1

Data Complexity and Evolutionary Learning: Classifier's Behavior and Domain of Competence

Ester Bernadó-Mansilla[1], Tin Kam Ho[2], Albert Orriols[1]

[1] Computer Engineering Department
Enginyeria i Arquitectura La Salle, Ramon Llull University
Quatre Camins, 2. 08022 Barcelona, Spain
e-mail:{esterb,aorriols}@salleurl.edu
[2] Computing Sciences Research Center
Bell Laboratories, Lucent Technologies
Murray Hill, NJ 07974-0636 USA
e-mail: tkh@research.bell-labs.com

Summary. We study the behavior of XCS, a classifier based on genetic algorithms. XCS summarizes the state-of-the-art of the genetic based machine learning field and benefits from long experience and research in the area. We describe the learning mechanisms of XCS by which a set of rules describing the class boundaries is evolved. We study XCS's behavior related to data complexity and identify conditions of difficulty for XCS in the complexity measurement space as those with long boundaries, high class interleaving, and high nonlinearities. Comparison with other classifiers in the complexity space allows to identify domains of competence for XCS as well as domains of poor performance. The study lays the basis to further apply the same methodology to analyze the domains of competence of other classifiers.

Key words: Genetic algorithms, evolutionary computation, evolutionary learning classifier systems, data complexity, domains of competence, classification.

1.1 Introduction

Genetic algorithms (GAs) are search algorithms based on the mechanisms of natural selection and genetics [18, 14, 15]. They have been applied to search, optimization and machine learning problems with great success. GAs explore the search space by using a population of solutions, instead of a single point. This population is evaluated and then, potentially improved by the mechanisms of selection, crossover and mutation. One of the abilities of GAs is to keep a good balance between exploration of the search space and exploitation of the best found solutions. This equilibrium allows to explore large search spaces efficiently, tending to avoid local minima. GAs can also be applied to a wide range of domains, since it does not require much

assumptions on the data model. They can also work with different representations allowing even wider applicability.

This potential is also exploited in the machine learning field, leading to very successful applications. The GA's capability to use different types of representations has resulted in applications as diverse as induction of decision trees [12], instance sets [21], rule sets [6, 10], evolution of neural networks [29, 23], etc. Particularly, the evolution of rule sets has been experimenting a growing interest in the last decades. Since the first proposal, developed by Holland in 1975 [18], the field has benefited from numerous research and development, which have resulted in effective classifiers such as XCS [26]. Currently, XCS is mature enough to be considered as a competitive classifier. Its background consists of experimental studies demonstrating its efficiency in real problems [6, 3], as well as theoretical studies giving light in the functioning of their mechanisms and providing guidelines to exploit its potential by the use of appropriate parameter settings [10]. XCS has also been improved from its first version, with the inclusion of generalization mechanisms [27], new representations [19, 20, 25, 28], improved components [10], etc.

At this stage of maturity, researchers have started to analyze the domain of competence of XCS, i.e., where XCS is applicable and whether it is best of worst than other classifiers for certain types of problems. Several studies have approached this subject by the comparison of XCS's performance with that of other classifiers in a varied range of classification problems [6, 4, 3]. These studies rely their conclusions on observable measures of the datasets, such as the number or types of attributes or the number of classes. But this approach was revealed insufficient for relating XCS's performance, and relative performances between classifiers, to the dataset complexity. A more recent approach [5] started to analyze XCS's performance to data complexity, benefiting from previous studies proposing complexity metrics for classification problems [17]. The aim of this contribution is to summarize this study and enhance the investigation on the domain of competence of XCS.

First, we introduce XCS and its learning mechanisms, showing how XCS evolves rules approximating the class boundaries. The study by Bernadó and Ho [5] performed an extended analysis on XCS's performance related to data complexity, and introduced an analysis on relative performances by making pairwise comparisons of XCS with other classifiers. In this paper, we briefly summarize this study, by showing how XCS adapts to data complexity. Then, we extend it by showing the best and worst domain of XCS, related to a particular choice of representative classifiers.

The paper is structured as follows. Section 1.2 gives a brief introduction to genetic algorithms and evolutionary learning classifier systems. It sets the framework and defines the basic GA's terminology which will be used along the paper. Section 1.3 describes the learning mechanisms of XCS, and section 1.4 introduces the available knowledge representations, centering on the hyperrectangle representation which is the approach taken in this paper. Next, we study XCS's behavior in two classification problems designed artificially and we show graphically how classification problems may imply different degrees of difficulty to different types classifiers. Then, we evaluate XCS's performance on data complexity and identify the complexity measures most relevant for XCS. We also aim to identify problem conditions where XCS is optimal. Thus, we analyze XCS's behavior compared with that of other popular classifiers in the complexity measurement space. Although there are other types of classifiers based on GAs, we center our study on XCS for being one of the best representatives of evolutionary learning classifier systems (LCSs). Section

1.7 outlines how this study can be extended to other types of evolutionary learning classifier systems and summarizes the main conclusions.

