of rules and all the matching examples are removed from the training data set. This process is repeated until the training data set is empty. In test mode, the predicted class of a new example is given by the first rule in the decision list that matches the example. One of the most recent representatives of genetic-based IRL systems is HIDER [2, 3]. For fuzzy representation, SLAVE [18, 45] is one of the most appealing proposals of IRL GFRBS.

In addition of these three approaches, in the last few years, EAs have been included as robust search systems in different machine learning fields, broadening the meaning of *GBML*. For example, EAs have been applied in the field of statistical learning to discover new promising rules in different boosting algorithms such as AdaBoost [35] and LogitBoost [36].

3 Description of the GBML systems used in the comparison

This section briefly describes the six highly-competent GBML systems included in the comparison: (1) UCS [10, 70], (2) GAssist [8], (3) HIDER [2, 3], (4) HMOF [54, 56], (5) SLAVE [46], and (6) Fuzzy LogitBoost [71]. For each system, we explain the type of knowledge representation, the inference process followed to predict the class of previously unseen instances, and the learning process used to evolve the rule set. Table 1 summarizes the main characteristics of each GBML method. For more details, the reader is referred to the original papers of these methods.

3.1 UCS

UCS [10] is an on-line, model-free Learning Classifier System which inherits the main components of XCS [87,

Table 1 Summary of the main characteristics of the GBML methods

	Rule type and knowledge rep.	Inference method	Type of GBML
UCS	Interval-based rules with fitness Population of independent rules	Voting policy according to rule's fitness	On-line Michigan-style GBML
GAssist	Interval-based rules (discretization) Decision list with an explicit default rule	First matching rule in the decision list	Pittsburgh-style GBML
HIDER	Interval-based rules (natural coding) Decision list with an implicit default rule	First matching rule in the decision list	IRL GBML
HMOF	Fuzzy rule with hierarchical fuzzy semantics and a weight Set of independent classifiers	Winner rule inference considering fitness	Pittsburgh-style GFRBS with local search based on EAs
SLAVE	Fuzzy rule with number of linguistic terms prefixed and no weights Set of independent classifiers	Winner rule inference	IRL GFRBS
LogitBoost	Fuzzy rule with number of linguistic terms prefixed and a weight per class Set of independent classifiers (prefixed number of classifiers)	Voting policy according to matching degree and weight per class	Statistical learning

88], but specializes them for supervised learning tasks. The competitiveness of the system has been clearly demonstrated, specially in imbalanced domains [69]. In the following, we describe the knowledge representation, the inference methodology, and the learning process.

3.1.1 Knowledge representation

UCS evolves a population of classifiers which together cover the input space. Each classifier consists of a production rule of the form *condition* →*class* and a set of parameters. The condition specifies the set of inputs where the classifier can be applied.

For continuous inputs, the condition is codified as a set of intervals $[l_i, u_i]^n$, which globally represents a hyper rectangle in the feature space. The class c^k of the rule specifies the class predicted when the condition is satisfied:

$$\textbf{if } x_1 \in [l_1, u_1] \wedge \ldots \wedge x_n \in [l_n, u_n] \quad \textbf{then } c^k. \tag{1}$$

Each rule has the following parameters: (a) accuracy *acc*; (b) fitness *F*; (c) correct set size *cs*; (d) numerosity *num*; and (e) experience *exp*. Accuracy and fitness are measures of the quality of the classifier. The correct set size is the estimated average size of all the correct sets where the classifier has belonged to. Numerosity is the number of copies of the classifier and experience is the number of times that a classifier has been evaluated. In [11], the limitations of this type of representation with respect to certain indicators of problem complexity is carefully analyzed.

3.1.2 Class inference

To classify a test instance, all matching classifiers emit a vote proportional to their accuracy and fitness, i.e.,

$$\forall_{c^i} \mathrm{vote}_{c^i} = \sum_{k | c^k = c^i} F^k \times acc^k. \tag{2}$$

The most voted class is selected as output.

3.1.3 Learning process

During training, UCS incrementally evolves a set of classifiers. At each learning iteration, the system receives an input example *e* and its class *c*. Then, the system creates the match set [M], which contains all the classifiers in the population [P] whose condition matches *e*. From that, the correct set [C] is formed, which consists of all the classifiers in [M] that predict class *c*. If [C] is empty, the covering operator is activated, creating a new classifier with a generalized condition matching *e*, and predicting class *c*.

Next, the parameters of all the classifiers in [M] are updated. The experience of each classifier is increased, and its accuracy is updated depending on whether the given prediction was correct. The correct set size *cs* is calculated if the classifier belongs to [C]. Then, the fitness is shared among all the classifiers that participate in [C].

After one learning cycle, a genetic algorithm (GA) is triggered if the average time since the last application of the GA on the classifiers in [C] is greater than θ_{GA}. In this case, the GA selects two parents from [C] with a probability that depends on the classifier's fitness. The two parents are copied, creating two new children, which are recombined and mutated with probabilities χ and μ, respectively. Recombination crosses the parent's conditions by two points. Mutation modifies the lower and upper bound of an interval according to a uniform distribution. Finally, each offspring is introduced into the population, removing another classifier if the population is full.

3.2 GAssist

GAssist [8] is one of the most competitive current Pittsburgh-style GBML systems. GAssist was initially derived from GABIL [23], introducing several modifications that enabled the system to overcome scalability problems detected in the first Pittsburgh-style GBML approaches [34]. The rule representation, the inference methodology, and the learning process are described as follows.

3.2.1 Knowledge representation

GAssist evolves a set of individuals, each of them represented by a rule set of variable length, where each rule consists of a condition and a predicted class c^k:

$$\begin{aligned} &\textbf{IF } (x_1 = V_1^1 \vee \ldots \vee x_1 = V_m^1) \wedge \ldots \\ &\wedge (x_n = V_1^n \wedge \ldots \wedge x_n = V_m^n) \quad \text{THEN } c^k. \end{aligned} \tag{3}$$

That is, each input variable x_i is represented by a disjunction of feasible values for this variable. For nominal variables, $(V_1^i, \ldots, V_j^i)$ are the *j* possible values that the variable can take. For continuous variables, GAssist applies a discretization technique to transform the input space into intervals of values. Several discretization techniques have been proposed for GAssist. In our experiments, we used a uniform discretization.

3.2.2 Class inference

The best evolved individual is considered to classify new input instances. This individual is treated as a decision list [75]. Therefore, the class of the first matching rule of the individual is selected as output.

3.2.3 Learning process

The core of the system is a near-standard generational genetic algorithm similar to the one applied in GABIL [23]. The offspring are evaluated by means of a fitness function based on the *minimum description length* principle (MDL) [74]. GAssist uses the same crossover operator defined by GABIL, i.e., a *semantically correct crossover* operator [23]. This is a multiple-point crossover operator that forces that the selected points cut both parents in the same position of the variable. The mutation operator randomly adds or removes one value of a given variable.

GAssist introduces a new *deletion operator* that permits to remove rules from individuals, and so, to control their size. This operator is activated after a predefined number of iterations and it removes the rules of an individual that do not match any input example. To avoid an excessive loss of diversity, this operator is not applied if the individual does not have a minimum number of rules.

Finally, GAssist controls the runtime of the system by means of a windowing scheme addressed as *Incremental Learning with Alternating Strata* (ILAS). This mechanism splits the training data set into several non-overlapping subsets of examples, and selects a different subset at each GA iteration. Thus, ILAS permits to reduce the training time of a single GA iteration since fewer examples need to be matched with the new individuals in the evaluation process. Moreover, in [8], it was empirically shown that this technique allows for a better generalization.

3.3 HIDER

HIDER [3] is an iterative rule learning approach that evolves a set of rules which is made available as a decision list [75]. HIDER combines ideas of both Michigan- and Pittsburgh-style GBMLs with the aim of evolving rule sets that are similar to Pittsburgh-style GBMLs, but reducing the search space of possible solutions. Moreover, HIDER incorporates a brand new rule coding addressed as *natural coding* [2]. As follows, we provide more details about the knowledge representation, the inference methodology, and the learning process.

3.3.1 Knowledge representation

Similarly to GAssist, HIDER represents the knowledge as a set of rules which takes the form of a decision list. The main difference between GAssist and HIDER is that the latter one uses the so-called *natural coding* to represent each rule [2]. That is to say, each rule is encoded as

$$\textbf{IF } x_1 = L_1 \wedge \ldots \wedge x_n = L_n \quad \textbf{THEN } c^k, \tag{4}$$

where L_i is a label that is used to map the genotype to the phenotype of this attribute. This type of representation permits a one-to-one mapping, i.e., each label L_i identifies one phenotype and each phenotype is identified by a single label. The possible phenotypes are obtained differently for categorical and continuous attributes. For categorical attributes, a different label L_i is assigned for each possible combination of categorical values. Therefore 2^ℓ possible combinations are considered, where ℓ is the number of categorical values for the given attribute. For continuous attributes, a discretization technique is employed to reduce the cardinality of the alphabet. The discretization technique returns a set of cut-points. Then, a label L_i is assigned to each possible combination of two cut-points, which determine an interval of possible values for the given attribute. In our experiments we used the *Unparameterized Supervised Discretization* algorithm (USD) [42] to obtain the cut points.

3.3.2 Learning process

The learning process follows a *sequential covering method* [40], that is, a separate-and-conquer strategy that progressively discovers rules that explain part of the training instances. The process is detailed as follows. The system initializes an empty rule set. Then, at each learning iteration, HIDER invokes a genetic algorithm which supplies a new rule. This rule is included at the end of the hierarchical rule set and the examples covered by this rule are removed from the data set. This process is repeated until the data set is empty.

The system employs a generational genetic algorithm to evolve single rules, in which new crossover and mutation operators have been designed to effectively deal with the natural coding. For categorical attributes, the crossover operator generates offspring whose crossed attributes are obtained from the intersection of a mutation of the parents values for this attribute. The mutation operator creates new individuals that are phenotypically close to the parents. For continuous attributes, similar ideas are followed. The crossover operator produces offspring whose supports are inherited from the parent's supports. The mutation operator selects one of the closest discretized intervals to the parent's intervals.

3.3.3 Class inference

Given a new unseen instance, the hierarchical rule set is searched in order. The class of the first matching rule is returned as output.

3.4 HMOF

In [55], a Pittsburgh-style GFRBS that introduces a local search operator inspired by Michigan-style GBML was presented. The system evolved a set of linguistic fuzzy rules whose fitness was calculated as the classification accuracy of each rule. Later on, Ishibuchi and Nojima moved the system to a multi-objective architecture, and showed the many benefits that the evolutionary multi-objective learning method provided with respect to its former proposal [54] . We used the latter approach in our experiments, which we address as *Hybrid Multi-Objective Fuzzy* system (HMOF). In the following, we provide more details about the knowledge representation, the learning process, and the inference methodology.

3.4.1 Knowledge representation

HMOF evolves a population of individuals, as done in Pittsburgh-style GBMLs. Each individual consists of a set of fuzzy rules, which take the following form

$$\textbf{IF } x_1 \text{ is } A_1^k \quad \text{and}\ldots\text{and } x_n \text{ is } A_n^k \quad \textbf{THEN } c^k \quad \textbf{WITH } w^k, \tag{5}$$

where each input variable x_i is represented by a *linguistic term* or *label*. The system defines 14 possible linguistic terms for each attribute, which correspond to Ruspini's strong fuzzy partitions with two, three, four, and five uniformly distributed triangular-shaped membership functions. Moreover, the system also uses "*don't care*" as an additional linguistic term, which indicates that the variable matches any input value with maximum matching degree.

The *matching degree* $\mu_{A^k}(e)$ of an example e with a single fuzzy rule k is computed as follows. For each variable x_i, we compute the membership degree of e_i with the corresponding linguistic term of the variable. Then, the matching degree of the rule with the example e is determined by the T-norm (conjunction) of the matching degree of each input variable. In our experiments, we used the product as T-norm.

To determine the class predicted by the rule c^k and the weight w^k, the system first calculates the confidence with which the fuzzy rule k predicts each class h, i.e.,

$$\text{confidence}(k|h) = \sum_{\forall e|\text{class}(e)=h} \mu_{A^k(e)} \Big/ \sum_{\forall e} \mu_{A^k(e)}. \tag{6}$$

That is, the confidence in the class h depends on the ratio of the sum of the matching degrees with the instances of class h to the sum of the matching degrees with all the instances of the data set. The class of the rule c^k is the class with maximum confidence. Finally, w^k is computed as the maximum confidence value minus the sum of the remaining confidence values.

The learning is guided by three objectives, which are encoded with each individual: (1) the number of training patterns correctly classified by the rule set, (2) the number of fuzzy rules that the individual contains, and (3) the total number of antecedent conditions of the fuzzy rules. The multi-objective approach aims at maximizing the first and third objectives and minimizing the second objective.

3.4.2 Class inference

Given a new input pattern e and the rule set S, the rule k that maximizes the product of the matching degree with e and the weight w^k is selected. The system returns the class c^k predicted by the winner rule k.

