Soft Computing manuscript No.
(will be inserted by the editor)

Evolutionary Rule-Based Systems for Imbalanced Datasets

Albert Orriols-Puig, Ester 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,esterb}@salle.url.edu

Abstract This paper investigates the capabilities of evolutionary online rule-based systems, also called Learning Classifier Systems (LCSs), for extracting knowledge from imbalanced data. While some learners may suffer from class imbalances and instances sparsely distributed around the feature space, we show that LCSs are flexible methods that can be adapted to detect such cases and find suitable models. Results on artificial datasets specifically designed for testing the capabilities of LCSs in imbalanced data show that LCSs are able to extract knowledge from highly imbalanced datasets. When LCSs are faced with real-world problems, they demonstrate to be one of the most robust methods compared with instance-based learners, decision trees and support vector machines. Moreover, all the learners benefit from resampling techniques. Although there is not a resampling technique that performs best in all datasets and for all learners, those based in oversampling seem to perform better in average. The paper adapts and analyses LCSs for challenging imbalanced datasets and sets the bases for further studying the combination of resampling techniques plus learner best suited to a specific kind of problem.

Keywords. Imbalanced data, rule-based systems, data preprocessing, classification.

1 Introduction

During the last few years, machine learning techniques have been applied to complex real-world problems with the aim of extracting novel and useful knowledge. Due to either the rarity or the cost to obtain them, many real-world problems contain few examples of the concept to be described. This results in datasets with either *rare classes* or *rare cases* [30], and learning from these rarities has been identified as one of the main challenges in data mining. It has been shown that some learners such as C4.5 or multi-layered perceptrons may suffer when learning from datasets that contain rare classes[1], since they are biased toward the majority class [14, 15]. On the other hand, rare cases produce *small disjuncts*[2] [16], which concentrate the most part of classification error. In supervised learning, *rare classes* and *rare cases* are closely related; learners tend to create small disjuncts when learning from datasets with rare classes, and so, their effect can be hardly studied separately.

Evolutionary rule-based systems are a type of learners that evolve a set of rules by means of evolutionary algorithms. Among the different approaches that fit this definition, the so-called Learning Classifier Systems (LCSs) approach [12] is one of their best representatives. LCSs are online learners which evolve a set of rules that jointly represent the target concept. Although the robustness of evolutionary algorithms in imbalanced data has been widely shown [8], no systematic analyses have been conducted on Learning Classifier Systems (LCSs), which intrinsically use evolutionary algorithms to evolve the rule-based knowledge.

This paper studies the behavior of XCS [32, 33] and UCS [3], two accuracy-based LCSs that have demonstrated to perform competitively in classification tasks [6, 3]. First, we review the theory for learning from imbalanced data in XCS and UCS. The theoretical analysis states that both LCSs should be robust to class imbalances if they are properly configured. So, we summarize the guidelines to configure LCSs for an imbalanced dataset, given its imbalance ratio. Furthermore, we propose an algorithm that allows XCS/UCS to self-adapt if imbalances are detected during learning. This approach is essential in real-world problems, since the presence of small disjuncts is unknown a priori. The performance of XCS and UCS is tested on artificial problems that permit to vary separately the concept complexity and the imbalance level. After that, both LCSs are tested over

[1] Also referred as datasets with class imbalances.

[2] A *disjunct* is the definition of a subconcept of the original concept made by a specific learner.

a large set of real-world domains with different imbalance ratios and compared with three other well-known learners: C4.5 [26], SMO [25], IBk [1].

In highly imbalanced datasets, problems caused by *rare classes* and *rare cases* have been usually tackled by resampling the training datasets [2]. We investigate whether resampling techniques are valuable with LCSs and the other learners, and which of them offer better improvements.

The remainder of this paper is organized as follows. Section 2 introduces the problem of mining from rarities and reviews the main approaches proposed in the literature to deal with class imbalances. Section 3 briefly introduces both LCSs. Next, the theory of LCSs in imbalances is reviewed, and the algorithm that automatically adjusts XCS and UCS is proposed (Sect. 4). Section 5 shows the behavior of both LCSs on artificially imbalanced problems, and then, LCSs are compared to C4.5, SMO and IBk in real world problems. In Sect. 7, four resampling techniques are selected and introduced in the comparison. Finally, Sect. 8 summarizes, concludes and discusses further work.

2 Mining from Rarity: Class Imbalance and Small Disjuncts

In the recent years, several investigations have been conducted on the detection of two types of rarity: *rare classes* and *rare cases*. The concept of *rare classes* refers to datasets that contain different proportion of instances per class. The topic, which is mainly associated to supervised learning tasks, has also been addressed as the *class imbalance problem* [14]. Learning from datasets with *rare classes* usually hinders the performance of different learners. It has been shown that some learners, such as C4.5 or multi-layer perceptrons, are biased toward the majority class since they aim at minimizing a global measure of error [15]. The concept of *rare cases* is associated to both supervised and unsupervised tasks and refers to the sparse distribution of examples in the feature space. Specifically, it analyzes the problems derived from the presence of a small number of examples belonging to one class laying in a particular area of the feature space surrounded by examples of other classes. Usually, learners define a concept by means of several *disjuncts*[3]. In [13], *small disjuncts* were shown to hinder the performance of some learners; lately, some studies (e.g., [29]) indicated that most of the test error tends to concentrate around the *small disjuncts*.

In classification tasks, *rare classes* and *rare cases* are closely related. Jo and Japkowicz [17] argued that the performance degradation in imbalanced datasets was actually due to the presence of *small disjuncts*. Lately, Weiss [30] presented a unifying framework for both perspectives, suggesting that imbalanced datasets and small disjuncts may produce the same problems to data mining techniques. In fact, imbalanced datasets tend to present small disjuncts, as long as they consist of few instances of one class. In this paper, we consider both perspectives, and analyze the effect of class imbalances and small disjuncts as a whole.

Different approaches have been proposed to deal with class imbalances, which can be grouped in methods working at (i) the learner level, or (ii) the sampling level. Learner-level methods modify the learner to increase the pressure toward the discovery of the minority class. The main drawback of these methods is that they are designed for specific learners, and so, can hardly be transported to other learning schemes. Sampling-level methods, usually known as *resampling techniques*, resample the training dataset to balance the proportion of examples per class. As they are data-preprocessing methods, they can be generally used for any learner. Due to their flexibility, we only consider resampling methods in the remainder of this paper, and analyze whether they can improve the performance on several learners.

3 Learning Classifier Systems

Learning Classifier Systems (LCSs) are evolutionary online rule-based learners characterized by evolving a single set of rules. The ruleset is incrementally updated through the interaction with the environment and eventually improved by the action of evolutionary algorithms. XCS [32, 33], one of the best representatives of LCSs, uses a *reinforcement learning scheme* to evaluate the ruleset, while UCS [3] uses a *supervised learning scheme*. In the following, both systems are described in more detail.

