The Class Imbalance Problem in UCS Classifier System: A Preliminary Study

Albert Orriols-Puig and Ester Bernadó-Mansilla

Enginyeria i Arquitectura La Salle
Universitat Ramon Llull
Quatre Camins, 2. 08022, Barcelona, Spain
{aorriols,esterb}@salleURL.edu

Abstract. The class imbalance problem has been said recently to hinder the performance of learning systems. In fact, many of them are designed with the assumption of well-balanced datasets. But this commitment is not always true, since it is very common to find higher presence of one of the classes in real classification problems. The aim of this paper is to make a preliminary analysis on the effect of the class imbalance problem in learning classifier systems. Particularly we focus our study on UCS, a supervised version of XCS classifier system. We analyze UCS's behavior on unbalanced datasets and find that UCS is sensitive to high levels of class imbalance. We study strategies for dealing with class imbalances, acting either at the sampling level or at the classifier system's level.

1 Introduction

Learning Classifiers Systems (LCSs) [11,12] are rule-based systems that have been demonstrated to be highly competitive in classification problems with respect to other machine learning methods. Nowadays, XCS [28,27], an evolutionary online learning system, is one of the best representatives of LCSs.

The performance of XCS on real classification problems has been tested extensively in many contributions [18,19,3,5,17]. In addition, some analyses on the factors that make a problem hard for XCS have been made [15], and some theories have been formulated [5]. This work focuses on one of the complexity factors which is said to hinder the performance of standard learning methods: the *class imbalance* problem.

The class imbalance problem corresponds to classification domains for which one class is represented by a larger number of instances than other classes. The problem is of great importance since it appears in a large number of real domains, such as fraud detection [9], text classification [6], and medical diagnosis [20]. Traditional machine learning approaches may be biased towards the majority class and thus, may predict poorly the minority class examples. Recently, the machine learning community have paid increasing attention to this problem and how it affects the learning performance of some well-known classifier systems such as *C5.0*, *MPL*, and *support vector machines* [22,23,10]. In the LCS's framework, some approaches have been proposed to deal with class imbalances

X. Llorà et al. (Eds.): IWLCS 2003-2005, LNAI 4399, pp. 161–180, 2007.

in epidemiological data [13]. The aim of this paper is to enhance the analysis on class imbalances into the LCS's framework, and debate whether this problem affects LCSs, to what degree, and, if it is necessary, study different methods to overcome the difficulties.

Our analysis is centered on Michigan-style learning classifier systems. We choose UCS [7] as the test classifier system for our analysis, with the expectation that our results and conclusions can also be extended to XCS and other similar LCSs. UCS is a version of XCS that learns under a supervised learning scheme. In order to isolate the class imbalance problem and control its degree of complexity, we designed two artificial domains. We study UCS's behavior on these problems and identify factors of complexity when the class imbalance is high, which makes us to analyze different approaches to deal with these difficulties.

The remainder of this paper is organized as follows. Section 2 describes the UCS classifier system, focusing on the differences with XCS. Section 3 gives the details on the domain generation. In section 4, UCS is trained in the designed problems, and the class imbalance effects are analyzed. Section 5 describes the main approaches for dealing with the class imbalance problem, and sections 6 and 7 analyze these approaches under UCS's framework. Finally, we summarize our main conclusions, give limitations of the current study, and provide directions for further work.

2 Description of UCS

UCS [18,7] is a Michigan-style classifier system derived from XCS [28,27]. The main difference is that UCS was designed under a supervised learning scheme, while XCS follows a reinforcement learning scheme. In the following, we give a brief description of UCS, emphasizing the main differences with XCS. For more details, the reader is referred to [7].

2.1 Representation

UCS evolves a population of [P] classifiers. Each classifier has a rule of type $condition \rightarrow class$, as in XCS, and a set of parameters estimating the quality of the rule.

The main parameters of a rule are: a) the rule's accuracy acc, b) the fitness F, c) the experience exp, d) the niche size ns, f) the last time of the GA activation ts, and g) the numerosity num.

2.2 Performance Component

UCS learns incrementally according to a supervised learning scheme. During *learning*, examples are provided to the system. Each example comes with its attributes $x = (x_1,... x_n)$ and its corresponding class c. Then, the system creates

a match set [M] consisting of those classifiers whose conditions match the input example. From [M], the classifiers that correctly predict the class c form the correct set [C]. The remaining classifiers belong to [!C]. If [C] is empty, the covering operator is activated, creating a new classifier with a generalized condition matching x and class c.

In *exploit* or *test* mode, an input x is presented and UCS must predict its associated class. In this case, the match set [M] is formed, and the system selects the best class from the vote (weighted by fitness) of all classifiers present in [M].

2.3 Parameter Updates

In learning mode, the classifier parameters are updated. First of all, the classifier's accuracy is updated:

$$acc = \frac{number\ of\ correct\ classifications}{experience}$$

Fitness is calculated as a function of accuracy:

$$F = (acc)^{\nu}$$

where ν is a parameter set by the user. A typical value is 10. Thus, accuracy accumulates the number of correct classifications that each classifier has done, and fitness scales exponentially with accuracy.

The experience of a classifier exp is updated every time a classifier participates in a match set. The niche set size ns stores the average number of classifiers in [C]; it is updated each time the classifier belongs to a correct set.

2.4 Discovery Component

In UCS, the genetic algorithm (GA) is used as the search mechanism in a similar way to that in XCS. The GA is applied to [C] instead of all the population. It selects two parents from [C] with a probability proportional to fitness and copies them. Then, the copies are recombined and mutated with probabilities χ and μ respectively. The resulting offspring are introduced into the population. First, each offspring is checked for subsumption. In an offspring can not be subsumed, it is inserted in the population, deleting potentially poor classifiers if the population is full. The deletion probability is computed in the same way as in XCS (see [14]).

3 Dataset Design

In order to isolate the class imbalance problem from other factors that affect UCS's performance [15], two artificial domains were generated. Each one tries to highlight different traits of the system. These are the checkerboard problem

(denoted as *chk*) and the position problem (denoted as *pos*). They are described in the following.