3.4.3 Learning process

The learning process of the system follows closely the proposal of NSGA-II [24]. That is to say, at the beginning of the run, the system initializes a population from a set of randomly selected training examples. The new examples are evaluated according to the Pareto ranking and the crowding measure of NSGA-II. Then, at each cycle, the system repeats *Npop* times the following three steps:

- Select a pair of parents using binary tournament selection.
- Apply crossover and mutation, with probabilities P_{chi} and P_{μ}, respectively, to generate an offspring. The crossover operator randomly selects S_1 and S_2 rules from each parent to create the offspring, where S_1 and S_2 are random numbers ranging from 1 to the length of each parent. Mutation randomly changes a variable of the antecedent of the fuzzy rule.
- Apply a single iteration of a Michigan-style GBML algorithm to the offspring rule set with probability P_M.

Then, both the offspring and the parent population are combined in a single one and the best N_{pop} individuals from the merged population are selected to build the next population. Again, the new individuals are evaluated according to the Pareto ranking and the crowding distance of NSGA-II. This procedure is repeated for a certain number of iterations.

The main novelty of this approach with respect to Pittsburgh-style GBML systems is that it performs a local search iteration after crossover and mutation. In this cycle, the system follows the next steps on the selected individual:

- Compute the fitness of each rule as the number of examples correctly classified by the rule.
- Apply genetic operators to generate N_r fuzzy rules.

- Replace the worst N_r rules of the individual with the new generated offspring.

When the stop criterion is met, the system returns all the non-dominated rule sets.

3.5 SLAVE

SLAVE [18, 45] is an inductive learning algorithm based on a fuzzy-rule representation. SLAVE follows an iterative rule learning scheme with the aim of reducing the large search spaces of Pittsburgh-style GBML systems. As follows, the knowledge representation, the inference process, and the learning process are explained.

3.5.1 *Knowledge representation*

SLAVE creates a set of individuals that consist of a single rule. The condition of each rule is represented in *conjunctive normal form*

$$\textbf{IF } x_1 \text{ is } \widetilde{A_1^k} \quad \text{and...and } x_n \text{ is } \widetilde{A_n^k} \quad \textbf{THEN } c^k,$$

where each input variable x_i is represented by a disjunction of linguistic terms $\widetilde{A_i^k} = \{A_{i1} \text{ or}, \ldots, \text{or } A_{in_i}\}$, and the consequent c^k is the class predicted by the rule. In our experiments, all the variables share the same semantics, which are defined by means of triangular-shaped fuzzy membership functions with five linguistic terms. Notice the differences between these rules and the ones evolved by HMOF. In SLAVE, each variable is represented by an arbitrary number of linguistic terms (in our experiments, a maximum of five linguistic terms). Therefore, rules can be generalized by including several linguistic terms per variable. SLAVE simulates the absence of a variable by letting it take all the possible linguistic terms. On the other hand, variables of HMOF are represented by a single linguistic term. Nonetheless, note that, as different fuzzy partitions are defined, the rules can be easily generalized by taking linguistic terms that cover larger regions of the feature space or by taking the "*don't care*".

The *matching degree* $\mu_{A^k}(e)$ of an example e with a rule k is computed as follows. For each variable x_i, we compute the membership degree for each of its linguistic terms, and aggregate them by means of a T-conorm (disjunction). Then, the matching degree of the rule is determined by the T-norm (conjunction) of the matching degree of all the input variables. In our experiments, we used the maximum and the product operators as T-conorm and T-norm, respectively.

3.5.2 *Inference process*

The class of a new example is determined by the rule that maximizes the matching degree with this example. In case of having more than one rule with maximum matching degree, the class of the first matching rule is selected as the output.

3.5.3 *Learning process*

Figure 1 illustrates the learning scheme of SLAVE, which is based on two steps: i) learn one rule from the data set, and ii) penalize the data covered by the rule. Given a data set E and a specific class B, the system searches for the rule that describes more accurately this class. This process is performed by a steady-state genetic algorithm. The fitness function of the GA is determined by the training error and the generality of the rule. Then, this rule is aggregated to the fuzzy-rule set. If more rules are required to represent all the examples of class B, the examples covered by the current rules of class B are removed, and the GA is run again providing a new rule for class B. The same procedure is repeated until no more rules are required for class B. Then, the same algorithm is followed to learn rules for the other classes of the domain, resulting in a rule set that covers all the instances in the training data set.

3.6 Fuzzy LogitBoost

The LogitBoost algorithm [36] is a boosting method similar to AdaBoost [35] that uses a greedy version of generalized backfitting [36] to build an additive model. It

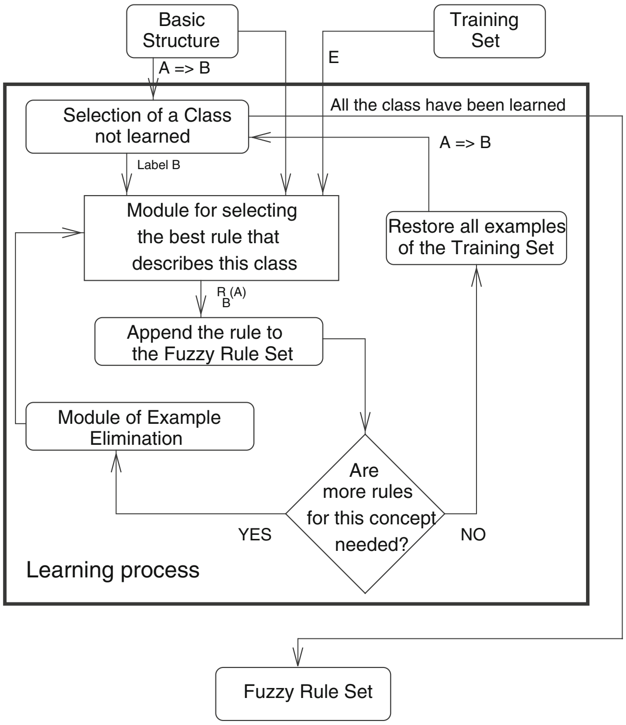
Basic Structure
Training Set
A => B
E
Selection of a Class not learned
All the class have been learned
A => B
Label B
Module for selecting the best rule that describes this class
Restore all examples of the Training Set
R_B(A)
Append the rule to the Fuzzy Rule Set
Module of Example Elimination
Are more rules for this concept needed?
YES
NO
Learning process
Fuzzy Rule Set

Fig. 1 Illustrative scheme of the learning process of SLAVE

has been experimentally shown that LogitBoost outperforms AdaBoost, especially in multi-class problems. Due to these improvements, LogitBoost was extended to induce fuzzy classifiers, resulting in the so-called Fuzzy LogitBoost algorithm [71]. In the following, we describe the knowledge representation, the inference methodology, and the learning process of Fuzzy LogitBoost.

3.6.1 Knowledge representation

Fuzzy LogitBoost creates a set of *weak classifiers*, which take the following form

$$\begin{aligned}\textbf{IF } x_1 \text{ is } A_1^k \quad \text{and}\ldots\text{and } x_n \text{ is } A_n^k \quad \textbf{THEN}\\ c_1^k \textbf{ WITH } s_1^k, \ldots, c_p^k \quad \textbf{WITH } s_p^k.\end{aligned} \tag{7}$$

Each input variable x_i is represented by a linguistic term A_i^k. All variables share the same semantics, represented by triangular-shaped membership functions. The method permits the absence of a variable by not assigning any linguistic term to this variable. In the consequent, the rule maintains one weight s_j^k for each class j that indicates the soundness with witch the weak classifier predicts class j.

3.6.2 Class inference

Given a new unknown example e, each rule k votes for each class according to the weights per class. That is, the vote for the class c is computed as

$$\text{vote}_c = \sum_{\forall k} s^k \times \mu_{A^k}(e). \tag{8}$$

The most voted class is returned as output.

3.6.3 Learning process

Fuzzy LogitBoost iteratively invokes an algorithm that provides a *low quality classifier*, addressed as *weak hypothesis* in the boosting literature. This weak hypothesis is added to a *compound classifier*. The goal of the algorithm is to minimize the likelihood of this compound classifier. For details on the statistical formulation of the problem, the reader is referred to [36, 71, 78]. Moreover, the pseudo code of the algorithm is presented in [71]. Instead of giving all this formulation, in this section we briefly describe the process used to learn the compound classifier.

Fuzzy LogitBoost generates N classifiers, where N is a user-set parameter. Each example i in the training dataset has associated a weight w^i that represents the importance of the example. At each learning iteration, a genetic algorithm is applied to find a rule that fits accurately the training data according to the weights of the different examples. Then, the new generated weak hypothesis is added to the compound classifier, and the examples in the training dataset are re-weighted. In that way, in the following iterations, the search will be focused on examples that are more difficult to learn. Moreover, at the end of each iteration, a voting strength s^j for each class j is assigned to the new rule. s^j is calculated according to the confidence with which the rule predicts class j.

4 Experimental methodology

This section presents the experimental methodology followed to evaluate the six analyzed GBML techniques. First, we compare the accuracy and the size of the models evolved by the six GBML learning systems . This analysis enables us to emphasize the benefits and the drawbacks of the different systems. Then, the six systems are compared with several of the most influential learning algorithms that belong to a wide variety of learning paradigms. In the following, we provide details of (i) the real-world problems chosen for the experimentation; (ii) the metrics used to evaluate the performance and the interpretability of the models built by the different techniques; (iii) the configuration parameters of the GBML methods; (iv) the learning techniques included in the comparison of GBML methods with non-evolutionary machine learning algorithms; and (iv) the statistical analysis applied to compare the results obtained with the learning systems.

4.1 Test bed

We selected a collection of twenty real-world data sets whose characteristics are detailed in Table 2. The problems represent different characteristics which may pose particular challenges to the different learning techniques. All these data sets were obtained from the UCI repository [7], except for *tao*, which was chosen from a local repository [12].

4.2 Comparison metrics

We evaluated the performance and the interpretability of the models evolved by each learning system. We used the test accuracy, i.e., the proportion of correct classifications on previously unseen examples to measure the performance of the method. To obtain reliable estimates of this indicator, we used a ten-fold cross validation procedure [27]. The partitioned data sets are available at the authors;[2] thence, researchers can easily compare their own machine learning techniques with the results supplied in our comparative analysis. The results provided in the next section

[2] http://www.salle.url.edu/~aorriols/DataSets.tgz.

Table 2 Properties of the data sets

Id.	Data set	#Inst	#Fea	#Re	#In	#No	#Cl	%Miss
ann	Annealing	898	38	6	0	32	5	0
aut	Automobile	205	25	15	0	10	6	22.4
bal	Balance	625	4	4	0	0	3	0
bpa	Bupa	345	6	6	0	0	2	0
cmc	Contracep. method choice	1,473	9	2	0	7	3	0
col	Horse colic	368	22	7	0	15	2	98.1
gls	Glass	214	9	9	0	0	6	0
h-c	Heart-c	303	13	6	0	7	2	2.3
h-s	Heart-s	270	13	13	0	0	2	0
irs	Iris	150	4	4	0	0	3	0
pim	Pima	768	8	8	0	0	2	0
son	Sonar	208	60	60	0	0	2	0
tao	Tao	888	2	2	0	0	2	0
thy	Thyroid	215	5	5	0	0	3	0
veh	Vehicle	846	18	18	0	0	4	0
wbcd	Wisc. breast-cancer	699	9	0	9	0	2	2.3
wdbc	Wisc. diag. breast-cancer	569	30	30	0	0	2	0
wne	Wine	178	13	13	0	0	3	0
wpbc	Wisc. prog. breast-cancer	198	33	33	0	0	2	2
zoo	Zoo	101	17	0	1	16	7	0

The columns describe: the identifier of the Data set (Id.), the name of the data set (Data set), the number of instances (*#Inst*), the total number of features (*#Fea*), the number of continuous features (*#Re*), the number of integer features (*#In*), the number of nominal features (*#No*), the number of classes (*#Cl*), and the proportion of instances with missing values (*%Miss*)

are averages over ten runs of the GBML systems with different random seeds.

The comparison of the readability of the models evolved by the GBML systems is more complicated since the methods included in the analysis use different rule representations (see Sect. 3). In our comparison, we use the number of rules evolved by the different learning methods to supply an approximate idea of the model complexity. We use this indicator to qualitatively compare the model readability, also considering the intrinsic differences of interpretability among the rules created by the different learning techniques. In addition, we also provide the average number of relevant variables per rule to complement this information.

4.3 Configuration of GBML techniques

The experiments were ran with the following source codes. For UCS, we used our own implementation. For GAssist, we used the open source code[3] which was developed by the authors of the learning system. The results of HIDER, HMOF, and SLAVE were kindly provided by the authors of the corresponding methods. For Fuzzy LogitBoost, we used the KEEL open source platform [5]. For the sake of repeatability, as follows we detail how the different learning algorithms were configured. In all cases, we searched for the best configuration, that is, the configuration which could yield the better results on average. For HIDER, HMOF, and SLAVE, the authors of the methods adjusted the configuration of the methods with the aim of achieving maximum accuracy. For GAssist and UCS, we used the default configurations reported in the literature (see [8, 69]). For LogistBoost, we employed the default configuration provided in the KEEL platform, which is set according to [71]. In any case, each method used the same configuration parameters for all the test bed, with the only exception of HMOF, which varied one parameter for three specific data sets. Therefore, we employ configurations that are able to evolve accurate models for different data sets, showing in this way the robustness of the different learning systems to the configuration parameters. More specifically, we used the following configuration parameters for each method.