3.1 Description of XCS

In the following, we provide a brief description of the different components of XCS. The reader is referred to [32, 33] for more details about the system, and to [7] for an algorithmic description.

Representation. XCS evolves a population [P] of classifiers, where each classifier has a rule and a set of associated parameters estimating the quality of the rule. Each rule has the form: $condition \rightarrow class$. The condition specifies the set of inputs where the classifier can be applied. For binary inputs, the condition is usually represented in the ternary alphabet: $\{0, 1, \#\}^n$, where n is the length of the input string. In this case, a condition $(c_1, c_2, ..., c_n)$ matches an input example $(x_1, x_2, ..., x_n)$, if and only if $\forall i\ c_i = x_i \vee c_i = \#$. The symbol $\#$, called *don't care*, allows the formation of generalizations in the rule's condition. If the input attributes are real, the condition is codified as a set of intervals $[l_i, u_i]^n$, which globally represents a hyperrectangle in the feature space. The

[3] A *disjunct* is a definition of a subconcept of the original concept.

class part of the rule specifies the class that is being predicted when the condition is satisfied.

Each classifier has a set of parameters estimating the quality of the rule. The most important ones are: a) the payoff prediction p, an estimate of the payoff that the classifier will receive if its condition matches and its class is selected, b) the prediction error ϵ, which estimates the average error between the classifier's prediction and the received payoff, c) the fitness F, an estimate of the accuracy of the payoff prediction, and d) the numerosity num, the number of copies of the classifier in the population.

Performance Component. At each time step, a training example x is sampled. Given x, the system builds a match set [M], which is formed by all the classifiers in [P] whose conditions are satisfied by x. If the number of classes represented in [M] is less than a threshold θ_{mna}, new classifiers are created through the covering operator. From [M], a class is selected and sent to the environment. If XCS is in training mode, the class is selected randomly. Thus, XCS explores the consequences of all classes for each possible input. Otherwise, when XCS is under testing, the selected class is that maximizing the expected payoff from the environment. The chosen class determines the action set [A], which consists of all classifiers advocating that class. The action set works as a *niche* where the parameter's update and the genetic algorithm take place.

Parameters Update. Once the class is sent to the environment, the environment returns a reward which is maximum if the proposed class is the same as the training example, and minimum (usually zero) otherwise. The reward r is used to update the parameters of the classifiers in [A]. Thus, the prediction of each classifier is updated according to: $p \leftarrow p + \beta(r - p)$, where β $(0 < \beta \leq 1)$ is the learning rate. Next, the prediction error: $\epsilon \leftarrow \epsilon + \beta(|r - p| - \epsilon)$. Then, we compute the accuracy of the classifier as an inverse function of the error, and finally, we update the fitness of each classifier as $F \leftarrow F + \beta(k' - F)$, where k' is the classifier's accuracy relative to the action set. Thus, fitness is an estimate of the accuracy of the classifier's prediction relative to the accuracies of the overlapping classifiers. This provides sharing among the classifiers belonging to the same action set.

Search Component. The search component in XCS is based on a genetic algorithm. The GA triggers with a frequency fixed by θ_{GA} and takes place in the action set. It selects two parents from the current [A] with probability proportional to fitness and copies them. The copies undergo crossover with probability χ and mutation with probability μ per allele.

Each offspring is introduced in the population, removing a classifier if the population is full. The deletion probability of a classifier is proportional to the size of the action sets where the classifier has participated and inversely proportional to its fitness [18]. This biases the search towards highly fit classifiers, and at the same time balances the classifiers' allocation in the different action sets.

3.2 Description of UCS

UCS [3] is a learning classifier system derived from XCS. It inherits the main features of XCS, but specialize them for supervised learning tasks. UCS mainly differs from XCS in two perspectives. Firstly, the learning interaction is adjusted to a supervised learning scheme. UCS benefits from knowing the class of the input example since it only explores the correct class. Secondly, the accuracy in UCS is computed as the percentage of correct predictions of the rule.

In the following, we briefly describe each component of the system. For further details, the reader is referred to [3, 24].

Representation. UCS inherits the rule representation of XCS. Thus, each rule has the form: $condition \longrightarrow class$. Moreover, each rule consists of 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 participates. Numerosity is the number of copies of the classifier, and experience is the number of times that a classifier has belonged to a match set.

Performance Component. In train mode, at each learning iteration, UCS receives an input example x and its class c. Then, the system creates the match set [M], which contains all classifiers in the population [P] whose condition matches x. From that, the correct set [C] is formed, which consists of the classifiers in [M] that predict the correct class. If [C] is empty, the covering operator is activated, creating a new classifier with a generalized condition matching x, and predicting class c. The remaining classifiers form the incorrect set ![C].

In test mode, a new input example x is provided, and UCS must predict the associated class. To do that, the match set [M] is created. All classifiers in [M] emit a vote, weighted by their fitness, for the class they predict. The most-voted class is chosen as the output.

Parameter Updates. Each time a classifier participates in a match set, its experience, accuracy, and fitness are updated. Firstly, the experience is increased. Then, the accuracy is computed as the percentage of correct classifications:

$$acc = \frac{\#correct\ classifications}{experience} \tag{1}$$

Thus, accuracy is a cumulative average of correct classifications over all matches of the classifier. Next, the

accuracy of the classifier relative to the action set is computed as follows:

$$k' = \frac{acc_{cl} \cdot num_{cl}}{\sum_{cl_i \in [M]} acc_{cl_i} \cdot num_{cl_i}} \quad (2)$$

and then, the fitness is updated: $F = F + \beta \cdot (k' - F)$, where $(0 < \beta \leq 1)$ is the learning rate. Finally, each time the classifier participates in [C], the correct set size *cs* is updated. *cs* is computed as the arithmetic average of the size of the correct sets where the classifier has taken part.

Search Component. The discovery component is copied from XCS, and is applied to the correct set. It 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. Finally, each offspring is introduced into the population, removing another classifier if the population is full.

3.3 Evolutionary Pressures in LCS

Several studies [33, 3] show experimentally that both XCS and UCS tend to evolve rulesets which are *complete, consistent* and *minimal representations* of the target concept. This behavior has been supported theoretically [6] by the interaction of two types of evolutionary pressures: the accuracy pressure, which moves the search towards accurate rules, and the generalization pressure, which guides the search towards the most general representations. Briefly summarizing, theoretical studies show that basing fitness on accuracy results in a pressure towards the specificity of maximally general classifiers. Also mutation results in a pressure towards specificity. The fact that the GA is applied in niches, while deletion is done over the whole population, tends to make rules more general. The global interaction of all these components favors the evolution of compact rulesets consisting of accurate and maximally general rules.