3.1 Chk Domain Generation

The *chk* problem is based on the balanced checkerboard problem, used as a benchmark in [8,26]. It has two real attributes (x and y) that can take values from zero to one ($x, y \in [0, 1]$). Instances are grouped in two non-overlapping classes, drawing a checkerboard in the feature space.

The complexity of the problem can be varied along three different dimensions (similarly to [23]): the degree of *concept complexity* (c), the *dataset size* (s), and the *imbalance level* between the two classes (i). *Concept complexity* defines the number of class boundaries, pointed as a complexity factor in XCS [8]. The *dataset size* is the size of the balanced dataset. The *imbalance level* determines the ratio between the number of minority class instances and the number of majority class instances.

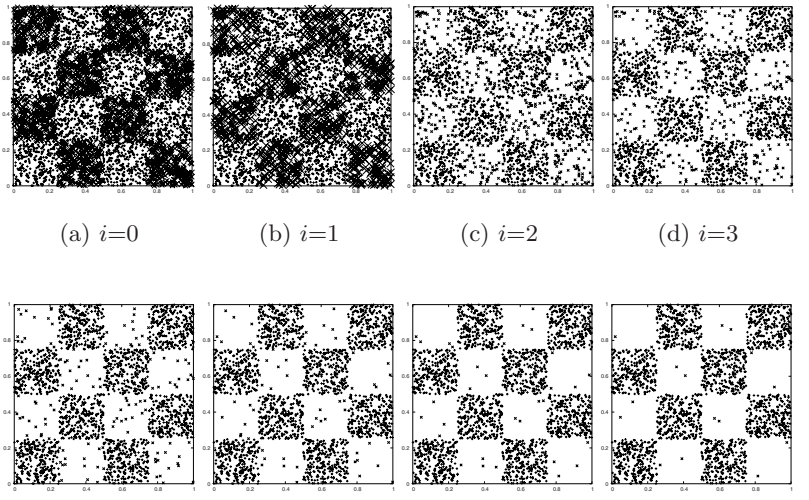
The generation process creates a balanced two-class domain, and then proceeds to unbalance it by removing some of the minority class instances. The original balanced problem is defined by the dataset size s, and the concept complexity c, which defines c^2 alternating squares. We randomly drew points into the feature space so that each checkerboard square received s/c^2 instances. For the balanced dataset, i=0.

An imbalance degree i corresponds to the case where the minority class has $1/2^i$th of its normally entitled points, while the majority class maintains the same points as in the original dataset. This means that the ratio between the minority class instances and the majority class instances is $1/2^i$. Given s, c, and an imbalance level i, each square of the majority class has s/c^2 instances, while each square of the minority class has $s/(c^2 \cdot 2^i)$ instances. For example, for $c = 4$, $s = 4096$, and $i = 3$, each square of the majority class is represented by 256 examples, and each square of the minority class is represented by 32 examples. The domain generation unbalances the dataset iteratively. For $i = 1$, it takes the balanced dataset and removes half of the instances from the minority class. For $i = 2$, it takes the dataset obtained in the previous step and again removes half of the minority class instances, and so on.

3.2 Pos Domain Generation

The *pos* problem [7] has multiple classes and different proportions of examples per class. Given a binary input x of fixed length l, the output class corresponds to the position of the leftmost one-valued bit. If there is not any one-valued bit, the class is zero. The length of the string l determines both the complexity of the problem and the imbalance level.

In the position problem, the most specific rules are activated very sparsely and thus they have very few opportunities to reproduce. Our motivation to include such an extreme domain in the current analysis, rather than trying to solve this particular problem, is to validate our findings in the checkerboard domain.



(a) i=0 (b) i=1 (c) i=2 (d) i=3

(e) i=4 (f) i=5 (g) i=6 (h) i=7

Fig. 1. Training datasets for the *chk* problem, generated with parameters s=4096, c=4, and imbalance levels from 0 to 7

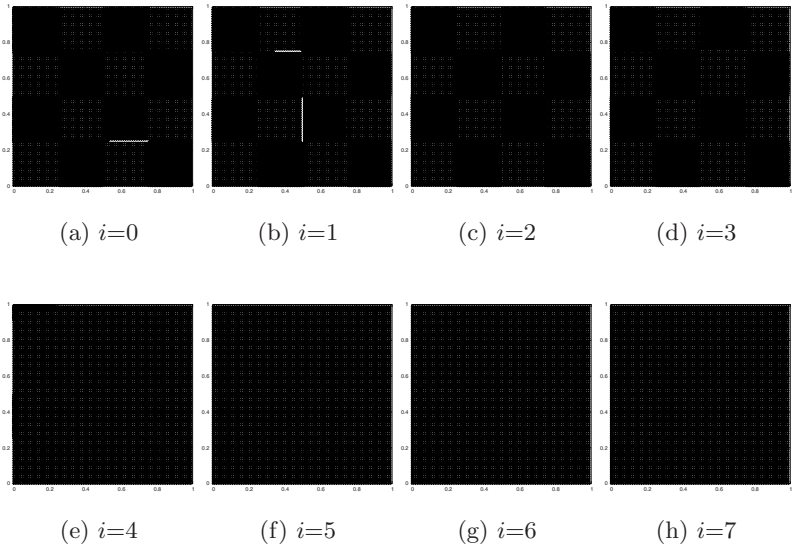
4 UCS's Behavior on Unbalanced Datasets

In this section we analyze UCS's performance with unbalanced datasets, using the artificial problems described in the last section.

4.1 *Chk* Problem

We ran UCS in the checkerboard problem with a fixed dataset size s =4096 and concept complexity c=4, which corresponds to sixteen alternating squares. We varied the imbalance level from i=0 to i=7. For i=0, the dataset is balanced, with 2048 instances per class. For increasing i values we took out half of the instances of the minority class. Thus, the last configuration (i=7) corresponds to 256 instances per square belonging to the majority class, and only two instances for each square of the minority class. Figure 1 depicts all the datasets generated, showing the location of each training point in the feature space.

Since there is not overlapping among instances of different classes, the minority class instances should not be dealt as noise. Even in the most unbalanced dataset (figure 1(h)), all the minority class instances are isolated from the regions containing the majority class instances, and this should be enough to let the system discover the minority class regions. However, it is reasonable to expect that some regions will be more difficult to evolve, depending on the distance of the training points of the minority class to the training points of the majority class.