– We ran UCS with a maximum population size of 6,400 classifiers during 100,000 iterations. The probabilities of crossover and mutation were set to 0.8 and 0.4, respectively. The other parameters of UCS were set to standard values (see [10, 70] for notation details): $acc_0 = 0.99$, $\nu = 10$, $\{\theta_{\mathrm{GA}}, \theta_{\mathrm{del}}, \theta_{\mathrm{sub}}\} = 50$, $\delta = 0.1$, $p_{\#} = 0.6$.
– GAssist was configured with a maximum population size of 400 rule sets and was run during 1,500 learning iterations. The probabilities of crossover and individual mutation were set both to 0.6. The remaining parameters were configured as follows (see [8] for notation details): *initialNumberOfRules* = 20, *probOne* = 0.90, *initMethod* = cwinit, *numStrata* = 2, *sizePenaltyMinRules* = 4, *tournamentSize* = 3, *defaultClass* = auto, *discretizer* = UniformWidth, *probMerge* = 0.05, *probReinitializeBegin* = 0.03, *probReinitializeEnd* = 0, *probSplit* = 0.05, *hierarchicalSelectionThreshold* = 0, *iterationHierarchicalSelection* = 24, *iterationMDL* = 25, *useMDL* = true, *initialTheoryLengthRatio* = 0.075, *weightRelaxFactor* = 0.90, *iterationRuleDeletion* = 5, *ruleDeletionMinRules* = 12.
– HIDER was configured as follows. Each genetic algorithm was run during 100 iterations with a population of 70 rules. The proportion of crossover was set to 0.8 and the probability of mutation to 0.5. Moreover, variables were mutated to open/close intervals with

[3] http://www.asap.cs.nott.ac.uk/~jqb/PSP/GAssist-Java.tar.gz.

probability 0.05. A maximum of 20% of replicated individuals were allowed in a GA run.

- HMOF was run during 5,000 learning iterations with a population size of 200 individuals. The crossover and mutation probabilities of both the Pittsburgh-style GBML and the local search inspired by a Michigan-style GBML were set to 0.9 and $1/\ell$, respectively (where ℓ is the number of input variables). The maximum number of rules per individual was fixed to 20. The probability of "*don't care*" was set to 0.8 for all the data sets except for *bal*, *irs*, and *tao*; in these three data sets, the probability of "*don't care*" was set to 0.5.
- SLAVE used a steady state GA with maximum population size of 100 individuals. At each iteration, two new classifiers were generated, which replaced the two worst rules in the population. To generate these offspring, two point crossover was applied with probability 1 and uniform crossover was triggered with probability 0.01. The GA was stopped after 500 iterations without improving the best individual in the population.
- For Fuzzy LogitBoost, we fixed the number of rules to 50. To generate each rule, the system used a steady-state genetic algorithm with ten subpopulations of 100 individuals. The best individual after 2,500 crossover operations was added to the compound classifier.

Furthermore, the methods implemented different policies to deal with missing values. UCS, GAssist, and HIDER replaced the missing values of an attribute with (i) the average value of the attribute for the class of the instance in case of continuous attributes or (ii) the most frequent value of the attribute for the class of the instance in case of nominal attributes. HMOF, SLAVE, and Fuzzy LogitBoost did not use a replacement strategy; instead of this, they assumed that the matching of a rule variable with a missing value was maximum (i.e., $\mu_{A^k}(e_i) = 1$, if e_i was missing).

4.4 Machine learning techniques included in the comparison

We also compared the competence of the six GBML techniques with several widely-known machine learning techniques that represent a large variety of learning paradigms. More specifically, we compared the GBML methods with C4.5, IBk, Naïve Bayes, PART, and SMO. C4.5 [73] is a decision tree that enhances ID3 by introducing methods to deal with continuous variables and missing values. IBk [4] is a nearest neighbor algorithm; it classifies a test instance with the majority class of its k nearest neighbors. Naïve Bayes [57] is a probabilistic classifier that estimates the parameters of a Bayesian model. PART [33] is a learning architecture that combines the extraction of rules from partial decision trees and the separate-and-conquer rule learning technique to create a rule-based classifier without performing global optimization. SMO [72] is a widely-used implementation of *support vector machines* [84]. In our experiments, we used SMO with polynomial kernels and SMO with Gaussian kernels. Table 3 summarizes the main characteristics of these learning systems. It is worth mentioning that C4.5, nearest neighbor classifiers, support vector machines, and Naïve Bayes have been selected among the ten most influential machine learning techniques in data mining [94].

All these methods were run using WEKA [93]. For all of them, we used the configuration recommended in the WEKA platform, with the following exceptions. For IBk, we ran the experiments with $k = \{1,3,5,7\}$ and ranked the results; we selected the configuration that yielded the higher average rank, that is, $k = 5$. For SMO with polynomial kernels, we followed a similar strategy. We ran the system with polynomial kernels of order $\{1,3,5,10\}$ and chose the results of the configuration that provided the best average rank, i.e., polynomial kernels of order 3.

Table 3 Summary of the main characteristics of the machine learning techniques included in the comparison: C4.5, IBk, Naive Bayes (NB), Part, and SMO

	Paradigm	Knowledge representation and inference method
C4.5	Decision-tree induction	Decision-tree *Inference*: class given by the corresponding leaf
IBk	Instance-based learning	No general model *Inference*: class determined by the majority class of the k nearest neighboors
NB	Statistical modeling	Probabilities of a Bayesian model *Inference*: the output is the class with maximum probability
Part	Rule induction based on decision-tree induction and a separate-and-conquer approach	Ordered list of rules. Continuous variables represented by float-coded attributes *Inference*: the output is the class of the first matching rule in the ordered list
SMO	Neural networks (support vector machines)	Weights of the support vector machines *Inference*: The class is determined by the decision function represented by the SVM

4.5 Statistical analysis

We followed the recommendations pointed out by Demšar [26] to perform the statistical analysis of the results. As suggested by Demšar, we used non-parametric statistical tests to compare the accuracies and sizes of the models built by the different learning systems. We avoided using parametric tests since they require strong conditions on the input data and the tests to check these conditions are ineffective unless large volumes of data are provided.

To compare multiple learning methods, we first applied a multi-comparison statistical procedure to test the null hypothesis that all the learning algorithms obtained the same results on average. Specifically, we used the Friedman test [37, 38], the non-parametric equivalent to the analysis of variance ANOVA test [32]. If the Friedman test rejected the null hypothesis, post-hoc tests were applied. Our first concern was to study whether each learning algorithm significantly differed from the best technique. For this purpose, we applied several post-hoc statistical tests. Firstly, we applied the Bonferroni–Dunn test [30], which defines that one learning method performs significantly differently from a control method (in our experiments, we used the best ranked system as the control method) if the corresponding average rank differs by, at least, a critical distance CD computed as

$$CD = q_\alpha \sqrt{\frac{n_\ell(n_\ell+1)}{6n_{\mathrm{ds}}}} \tag{9}$$

where n_ℓ is the number of learning techniques, n_{ds} is the number of data sets, and q_α is the critical value based on the Studentized range statistic [79]. The Bonferroni–Dunn test enables us to graphically illustrate the results by means of the critical distance; thus, it permits to easily visualize the significant differences among groups of learning systems. Nonetheless, the Bonferroni–Dunn test is said to be less powerful than other non-parametric tests. For this reason, to complement the multi-comparison statistical analysis, we used the Holm's step-down procedure [49, 53]. Holm's test compares each classifier i with a control learning technique, which has to be the best ranked technique, by computing the z-value

$$z_{ib} = (R_i - R_b) \Big/ \sqrt{\frac{n_\ell(n_\ell+1)}{6n_{ds}}} \tag{10}$$

where R_b is the rank of the best ranked learning technique, and R_i is the rank of the learning method that is to be compared with the best technique. Next, the p-value p_i is calculated from each z_{ib}. All the p_i are sorted so that $p_1 < p_2 < \cdots < p_{n_\ell}$. Then, the Holm's procedure starts with the most significant p_i, i.e., p_1. If p_1 is below $\alpha/(n_\ell - 1)$ the corresponding hypothesis is rejected and the second hypothesis is tested. This strategy is repeated until we find the first hypothesis that cannot be rejected. In this case, all the remaining hypothesis are retained as well. In the next section, we apply the Bonferroni–Dunn test at $\alpha = 0.10$ and the Holm's procedure at $\alpha = 0.05$. We use a larger significance level in the Bonferroni–Dunn test since it is more conservative than the Holm's procedure.

Finally, we also applied pairwise comparisons to compare pairs of learning algorithms. For this purpose, we used the non-parametric Wilcoxon signed-ranks test [86] and provided the approximative p-values computed as indicated in [79].

5 Comparison

Firstly, the analysis provided in this section compares the behavior of the six GBML techniques, highlighting the advantages and weaknesses of the different GBML paradigms. Next, the GBML techniques are compared with some of the most influential machine learning techniques for pattern recognition. The experimental results show the robustness and competitiveness of the GBML approaches in the realm of pattern recognition.

5.1 Comparison of the six analyzed GBML techniques

Tables 4 and 5 show the test accuracies and the number of rules of the models obtained with the GBML methods. For HMOF, we supply the results that correspond to the solution of the Pareto front that has the best training accuracy. The accuracy and size of the models created by HIDER in the *son* problem are not provided since the system could not evolve rules that covered the test instances; this behavior was due to the high dimensionality of the problem with respect to the number of instances. The last two rows of the table show the average rank and the absolute position in the ranking of each method. To complement the information of the model sizes, Table 6 provides the average number of variables used per rule in the GBML models.

We statistically analyzed the results to detect significant differences among the accuracy and the size of the models evolved by the different learning methods. The multi-comparison Friedman's test rejected the null hypotheses that (i) all the systems performed the same on average with $p = 1.22 \times 10^{-4}$, (ii) the number of rules of the models was equivalent on average with $p = 5.55 \times 10^{-15}$, and (iii) the average number of variables per rule was equivalent on average with $p = 6.05 \times 10^{-8}$. Therefore, we applied the post-hoc Bonferroni–Dunn test (at $\alpha = 0.10$) to detect (i) which learning methods performed equivalently

Table 4 Comparison of percentage test accuracy achieved by the six GBML techniques on the twenty real-world problems

Problem	UCS	GAssist	HIDER	HMOF	SLAVE	LogitBoost
ann	99.05 ± 1.39	97.88 ± 1.70	96.17 ± 2.23	97.56 ± 1.63	96.78 ± 2.21	76.20 ± 1.66
aut	77.41 ± 7.23	68.63 ± 4.13	68.80 ± 12.09	66.87 ± 11.26	70.71 ± 6.91	32.63 ± 3.22
bal	77.32 ± 4.03	79.57 ± 2.10	70.54 ± 3.37	92.18 ± 1.41	72.01 ± 6.03	88.30 ± 3.68
bpa	67.54 ± 9.63	62.24 ± 4.82	62.94 ± 3.76	65.09 ± 7.69	59.99 ± 4.75	64.46 ± 9.02
cmc	50.27 ± 4.32	53.58 ± 4.19	48.52 ± 3.29	53.16 ± 5.72	46.09 ± 3.82	51.10 ± 3.65
col	96.26 ± 3.21	94.30 ± 2.29	77.00 ± 4.92	84.50 ± 5.23	82.88 ± 3.92	63.06 ± 1.08
gls	70.04 ± 9.08	65.06 ± 7.99	66.27 ± 3.35	61.01 ± 9.49	57.62 ± 13.18	68.18 ± 8.06
h-c	79.72 ± 7.79	80.09 ± 6.16	74.93 ± 5.20	79.17 ± 4.51	77.86 ± 7.14	62.09 ± 4.52
h-s	74.63 ± 6.83	77.67 ± 5.82	74.11 ± 9.58	84.07 ± 4.29	76.30 ± 4.74	59.33 ± 4.43
irs	95.40 ± 4.51	96.20 ± 3.08	96.67 ± 4.71	95.50 ± 4.16	93.33 ± 5.96	95.33 ± 3.55
pim	74.61 ± 5.01	73.76 ± 2.99	74.47 ± 4.20	76.17 ± 2.03	72.28 ± 5.71	71.84 ± 4.43
son	76.49 ± 10.65	75.81 ± 10.37	–	75.43 ± 10.88	73.07 ± 7.51	53.38 ± 1.63
tao	87.00 ± 3.55	91.59 ± 2.29	86.44 ± 2.85	91.10 ± 2.39	82.14 ± 2.25	91.73 ± 2.56
thy	95.13 ± 4.22	92.52 ± 3.06	93.84 ± 5.26	94.16 ± 4.60	91.29 ± 7.19	97.08 ± 3.94
veh	71.40 ± 4.50	67.00 ± 2.82	64.30 ± 2.12	63.35 ± 2.71	66.55 ± 4.85	37.25 ± 4.76
wbcd	96.28 ± 1.92	95.59 ± 2.04	96.27 ± 2.65	95.39 ± 2.44	96.43 ± 2.04	94.12 ± 2.72
wdbc	95.96 ± 2.75	94.24 ± 2.49	88.08 ± 4.78	95.77 ± 2.41	91.74 ± 3.59	62.74 ± 0.65
wne	96.13 ± 4.14	93.19 ± 6.35	91.68 ± 4.54	96.60 ± 4.72	94.35 ± 3.62	85.02 ± 9.74
wpbc	69.40 ± 8.70	72.33 ± 5.26	63.73 ± 5.59	79.48 ± 8.52	75.73 ± 5.83	76.35 ± 2.23
zoo	96.78 ± 5.35	93.97 ± 5.44	95.80 ± 4.87	92.50 ± 7.20	96.50 ± 6.26	41.89 ± 6.65
Rank	2.20	3.00	4.30	2.85	4.15	4.50
Pos	1	3	5	2	4	6