4 Facetwise Analysis of Learning Classifier Systems

Goldberg emphasizes the big importance of the *design decomposition* and *facetwise analysis* for the understanding of *complex systems*, which permit a more *effective* and *efficient* design to solve bounded difficult problems *quickly*, *accurately*, and *reliably* [11]. This approach has been closely followed to understand the impact that class imbalances cause in the different mechanisms of XCS and propose new approaches that overcome the detected drawbacks [22, 23]. Specifically, facetwise models have been developed that predict (i) the maximum class imbalance until which XCS would not overgeneralize toward the majority class, and (ii) the minimum population size that permits enough diversity of rules of the minority class to let the genetic pressures take off. In the following, the theoretical analysis is rewritten to be valid for both LCSs, and an algorithm is proposed to let both LCSs self-adapt depending on the imbalance level detected during learning.

Imbalance bound to prevent overgeneralization. In [22], a bound on the maximum imbalance ratio allowed in XCS is derived. The imbalance ratio is defined as the fraction between the number of instances of the majority class and the minority class. The bound defines the maximum imbalance ratio with which XCS can deal without overgeneralizing toward the majority class:

$$ir \leq \frac{2R_{max}}{\epsilon_0} \quad (3)$$

where R_{max} is the maximum reward that the system can receive (in classification tasks, $R_{max} = 1000$), and ϵ_0 is the maximum error that a rule can have to be considered accurate (usually, $\epsilon_0 = 1$). Without loss of generality, this bound can be extended for UCS by recognizing that $\epsilon_0 = 1 - acc_0$. If the inequality of equation 3 holds, it guarantees that neither XCS nor UCS will overgeneralize toward the majority class. Moreover, the learning rate β and θ_{GA}, which controls the frequency of activation of the GA, were detected as two critical parameters that need to be configured properly to satisfy the imbalance bound.

Population size bound. Next, in [23] a bound was derived on the minimum population size required to guarantee that XCS would initially be supplied with enough rules, and so, the genetic search would pressure toward the discovery of the minority class. The same bound is valid for UCS, which can be written as follows:

$$N = O\left[n \cdot (1 + ir)\right] \quad (4)$$

in which n is the number of classes of the problem and ir the imbalance ratio. This bound shows up the robustness of XCS and UCS when dealing with imbalances, indicating that the population size only needs to increase linearly with the imbalance ratio to ensure the discovery of the minority class.

Online adaptation algorithm. Both bounds were individually validated using artificial problems. The *patch-quilt integration* of them resulted in a theory providing guidelines on how to set the critical parameters of both LCSs. For a fixed population size, β and θ_{GA} should be configured according to the imbalance ratio between big niches and small niches[4] that lay closely on the feature space (ir_n). Nonetheless, ir_n is unknown for real-world problems, and can hardly be estimated before running LCSs. Thus, we propose an algorithm that estimates

[4] Note that, in LCSs terms, a *disjunct* equals to a *niche*. Thus ir_n reflects the imbalance ratio between big and small disjuncts.

Algorithm 4.1: Pseudocode for the *online adaptation algorithm.*

```
1 Algorithm: OnlineAdaptation ( cl is classifier )
2 if cl is overgeneral then
3 |   ir_n := exp_maj(cl) / (exp_maj(cl) + exp_min(cl))
4 |   if ( ir_n < 2R_max/ε_0 ∧ num_cl > num_[P] ) then
5 |   |   Adapt β and θ_GA based on ir_n
6 |   end
7 end
```

ir_n from information that intrinsically resides in overgeneral classifiers. Overgeneral classifiers cover several niches that lay nearby on the feature space. By computing the number of examples covered per class of an overgeneral classifier, we can estimate the imbalance ratio between these niches. Note that this strategy permits not only to detect *small disjuncts*, but also to calculate an estimate of the imbalance ratio between these small disjuncts and their neighbors.

The online adaptation algorithm works as follows (see the pseudocode in Alg. 4.1). After checking that the classifier is overgeneral, it estimates ir_n from the number of instances covered per class (labeled as *exp* in the algorithm). Next, if ir_n satisfies the imbalance bound (see formula 3), and the classifier is numerous enough ($num_{cl} > \overline{num}_{[P]}$), β and θ_{GA} are updated according to the formulas presented in [22]. Next section analyzes the behavior of XCS and UCS with the online adaptation algorithm on highly imbalanced datasets.

5 LCSs in Artificial Domains

This section explores the competence of XCS and UCS with online adaptation of parameters to discover cases that are infrequently sampled. For this purpose, we use the *imbalanced multiplexers*, a family of problems of *bounded difficulty* that permits to control separately the concept complexity and the imbalance complexity. The *multiplexer* [32] is one of the most used benchmarks in the LCS field. By using the multiplexer problem, we enable replication of studies on standard XCS and allow comparison with previous results.

5.1 The Imbalanced Multiplexer

The multiplexer is defined for binary strings of size ℓ, where the first $\log_2 \ell$ bits are the *address bits*, and the remaining bits are the *position bits*. Then, the output is the value of the position bit referred by the decimal value of the address bits. For example, in the 6-bit multiplexer (i.e., $\ell = 6$), f(00 1001) = 1 or f(10 0101) = 0. The concept complexity of the multiplexer is controlled by the input length ℓ. To obtain the correct classification model, learners need to discover the linkages between the address bits and the position bits, which increase exponentially with ℓ. For this reason, multiplexers pose a big challenge to many well-known learners [3], specially as ℓ increases.

In the *imbalanced multiplexer* [22], the imbalance complexity is controlled by undersampling instances of the class labeled as '1'. That is, when required, a new input example is selected randomly. If the example belongs to the class '0', it is given to the system. Otherwise, it is accepted with a certain probability. In the remainder of the paper, we use the imbalance ratio *ir*—that is, the ratio between the number of instances of class '0' (majority class) and class '1' (the minority class)—to refer to the imbalance complexity.