(a) i=0 (b) i=1 (c) i=2 (d) i=3

(e) i=4 (f) i=5 (g) i=6 (h) i=7

Fig. 2. Boundaries evolved by UCS in the *chk* problem with imbalance levels from 0 to 7. Plotted in black are the regions belonging to the minority class and plotted in gray are the regions of the majority class.

UCS was trained with each of the datasets for 200,000 learning iterations, with the following parameter settings (see [21] for the notation): N=400[1], α=0.1, β=0.2, δ=0.1, ν=10, θ_{del}=20, θ_{sub} = 20, acc_0=0.99, χ=0.8, μ=0.04, θ_{GA}=25, GASub = true, [A]Sub=false. Specify was enabled with parameters Ns_P =20 and Ps_P =0.5 (see [16]).

Figure 2 shows the classification boundaries obtained by UCS in all the datasets. Figure 2(a) corresponds to the balanced dataset. As expected, UCS is able to discover the optimal ruleset. A similar behavior is shown for imbalance levels i={1,2,3}, as seen in figures 2(b), 2(c), and 2(d) respectively. In these cases, the class imbalance does not prevent UCS from evolving the correct boundaries.

The problem arises with imbalance levels equal or greater than 4. Figure 2(e) shows that the system is not able to classify correctly any of the minority class squares. Looking at the training dataset, shown in figure 1(e), it seems feasible that UCS could learn the minority class regions since the number of instances representing these regions define a distinguished space in the eyes of a human beholder. In addition, any model evolved with higher imbalance levels is not able to discover any minority class region, as shown in figures 2(f), 2(g), and 2(h). This abrupt change in the UCS's behavior led us to make a deeper analysis.

Table 1 shows the rules evolved by UCS in the *chk* problem for imbalance level i=4. The table shows, to our surprise, that UCS evolved accurate and maximally general classifiers that cover the minority class regions. Actually, the

[1] The value was set to sixteen times the optimal population size, as suggested in [7].

Table 1. Most numerous rules evolved by UCS in the *chk* problem with imbalance level i=4, sorted by class and numerosity. Columns show respectively: the rule number, the condition and class of the classifier, where 0 is the majority class and 1 the minority class, the accuracy (acc), fitness (F), and numerosity (num).

id	condition	class	acc	F	num
1	[0.509, 0.750] [0.259, 0.492]	: 1	1.00	1.00	39
2	[0.000, 0.231] [0.252, 0.492]	: 1	1.00	1.00	38
3	[0.000, 0.248] [0.755, 1.000]	: 1	1.00	1.00	35
4	[0.761, 1.000] [0.000, 0.249]	: 1	1.00	1.00	34
5	[0.255, 0.498] [0.520, 0.730]	: 1	1.00	1.00	33
6	[0.751, 1.000] [0.514, 0.737]	: 1	1.00	1.00	31
7	[0.259, 0.498] [0.000, 0.244]	: 1	1.00	1.00	27
8	[0.501, 0.743] [0.751, 1.000]	: 1	1.00	1.00	18
9	[0.500, 0.743] [0.751, 1.000]	: 1	1.00	1.00	9
10	[0.751, 1.000] [0.531, 0.737]	: 1	1.00	1.00	8
	...				
18	[0.509, 0.750] [0.246, 0.492]	: 1	0.64	0.01	1
19	[0.000, 1.000] [0.000, 1.000]	: 0	0.94	0.54	20
20	[0.000, 1.000] [0.000, 0.990]	: 0	0.94	0.54	13
21	[0.012, 1.000] [0.000, 0.990]	: 0	0.94	0.54	10
	...				
64	[0.012, 1.000] [0.038, 0.973]	: 0	0.94	0.54	1

eight most numerous rules are those that cover the eight minority class regions, and all of them are accurate. Besides these rules, the table also shows some less numerous rules predicting the majority class. These are overgeneral rules; they cover inaccurately almost all the feature space. From these results two questions arise. Why does UCS evolve these overgeneral rules? And why the system does not properly use the specific rules to classify the minority class regions instead of these overgeneral rules?

To explain UCS's tendency to evolve these overgeneral rules, other populations evolved with lower imbalance levels were checked. We found that all populations evolved with an imbalance level higher than 1 contained the most general rule ($at_1 \in [0, 1]$ and $at_2 \in [0, 1]$), as shown in figure 2 for imbalance level i=3.

We hypothesize that the generalization pressure produced by the GA induces the creation of these overgeneral rules. Once created, these rules are activated in nearly any action set, because of its overgeneral condition. In balanced or low-unbalanced datasets, these overgeneral rules tend to have low accuracy and consequently, low fitness. For example, the most general rule has a 0.50 of accuracy in a balanced dataset. Thus, overgeneral rules with low fitness tend to have low probabilities of participating in reproductive events and finally, they are removed from the population. The problem comes out with high imbalance levels, where overgeneral rules have a high tendency to be maintained in the population. The reason is that the data distribution does not allow to penalize the classifier's accuracy so much, as long as the minority class instances are sampled in a lower frequency. So, the higher the imbalance level, the more accurate an

Table 2. Most numerous rules evolved by UCS in the *chk* problem with imbalance level i=3, sorted by class and numerosity. Columns show respectively: the rule number, the condition and class of the classifier, where 0 is the majority class and 1 the minority class, the accuracy (acc), fitness (F), and numerosity (num).

id	condition	class	acc	F	num
1	[0.251, 0.498] [0.000, 0.244]	: 1	1.00	1.00	39
2	[0.501, 0.751] [0.760, 1.000]	: 1	1.00	1.00	37
3	[0.000, 0.246] [0.259, 0.500]	: 1	1.00	1.00	36
4	[0.259, 0.499] [0.504, 0.751]	: 1	1.00	1.00	33
5	[0.506, 0.746] [0.263, 0.498]	: 1	1.00	1.00	30
6	[0.751, 1.000] [0.502, 0.749]	: 1	1.00	1.00	29
7	[0.752, 1.000] [0.000, 0.240]	: 1	1.00	1.00	27
8	[0.000, 0.246] [0.759, 1.000]	: 1	1.00	1.00	20
	...				
25	[0.000, 0.233] [0.584, 1.000]	: 1	0.13	0.00	1
26	[0.000, 1.000] [0.000, 1.000]	: 0	0.89	0.31	13
27	[0.010, 1.000] [0.000, 1.000]	: 0	0.89	0.31	12
	...				
60	[0.051, 1.000] [0.017, 0.926]	: 0	0.89	0.31	1

overgeneral classifier is considered (and also the higher fitness it has). This effect is clearly seen in tables 2 and 1. Observe that for i=3 (table 2), overgeneral rules have accuracies of 0.89. Similar overgeneral rules for i=4 (table 1) have 0.94 of accuracy. Consequently, in high imbalance levels overgeneral rules tend to have higher accuracies, presenting more opportunities to be selected by the GA, and also lower probabilities of being removed by the deletion procedure.