The two last rows show the average rank of each learning algorithm (*Rank*) and its position in the ranking (*Pos*)

Table 5 Comparison of the number of rules of the models created by the six GBML techniques on the twenty real-world problems

Problem	UCS	GAssist	HIDER	HMOF	SLAVE	LogitBoost
ann	4410.2 ± 200.7	5.3 ± 0.3	11.8 ± 1.0	6.8 ± 7.2	8.0 ± 0.9	50.0 ± 0.0
aut	4064.2 ± 69.4	6.9 ± 0.4	18.1 ± 2.6	7.5 ± 6.6	17.4 ± 2.0	50.0 ± 0.0
bal	1712.1 ± 77.5	8.2 ± 0.6	6.0 ± 0.0	3.3 ± 4.1	22.2 ± 2.4	50.0 ± 0.0
bpa	2602.7 ± 220.3	6.4 ± 0.6	4.3 ± 0.2	7.0 ± 6.0	5.8 ± 1.5	50.0 ± 0.0
cmc	3175.1 ± 62.6	15.4 ± 1.9	50.9 ± 3.9	6.5 ± 4.5	48.6 ± 6.0	50.0 ± 0.0
col	3445.6 ± 249.3	4.5 ± 0.5	10.8 ± 0.6	7.3 ± 6.2	7.0 ± 1.1	50.0 ± 0.0
gls	3013.0 ± 114.4	4.7 ± 0.5	11.2 ± 0.6	6.6 ± 9.1	14.6 ± 2.3	50.0 ± 0.0
h-c	2892.6 ± 146.2	6.4 ± 0.7	6.1 ± 0.2	6.9 ± 7.9	6.4 ± 0.9	50.0 ± 0.0
h-s	3499.2 ± 115.6	5.5 ± 0.3	4.2 ± 0.3	6.2 ± 7.2	7.1 ± 0.9	50.0 ± 0.0
irs	634.3 ± 201.6	3.1 ± 0.1	3.0 ± 0.0	4.1 ± 2.2	3.0 ± 0.0	50.0 ± 0.0
pim	3225.1 ± 138.7	7.2 ± 0.6	5.4 ± 0.2	7.3 ± 8.4	13.3 ± 3.2	50.0 ± 0.0
son	5999.1 ± 95.9	4.6 ± 1.0	–	6.3 ± 8.0	8.5 ± 1.1	50.0 ± 0.0
tao	609.5 ± 123.7	5.9 ± 0.4	3.0 ± 0.0	7.1 ± 2.4	3.0 ± 0.0	50.0 ± 0.0
thy	1282.9 ± 202.9	4.1 ± 0.3	4.0 ± 0.4	5.2 ± 1.9	4.6 ± 0.5	50.0 ± 0.0
veh	4601.2 ± 56.0	6.9 ± 0.7	19.8 ± 0.6	6.6 ± 8.2	25.9 ± 4.0	50.0 ± 0.0
wbcd	1799.1 ± 118.1	3.5 ± 0.2	2.0 ± 0.0	5.3 ± 3.8	5.0 ± 1.5	50.0 ± 0.0
wdbc	5078.8 ± 36.9	4.3 ± 0.3	20.7 ± 0.7	5.8 ± 3.7	5.1 ± 0.8	50.0 ± 0.0
wne	3412.6 ± 290.7	3.2 ± 0.2	5.5 ± 0.3	3.9 ± 4.0	3.6 ± 0.9	50.0 ± 0.0
wpbc	5077.6 ± 26.1	3.9 ± 0.3	18.1 ± 0.3	4.6 ± 10.0	9.9 ± 2.8	50.0 ± 0.0
zoo	1243.7 ± 63.3	6.2 ± 0.3	7.5 ± 0.3	8.0 ± 8.5	7.3 ± 0.5	50.0 ± 0.0
Rank	6.00	1.70	2.63	2.80	2.93	4.95
Pos	6	1	2	3	4	5

The two last rows show the average rank of each learning algorithm (*Rank*) and its position in the ranking (*Pos*)

Table 6 Comparison of the average number of variables of the rules created by the six GBML techniques on the twenty real-world problems

Problem	UCS	GAssist	HIDER	HMOF	SLAVE	LogitBoost
ann	16.57 ± 0.44	9.51 ± 2.53	10.18 ± 0.30	3.68 ± 1.01	2.54 ± 0.45	5.36 ± 0.13
aut	16.20 ± 0.24	7.74 ± 1.04	22.60 ± 0.21	4.17 ± 0.92	4.87 ± 0.85	5.38 ± 0.21
bal	3.76 ± 0.04	2.43 ± 0.53	1.17 ± 0.00	2.52 ± 0.47	2.70 ± 0.05	2.10 ± 0.03
bpa	5.17 ± 0.22	3.68 ± 0.58	2.27 ± 0.20	3.52 ± 0.52	4.08 ± 0.54	1.68 ± 0.12
cmc	6.22 ± 0.06	5.32 ± 1.61	4.10 ± 0.08	3.32 ± 0.57	4.22 ± 0.15	2.61 ± 0.23
col	9.99 ± 0.36	4.81 ± 1.76	8.59 ± 0.16	2.10 ± 0.45	3.01 ± 0.59	2.61 ± 0.23
gls	7.78 ± 0.19	2.85 ± 0.54	6.52 ± 0.11	3.42 ± 0.74	3.71 ± 0.40	3.41 ± 0.17
h-c	7.37 ± 0.21	4.92 ± 0.71	6.14 ± 0.12	2.63 ± 0.87	3.68 ± 0.38	2.89 ± 0.24
h-s	10.75 ± 0.14	3.23 ± 0.33	3.62 ± 0.14	3.31 ± 0.49	3.93 ± 0.37	2.80 ± 0.22
irs	3.28 ± 0.27	1.55 ± 0.22	1.00 ± 0.01	1.93 ± 0.26	1.50 ± 0.17	1.83 ± 0.11
pim	6.90 ± 0.21	3.56 ± 0.59	4.31 ± 0.20	3.50 ± 0.48	3.90 ± 0.25	2.57 ± 0.14
son	27.11 ± 1.31	2.32 ± 0.32	–	3.36 ± 0.77	6.64 ± 1.86	4.05 ± 0.19
tao	1.93 ± 0.05	2.03 ± 0.32	1.00 ± 0.00	1.80 ± 0.16	1.33 ± 0.00	0.27 ± 0.03
thy	4.30 ± 0.20	2.09 ± 0.33	1.58 ± 0.11	2.36 ± 0.32	2.80 ± 0.24	2.05 ± 0.12
veh	15.70 ± 0.17	3.31 ± 0.53	14.82 ± 0.14	6.47 ± 1.11	6.33 ± 0.63	4.55 ± 0.20
wbcd	5.86 ± 0.36	3.74 ± 0.55	2.90 ± 0.32	2.83 ± 0.48	3.69 ± 0.70	4.00 ± 0.25
wdbc	24.97 ± 0.34	3.49 ± 0.66	28.26 ± 0.13	2.59 ± 0.55	3.40 ± 0.51	3.64 ± 0.21
wne	9.76 ± 0.39	0.62 ± 0.00	8.80 ± 0.22	3.10 ± 0.77	3.87 ± 0.51	4.88 ± 0.21
wpbc	28.00 ± 0.24	4.97 ± 0.77	28.79 ± 0.16	4.70 ± 1.31	6.91 ± 1.37	3.98 ± 0.29
zoo	5.44 ± 0.29	1.72 ± 0.20	1.16 ± 0.07	3.06 ± 0.60	1.26 ± 0.16	3.71 ± 0.13
Rank	4.75	2.10	2.80	1.55	2.35	1.45
Pos	6	3	5	2	4	1

The two last rows show the average rank of each learning algorithm (*Rank*) and its position in the ranking (*Pos*)

to the best ranked method according to the test accuracy (i.e., UCS); (ii) which methods evolved models whose number of rules was equivalent to the best ranked method according to the model size (i.e., GAssist); and (iii) which methods had an equivalent number of variables per rule to the best ranked method in this respect (i.e., LogitBoost). Figure 2 illustrates the results of this statistical analysis, comparing all systems by means of both the accuracy and the number of rules of the model. For the sake of clarity, the plot does not consider the number of variables per rule since the post-hoc test only detected a significant difference: UCS created rules with a higher number of variables than LogitBoost. To contrast the results, we also applied the Holm's step-down procedure, which is said to be more powerful than the Bonferroni–Dunn test and does not make any additional assumptions on the data. The Holm's test at $\alpha = 0.05$ detected the same significant differences among learning methods.

The following observations can be drawn from the statistical analysis of the results:

- UCS has the best average rank in terms of test accuracy, followed by HMOF and GAssist, which performed equivalently to UCS. UCS significantly outperformed SLAVE, HIDER, and Fuzzy LogitBoost. Nonetheless, UCS created the largest models among all the methods. UCS's models consisted of thousands of

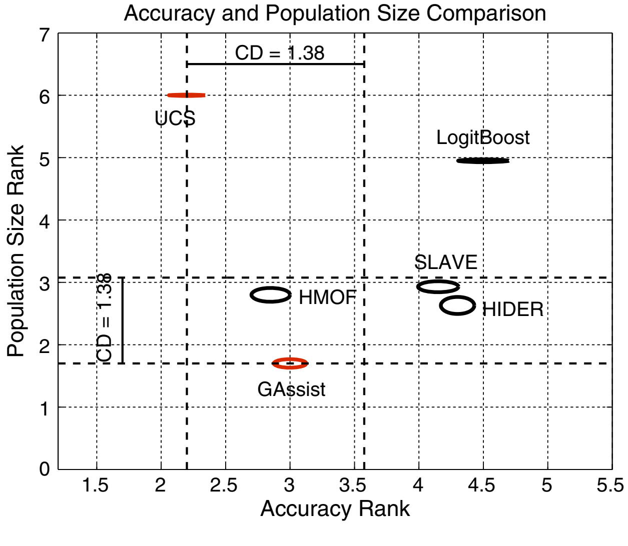

Fig. 2 Visual comparison of the accuracy and the number of rules of the models evolved by the six GBML systems. The figure plots an ellipse for each learning technique, which is centered on its average performance rank and its average rank of the number of rules. The semi-major and semi-minor axes of each ellipse have been sized proportionally to the standard deviation of the accuracy and population size rank, respectively. Dashed lines indicate the regions for which either the population size or the accuracy is equivalent to the best ranked method according to a Bonferroni–Dunn test at $\alpha = 0.10$

interval-based rules, each one with a large number of variables, in the majority of the problems. Thus, even using a rule-based representation, this high number of rules may hinder the readability of the evolved knowledge. Therefore, UCS appeared to be the best alternative if we need highly accurate models regardless of its interpretability. It is worth noting that UCS is an on-line learning algorithm, so that, differently from all the other presented approaches, it can learn from data streams [1, 21]. In fact, this is one of the reasons why UCS models consisted of a large number of rules.

- GAssist created the populations with the smallest number of rules, which were significantly smaller than the rule sets evolved by Fuzzy LogitBoost and UCS. Moreover, the rules contained few variables in their conditions. The sizes of the models built by HIDER, HMOF, and SLAVE did not significantly differ from the sizes of the rule sets constructed by GAssist.
- In general, GAssist and HMOF turned out to be the most robust methods of the comparison regarding the two objectives that we are analyzing here. Both learning methods built models whose accuracy and size did not differ from the best ranked method for each indicator. Besides, their rules contained few relevant variables. Thence, they reached a good equilibrium between accuracy and interpretability, being the best alternatives when the user searches for highly interpretable and accurate models in pattern classification tasks.
- SLAVE and HIDER emerged as good alternatives in terms of model size, although the accuracy of their models significantly differed from UCS, the most accurate system of the comparison. Below, we further compare the readability of the models evolved by the different learning systems by analyzing carefully the types of rules created by them.
- Fuzzy LogitBoost presented the poorest results of all the comparison. It degraded the accuracy of the other two fuzzy learning methods. Moreover, it evolved considerably larger model sizes, since the population size was a fixed parameter.

if $x \in [-6.00, -0.81]$ **and** $y \in [-6.00, 0.40]$ **then** red **with** acc= 1.00

if $x \in [2.84, 6.00]$ **and** $y \in [-5.26, 4.91]$ **then** blue **with** acc =1.00

if $x \in [-6.00, -0.87]$ **and** $y \in [-6.00, 0.74]$ **then** red **with** acc =1.00

...