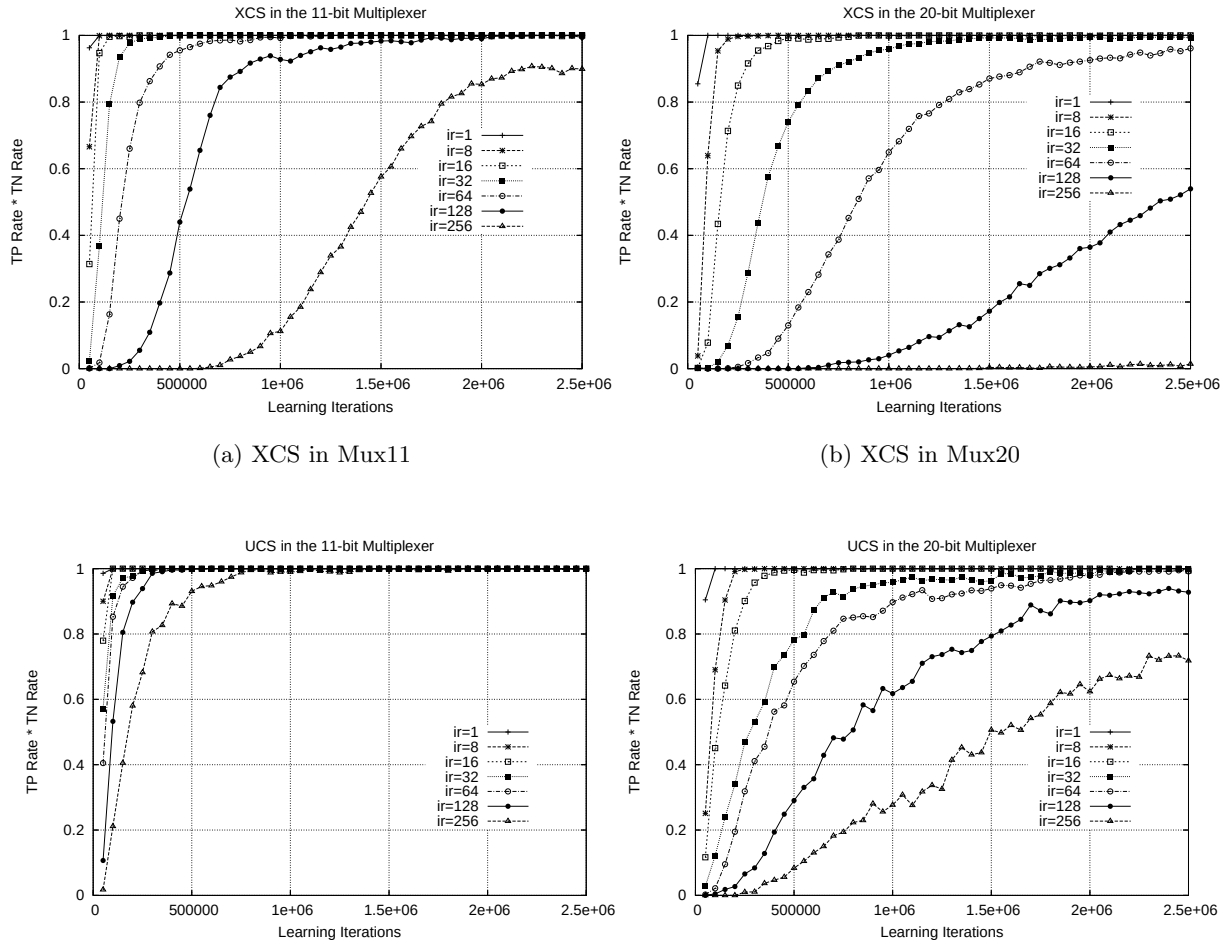
5.2 Experimentation

In [22], XCS was shown to be sensitive to moderate imbalance ratios; particularly, XCS could discover the minority class for imbalance ratios up to ir=32 in the 11-bit multiplexer. To analyze the improvement introduced by the online adaptation algorithm, we ran XCS and UCS[5] in the 11-bit and 20-bit multiplexers and imbalance ratios from $ir = 1$ (completely balanced dataset) to $ir = 256$. Populations were sized to N={800, 2000} for XCS and to N={400, 1000} for UCS in the 11-bit and the 20-bit multiplexer respectively. As UCS works under a supervised learning scheme, and so, does not need to explore all the classes in the feature space, we configured smaller population sizes for UCS as suggested in [3].

As a metric of performance for imbalanced datasets, the average accuracy rate is biased toward the majority class. Instead, we measured the performance with the proportion of instances of the minority class correctly classified (TP rate) and the proportion of instances of the majority class correctly classified (TN rate). Figure 1 shows the product of TP rate and TN rate of XCS and UCS averaged over 10 runs. Note that the graph shows the incremental improvement of both systems over the training iterations, where a training iteration corresponds to sampling a single example of the dataset.

In the 11-bit multiplexer, we note that XCS and UCS need more learning iterations to achieve 100% performance as *ir* increases (see Figs. 1(a) and 1(c)). Specifically, the TN rate needs only about 5,000 iterations to reach 100% in all runs. Thus, all the error is concentrated on the prediction of the minority class. This is because minority class instances are sampled less frequently, and so, accurate rules of the minority class receive a smaller number of genetic events. Note that the self-adaptive algorithm allows LCSs to discover the minority class for

[5] To allow replicability, XCS's parameters were configured with the standard values typically used in the literature: α=0.1, $\epsilon_0 = 1$, ν=5, θ_{GA}=25, χ=0.8, μ=0.04, θ_{del}=20, δ=0.1, θ_{sub}=200, $P_{\#}$=0.8. For UCS, the same parameters were used but: ν=10 and acc_0=0.99. See [33, 3] for notation details.



(a) XCS in Mux11

(b) XCS in Mux20

(c) UCS in Mux11

(d) UCS in Mux20

Fig. 1 Incremental TP rate of XCS and UCS in the 11-bit and 20-bit multiplexers for imbalance ratios ranging from $ir = 1$ to $ir = 256$.

high imbalance ratios, while previous results in [22] indicated that XCS only learnt with ratios up to $ir = 32$. Results also illustrate that UCS converges more quickly than XCS, specially for the highest ir. In fact, as UCS is specialized for classification tasks, its convergence time was expected to be lower than XCS's time.

Figures 1(b) and 1(d) show the behavior of XCS and UCS on the 20-bit multiplexer. Results show that, with a higher concept complexity, both LCSs need more learning iterations to solve an experiment with the same ir as before. Again, we observe that (i) the convergence time is higher as ir increases and (ii) UCS needs lower convergence time than XCS.

6 LCSs in Data Mining

This section analyzes the performance of XCS and UCS in various real-world imbalanced problems. The understanding of LCSs behavior on real-world problems is really complicated since they may have different sources of complexity which can be hardly identified; the interaction of all these complexities may limit the maximum performance that can be achieved. To evaluate the competence of XCS and UCS, we compare their performance to three highly-competent learners. In the following, we first present the methodology and then, we compare XCS and UCS with the other learners.

6.1 Methodology

We used a collection of 25 real-world problems with different characteristics and imbalance ratios, which were constructed as follows. We selected the following twelve problems: *balance-scale*, *bupa*, *glass*, *heart disease*, *pima indian diabetes*, *tao*, *thyroid disease*, *waveform*, *Wisconsin breast cancer database*, *Wisconsin diagnostic breast cancer*, *wine recognition data*, and *Wisconsin pronostic breast cancer*. All the real-world problems were obtained from the UCI repository [5], except for *tao*, which was selected from a local repository [3]. To force higher imbalance ratios, we discriminated each pair of classes in

Table 1 Description of the datasets properties. The columns describe the dataset identifier (Id.), the original name of the dataset (Dataset), the number of problem instances (#Ins.), the number of attributes (#At.), the proportion of minority class instances (%Min.), the proportion of majority class instances (%Maj.), and the imbalance ratio (ir).