After analyzing why overgeneral rules are created and maintained in the population as the imbalance level increases, let's consider why the system does not predict the minority class even though it evolved the appropriate rules. For i=3, UCS was able to predict the minority class regions but not for i=4, although apparently the populations evolved were similar. For i=3, there are several numerous and overgeneral rules, but their vote in the prediction array is not enough to overcome the vote of the accurate rules predicting the minority class. Therefore, UCS is able to predict accurately the minority class. The problem arises at a certain point in the imbalance level that makes the vote of the overgeneral rules higher than the vote of the accurate rules. In our datasets, this happens at imbalance level 4. The population evolved in this dataset consists of 64 macroclassifiers, 46 of them predicting the majority class and only 18 predicting the minority class. Taking into account the classifiers' numerosities, there are more than 100 microclassifiers covering all the feature space with the majority class, and only 32 microclassifiers in average for each of the minority class squares. When an instance belonging to a minority class region is shown to UCS, the prediction vote for the majority class is greater, and this makes UCS to classify this instance wrongly.

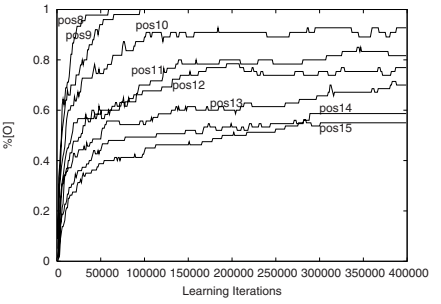


Fig. 3. Percentage of optimal population evolved by UCS in the *pos* problem. Each curve corresponds to the average of five seeds.

4.2 *Pos* Problem

UCS was trained with the *pos* problem with condition lengths l from 8 to 15. We ran UCS for 400000 iterations with the the following parameter settings: N=25·(l+1), α=0.1, β=0.2, δ=0.1, ν=10, θ_{del}=20, θ_{sub} = 20, acc_0=0.99, χ=0.8, μ=0.04, θ_{GA}=25, GASub = true, [A]Sub=false, Specify=true, Ns_P =20, Ps_P =0.5. The experiments were averaged over five different runs.

To analyze UCS's behavior on this problem, we show the curves of performance of UCS during training. We consider here the percentage of optimal classifiers (%[O]) [15] achieved by UCS along the iterations. We use this metric instead of accuracy, because accuracy is biased towards the majority classes. Since we aim to evaluate the system's capability to evolve all the classes, we use a measure that gives equal importance to each of the classes independently of the *a priori* probabilities of each class. Alternatively, a measure of cost per class could be used.

Figure 3 depicts the percentage of optimal population achieved during training. Each curve represents a different level of complexity in the *pos* problem, ranging from l=8 until l=15. It shows that UCS has difficulties in learning all the optimal classifiers as the condition length grows. Table 3 shows an example of the population evolved by UCS for the *pos* problem at l=12. Note that the system can discover the most general optimal rules, being not able to discover the three most specific ones. This behavior is also observed in other evolved populations. As expected, more general rules have higher numerosities. This behavior is attributed to the fact that specific rules activate less often than more general ones, and thus they have fewer reproductive opportunities. Therefore, in a problem with class imbalances the system has more opportunities to learn rules that cover the majority class than those that cover the minority class [7].

5 How to Deal with Class Imbalances

This section reviews different strategies for dealing with the class imbalance problem. The first two are general methodologies applicable to any type of classifier

Table 3. Most numerous rules evolved by UCS in the *pos* problem for l=12, sorted by numerosity. Columns show respectively: the rule number, the condition and class of the classifier, the accuracy (acc), fitness (F), and numerosity (num).

condition	class	acc	F	num
1###########	:12	1.00	1.000	56
0001########	: 9	1.00	1.000	49
01##########	:11	1.00	1.000	46
001#########	:10	1.00	1.000	43
00001#######	: 8	1.00	1.000	32
000001######	: 7	1.00	1.000	24
0000001#####	: 6	1.00	1.000	16
00000001####	: 5	1.00	1.000	11
000000001###	: 4	1.00	1.000	10
0000000001##	: 3	1.00	1.000	5
00#00#00000#	: 1	0.20	0.000	4
0000000#01##	: 3	0.70	0.028	2
0000000000#0	: 2	0.66	0.017	2

since they are based on dataset resampling. The aim of resampling is to balance the *a priori* probabilities of the classes. The last methods are specially designed for UCS, although they can also be adapted to other classifier schemes.

Oversampling. This method consists of oversampling the examples of the minority class until their number is equal to the number of instances in the majority class [22]. Two variants can be considered. *Random oversampling* resamples at random the minority class examples. *Focused resampling* oversamples mainly those instances closer to class boundaries.

Undersampling. Undersampling consists of eliminating some of the majority class instances until we reach the same number of majority class instances as minority class instances [22]. Two classic schemes are *random undersampling*, which removes at random majority class instances, and *focused resampling*, which removes only those instances further away from class boundaries.

Adaptive sampling. This method, initially proposed in [4], is inspired in *oversampling* and in the way *boosting* [25] works. It proposes to maintain a weight for each dataset instance (initially set to 1), which indicates the probability that the sample process selects it. Weights are updated incrementally when the system makes a prediction under exploit mode. Depending on whether the system has made a correct prediction or not on a given example, the weight for that example will be decreased or increased by a factor α (in the experiments we fixed $\alpha = 0.1$).