(a) UCS

if $x > 2.72$ **and** ($y \in [0.92, 4.61]$ or $y > 5.07$) **then** blue

else if ($x \in [-0.54, 0.54]$ or $x > 2.72$) **and** $y \in [-4.28, -2.57]$ **then** blue

...

otherwise red

(b) GAssist

if $x \in [-2.87, 6.00]$ **and** $y \in [-0.125, 6.00]$ **then** blue

else if $x \in [-6.00, 3.12]$ **and** $y \in [-6.00, 6.00]$ **then** red

...

otherwise blue

(c) HIDER

if x is ML^5 **and** y is S^4 **then** blue **with** 0.123

if x is S^4 **and** y is L^4 **then** red **with** 0.827

if x is S^4 **then** red **with** 0.962

...

(d) HMOF

if x is {XS or S or M} **and** y is {XS or S or M} **then** red

if x is XS **then** red

if x is {VL or M or H or VH} **then** blue

...

(e) SLAVE

if x is L **and** y is L **then** blue **with** -5.42 **and** red **with** 0.0

if x is M **and** y is XS **then** blue **with** 2.21 **and** red **with** 0.0

if x is M **and** y is XL **then** blue **with** -2.25 **and** red **with** 0.0

...

(f) LogitBoost

Fig. 3 Examples of part of the models evolved by **a** UCS, **b** GAssist, **c** HIDER, **d** HMOF, **e** SLAVE, and **f** Fuzzy LogitBoost for the two-dimensional tao problem. The fuzzy methods SLAVE and Fuzzy LogitBoost use the following five linguistic terms per variable: {XS, S, M, L, XL}. HMOF uses 15 linguistic terms per variable that correspond to the four fuzzy partitions {S^2, L^2}, {S^3, M^3, L^3}, {S^4, MS^4, ML^4, L^4}, and {S^5, MS^5, M^5, ML^5, L^5} and the "*don't care*" symbol

To further analyze the interpretability of the models created by the different learning techniques, Fig. 3 reports partial examples of the models evolved by the six GBML systems on the two-dimensional tao problem. Figure 4 complements this information by plotting an example of a rule set evolved by each learning system. That is, for each learner, we considered a single run and depicted all the rules contained in the final population as follows. For the interval-based techniques, we plotted the hyper rectangle defined by the intervals of the two variables of the rule. Note that GAssist and HIDER make the rules available as a decision list; thus, there are overlapping rules (the most specific rules tend to be at the beginning of the decision list). For the fuzzy systems, we depicted the hyper rectangle defined by the supports of the linguistic terms contained in each variable. In these systems, there are also overlapping rules; a rule inference mechanism that considers the information provided by the matching rules is applied to predict the final class. This analysis, together with the statistical comparison of the model sizes, enables us to qualitatively extract the following conclusions regarding the readability of the models created by the different learning systems:

- SLAVE uses the simplest rules to represent the knowledge (see Fig. 3). It creates linguistic fuzzy rules whose variables are defined by a disjunction of five linguistic terms. Rules can be regarded as independent

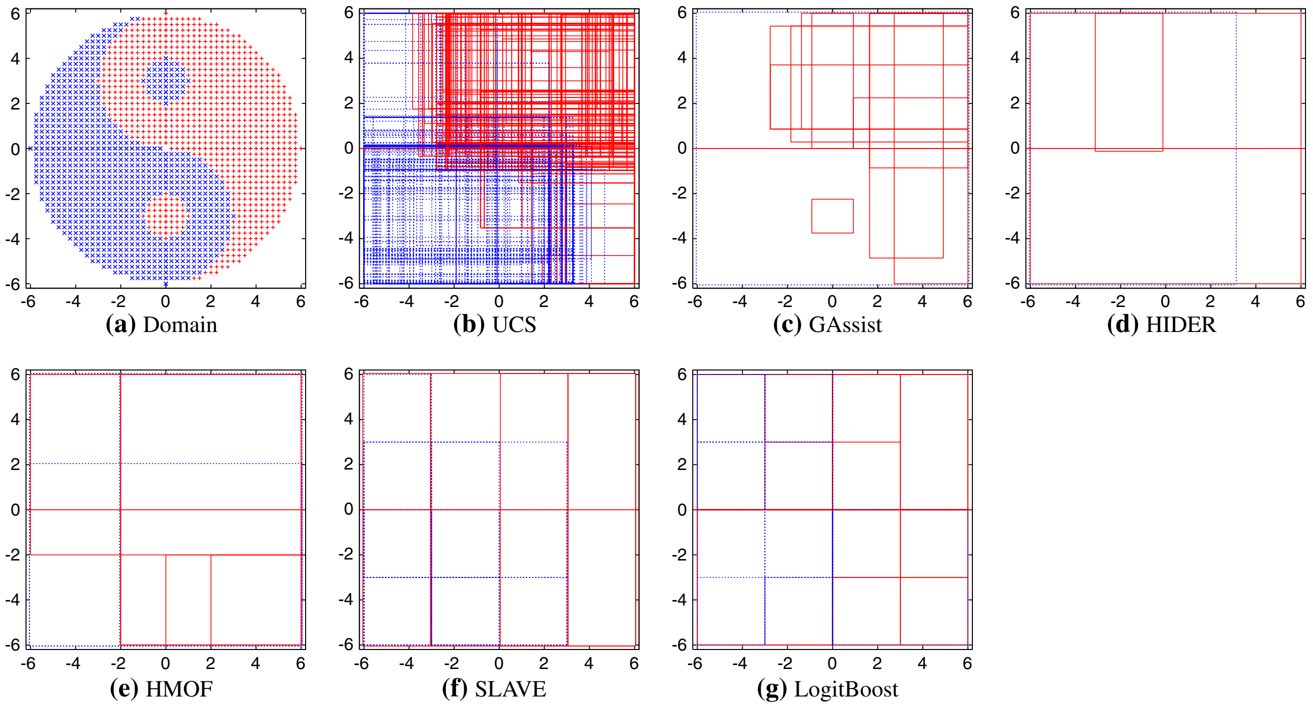

Fig. 4 Examples of the rules evolved by **b** UCS, **c** GAssist, **d** HIDER, **e** HMOF, **f** SLAVE, and **g** Fuzzy LogitBoost for the two-dimensional tao problem (**a**). In the three fuzzy systems, in addition, the fuzzy inference is applied to finally infer the class

classifiers that are experts in the region of the feature space that they cover. Rules does not have any weight associated. Then, the inference process chooses the class of the rule that maximizes the matching with the new unknown example. If more than one rule maximally matches the input instance, the class of the first matching rule is returned as the output. Thus, this simple inference process facilitates the readability of the evolved knowledge.

- HMOF representation differs from SLAVE rules in two aspects (see Fig. 3). First, each variable is defined by a single label, which is chosen among fifteen possible linguistic terms. Second, the HMOF rules have a weight associated, which is used to infer the class of any test instance. We consider that this knowledge representation is slightly more complex than the representation used by SLAVE. Nevertheless, note that, in general, HMOF needed a smaller number of rules than SLAVE in the tested problems. Thus, in global, both knowledge representations may be similarly interpretable.
- Fuzzy LogitBoost employs rules whose variables are defined by a single linguistic term. Each rule maintains a weight per class, which indicates the confidence of the rule on the class. Thus, these rules are the least interpretable among those evolved by the fuzzy learning techniques.
- GAssist and HIDER create a similar knowledge representation, which is made available as a decision list. The main difference between both is that, in GAssist, each variable can be represented by a disjunction of values, which enables the system to evolve slightly smaller rule sets than those created by HIDER. Moreover, GAssist forces the creation of a default rule. On the other hand, HIDER does not require to create this default rule, but the system architecture makes a strong pressure toward its creation; therefore, this default rule is usually present in HIDER models. Note that this type of representation is by far less interpretable than the presented fuzzy models for two main reasons. Firstly, because rules' variables are defined by intervals instead of linguistic terms. Secondly, because the rules of GAssist and HIDER depend on the context, while the rules of the three fuzzy learning methods are independent classifiers. That is to say, in GAssist and HIDER, a rule is used to infer the class of a new test instance only if all the previous rules in the decision list do not match the input instance. Thus, the context of the rules (i.e., the conditions of the previous rules in the decision list) has to be considered to read the whole rule set. For example, note that in Fig. 4c and d there are several rules of one class that contain rules of other classes; this aspect is not present in UCS's rule sets (see Fig. 4b).

Therefore, this makes evident a trade-off between the size of the models and the interpretability of individual rules. In some cases, slightly bigger populations of more interpretable rules may be preferred by human experts.

- UCS uses interval-based rules which predict a class for the region of the feature space that they cover. These rules can be easily interpreted individually. Nonetheless, as mentioned in the above discussion, the main problem of UCS is that it tends to evolve large rule sets, hindering the readability of the whole model. Although several reduction techniques have been proposed over the years [28, 39, 91], reduced populations are still bigger than those evolved by GBML systems [67], such as GAssist, HIDER, HMOF, and SLAVE.
- The three fuzzy learning techniques define the rule variables with linguistic terms that belong to a shared semantic. For this reason, the rules partition the feature space in uniformly distributed hyper rectangles (see Fig. 4e, f and g). On the other hand, the interval-based learning methods can define independent intervals for each attribute, which enables these systems to build hyper rectangles that fit the decision boundaries more accurately (see Fig. 4b, c, and d).

The study performed herein has highlighted the strengths and weaknesses of the different approaches for pattern recognition tasks. We showed that the rule-based representation permits the evolution of highly accurate models that can be easily interpreted by human experts. In the next section, we compare these systems to some of the most used machine learning techniques. The aim of the following study is to analyze whether GBML is a competitive alternative to deal with new challenging classification problems.

5.2 Comparison with other machine learning techniques

We now study the competitiveness of the six GBML systems with respect to some of the most influential methods from several learning paradigms. Table 7 provides the test accuracy achieved by (1) the decision tree C4.5, (2) the instance-based algorithm IBk, (3) the probabilistic classifier Naïve Bayes, (4) the rule-induction method PART, and the support vector machine SMO with (5) polynomial kernels of order 3 and (6) Gaussian kernels. Table 8 ranks the performance obtained with the six machine learning techniques and the six GBML methods (see Table 4). The last two rows of the table show the average rank and the absolute position in the ranking of each of the twelve learning techniques.

The multi-comparison Friedman test rejected the null hypothesis that all the twelve learning systems performed the same on average with $p = 1.56 \times 10^{-5}$. We applied the post-hoc Bonferroni–Dunn test at $\alpha = 0.10$ to detect

Table 7 Test accuracy achieved by C4.5, IB5, Naïve Bayes, PART, SMO with polynomial kernels of order 3 (SMOp3), and SMO with Gaussian kernels

Problem	C4.5	IB5	Naïve Bayes	PART	SMOp3	SMOrbf
ann	98.90 ± 0.88	97.34 ± 1.66	86.33 ± 3.60	98.57 ± 1.25	99.34 ± 0.57	91.90 ± 2.39
aut	80.94 ± 10.30	64.03 ± 7.57	58.79 ± 13.49	74.41 ± 8.66	78.09 ± 5.58	45.55 ± 7.47
bal	77.42 ± 3.60	88.18 ± 3.50	90.57 ± 2.27	82.86 ± 3.66	91.20 ± 2.93	88.30 ± 3.42
bpa	62.31 ± 5.19	58.85 ± 7.65	55.97 ± 14.33	67.56 ± 5.46	59.97 ± 3.03	57.99 ± 1.17
cmc	52.62 ± 4.52	46.51 ± 2.57	50.65 ± 4.50	50.04 ± 2.52	48.75 ± 3.91	42.70 ± 0.61
col	85.32 ± 5.08	81.49 ± 7.76	78.23 ± 5.86	84.51 ± 3.70	75.59 ± 6.72	82.41 ± 5.98
gls	66.15 ± 6.87	64.68 ± 11.09	48.95 ± 7.41	66.62 ± 9.56	66.15 ± 10.40	35.65 ± 4.83
h-c	78.45 ± 9.30	83.16 ± 6.68	82.80 ± 5.45	74.20 ± 9.11	78.59 ± 6.55	82.48 ± 5.95
h-s	79.26 ± 7.65	80.74 ± 7.16	83.33 ± 6.59	80.00 ± 7.45	78.89 ± 7.21	82.59 ± 8.38
irs	94.00 ± 5.84	96.00 ± 4.66	96.00 ± 3.44	94.00 ± 7.98	92.67 ± 5.84	93.33 ± 6.29
pim	74.23 ± 2.85	73.32 ± 4.37	75.80 ± 6.12	74.88 ± 4.23	76.70 ± 3.66	65.11 ± 0.54
son	71.07 ± 12.99	84.05 ± 11.79	69.71 ± 9.87	74.38 ± 9.90	85.52 ± 11.19	69.26 ± 9.25
tao	95.92 ± 1.51	97.14 ± 1.01	80.98 ± 2.45	94.33 ± 1.56	84.22 ± 2.03	83.63 ± 2.20
thy	94.91 ± 3.35	94.85 ± 5.16	97.16 ± 3.32	94.33 ± 6.63	88.91 ± 7.64	69.83 ± 2.23
veh	71.14 ± 4.40	68.91 ± 3.44	46.28 ± 4.06	73.39 ± 2.86	83.30 ± 4.68	41.71 ± 2.66
wbcd	94.99 ± 3.09	97.14 ± 2.02	96.15 ± 2.12	95.71 ± 2.15	96.42 ± 1.94	96.13 ± 2.27
wdbc	94.40 ± 3.58	96.78 ± 1.65	93.13 ± 2.76	94.46 ± 3.73	97.58 ± 1.71	92.88 ± 3.82
wne	93.89 ± 6.65	96.67 ± 3.88	97.19 ± 2.96	93.30 ± 6.30	97.75 ± 2.91	39.93 ± 2.60
wpbc	71.61 ± 15.26	78.85 ± 10.39	69.45 ± 11.50	70.05 ± 10.70	81.25 ± 9.12	72.97 ± 11.40
zoo	92.81 ± 6.92	90.47 ± 8.37	94.47 ± 5.98	93.81 ± 7.19	97.83 ± 4.72	76.03 ± 14.60