Id.	Dataset	#Ins.	#At.	%Min.	%Maj.	ir
bald1	balance-scale disc. 1	625	4	7.84%	92.16%	11.76
bald2	balance-scale disc. 2	625	4	46.08%	53.92%	1.17
bald3	balance-scale disc. 3	625	4	46.08%	53.92%	1.17
bpa	bupa	345	6	42.03%	57.97%	1.38
glsd1	glass disc. 1	214	9	4.21%	95.79%	22.75
glsd2	glass disc. 2	214	9	6.07%	93.93%	15.47
glsd3	glass disc. 3	214	9	7.94%	92.06%	11,59
glsd4	glass disc. 4	214	9	13.55%	86.45%	6.38
glsd5	glass disc. 5	214	9	32.71%	67.29%	2.06
glsd6	glass disc. 6	214	9	35.51%	64.49%	1.82
h-s	heart-disease	270	13	44.44%	55.56%	1.25
pim	pima-inidan	768	8	34.90%	65.10%	1.87
tao	tao-grid	1888	2	50.00%	50.00%	1.00
thyd1	thyroid disc. 1	215	5	13.95%	86.05%	6.17
thyd2	thyroid disc. 2	215	5	16.28%	83.72%	5.14
thyd3	thyroid disc. 3	215	5	30.23%	69.77%	2.31
wavd1	waveform disc. 1	5000	40	33.06%	66.94%	2.02
wavd2	waveform disc. 2	5000	40	33.84%	66.16%	1.96
wavd3	waveform disc. 3	5000	40	33.10%	66.90%	2.02
wbcd	Wis. breast cancer	699	9	34.48%	65.52%	1.90
wdbc	Wis. diag. breast cancer	569	30	37.26%	62.74%	1.68
wined1	wine disc. 1	178	13	26.97%	73.03%	2.71
wined2	wine disc. 2	178	13	33.15%	66.85%	2.02
wined3	wine disc. 3	178	13	39.89%	60.11%	1.51
wpbc	wine disc. 4	198	33	23.74%	76.26%	3.21

each dataset, considering each discrimination as a new problem. Thus, $\binom{n}{2}$ two-class problems were created from a problem with n classes, resulting in a testbed that consisted of 25 two-class real-world problems. Table 1 gathers the most relevant features of the problems. Note that the imbalance ratio between niches ir_n can be much higher than the imbalance ratio of the learning dataset reported in the table.

The performance was measured by the product of TP rate and TN rate. To have good estimates, we ran the experiments on a ten-fold cross validation [27]. We used a multiple comparison Friedman procedure [19, 20] to test whether all the learning algorithms performed equivalently on average. Moreover, the performance of each pair of learning algorithms on each problem was compared using a Wilcoxon signed-ranks test [31].

Both LCSs were compared to three of the most competent learners: C4.5 [26], SMO [25], and IBk [1]. C4.5 is a decision tree derived from the ID3 algorithm. SMO is a support vector machine that implements the *Sequential Minimal Optimization algorithm*. IBk is a nearest neighbor algorithm. All these machine learning methods were run using WEKA [34], and the recommended default parameters were used. We selected the model for SMO as follows. We ran SMO with polynomial kernels of order 1, 5, and 10, and with Gaussian kernels. We first discarded SMO with Gaussian kernels since it achieved 0% performance in the majority of problems as it misclassified all the instances of the minority class. Then, we ranked the results obtained with the three polynomial kernels, and chose the model that maximized the average ranking: SMO with lineal kernels. In this way we avoid using particular configurations for each problem. We followed the same process with IBk, and provide the results with k=5. XCS and UCS were configured as previously specified, except for N=6400, r_0=0.6, and m_0=0.1. Finally, we did not introduce asymmetric cost functions in any system, although the majority of them permit it. In this way, we aim at analyzing the intrinsic capabilities of each method to deal with class imbalances.

6.2 Results

Table 2 summarizes the performance of the different learners on the 25 datasets. The overall results highlight which problems are more complex. All learners presented poor performance in the problems *bald1*, *bpa*, *glsd1*, *glsd3*, *pim*, and *wpbc*. Examining the measure of performance, we observed that all learners had a low TP rate, which indicates that the minority class is not well defined in these problems. Most of these datasets are highly imbalanced; so, the imbalance ratio turns up to be an important factor that hinders the performance of the tested learners. Nonetheless, the problems *bpa* and *pim*

Table 2 Comparison of C4.5, SMO, IBk, XCS and UCS on the 25 real-world problems. Each cells depicts the average value of TP rate * TN rate and the standard deviation. The row labeled *Avg* gives the performance average (and standard deviation) of each method over the 25 datasets.

	C4.5	**SMO**	**IB5**	**XCS**	**UCS**
bald1	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
bald2	69.28 ± 7.68	83.98 ± 7.30	81.16 ± 5.54	71.22 ± 5.02	69.77 ± 8.19
bald3	71.21 ± 5.80	85.69 ± 8.40	82.11 ± 8.67	70.07 ± 7.23	73.65 ± 6.66
bpa	33.50 ± 10.30	0.00 ± 0.00	32.40 ± 9.44	47.22 ± 10.92	47.21 ± 11.22
glsd1	79.60 ± 41.93	0.00 ± 0.00	69.32 ± 48.30	20.00 ± 42.16	59.11 ± 50.87
glsd2	33.95 ± 46.69	15.00 ± 33.75	24.13 ± 35.36	59.40 ± 45.02	74.25 ± 41.89
glsd3	28.78 ± 41.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	19.39 ± 25.17
glsd4	73.36 ± 32.31	80.33 ± 24.33	77.07 ± 24.98	80.33 ± 24.33	83.61 ± 19.53
glsd5	65.35 ± 20.36	9.58 ± 9.42	62.26 ± 21.14	67.82 ± 18.71	64.45 ± 21.46
glsd6	52.03 ± 17.13	0.00 ± 0.00	61.74 ± 18.23	61.08 ± 11.21	57.90 ± 14.20
h-s	63.70 ± 11.02	68.80 ± 8.87	64.40 ± 14.65	60.32 ± 15.59	54.87 ± 13.61
pim	44.96 ± 5.77	48.36 ± 5.60	46.91 ± 4.84	46.06 ± 6.37	47.88 ± 6.60
tao	91.00 ± 2.37	70.57 ± 6.45	94.25 ± 2.10	82.90 ± 5.42	78.79 ± 7.18
thyd1	87.53 ± 16.53	76.67 ± 22.50	76.67 ± 22.50	78.69 ± 22.01	92.32 ± 13.66
thyd2	93.12 ± 13.21	54.17 ± 24.92	77.90 ± 21.40	82.50 ± 24.98	93.12 ± 12.09
thyd3	87.31 ± 13.18	33.81 ± 21.35	81.12 ± 16.16	89.74 ± 11.75	87.97 ± 14.89
wavd1	67.80 ± 3.82	78.65 ± 4.27	72.28 ± 3.97	80.43 ± 2.97	76.35 ± 2.10
wavd2	62.54 ± 3.53	72.35 ± 2.71	67.49 ± 1.75	73.48 ± 2.88	71.50 ± 3.83
wavd3	68.61 ± 2.18	79.61 ± 2.04	74.14 ± 2.86	81.01 ± 3.99	76.62 ± 4.14
wbcd	89.10 ± 4.57	92.72 ± 5.32	92.72 ± 5.36	92.29 ± 5.50	94.11 ± 4.23
wdbc	88.83 ± 4.98	94.27 ± 3.28	93.47 ± 3.64	90.30 ± 4.61	89.67 ± 5.61
wined1	85.58 ± 14.57	98.46 ± 3.24	94.98 ± 8.29	99.23 ± 2.43	99.23 ± 2.43
wined2	91.83 ± 8.50	97.51 ± 5.62	97.50 ± 4.03	99.17 ± 2.64	91.76 ± 10.02
wined3	87.64 ± 11.83	97.14 ± 6.02	87.94 ± 12.53	93.43 ± 7.15	85.36 ± 9.55
wpbc	33.96 ± 11.01	9.37 ± 16.98	28.98 ± 16.49	20.99 ± 16.38	16.97 ± 21.63
Avg	66.02 ± 14.01	53.88 ± 8.90	65.64 ± 12.49	65.91 ± 11.97	68.23 ± 13.23