Class-sensitive accuracy. *Class-sensitive accuracy* modifies the way in which UCS computes accuracy so that each class is considered equally important regardless of the number of instances representing each class. UCS was slightly

modified to compute the experience of a classifier per each class. The proportion of examples covered per each class is taken in account to calculate the classifier's fitness, counterbalancing the bias produced by class imbalances. See [1] for further details.

Selected techniques. We chose to analyze three main representative approaches: random oversampling, adaptive sampling, and class-sensitive accuracy with weighted experience. Undersampling was not considered for being too extreme in the case of highly imbalanced datasets. Under our point of view, undersampling majority class instances to the same degree as minority class instances may produce a loss of valuable information and may change class boundaries unnecessarily. The problem may degenerate into a problem of sparsity, for which classifier schemes in general are expected to show poor generalization capability. Next section compares each of the selected strategies under the checkerboard problem.

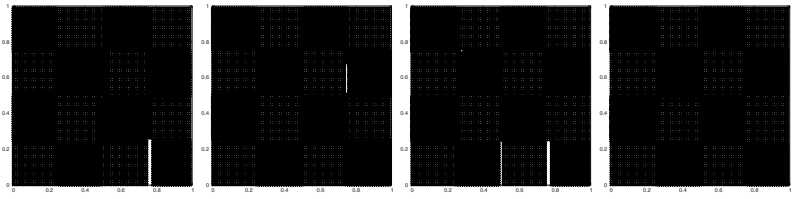
6 UCS in the *Chk* Problem

We run UCS in the checkerboard problem for imbalance levels from i=0 to i=7 using the three aforementioned strategies. We use the same parameter settings as those in section 4, adding the new parameter θ_{acc} for the case of class-sensitive computation. θ_{acc} is set to 50 to protect the fitness decrease of young classifiers.

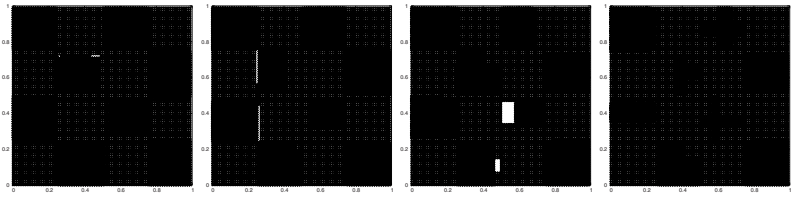
6.1 Random Oversampling

Figure 4 shows the boundaries evolved by UCS under random oversampling. Note that UCS was able to evolve some boundaries for the minority class examples even for the highest imbalance levels. In many cases, these boundaries do not reach the real boundaries of the original balanced dataset. But this result is reasonable since the distribution of training points has changed with respect to the original dataset.

Under oversampling, UCS works as the problem was a well balanced dataset, because the proportion of minority class examples has been adjusted a priori. UCS sees a dataset with the same number examples per class, but with some gaps in the feature space that are not represented by any example. These gaps are mostly covered by rules predicting the majority class rather than by minority class rules. In fact, what happens is that rules from both classes tend to expand as much as possible into these gaps until they reach points belonging to the opposite classes. That is, rules tend to expand as long as they are accurate. Thus, there are overlapping rules belonging to different classes in the regions that are not covered by any example. When we test UCS in these regions, the majority class rules have higher numerosities and their vote into the prediction array is higher. The reason why majority class rules have higher numerosities is that their boundaries are less complex, so in many cases a single rule suffices for all the region. This rule tends to cover all the examples of a majority class square and benefits from long experience and numerosity. On the contrary, minority



(a) i=0 (b) i=1 (c) i=2 (d) i=3



(e) i=4 (f) i=5 (g) i=6 (h) i=7

Fig. 4. Class boundaries evolved by UCS with random oversampling in the chk problem with imbalance levels from 0 to 7

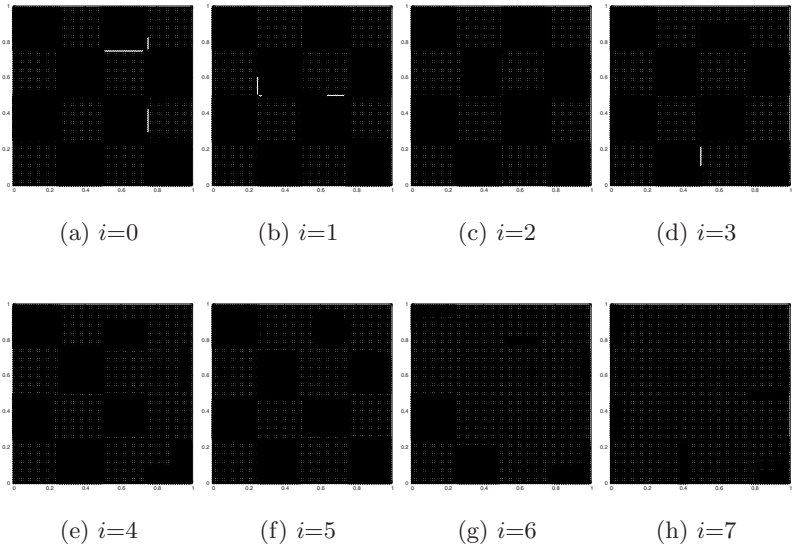
class regions are more complex, and rules covering these regions tend to have less experience and numerosity.

6.2 Adaptive Sampling

Figure 5 shows the results obtained by UCS under the adaptive sampling strategy. UCS evolved part of the squares belonging to the minority class. For i=4 and i=5, the boundaries evolved by UCS almost approximate the real boundaries of the original problem. In these cases, adaptive sampling allowed UCS to evolve fairly good approximations with respect to the original results shown in figure 2. For higher imbalance levels, UCS found more difficulties in finding good approximations for the minority class squares. In these cases, the result achieved under the adaptive sampling strategy is worse than that achieved by UCS under oversampling (see figure 4).

We tried to use a more disruptive function for the weight computation of the adaptive sampling strategy but we found no improvements. On the contrary, trying to use a higher α parameter so that weights could be further increased if instances were poorly classified led to oscillations in the weights and difficulties to stabilize the boundaries evolved.