Table 8 Ranking of the performance achieved by C4.5, IB5, Naïve Bayes, PART, SMO with polynomial kernels of order 3 (SMOp3), SMO with Gaussian kernel (SMOrbf), and the six GBML techniques

	C4.5	IB5	Naïve Bayes	PART	SMOp3	SMOrbf	UCS	GAssist	HIDER	HMOF	SLAVE	LogitBoost
ann	3	7	11	4	1	10	2	5	9	6	8	12
aut	1	9	10	4	2	11	3	7	6	8	5	12
bal	9	6	3	7	2	4.5	10	8	12	1	11	4.5
bpa	6	10	12	1	9	11	2	7	5	3	8	4
cmc	3	10	5	7	8	12	6	1	9	2	11	4
col	3	8	9	4	11	7	1	2	10	5	6	12
gls	5.5	8	11	3	5.5	12	1	7	4	9	10	2
h-c	8	1	2	11	7	3	5	4	10	6	9	12
h-s	6	4	2	5	7	3	10	8	11	1	9	12
irs	8.5	3.5	3.5	8.5	12	10.5	6	2	1	5	10.5	7
pim	7	9	3	4	1	12	5	8	6	2	10	11
son	8	2	9	6	1	10	3	4	12	5	7	11
tao	2	1	12	3	9	10	7	5	8	6	11	4
thy	4	5	1	6	11	12	3	9	8	7	10	2
veh	4	5	10	2	1	11	3	6	8	9	7	12
wbcd	11	1	6	8	3	7	4	9	5	10	2	12
wdbc	6	2	8	5	1	9	3	7	11	4	10	12
wne	7	3	2	8	1	12	5	9	10	4	6	11
wpbc	8	3	10	9	1	6	11	7	12	2	5	4
zoo	8	10	5	7	1	11	2	6	4	9	3	12
Rank	5.90	5.38	6.73	5.63	4.73	9.20	4.60	6.05	8.05	5.20	7.93	8.63
Pos.	6	4	8	5	2	12	1	7	10	3	9	11

The two last rows show the average rank of each system (*Rank*) and its position in the ranking (*Pos*)

which learning techniques performed differently from the best ranked method. Figure 5 graphically represents the rank of each technique and connects with a line the classifiers whose performance does not statistically differ from the best ranked method. We also applied the Holm's step-down procedure, which yielded the same statistical conclusions.

The statistical analysis shows that UCS is the best ranked method of the comparison, achieving a better average rank than widely-used machine learning techniques such as the support vector machine SMO, the instance-based classifier IBk, and the decision tree C4.5 among others. Moreover, the study also detects that seven learning methods perform equivalently to UCS: the two GBML methods HMOF and GAssist; and the five machine learning techniques SMO with polynomial kernels of order 3, IBk, PART, C4.5, and Naïve Bayes.

We complemented the statistical study by comparing each pair of learning systems. It is well known that pairwise comparisons increase the risk of rejecting null hypotheses that are actually true. Herein, we assume this risk with the aim of providing further information about the excellence of the different methods. Table 9 shows the approximate p-values of the pairwise comparison according to a Wilcoxon signed-ranks test. The symbols ⊕ and ⊖ indicate that the method in the row significantly improves/degrades the performance obtained with the method in the column at a significance level of 0.05. Similarly, the symbols + and − are used to denote a non-significant

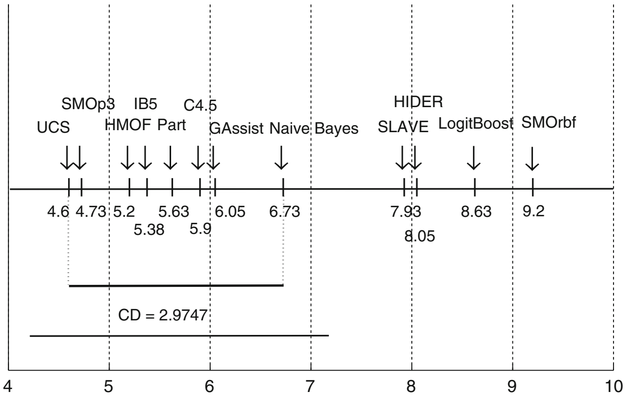

Fig. 5 Visual comparison of the performance of the non-evolutionary and the GBML techniques. The learning systems connected with a line are those that perform equivalently to the best ranked learning method according to a Bonferroni–Dunn test at $\alpha = 0.10$

Table 9 Pairwise comparisons of the learning methods by means of a Wilcoxon signed-ranks test

	C4.5	IB5	Naïve Bayes	PART	SMOp3	SMOrbf	UCS	GAssist	HIDER	HMOF	SLAVE	LogitBoost
C4.5		0.709	0.263	0.904	0.421	0.007	0.204	0.941	0.004	0.478	0.012	0.011
IB5	=		0.053	0.794	0.411	0.000	0.852	0.455	0.040	0.911	0.019	0.005
Naïve Bayes	−	−		0.247	0.067	0.033	0.135	0.108	0.695	0.012	0.601	0.021
Part	+	−	=		0.296	0.006	0.232	0.433	0.004	0.641	0.014	0.006
SMOp3	+	+	+	+		0.003	0.654	0.296	0.021	0.709	0.004	0.007
SMOrbf	⊖	⊖	⊖	⊖	⊖		0.006	0.003	0.263	0.000	0.027	0.573
UCS	+	=	+	+	−	⊕		0.218	0.000	0.550	0.002	0.007
GAssist	−	−	+	−	−	⊕	−		0.012	0.765	0.017	0.017
HIDER	⊖	⊖	−	⊖	⊖	+	⊖	⊖		0.014	0.490	0.167
HMOF	+	=	⊕	=	−	⊕	−	−	⊕		0.011	0.001
SLAVE	⊖	⊖	+	⊖	⊖	⊕	⊖	⊖	+	⊖		0.057
LogitBoost	⊖	⊖	⊖	⊖	⊖	−	⊖	⊖	−	⊖	−	

The above diagonal contains the approximate p-values. The below diagonal shows a symbol $\oplus/\ominus$ if the method in the row significantly outperforms/degrades the method in the column at a significance level of .05 and $+ / = / -$ if there is no significant difference and performs better/equal/worst

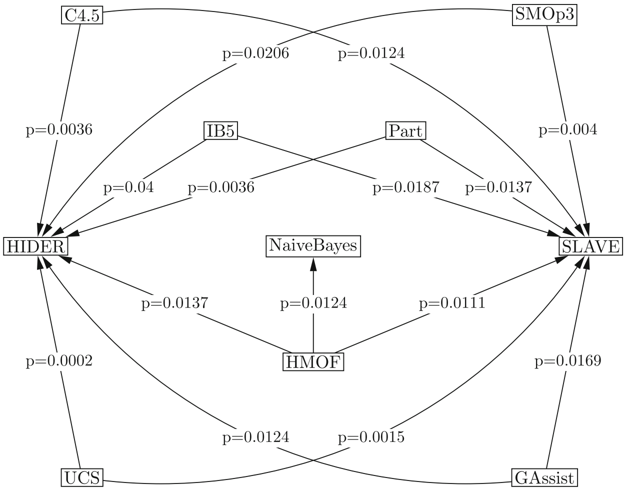

Fig. 6 Illustration of the significant differences of the performance among the GBML systems and the non-evolutionary learning techniques according to a Wilcoxon signed-ranks test at $\alpha = 0.05$. An edge $L_1 \xrightarrow{p_{\text{value}}} L_2$ indicates that the method L_1 outperforms the method L_2 with the corresponding p_{value}. To facilitate the visualization, LogitBoost and SMO with Gaussian kernel, the two most outperformed algorithms, are not included in the graph

improvement/degradation. The symbol = indicates that each method outperforms and degrades the other in the same number of data sets. The same information is graphically illustrated in the graph of Fig. 6, where each method is depicted in one vertex of the graph, and significant improvements (at $\alpha = 0.05$) of one learning method with respect to another are plotted with a directed edge labeled with the corresponding p-value. To facilitate the visualization, SMO with Gaussian kernel and Fuzzy LogitBoost were not included in the graph, since they were outperformed by most of the learning techniques. The pairwise analysis confirms the aforementioned conclusions. Moreover, it detects further differences between each pair of learning systems. Note that UCS, GAssist, and HMOF are never significantly outperformed. Moreover, HMOF significantly surpasses Naïve Bayes.

So far, we have shown the high performance of the models evolved by the GBML methods. Now, we qualitatively evaluate their readability with respect to the models evolved by the other six machine learning techniques. For this purpose, Fig. 7 provides examples of the models built by SMO, PART, C4.5, and Naïve Bayes. The model of IBk is not supplied since it does not build any global model; IBk is a lazy learning method that searches the k nearest neighbors for each test instance, and returns the majority class among them.

The least interpretable models are given by SMO. It creates models that consist of $\binom{n_c}{2}$ support vector machines (SVM), where n_c is the number of classes (see Fig. 7a). Each SVM is formed by $\ell + 1$ real-valued weights, where ℓ is the number of attributes of the problem. Thus, this type of model has low explanatory capabilities. On the other hand, Naïve Bayes builds models formed by a set of parameters which estimate the independent probability functions and the so-called class-priors of a Bayesian model (see Fig. 7b). These types of models can be easily interpreted from a probabilistic framework. The comparison between this type of knowledge and rule-based systems is not trivial and it is out of the scope of our study. For further details on the probabilistic models, the reader is referred to [65], where a framework that maps a Naïve Bayes classifier into a Neuro-Fuzzy Classifier is developed.

Fig. 7 Examples of part of the models evolved by **a** SMO, **b** Naïve Bayes, **c** PART, and **d** C4.5 for the two-dimensional tao problem

```
-   1.000     * <0.229 0.875 > * X]
-   0.298     * <0.708 0.437 > * X]
                    ...
```

(a) SMO

Class red: Prior probability $= 0.5$
  x: Normal Dist. $\bar{x}$ = -1.92 $s_x = 2.27$ $\sum w = 850$ prec. $= 0.25$
  y: Normal Dist. $\bar{y}$ = -1.13 $s_y = 2.73$ $\sum w = 850$ prec. $= 0.25$

Class blue: Prior probability $= 0.5$
  x: Normal Dist. $\bar{x} = 1.94$ $s_x = 2.27$ $\sum w = 850$ prec. $= 0.25$
  y: Normal Dist. $\bar{y} = 1.05$ $s_y = 2.78$ $\sum w = 850$ prec. $= 0.25$

(b) Naïve Bayes

if x $\leq$ -3.25 **then** red (308)
else if x $>$ 2.75 **then** blue (347/1)
else if y $\leq$ 0 **and** x $\geq$ -1 **then** red (192/1)
...