are almost balanced, so there may be other complexity factors affecting the learning performance such as small disjuncts.

The Friedman multiple comparison test did not permit to reject the null hypothesis that all the learning methods performed the same on average with $p = 0.2519$. Consequently, post-hoc tests could not be applied since no significant differences between the multiple learners were found [10]. This result is not surprising; in fact, in general terms, the no-free-lunch theorem [35, 36] justifies that no learning algorithm can systematically outperform the others. However, we are interested in methods that are robust in a wide range of problems. To analyze that, we applied statistical pairwise comparisons according to a Wilcoxon signed-ranks test at 0.95 significance level. Table 3 shows the results. The • and ◦ symbols denote a significant degradation/improvement of the given learning algorithm with respect to another in a particular dataset.

The overall degradation-improvement comparison (see the row labeled *Score*) permits to rank the quality of the five learners. Under this criteria, XCS appears as the most robust method with a ratio of degradation-improvement of 8-20, followed closely by IBk and UCS. Both LCSs show the poorest results with respect to the other learners in the problems *bald*2, *bald*3, and *tao*, which have a low imbalance ratio. In [4], it is shown that the hyperrectangle codification used by XCS and UCS is not adequate when the boundary between classes in the learning dataset is curved. This is the case of the *tao* problem [4]. We hypothesize that *bald*2 and *bald*3 are also characterized by curved boundaries, which would explain the degradation in performance of both LCSs. This hypothesis is also supported by the results obtained with IBk, which improves XCS and UCS in the three problems mentioned. IBk is not affected by curved boundaries since it decides the output as the majority class of the k nearest neighbors.

The two last methods in the ranking are C4.5 and SMO. The surprisingly poor rank of C4.5 is mainly caused by the results obtained in the problems *wavd*1, *wavd*2, and *wavd*3, in which C4.5 is outperformed by all the other learners. These results are not correlated with the imbalance ratio, so there may be other types of complexity that makes C4.5 perform poorly in these problems. Finally, SMO is the last method in the rank. It shows a tendency to overgeneralize toward the majority class in problems with moderate and high class imbalances such as *glsd*1, *glsd*3, and *glsd*6, in which the TP rate is zero. The same behavior is shown in problems with low imbalance ratios such as the *bpa* problem, which we identified as a difficult problem perhaps due to small disjuncts.

Table 3 Comparison of C4.5, SMO, IBk, XCS and UCS on the 25 real-world problems. For a given problem, the • and ○ symbols indicate that the learning algorithm of the column performed significantly worse/better than another algorithm at 0.95 significance level (pairwise Wilcoxon signed-ranks test). *Score* counts the number of times that a method performed worse-better, and $Score_{ir>5}$ does the same but only for the highest imbalanced problems ($ir > 5$).

	C4.5	**SMO**	**IBk**	**XCS**	**UCS**
bald1					
bald2	••	○ ○ ○	○ ○ ○	••	••
bald3	••	○ ○ ○	○ ○ ○	••	••
bpa	• • ○	• • ••	• • ○	○ ○ ○	○ ○ ○
glsd1	○○	• • •	○	•	○
glsd2		••	•	○	○○
glsd3					
glsd4					
glsd5	○	• • ••	○	○	○
glsd6	○	• • ••	○	○	○
h-s		○	○		••
pim					
tao	• ○ ○○	• • ••	○ ○ ○○	• • ○○	• • •○
thyd1					
thyd2	○	• • ••	○	○	○
thyd3	○	• • ••	○	○	○
wavd1	• • ••	○○	• • •○	○ ○ ○	• ○ ○
wavd2	• • ••	○○	• • •○	○○	○○
wavd3	• • ••	○○	• • ○	○ ○ ○	•○
wbcd	• • •	○	○		○
wdbc	•	○○	○	•	•
wined1	• • •	○		○	○
wined2					
wined3		○		○	••
wpbc					
Score	26-10	29-18	11-22	8-20	14-18
Score$_{ir>5}$	0-3	9-0	1-2	1-2	0-4

However, we can also find significant improvements with respect to other learners in the problems: *bald*2, *bald*3 and *wdbc*. Thus, these results indicate that SMO performs competitively in a restricted set of problems, but it is affected by some complexities, among which we may find the imbalance ratio.

Finally, let's compare the learners in terms of imbalance robustness. To do that, we consider the most imbalanced problems: *glsd1*, *glsd2*, *bald1*, *glsd3*, *glsd4*, *thyd1*, and *thyd2*, which have imbalance ratios ranging from ir=5 to ir=23. In these problems, UCS appears to be the best learner, with a degradation-improvement ratio of 0-4, followed closely by C4.5. These results agree with several papers which indicate that C4.5 can deal with high amounts of class imbalance [15, 2]. IBk and XCS are the two next methods in the rank. IBk may suffer from *small disjuncts*, since minority class regions are surrounded by many instances of the majority class, concentrating a high amount of the test error around the small disjuncts. XCS also appears to be more sensible to class imbalances than UCS and C4.5. This confirms the results observed in section 4, which indicate that XCS is less robust than UCS in problems with the highest imbalance ratios. Finally, SMO performs poorly in the most imbalanced datasets. As mentioned above, we tried other orders of polynomial kernels, as well as a Gaussian kernel, but no significant improvement was found.

7 Resampling the Training Datasets

Resampling techniques have been said to boost the performance of several learners on imbalanced datasets [9, 15]. They are based on balancing the proportion of instances per class in the training dataset, by either oversampling instances of the minority class or undersampling instances of the majority class. In this section, we aim at analyzing if resampling techniques also improve the performance of LCSs and which of them are preferred.