Analyzing the behavior of UCS under these two strategies (not detailed for brevity), we found that under adaptive sampling there is less generalization pressure towards the minority class rules than with oversampling. The reason is that, with adaptive sampling, once all instances are well classified, weights stabilize and then, all instances are sampled as the original a priori probabilities. Under oversampling, minority class instances are always sampled at the same



(a) i=0 (b) i=1 (c) i=2 (d) i=3

(e) i=4 (f) i=5 (g) i=6 (h) i=7

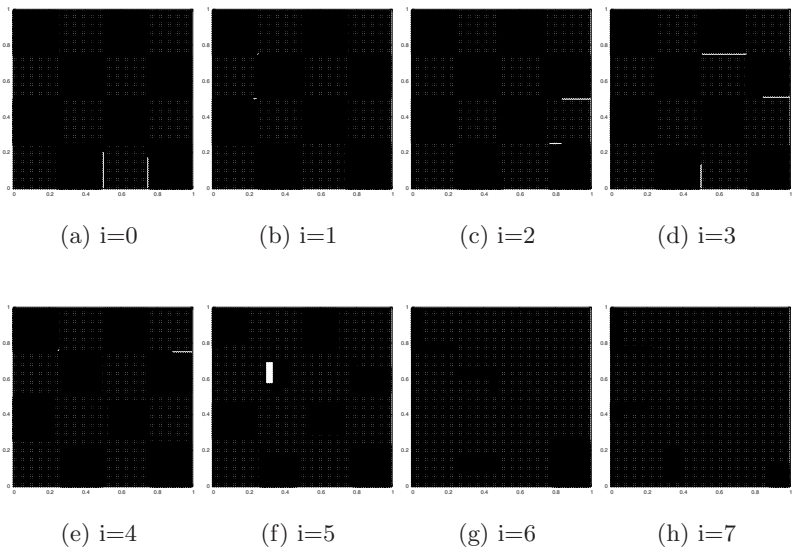
Fig. 5. Class boundaries evolved by UCS with adaptive sampling in the chk problem with imbalance levels from 0 to 7

a priori probability as majority class instances, keeping the same generalization pressure towards both classes. This may justify why, under adaptive sampling, UCS finds more difficulties in the generalization of rules covering the minority class instances, especially for the highest imbalance levels.

6.3 Class-Sensitive Accuracy

Figure 6 shows the results of UCS under class-sensitive accuracy. Note that the boundaries evolved in all imbalance levels are better at discovering minority class regions than those evolved by raw UCS. However, for the lowest imbalance levels (i.e., i=[1-3]), there is a little tendency to leave some blank spaces near the class boundaries. These gaps predominantly belong to the minority class regions. The reason is that rules covering minority class regions easily get inaccurate when they overpass slightly into the majority class regions. Rules classifying the majority class squares and getting inside minority class squares have less probability to cover minority instances (because they are less frequent) so that their accuracy is not penalized as much. This gap effect in the class boundaries was even stronger for class-sensitive accuracy without the weighted experience modification, as shown in [1]. Note that for the balanced dataset, i.e., i=0, the gap effect is also present although with few incidence. These gaps also appeared slightly under oversampling and adaptive sampling, although in the latter to a lower extent.

For imbalance levels i=4 and i=5, UCS with class-sensitive accuracy clearly improves raw UCS in terms of the boundaries evolved. Furthermore, the



(a) i=0 (b) i=1 (c) i=2 (d) i=3

(e) i=4 (f) i=5 (g) i=6 (h) i=7

Fig. 6. Boundaries evolved by UCS with *class-sensitive accuracy* for 0 to 7. Black regions are those classified as minority class. Grey regions are regions classified as majority class. White regions are non-covered domain regions.

tendency of evolving overgeneral rules is restrained. Table 4 depicts the most numerous rules evolved by UCS for imbalance level i=4. Note that overgeneral rules are not evolved. Instead there are maximally general rules covering each of the alternating squares.

Finally, figures 6(g) and 6(h) show the models evolved with imbalance levels i=6 and i=7. As the imbalance level increases, the system finds it harder to evolve the minority class regions. For the highest class imbalances, UCS can only draw partially four of the minority class regions. Looking at the evolved population, not shown for brevity, we confirm that the problem is not attributable to the evolution of overgeneral rules but to the fact that the imbalance ratio is so high (1:128) that it could be considered as an sparsity problem. There are so few representatives of the minority class regions that we may debate whether these points are representative of a sparse class region or whether they belong to noisy cases. In the latter case, we would acknowledge that UCS should not find any distinctive region.

7 Contrasting Results with *Pos* Problem

In last section, we isolated and analyzed the imbalance class problem under the *chk* domain. Now, we analyze the *pos* problem, which combines jointly different complexity factors. Increasing the condition length increases not only its imbalance level but also other identified complexity factors such as the number of classes, the size of the training dataset and the condition length itself. Our

Table 4. Most numerous rules evolved by UCS with class-sensitive accuracy in the *chk* problem for imbalance level i=4, sorted by numerosity. Columns show respectively: the rule number, the condition and class of the classifier, where 0 is the majority class and 1 the minority class, the accuracy for each class (acc_0 and acc_1), fitness (F), and numerosity (num).

id	condition	class	acc_0	acc_1	F	num
1	[0.000 - 0.200] [0.492 - 0.756]	:0	1	-	1.00	34
2	[0.754 - 1.000] [0.757 - 1.000]	:0	1	-	1.00	32
3	[0.000 - 0.253] [0.000 - 0.242]	:0	1	-	1.00	26
4	[0.730 - 1.000] [0.260 - 0.527]	:0	1	-	1.00	15
5	[0.491 - 0.759] [0.480 - 0.753]	:0	1	-	1.00	12
6	[0.000 - 0.250] [0.770 - 1.000]	:1	-	1	1.00	20
7	[0.519 - 0.748] [0.260 - 0.483]	:1	-	1	1.00	19
8	[0.751 - 1.000] [0.000 - 0.247]	:1	-	1	1.00	17
9	[0.257 - 0.454] [0.510 - 0.722]	:1	-	1	1.00	15
10	[0.000 - 0.246] [0.253 - 0.460]	:1	-	1	1.00	15
11	[0.763 - 1.000] [0.526 - 0.740]	:1	-	1	1.00	13
12	[0.482 - 0.786] [0.000 - 0.241]	:0	1	0	0.39	19
13	[0.264 - 0.565] [0.699 - 1.000]	:0	1	0	0.87	18
14	[0.114 - 0.547] [0.156 - 0.529]	:0	1	0	0.66	15
	...					
69	[0.156 - 0.547] [0.201 - 0.507]	:0	1	0	0.43	1

purpose here is to analyze the three strategies under a more difficult problem, as a previous step to the analysis of real-world problems which will be left as a future work.