(c) PART

```
x <= -2.75
|  x <= -3.25: red (308.0)
|  x > -3.25
|  |   y <= 1.75: red (55.0)
|  |   y > 1.75
|  |  |  x <= -3: red (11.0/1.0)
|  |  |  x > -3
|  |  |  |     y <= 4.25: blue (6.0)
|  |  |  |     y > 4.25: red (4.0)
                    ...
```

(d) C4.5

Table 10 Average number of leaves of the decision trees built by C4.5 and average number of rules created by PART

Problem	C4.5	PART
ann	49 ± 5	15 ± 2
aut	61 ± 10	21 ± 5
bal	90 ± 4	37 ± 3
bpa	49 ± 5	9 ± 2
cmc	269 ± 18	168 ± 13
col	7 ± 1	9 ± 1
gls	48 ± 3	15 ± 1
h-c	48 ± 6	21 ± 2
h-s	33 ± 2	18 ± 2
irs	8 ± 1	4 ± 1
pim	37 ± 5	7 ± 2
son	27 ± 2	8 ± 1
tao	72 ± 4	17 ± 4
thy	15 ± 1	4 ± 0
veh	136 ± 4	32 ± 5
wbcd	24 ± 3	10 ± 1
wdbc	21 ± 2	7 ± 1
wne	10 ± 1	5 ± 1
wpbc	23 ± 3	7 ± 3
zoo	18 ± 3	8 ± 1

C4.5 constructs decision trees in which the nodes represent a condition over one variable of the problem (see Fig. 7d). Decision trees and rule-based systems create similar models; in fact, the paths from the root node to each leaf of a decision tree can be regarded as an individual rule. PART builds rule sets which are interpreted as decision lists, similarly to GAssist and HIDER. To qualitatively compare these two knowledge representations to those of the GBML systems, Table 10 reports the average number of leaves of C4.5 decision trees and the average number of rules created by PART. Note that PART created smaller models than C4.5. Nonetheless, both approaches built larger models than those evolved by GAssist, HIDER, HMOF, and SLAVE according to a Wilcoxon signed-ranks test at $\alpha = 0.05$.

The overall analysis provided in this section emphasized the competitiveness of the different GBML approaches for supervised learning with respect to some of the best representative methods of different machine learning paradigms. UCS is the learning algorithm that resulted in the most accurate models in average. GAssist and HMOF also yielded competitive results in terms of accuracy. Besides, GBML approaches, and more specifically, GAssist and HMOF, resulted in models that were by far more readable than those created by any other of the tested machine learning techniques; moreover, the statistical analysis highlighted that these models were as accurate as the models evolved by UCS, the best performer.

5.3 Further considerations

Finally, we discuss two further aspects that typically have discouraged the use of GBML in new challenging problems: the parametrization and the run time of the GBML methods.

The number of parameters of the majority of GBML systems is one of the first arguments against their use. As described in the Sect. 3, several parameters need to be specified before running the learning techniques. These parameters permit to tune the system's behavior for particular learning problems. Nonetheless, note that the

GBML systems were run with the same configuration for all the data sets of the comparison and under this parameterization they empirically demonstrated to be, at least, as good as the non-evolutionary machine learning techniques. Therefore, the default configuration can be used as a valid approximation for any new problem. In this context, configuration parameters can be seen as an opportunity to tune the method for particular data sets. Moreover, several heuristics can be designed to tune the system based on either the problem characteristics or the signals received during evolution. For example, in [68] an heuristic procedure that adjusts XCS according the apparent niche imbalance observed during learning was designed, providing an impressive improvement in highly imbalanced domains.

The run time of the different algorithms is also a burden typically associated to GBML methods, although the current increasing amount of computational resources is diminishing this problem. In our analysis, the learning time was not compared since the source codes were implemented with different programming languages, and the experiments were run in different machines. However, by analyzing the learning process of the different methods, the following appreciations can be made. In general, Pittsburgh-style GBML consume a lot of computational resources since all the rules of each new individual have to be evaluated with all the training data set. Moreover, the search space is huge since individuals represent sets of rules. In our comparison, HMOF was probably the most time-consuming algorithm, since it combined the Pittsburgh-style approach with a local search. In contrast, GAssist consumed moderate resources thanks to the different approaches that have been expressly designed to reduce its run time [8]. Michigan-style systems also tended to present high run-times, mostly due to the cost of creating the subsequent match sets. Finally, incremental rule learning approaches had a low run time as they incrementally reduce the search space. HIDER was one of the quickest approaches, presenting the lower computational time among all the tested GBML systems. SLAVE had a higher computational cost since it considers all the original data set for learning each one of the classes. Fuzzy LogitBoost had a computational cost which was similar to SLAVE’s cost since it ran a genetic algorithm for discovering each weak hypothesis.

Finally, it is worth noting that all the research in parallel genetic algorithms [17] can be applied to speed up GBML systems. For example, first steps toward the parallelization of Michigan-style GBML methods [14], Pittsburgh-style GBML techniques [58], IRL GBML systems [59], and GFRBSs [64] have been taken. Besides, several techniques have been recently developed to speed up GBML systems such as fitness surrogates for Pittsburgh-style GBML [61] and fast rule matching schemes for GBML [60].

6 Summary and conclusion

6.1 Summary

In this paper, we studied whether *genetic-based machine learning* systems are competitive for pattern recognition. We reviewed different GBML families and chose six state-of-the-art GBML systems. This selection included methods that use fuzzy and non-fuzzy rule representations. Then, we carefully compared the accuracy and the readability of the models evolved by the six approaches. This analysis permitted us to highlight the strengths and weaknesses of the different learning techniques, also pointing out some recommendations on which GBML system should be use depending on the requirements of the user. When the priority is to obtain highly accurate models, UCS appeared to be the best choice, since it was the best performer of the comparison. When the model interpretability was the primary goal, SLAVE turned up to be the best alternative, since it could evolve a compact and highly legible model based on linguistic fuzzy rules. Besides, other approaches such as GAssist and HMOF showed a nice balance between accuracy and interpretability, being able to evolve highly accurate and readable models.

We further investigated on the competitiveness of these GBML approaches by comparing them with some of the representatives of several machine learning paradigms. The different methods were adjusted to maximize accuracy. For example, several kernels were used for the support vector machine approaches and different neighbor recovery strategies were tested for the instance-based algorithms. In both cases, the configuration with the best results was selected for the comparison. The analysis of the results clearly highlighted the competitiveness of the GBML approaches. Again, UCS was the method that yielded the most accurate models among the GBML systems and the non-evolutionary systems. Besides, GBML also excelled in terms of interpretability. For example, GAssist and HMOF resulted in more readable models than the remaining machine learning techniques, maintaining a statistically equivalent test accuracy. All these results may encourage machine learning practitioners to consider GBML approaches to face new challenging problems.

6.2 SWOT analysis

All the evidence provided through the experimentation is summarized in the SWOT analysis presented in Table 11, where *strengths* represent the main advantages of GBML,

Table 11 SWOT analysis of genetic-based machine learning

Strengths	Weaknesses
Highly accurate models; comparable with the state-of-the-art in classification	High requirements of computational resources
Capabilities to explore complex search spaces	Large number of configuration parameters
Capabilities to optimize multiple objectives	Lack of a theoretical framework
Highly legible rule-based representation	Poor visibility in the pattern recognition community
Good at incorporating best bits of other ML paradigms	
Opportunities	Threats
Evolutionary Algorithms naturally support parallelization	Other learning techniques may build accurate and interpretable models more quickly
Flexibility to deal with different knowledge representations (representation-blind techniques)	Specific-designed methods for dealing with unusual data types such as data streams
Prepared to deal with unusual data types: data streams (Michigan-style GBML) and vague data (GFRBSs)	More visible methods in pattern recognition may discourage the use of GBML systems
Possibility of forming part of ensembles. They can also be used as an ensemble themselves	Open source codes of traditional machine learning techniques available in Internet
Application to other types of learning tasks	

weaknesses show its drawbacks, *opportunities* outline some suggested further lines of investigation, and *threats* include some optional approaches considered by other methods that could compete with GBML.

GBML systems have five important *strengths*. Firstly, they present high performance, which supports their use to deal with new challenging problems. Secondly, the use of an EA enables the learning methods to optimize the rule sets over complex search spaces. Thirdly, EAs also permit to optimize multiple objectives. This issue is essential in pattern recognition, since different objectives such as the prediction error and the size of the classification model have to be minimized. Notice that dealing with multiple objectives is a non-trivial task for non-evolutionary algorithms. Fourthly, GBML is able to evolve highly legible models by creating either a reduced set of interval-based rules or slightly larger sets of linguistic fuzzy rules. Lastly, as shown in the different GBML approaches presented in this paper, GBML has the ability to incorporate characteristics of other machine learning paradigms.

GBML has a principal *weakness*: the high run-time typically associated to its learning algorithms. The most competent GBML approaches tested in our analysis presented a higher average run-time than C4.5, one of the quickest non-evolutionary systems of the comparison. Other technical issues also appear as weaknesses of the system. On one hand, GBML systems tend to have a large number of configuration parameters; this question has been carefully discussed in Sect. 3. On the other hand, it is said that complete theoretical frameworks are lacking for GBML, which has been historically inspired by the inventiveness of the natural processes. In response to this respect, first theoretical frameworks [29] and facet-wise analyses [15] have been developed. Finally, the last weakness of GBML is its low visibility in the machine learning community. Recent advances on both GBML theory and applications may help encourage GBML techniques usage as competent, accurate, and reliable machine learning systems.

There are also some *threats* to GBML that are worth mentioning. First, a user may prefer quicker learning techniques to build approximate models of the input data. Second, there are some non-evolutionary methods that are able to perform the tasks associated to different GBML families such as on-line learning from data streams [66]. Third, the wider visibility of some non-evolutionary systems, jointly with the availability of their open source implementations, may cloud GBML visibility in research fields. During the last few years, the GBML community has striven to make the source codes of the most competent GBML systems available. One of the most significant efforts toward this direction can be found in the KEEL platform[4] [5].

We finally mention the many *opportunities* of GBML. The first opportunity is that EAs can be easily parallelized, decreasing their convergence time (see, for example [14, 58, 61, 64]). Secondly, the flexibility of the EAs let GBML systems deal with different types of representations. For example, GBML approaches have been applied to evolve tree-based representations [62], hyper ellipsoidal representations [16], and gene expression programs [92]. In fact, some GBML architectures can be addressed as ensemble

[4] http://www.keel.es.

systems that evolve different weak classifiers. For instance, the Michigan-style architecture can be regarded as an ensemble of classifiers that are experts on the region of the feature space that they cover. This naturally supports the co-evolution of different representations. On the other hand, ensembles of several GBML systems can be designed to improve learning performance for both Michigan-style GBML systems (see [14]) and Pittsburgh-style GBML methods (see [9]).

The third opportunity is due to the inclusion of a fuzzy representation to GBML systems. The use of fuzzy logic permits GBML methods to be easily adapted to deal with vague and uncertain data [76, 77], which is very common in real-world problems. Finally, as GBML methods typically present a general-purpose architecture, they can be applied to other types of tasks such as regression [90] and clustering [82].

Acknowledgments The authors would like to warmly thank Raúl Giráldez (Pablo de Olavide University), Yusuke Nojima (Osaka Prefecture University), and Raúl Pérez (University of Granada) for providing us with the experimental results of HIDER, HMOF, and SLAVE, respectively. The authors also wish to acknowledge the *Ministerio de Educacioń y Ciencia* for its support under projects TIN2005-08386-C05-01 and TIN2005-08386-C05-04, and *Generalitat de Catalunya* for its support under grants 2005FI-00252 and 2005SGR-00302.