7.1 Methodology and Resampling Techniques

In our analysis, we chose four resampling techniques which have demonstrated to be highly competitive in reduced testbeds:

Random oversampling. This is a non-heuristic method that replicates the instances of the minority class until

Table 4 Intra-method ranking for original and rebalanced datasets for C4.5, SMO, IBk, XCS, and UCS. Rows 1*st* to 5*th* indicate the number of times that each resampling technique has been ranked in the correspondent position. The last column shows the average rank and its standard deviation.

	Resamp. Method	1st	2nd	3rd	4th	5th	Avg. ± Std.
C4.5	*original*	6	2	5	**9**	3	*3.04 ± 1.87*
	oversampling	7	4	**8**	4	2	*2.60 ± 1.60*
	undersampling TL	0	5	7	6	7	*3.60 ± 1.20*
	smote	**10**	**8**	3	2	2	***2.12 ± 1.54***
	csmote	2	6	2	4	**11**	*3.64 ± 2.07*
SMO	*original*	6	2	2	4	**11**	*3.48 ± 2.73*
	oversampling	**11**	**11**	3	0	0	***1.68 ± 0.46***
	undersampling TL	2	8	**9**	3	3	*2.88 ± 1.23*
	smote	3	3	8	7	4	*3.24 ± 1.46*
	csmote	3	1	3	**11**	7	*3.72 ± 1.56*
IBk	*original*	6	6	2	6	5	*2.92 ± 2.23*
	oversampling	4	**8**	**11**	1	1	***2.48 ± 0.89***
	undersampling TL	4	2	5	4	**10**	*3.56 ± 2.17*
	smote	**10**	4	2	**7**	2	***2.48 ± 2.09***
	csmote	1	5	5	**7**	7	*3.56 ± 1.45*
XCS	*original*	3	5	2	**6**	**9**	*3.52 ± 2.09*
	oversampling	7	5	4	1	8	*2.92 ± 2.63*
	undersampling TL	1	**8**	**10**	**6**	0	*2.84 ± 0.69*
	smote	**11**	3	2	**6**	3	***2.48 ± 2.33***
	csmote	3	4	7	**6**	5	*3.24 ± 1.62*
UCS	*original*	2	4	**8**	5	6	*3.36 ± 1.51*
	oversampling	6	5	5	**7**	2	*2.76 ± 1.70*
	undersampling TL	5	4	7	**7**	2	*2.88 ± 1.55*
	smote	**7**	**11**	4	1	2	***2.20 ± 1.28***
	csmote	5	1	1	5	**13**	*3.80 ± 2.48*

there is the same proportion of instances per class in the training dataset. Some authors have suggested that oversampling may produce a problem of overfitting, since it makes exact copies of the minority class. Nevertheless, this method has shown to perform competitively in many comparisons [15, 9].

Undersampling based on Tomek Links. This method consists in eliminating instances of the majority class that do not belong to any *Tomek Link* [28] until the dataset is balanced. A *Tomek Link* is a pair of instances (I_i, I_j) that lay on the class boundary.

SMOTE. The *Synthetic Minority Over-sampling TEchnique* [9] is an oversampling method that creates new minority class instances by interpolating several minority class examples that lay nearby in the feature space. It is said that this method avoids overfitting by creating rather than replying instances of the minority class.

CSMOTE. The *Clustered SMOTE* [21] is an oversampling method that derives from SMOTE, but introduces two modifications. First, new instances of the minority class are generated from minority class examples that belong to the same cluster. Second, it introduces a cleaning phase that removes all instances whose n neighbors belong to the same class.

We applied each resampling algorithm on the 10 folds of each dataset, obtaining 100 new problems, and ran C4.5, SMO, IBk, XCS, and UCS on these datasets. Learners were configured as specified in section 6.1.

7.2 Results

We analyzed the performance of each resampling technique and each learner (the complete tables are not shown for brevity). The multiple comparison Friedman test did not permit to reject the hypothesis that all resampling methods performed the same on average. However, significant improvements were shown in particular problems by using a pairwise t-test. To summarize the results, Table 4 ranks the performance obtained with the original and the resampled datasets for each learner. For each classifier, the resampling method that places first is marked in bold. The last column provides the average rank and the standard deviation for each resampling method.

The results show that, in general, resampling the training datasets yields better performance of the learners. On average, the best results are achieved with *random oversampling* and *SMOTE*. The empirical observations agree with some studies concluding that oversampling is more effective than undersampling in C4.5 [15, 2] and SMO [15]. The results obtained herein allow us to extend this conclusion to IBk, XCS, and UCS. We hypothesize that undersampling may cause a problem of sparsity as it removes instances that may be needed for learning.

In fact, undersampling is better ranked in the problems *pim*, *wavd*1, *wavd*2, and *wavd*3, which have the highest number of instances per dimension[6], and poorly ranked in the problems with the lowest number of instances per dimension: *wdbc*, *wined1*, *wined2*, *wined3*, and *wpbc*.

The standard deviation of the ranking somehow denote the dependency of each resampling method on the characteristics of the training domain. For C4.5, SMOTE is the best ranked resampling method with a low deviation. In most of the cases, SMOTE is the first or the second problem in the ranking. These results indicate that SMOTE should be used in combination with C4.5 to deal with class imbalances. For SMO and IBk, oversampling is the best ranked method and, at the same time, it shows a very low standard deviation. Consequently, SMO and IBk should be combined with random oversampling in imbalanced domains. Note that, for IBk, oversampling and SMOTE have the same average rank. However, SMOTE has a much higher standard deviation, which indicates that its behavior highly depends on the domain. For XCS, the best ranked resampling method, i.e., SMOTE, has one of the highest standard deviations. Thus, the behavior of this combination depends on the characteristics of the data. In this case, it should be more adequate to combine XCS with undersampling based on Tomek Links, since it has the second best average rank and a very low standard deviation. For UCS, the best and the most robust resampling method is SMOTE.

Finally, let's note that, in some cases, the best results are achieved with the original dataset. For example, a detailed inspection (not shown for brevity) revealed that the performance of many of the learners worsens when the datasets are resampled. This happens in *h-s*, *tao*, *wined1*, *wined2*, and *wined3*. This indicates that resampling the training instances may introduce other complexities, or even may create new small disjuncts around the feature space.

8 Summary and Conclusions

This paper showed that *evolutionary online rule-based systems*, usually called *Learning Classifier Systems*, can successfully deal with the challenges posed by learning from imbalances, mainly related to the disproportion of instances per class in the training dataset and to the need of learners to create *small disjuncts* (or *niches* in LCSs terms) in the knowledge model. Theoretical analyses indicated that XCS and UCS are robust to high imbalance ratios if some critical parameters are configured according to the ratio of the size between big and small disjuncts ir_n. As ir_n is not known a priori, and can hardly be estimated, we proposed a self-adaptive method that estimates ir_n online and lets LCSs adapt themselves so that accurate small disjuncts can be evolved for infrequent cases. Results on artificially imbalanced problems supported the theoretical analyses, demonstrating that both LCSs can model infrequent cases and classes.