7.1 Oversampling

UCS was run under oversampling with the same parameter settings as in the original *pos* problem (see section 4.2). Figure 7 shows the percentage of optimal population achieved by UCS in the *pos* problem for condition-lengths l from 8 to 15. Curves are averages of five runs. The figure shows high oscillations in the learning curves of UCS. Note that the learning curves have worsened significantly with respect to the original results with UCS (as shown in figure 3).

Making a deeper analysis, some harmful traits of oversampling, which were not observed in the *chk* problem, come out. In the *pos* problem, changing the a priori probabilities of examples makes accurate generalizations very hard to be evolved. UCS learns accurate and maximally general rules by the presence of the appropriate examples and counter-examples. While the presence of numerous examples favor the generalization of rules, counter-examples set the limit for these generalizations. If rules overgeneralize, the presence of counter-examples makes the rule inaccurate. Therefore rules generalize as long as they cover all the examples of the same class and cover no counter-examples. In the *pos* problem, we oversample minority class examples. Thus, the system gets a higher number of examples for the minority class rules, but on the contrary receives few

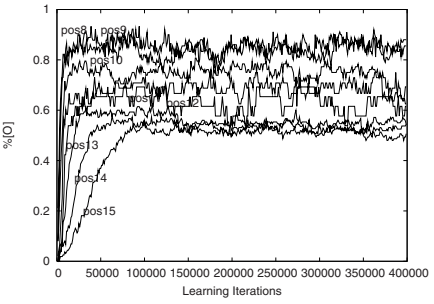


Fig. 7. Percentage of optimal population evolved by UCS under oversampling in the *pos* problem for l=8 until l=15

proportion of counter-examples for these rules. The result is that UCS tends to overgeneralize the rules covering the minority class examples. And the discovery of specific rules for the minority class examples remains unsolved.

We wonder why this effect did not arise in the *chk* problem. The reason is that the *chk* problem has originally the same generalization for each of the rules. Oversampling makes each rule to receive the same proportion of examples and counter-examples. So it is easier to find accurate generalizations.

The results of oversampling on the *position* problem suggest that this method could be harmful depending on the topology of the problem. So this is a method that should be applied with caution in real-world problems, at least for classifier schemes using similar learning patterns to those of UCS.

7.2 Adaptive Sampling

Figure 8 shows the percentage of optimal population achieved by UCS under adaptive sampling. Curves are averages of five runs. See that the oversampling effect does not appear here. If a rule is inaccurate because it does not classify properly an example, the probability of sampling that example is increased. Thus, this example will serve as a counter-example for overgeneral rules, and as an example to help discover rules covering it accurately. Note that the learning curves have improved with respect to the original problem (figure 3), although there is still a high difficulty in discovering the optimal populations for the highest complex levels.

7.3 Class-Sensitive Accuracy

Figure 9 shows the percentage of optimal population evolved by UCS with class-sensitive accuracy. The figure does not reveal significant improvements with respect to raw UCS, as shown in figure 3. The problem here is that UCS is receiving very few instances of the most specific classes, i.e., it receives exactly the same instances as in the original problem. For example, for l=15, UCS receives only one instance of class 0 each 32768 instances. Thus, even though UCS weighs

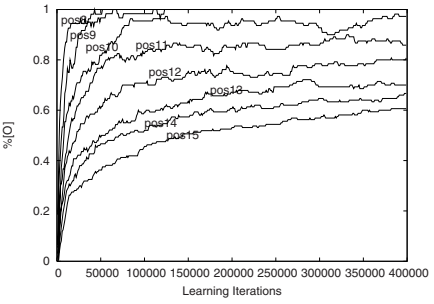


Fig. 8. Percentage of optimal population evolved by UCS under adaptive sampling in the *pos* problem from l=8 to l=15

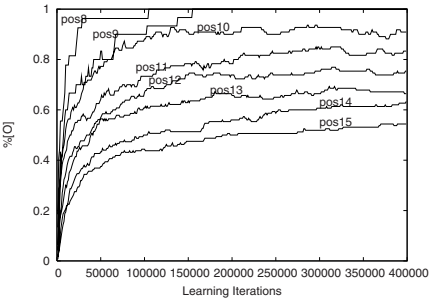


Fig. 9. Percentage of optimal population evolved by UCS with class-sensitive accuracy in the *pos* problem, from l=8 to l=15

the contribution of each class equally, if the minority class instances come very sparsely, rules covering them have few reproductive events. We find it interesting to analyze a hybrid strategy between adaptive sampling and class-sensitive accuracy so that the benefits of each approach could be combined.

8 Conclusions

We analyzed the class imbalance problem in UCS classifier system. We found that UCS has a bias towards the majority class, especially for high degrees of class imbalances. Isolating the class imbalance problem by means of artificially designed problems, we were able to explain this bias in terms of the population evolved by UCS.

In the checkerboard problem, we identified the presence of overgeneral rules predicting the majority class which covered almost all the feature space. For a given imbalance level, these rules overcame the vote of the most specific ones and thus, UCS predicted all instances as belonging to the majority class. Different strategies were analyzed to prevent the evolution of these overgeneral rules. We found that all tested strategies (oversampling, adaptive sampling, and class-sensitive accuracy)

prevented UCS from evolving these overgeneral rules, and class boundaries —for both the minority and majority classes— were approximated fairly better than with the original setting. However, the analysis on the position problem revealed many inconveniences in the oversampling strategy which make UCS's learning unstable. This leads us to discard this method for real-world datasets.

The study would be much enhanced with the analysis of the class imbalance problem on other LCSs. Preliminary experiments made with two other evolutionary learning classifier systems, *GAssist* [2] and HIDER [24], showed that they are even more sensitive to the class imbalance problem. Moreover, the analysis of the class imbalance problem in other classifier schemes such as nearest neighbors, support vector machines and C4.5, and their comparison with LCSs, could give higher understanding on how imbalance class problems affect classifier schemes, and whether they affect LCSs to a higher degree than others.