References

1. Aggarwal C (ed) (2007) Data streams: models and algorithms. Springer, Heidelberg
2. Aguilar-Ruiz JS, Giraldez R, Riquelme JC (2007) Natural encoding for evolutionary supervised learning. IEEE Trans Evol Comput 11(4):466–479
3. Aguilar-Ruiz JS, Riquelme JC, Toro M (2003) Evolutionary learning of hierarchical decision rules. IEEE Trans Syst Man Cybern B 33(2):324–331
4. Aha DW, Kibler DF, Albert MK (1991) Instance-based learning algorithms. Mach Learn 6(1):37–66
5. Alcalá-Fdez J, Sánchez L, García S, del Jesus MJ, Ventura S, Garrell JM, Otero J, Romero C, Bacardit J, Rivas VM, Fernández JC, Herrera F (2008) KEEL: a software tool to assess evolutionary algorithms to data mining problems. Soft Comput (forthcoming)
6. Anikow CZ (1993) A knowledge-intensive genetic algorithm for supervised learning. Mach Learn 13(2–3):189–228
7. Asuncion A, Newman DJ (2007) UCI Machine learning repository: http://www.ics.uci.edu/~mlearn/MLRepository.html. University of California
8. Bacardit J (2004) Pittsburgh genetic-based machine learning in the data mining era: representations, generalization and runtime. PhD thesis, Ramon Llull University, Barcelona, Catalonia, Spain
9. Bacardit J, Krasnogor N (2008) Empirical evaluation of ensemble techniques for a pittsburgh learning classifier system. In: Learning classifier systems, revised selected papers of the international workshop on learning classifier systems 2006–2007. Springer, Heidelberg
10. Bernadó-Mansilla E, Garrell JM (2003) Accuracy-based learning classifier systems: models, analysis and applications to classification tasks. Evol Comput 11(3):209–238
11. Bernadó-Mansilla E, Ho TK (2005) Domain of competence of XCS classifier system in complexity measurement space. IEEE Trans Evol Comput 9(1):1–23
12. Bernadó-Mansilla E, Llorà X, Garrell JM (2002) XCS and GALE: a comparative study of two learning classifier systems on data mining. In: Advances in learning classifier systems volume 2321 of LNAI. Springer, Heidelberg, pp 115–132
13. Bonelli P, Parodi A (1991) An efficient classifier system and its experimental comparison with two representative learning methods on three medical domains. In 4th international conference on genetic algorithms, pp 288–295
14. Bull L, Studley M, Bagnall A, Whittley I (2007) Learning classifier system ensembles with rule-sharing. IEEE Trans Evol Comput 11(4):496–502
15. Butz MV (2006) Rule-based evolutionary online learning systems: a principled approach to LCS analysis and design volume 109 of studies in Fuzziness and Soft Computing. Springer, Heidelberg
16. Butz MV, Lanzi PL, Wilson SW (2008) Function approximation with XCS: hyperellipsoidal conditions, recursive least squares, and compaction. IEEE Trans Evol Comput. doi:10.1109/TEVC.2007.903551 (forthcoming)
17. Cantú-Paz E (2001) Efficient and accurate parallel genetic algorithms. Kluwer, Dordrecht
18. Castillo L, González A, Pérez R (2001) Including a simplicity criterion in the selection of the best rule in a genetic fuzzy learning algorithm. Fuzzy Sets Syst 120:309–321
19. Corcoran AL, Sen S (1994) Using real-valued genetic algorithms to evolve rule sets for classification. In: International conference on evolutionary computation, pp 120–124
20. O. Cordón, Herrera F, Hoffmann F, Magdalena L (2001) Genetic fuzzy systems: evolutionary tuning and learning of fuzzy knowledge bases volume 19 of advances in fuzzy systems—aplications and theory. World Scientific
21. Dam HH, Lokan C, Abbass HA (2007) Evolutionary online data mining: an investigation in a dynamic environment. In: Evolutionary computation in dynamic and uncertain environments volume 51/2007 of studies in computational intelligence. Springer Berlin/Heidelberg, pp 153–178
22. de Jong KA, Spears W (1991) Learning concept classification rules using genetic algorithms. In: Proceedings of the international joint conference on artificial intelligence. Sidney, pp 651–656
23. de Jong KA, Spears WM, Gordon DF (1993) Using genetic algorithms for concept learning. Genetic algorithms for machine learning. In: John J, Grefenstette (eds) A special issue of machine learning, vol 13, 2–3, pp 161–188
24. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197
25. del Jesús MJ, Hoffmann F, Navascués LJ, Sánchez L (2004) Induction of fuzzy-rule-based classifiers with evolutionary boosting algorithms. IEEE Trans Fuzzy Syst 12(3):296–308
26. Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30
27. Dietterich TG (1998) Approximate statistical tests for comparing supervised classification learning algorithms. Neural Comput 10(7):1895–1924
28. Dixon PW, Corne DW, Oates MJ (2004) A ruleset reduction algorithm for the XCSI learning classifier system. In: Lecture Notes in Computer Science, vol 2661/2003. Springer, Heidelberg, pp 20–29

29. Drugowitsch J, Barry AM (2008) A formal framework and extensions for function approximation in learning classifier systems. Mach Learn 70(1):45–88
30. Dunn OJ (1961) Multiple comparisons among means. J Am Stat Assoc 56:52–64
31. Eiben AE, Smith JE (2003) Introduction to evolutionary computing. Springer, Berlin
32. Fisher RA (1959) Statistical methods and scientific inference. 2nd edn. Hafner Publishing Co, New York
33. Frank E, Witten IH (1998) Generating accurate rule sets without global optimization. In: Proceedings of the 15th international conference on machine learning. Morgan Kaufmann, San Francisco, pp 144–151
34. Freitas A (2002) Data mining and knowledge discovery with evolutionary algorithms. Springer, Heidelberg
35. Freund Y, Schapire RE (1996) Experiments with a new boosting algorithm. In: International conference on machine learning, pp 148–156
36. Friedman J, Hastie T, Tibshirani R (2000) Additive logistic regression: a statistical view of boosting. Ann Stat 32(2):337–374
37. Friedman M (1937) The use of ranks to avoid the assumption of normality implicit in the analysis of variance. J Am Stat Assoc 32:675–701
38. Friedman M (1940) A comparison of alternative tests of significance for the problem of m rankings. Ann Math Stat 11:86–92
39. Fu C, Davis L (2002) A modified classifier system compaction algorithm. In: GECCO'02: Proceedings of the 2002 genetic and evolutionary computation conference. Morgan Kaufmann Publishers Inc., San Francisco, pp 920–925
40. Fürnkranz J (1999) Separate-and-conquer rule learning. Artif Intell Rev 13(1):3–54
41. Giordana A, Neri F (1995) Search-intensive concept induction. Evol Comput 3(4):375–419
42. Giráldez R, Aguilar-Ruiz JS, Riquelme JC (2002) Discretization oriented to decision rules generation. In: Knowledge-based intelligent information engineering systems and allied technologies (KES'02). IOS Press, pp 275–279
43. Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. 1st edn, Addison Wesley, Reading
44. Goldberg DE (2002) The design of innovation: lessons from and for competent genetic algorithms. 1st edn. Kluwer, Dordrecht
45. González A, Pérez R (1998) Completeness and consistency conditions for learning fuzzy rules. Fuzzy Sets Syst 96:37–51
46. González A, Pérez R (1999) SLAVE: a genetic learning system based on an iterative approach. IEEE Trans Fuzzy Syst 7(2):176–191
47. Greene DP, Smith SE (1993) Competition-based induction of decision models from examples. Mach Learn 13:229–257
48. Herrera F (2008) Genetic fuzzy systems: taxonomy and current research trends and prospects. Evol Intell 1(1):27–46. doi: 10.1007/s12065-007-0001-5
49. Hochberg Y (1988) A sharper Bonferroni procedure for multiple tests of significance. Biometrika 75:800–802
50. Holland JH (1975) Adaptation in natural and artificial systems. The University of Michigan Press, Ann Arbor
51. Holland JH (1976) Adaptation. In: Rosen R, Snell F (eds) Progress in theoretical biology, vol 4. Academic Press, New York, pp 263–293
52. Holland JH, Reitman JS (1978) Cognitive systems based on adaptive algorithms. In: Waterman DA, Hayes-Roth F (eds) Pattern-directed inference systems. Academic Press, San Diego, pp 313–329
53. Holm S (1979) A simple sequentially rejective multiple test procedure. Scand J Stat 6:65–70
54. Ishibuchi H, Nojima Y (2007) Analysis of interpretability-accuracy tradeoff of fuzzy systems by multiobjective fuzzy genetics-based machine learning. Int J Approx Reason 44(1):4–31
55. Ishibuchi H, Yamamoto T (2005) Rule weight specification in fuzzy rule-based classification systems. IEEE Trans Fuzzy Syst 13(4):428–435
56. Ishibuchi H, Yamamoto T, Murata T (2005) Hybridization of fuzzy GBML approaches for pattern classification problems. IEEE Trans Syst Man Cybern B Cybern 35(2):359–365
57. John GH, Langley P (1995) Estimating continuous distributions in bayesian classifiers. In: 11th conference on uncertainty in artificial intelligence. Morgan Kaufmann, San Mateo, pp 338–345
58. Llorà X, Garrell JM (2001) Knowledge-independent data mining with fine-grained parallel evolutionary algorithms. In: GECCO'01: Proceedings of the 2th annual conference on genetic and evolutionary computation. Morgan Kaufmann Publishers, San Mateo, pp 461?'468
59. Llorà X, Reddy R, Matesic B, Bhargava R (2007) Towards better than human capability in diagnosing prostate cancer using infrared spectroscopic imaging. In GECCO'07: Proceedings of the 9th annual conference on genetic and evolutionary computation. ACM, New York, pp 2098–2105
60. Llorà X, Sastry K (2006) Fast rule matching for learning classifier systems via vector instructions. In: GECCO'06: Proceedings of the 8th annual conference on Genetic and evolutionary computation. ACM, New York, pp 1513–1520
61. Llorà X, Sastry K, Yu T-L, Goldberg DE (2007) Do not match, inherit: fitness surrogates for genetics-based machine learning techniques. In: GECCO'07: Proceedings of the 9th annual conference on Genetic and evolutionary computation. ACM, New York, pp 1798–1805
62. Llorà X, Wilson SW (2004) Mixed decision trees: minimizing knowledge representation bias in lcs. In: GECCO'04: Proceedings of the genetic and evolutionary computation conference. Springer, LNCS, vol 3103, pp 797–809
63. Michalewicz Z (1999) Genetic algorithms + data structures = evolution programs. 3rd edn. Springer, Heidelberg
64. Nojima Y, Ishibuchi H, Kuwajima I (2008) Parallel distributed genetic fuzzy rule selection. Soft Comput (forthcomming)
65. Nurnberger A, Borgelt C, Klose A (1999) Improving naive Bayes classifiers using neuro-fuzzy learning. In: Proceedings of the 1999 conference on neural information processing, vol 1, Perth, pp 154–159
66. Núñez M, Fidalgo R, Morales R (2007) Learning in environments with unknown dynamics: towards more robust concept learners. J Mach Learn Res 8:2595–2628
66. Orriols-Puig A, Bernadó-Mansilla E (2004) Analysis of reduction algorithms for XCS classifier system. In: Recent advances in artificial intelligence research and development number 113 in 1. IOS Press, pp 383–390
68. Orriols-Puig A, Bernadó-Mansilla E (2006) Bounding XCS parameters for unbalanced datasets. In: GECCO'06: Proceedings of the 2006 genetic and evolutionary computation conference. ACM Press, New York, pp 1561–1568
69. Orriols-Puig A, Bernadó-Mansilla E (2008) Evolutionary rule-based systems for imbalanced datasets. Soft Comput J. doi: 10.1007/s00500-008-0319-7
70. Orriols-Puig A, Bernadó-Mansilla E (2008) Revisiting UCS: description, fitness sharing and comparison with XCS. In: Advances at the Frontier of LCSs. Springer, Heidelberg
71. Otero J, Sánchez L (2006) Induction of descriptive fuzzy classifiers with the logitboost algorithm. Soft Comput 10(9):825–835
72. Platt J (1998) Fast training of support vector machines using sequential minimal optimization. In: Advances in Kernel methods—support vector learning. MIT Press, Cambridge

73. Quinlan JR (1995) C4.5: programs for machine learning. Morgan Kaufmann Publishers, San Mateo
74. Rissanen J (1978) Modeling by shortest data description. Automatica 14:465–471
75. Rivest RL (1987) Learning decision lists. Mach Learn 2(3):229–246
76. Sánchez L, Couso I (2007) Advocating the use of imprecisely observed data in genetic fuzzy systems. IEEE Trans Fuzzy Syst 15(4):551–562
77. Sánchez L, Couso I, Casillas J (2007) Modeling vague data with genetic fuzzy systems under a combination of crisp and imprecise criteria. In: Proceedings of the 2007 IEEE symposium on computational intelligence in multicriteria decision making, pp 346–353
78. Schapire RE, Singer Y (1999) Improved boosting algorithms using confidence-rated predictions. Mach Learn 37(3):297–336
79. Sheskin DJ (2000) Handbook of parametric and nonparametric statistical procedures. Chapman & Hall, London
80. Smith SF (1980) A learning system based on genetic adaptive algorithms. PhD thesis. University of Pittsburgh, USA
81. Stone C, Bull L (2003) For real! XCS with continuous-valued inputs. Evol Comput 11(3):299–336
82. Tammee K, Bull L, Ouen P (2007) Ycsc: a modified clustering technique based on lcs. J Digit Inf Manage 5(3):160–167
83. Theodoridis S, Koutroumbas K (2006) Pattern Recognition, 3rd edn. Elsevier, Amsterdam
84. Vapnik V (1995) The nature of statistical learning theory. Springer, New York
85. Venturini G (1993) SIA: a supervised inductive algorithm with genetic search for learning attributes based concepts. In: Brazdil PB (eds) Machine learning: ECML-93 - Proc. of the European conference on machine learning. Springer, Berlin, pp 280–296
86. Wilcoxon F (1945) Individual comparisons by ranking methods. Biometrics 1:80–83
87. Wilson SW (1995) Classifier fitness based on accuracy. Evol Comput 3(2):149–175
88. Wilson SW (1998) Generalization in the XCS classifier system. In: 3rd annual conf. on genetic programming. Morgan Kaufmann, San Mateo, pp 665–674
87. Wilson SW (2000) Get real! XCS with continuous-valued inputs. In: Learning classifier systems. From foundations to applications LNAI, Springer, Berlin, pp 209–219
90. Wilson SW (2002) Classifiers that approximate functions. J Nat Comput 1(2):211–234
91. Wilson SW (2002) Compact rulesets from XCSI. In: Advances in learning classifier systems, 4th international workshop, Lecture Notes in Artificial Intelligence, vol 2321. Springer, Heidelberg, pp 197–210
92. Wilson SW (2008) Classifier conditions using gene expression programming. Technical report, IlliGAL Report No. 2008001, Urbana-Champaign IL 61801, USA
93. Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques, 2nd edn. Morgan Kaufmann, San Francisco
94. Wu X, Kumar V, Quinlan JR, Ghosh J, Yang Q, Motoda H, McLachlan GJ, Ng A, Liu B, Yu PS, Zhou ZH, Steinbach M, Hand DJ, Steinberg D (2007) Top 10 algorithms in data mining. Knowl Inf Syst 14(1):1–37