In real-world problems, LCSs were among the best performers, compared with instance-based learners, induction trees and support-vector machines. Although the set of real-world problems used in the experiments did not contain high imbalance ratios, there is uncertainty about whether they contained small disjuncts or other mixed complexity factors. This is a common problem when we test the algorithms in real-world problems. Our proposal as a further work is to study measures that evaluate the presence of small disjuncts in the feature space and try to relate the algorithms' performance to such complexities. This would probably allow us to understand in which cases each algorithm is superior and provide guidelines towards the selection of particular algorithms given a dataset characterization.

Although the learners may be robust to class imbalances, resampling techniques usually favor better accuracy rates. In general, oversampling techniques were preferred over undersampling. Nevertheless, none of the resampling techniques systematically outperformed the others and, for a particular dataset, the best resampling method depended on the learner. In fact, resampling methodologies change the geometry of the dataset. Thus, to justify such dependencies, we need to seek for the geometrical characterization of the original dataset, and analyze the changes introduced by the different resampling techniques. Once we showed that LCSs are highly competitive methods for dealing with imbalances, our future work is to continue investigating on the imbalance characterization of real-world datasets, which can lead us to provide guidelines for resampling and learner selection.

Acknowledgements The authors are grateful to the three anonymous reviewers for their comments on earlier darfts of this paper. The authors thank the support of *Enginyeria i Arquitectura La Salle*, Ramon Llull University, as well as the support of *Ministerio de Ciencia y Tecnología* under project TIN2005-08386-C05-04, and *Generalitat de Catalunya* under Grants 2005FI-00252 and 2005SGR-00302.

References

1. D.W. Aha, D.F. Kibler, and M.K. Albert. Instance-Based Learning Algorithms. *Machine Learning*, 6(1):37–66, 1991.
2. G. Batista, R.C. Prati, and M.C. Monrad. A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data. *SIGKDD Explor. Newsl.*, 6(1):20–29, 2004.
3. E. Bernadó-Mansilla and J.M. Garrell. Accuracy-Based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks. *Evolutionary Computation*, 11(3):209–238, 2003.

[6] The ratio between the number of instances and the number of attributes of a problem has been proposed elsewhere [4] as a measure of sparsity.

4. E. Bernadó-Mansilla and T.K. Ho. Domain of Competence of XCS Classifier System in Complexity Measurement Space. *IEEE-TEC*, 9(1):1–23, 2005.
5. C.L Blake and C.J. Merz. *UCI Repository of machine learning databases: http://www.ics.uc.edu/ mlearn/MLRepository.html*. Univ. of California, 1998.
6. M.V. Butz. *Rule-Based Evolutionary Online Learning Systems: A Principled Approach to LCS Analysis and Design*, volume 109 of *Studies in Fuzziness and Soft Computing*. Springer, 2006.
7. Butz, M.V. and Wilson, S.W. An algorithmic description of XCS. In P.L. Lanzi, W. Stolzmann, and S.W. Wilson, editors, *Advances in Learning Classifier Systems: Proceedings of the Third International Workshop*, volume 1996 of *Lecture Notes in Artificial Intelligence*, pages 253–272. Springer, 2001.
8. D.R. Carvalho and A.A. Freitas. A Hybrid Decision Tree/Genetic Algorithm for Coping with the Problem of Small Disjuncts in Data Mining. In *GECCO'00*, pages 1061–1068. Morgan Kaufmann, 10-12 2000.
9. N.V. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.
10. J. Demsar. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research*, 7:1–30, 2006.
11. D.E. Goldberg. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Kluwer Academic Publishers, 1 edition, 2002.
12. J.H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.
13. R.C. Holte, L.E. Acker, and B.W. Porter. Concept Learning and the Problem of Small Disjuncts. In *IJCAI'89*, pages 813–818, 1989.
14. N. Japkowicz and S. Stephen. The Class Imbalance Problem: Significance and Strategies. In *IC-AI'00*, volume 1, pages 111–117, 2000.
15. N. Japkowicz and S. Stephen. The Class Imbalance Problem: A Systematic Study. *Intelligent Data Analisis*, 6(5):429–450, November 2002.
16. T. Jo and N. Japkowicz. Class imbalances versus small disjuncts. *SIGKDD Explorations*, 6(1):40–49, 2004.
17. T. Jo and N. Japkowicz. Class imbalances versus small disjuncts. *SIGKDD Explorations*, 6(1):40–49, 2004.
18. T. Kovacs. Deletion Schemes for Classifier Systems. In *GECCO'99*, pages 329–336. Morgan Kaufmann, 1999.
19. M. Friedman. The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. *Journal of the American Statistical Association*, 32:675–701, 1937.
20. M. Friedman. A Comparison of Alternative Tests of Significance for the Problem of m Rankings. *Annals of Mathematical Statistics*, 11:86–92, 1940.
21. A. Orriols-Puig. Facetwise Analysis of Learning Classifier Systems in Imbalanced Domains. Technical report, Ramon Llull University, 2006.
22. A. Orriols-Puig and E. Bernadó-Mansilla. Bounding XCS Parameters for Unbalanced Datasets. In *GECCO '06*, pages 1561–1568. ACM Press, 2006.
23. A. Orriols-Puig and E. Bernadó-Mansilla. Modeling XCS in Class Imbalances: Population Size and Parameters' Settings. In *GECCO'07*, 2007 (submitted).
24. A. Orriols-Puig and E. Bernadó-Mansilla. A Further Look at UCS Classifier System. In *Advances at the frontier of LCS*. Springer, Accepted.
25. J. Platt. Fast Training of Support Vector Machines using Sequential Minimal Opt. In *Adv. in Kernel Methods - Support Vector Lear.* MIT Press, 1998.
26. J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, California, 1995.
27. T.G. Dietterich. Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms. *Neural Comp.*, 10(7):1895–1924, 1998.
28. I. Tomek. Two Modifications of CNN. *IEEE Transactions on Systems, Man and Cybernetics*, 6:769–772, 1976.
29. G.M. Weiss. *The Effect of Small Disjuncts and Class Distribution on Decision Tree Learning*. PhD thesis, Graduate School New Brunswick - The State University of New Jersey, New Brunswick, New Jersey, 2003.
30. G.M. Weiss. Mining with rarity: a unifying framework. *SIGKDD Explorations*, 6(1):7–19, 2004.
31. F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1:80–83, 1945.
32. S.W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.
33. S.W. Wilson. Generalization in the XCS Classifier System. In *3rd Annual Conf. on Genetic Programming*, pages 665–674. Morgan Kaufmann, 1998.
34. I.H Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2005.
35. D.H. Wolpert. Stacked Generalization. *Neural Networks*, 5(2):241–259, 1992.
36. D.H. Wolpert. The Lack of A Priori Distinctions Between Learning Algorithms. *Neural Computation*, 8(7):1341–1390, 1996.