Also, we would like to extend this analysis to other artificial problems, as well as to real-world datasets. We suspect that the proposed strategies may be very sensitive in noisy problems, i.e, in problems having misclassified instances. The combination of noisy instances and strategies for dealing with class imbalances may worsen significantly the generalization of learners, resulting in too overfitted boundaries. As this is a feature often present in real-world datasets, this should be analyzed in detail as a previous step to understand the behavior of these strategies on real-world datasets.

Acknowledgements

The authors thank S.W. Wilson and two anonymous reviewers for useful comments on a previous version of the paper. Also thanks to 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, 2002SGR-00155, and 2005SGR-00302.

References

1. A. Orriols Puig and E. Bernadó Mansilla. The Class Imbalance Problem in UCS Classifier System: Fitness Adaptation. In *Congress on Evolutionary Computation*, volume 1, pages 604–611, Edinburgh, UK, 2-5 September 2005. IEEE.
2. J. Bacardit. *Pittsburgh genetic-based machine learning in the data mining era: representations, generalization and run-time*. PhD thesis, Department of Computer Science. Enginyeria i Arquitectura la Salle, Ramon Llull University, Barcelona, 2004.
3. J. Bacardit and M.V. Butz. Data Mining in Learning Classifier Systems: Comparing XCS with GAssist. In *Seventh International Workshop on Learning Classifier Systems (IWLCS-2004)*, 2004.
4. E. Bernadó Mansilla. Complejidad del Aprendizaje y Muestreo de Ejemplos en Sistemas Clasificadores. In *Tercer Congreso Espaol de Metaheursticas, Algoritmos Evolutivos y Bioinspirados (MAEB'2004)*, pages 203–210, 2004.

5. M.V. Butz. *Rule-based Evolutionary Online Learning Systems: Learning Bounds, Classification and Prediction*. PhD thesis, Illinois Genetic Algorithms Laboratory (IlliGAL) - University of Illinois at Urbana Champaign, Urbana Champaign 117, 2004.
6. N. Chawla, K. Bowyer, L Hall, and W. Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.
7. E. Bernadó Mansilla and J.M. Garrell Guiu. Accuracy-Based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks. *Evolutionary Computation*, 11(3):209–238, 2003.
8. E. Bernadó Mansilla and T.K. Ho. Domain of Competence of XCS Classifier System in Complexity Measurement Space. *IEEE Transactions on Evolutionary Computation*, 9(1):82–104, 2005.
9. R.E. Fawcett and F. Provost. Adaptive Fraud Detection. *Data Mining and Knowledge Discovery*, 3(1):291–316, 1997.
10. G.M. Weiss. *The effect of small disjunct and class distribution on decision tree learning*. PhD thesis, Graduate School - New Brunswick. Rutgers, The State University of New Jersey, New Brunswick, New Jersey, 2003.
11. J.H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.
12. J.H. Holland. Adaptation. In R. Rosen and F. Snell, editors, *Progress in Theoretical Biology*, volume 4, pages 263–293. New York: Academic Press, 1976.
13. J.H. Holmes. Differential negative reinforcement improves classifier system learning rate in two-class problems with unequal base rates. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 635–642. Morgan Kaufmann, 1998.
14. T. Kovacs. Deletion Schemes for Classifier Systems. In Wolfgang Banzhaf, Jason Daida, A.E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, (GECCO-99)*, pages 329–336. Morgan Kaufmann, 1999.
15. T. Kovacs and M. Kerber. What makes a problem hard for XCS. In *Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.), Advances in Learning Classifier Systems: Third International Workshop, IWLCS*, pages 80–99. Springer-Verlag, 2000.
16. P.L. Lanzi. A study of the generalization capabilities of XCS. In Thomas Bäck, editor, *Proc. of the Seventh Int. Conf. on Genetic Algorithms*, pages 418–425, San Francisco, CA, 1997. Morgan Kaufmann.
17. M. V. Butz, D. E. Golberg, and P. L. Lanzi. Bounding Learning Time in XCS. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2004)*, Berlin, 2004. Springer Verlag.
18. E. Bernadó Mansilla. *Contributions to Genetic Based Classifier Systems*. PhD thesis, Enginyeria i Arquitectura la Salle, Ramon Llull University, Barcelona, 2002.
19. E. Bernadó Mansilla, F.X. Llorà Fàbrega, and J.M. Garrell Guiu. XCS and GALE: a Comparative Study of Two Learning Classifier Systems on Data Mining. In P.L. Lanzi, W. Stolzmann, and S.W. Wilson, editors, *Advances in Learning Classifier Systems, 4th International Workshop*, volume 2321 of *Lecture Notes in Artificial Intelligence*, pages 115–132. Springer, 2002.
20. P.M. Murphy and D.W. Aha. UCI Repository of machine learning databases, University of California at Irvine, Department of Information and Computer Science, 1994.

21. M.V. Butz and S.W. Wilson. 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.
22. N. Japkowicz and S. Stephen. The Class Imbalance Problem: Significance and Strategies. In *2000 International Conference on Artificial Intelligence (IC-AI'2000)*, volume 1, pages 111–117, 2000.
23. N. Japkowicz and S. Stephen. The Class Imbalance Problem: A Systematic Study. *Intelligent Data Analisis*, 6(5):429–450, November 2002.
24. R. Giráldez, J.S. Aguilar-Ruiz, and J.E. Riquelme. Evolutionary Learning of Hierarchical Decision Rules. *IEEE Transactions on Systems, Man, and Cybernetics*, 33(2):324.331, 2003.
25. R.E. Schapire. A Brief Introduction to Boosting. In *Sixteenth International Joint Conference on Artificial Intelligence*, 1999.
26. C. Stone and L. Bull. For Real! XCS with Continuous-Valued Inputs. *Evolutionary Computation*, 11(3):299–336, 2003.
27. S.W. Wilson. Generalization in the XCS Classifier System. In J.R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. Fogel, M.H. Garzon, D.E. Goldberg, H. Iba, and R. Riolo, editors, *Genetic Programming: Proceedings of the Third Annual Conference*, pages 665–674. Morgan Kaufmann, 1998.
28. S.W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.