1.2 Genetic Algorithms for Classification

1.2.1 GA Basics

Genetic Algorithms (GAs) [18, 14, 15] are defined as search algorithms inspired by natural selection and genetics. GAs explore the search space by means of a *population* of candidate solutions to the problem. Each solution is called an *individual* and is codified in a *chromosome*, a data structure that keeps the genetic information of the solution in a representative way so that it can be manipulated by the genetic operators.

The population may be initialized at random and then incrementally evaluated and improved through *selection*, *crossover*, and *mutation*. Evaluation of each solution is performed by the *fitness* function, which provides the quality of the solution for the given problem. Fitness guides the evolution towards the desired areas of the search space. Individuals with higher fitness have higher chances to be selected and to participate in recombination (crossover) and mutation. Crossover combines the genetic material of two parent individuals to form new offspring. Thus, it exploits good solutions to potentially move the population towards even better solutions. Mutation is applied to single individuals, performing slight changes into their chromosomes. Its aim is to introduce diversity in the population. The new solutions thus obtained are evaluated and the cycle of selection, crossover and mutation is repeated until a satisfactory solution is found or a predefined time limit expires.

1.2.2 Evolutionary Learning Classifier Systems

Although GAs are primarily defined as search algorithms they can be applied to learning problems, where learning is expressed as a search in a space of models representing the target concept. In this sense, GAs must codify a model and evolve it by means of selection, recombination and mutation. The so-called *learning classifier systems* (LCSs) approach searches for a set of rules describing the target concept. In this context, there are two different approaches, called Pittsburgh and Michigan respectively, which differ mainly in their representation.

The Pittsburgh approach [13, 2] codifies each individual as a ruleset. Then the GA evolves a population of rulesets. Once convergence is achieved, the best individual is selected and their ruleset used as the result of learning. Evaluation of each individual (ruleset) is performed independently against the training set of examples, considering different aspects as the classification accuracy, the number of required rules, etc.

The Michigan approach [18, 26] codifies each individual as a single rule. Thus each individual represents a partial solution, and the whole population is needed to codify a ruleset. Evaluation differs from the Pittsburgh approach in that each individual's relative contribution to the whole target concept must be measured. Also the GA takes a different approach so that at convergence a set of diverse solutions are present which jointly codify a ruleset. The XCS classifier system, where we base the current study, takes this approach. Next section describes it in more detail.

1.3 The XCS Classifier System

XCS evolves a set of rules, by means of the interaction with the environment through a *reinforcement learning* scheme and a search mechanism based on a GA. Although XCS can be applied to both single-step and multi-step tasks, we restrict this analysis to XCS acting only as a classifier system. For more details, the reader is referred to [26] and [27] for an introduction of XCS, and to [11] for an algorithmic description.

1.3.1 Representation

XCS evolves a population [P] of classifiers. In the XCS context, a classifier[3] consists of a rule and a set of associated parameters. Each rule has a condition part and a class part: $condition \rightarrow class$. The condition specifies the set of input states where the rule can be applied. The class part specifies the classification that the rule proposes when its condition is satisfied.

The condition of each rule is a conjunction of tests over the features. If an example satisfies these tests, then it is classified with the class codified in the rule. The representation of these tests depends on the types of the features. It also depends on the particular setting of XCS, since several representations are available for a particular type of attribute. Section 1.4 gives an introduction to the most used representations.

Each classifier has a set of associated parameters that estimate the quality of the rule for the given problem. These are:

- the payoff prediction (p): an estimate of the payoff that the classifier will receive if its condition matches the input and its action is selected.
- the prediction error (ϵ): an estimate of the average error between the classifier's prediction and the payoff received from the environment.
- the fitness (F): an estimate of the accuracy of the payoff prediction.
- the experience (exp): the number of times that the classifier has participated in a classification.
- action set size (as): the average number of classifiers of the action sets where the classifier has participated.
- time-step (ts): time-step of the last application of the genetic algorithm.
- numerosity (num): the number of actual *microclassifiers* this *macroclassifier* represents[4].

These parameters are incrementally evaluated each time the classifier participates in the classification of an example. Their values serve as the basis to guide the search mechanisms.

[3] In XCS, the term *classifier* is used to refer to a rule and a set of associated parameters. In the machine learning and pattern recognition fields, a *classifier* refers to the whole system that classifies. In this section, we use this term in the sense of a rule and a set of associated parameters. The remaining sections use the term *classifier* as the whole system.

[4] Classifiers in XCS are in fact *macroclassifiers*, i.e., each classifier represents *num* *microclassifiers* having the same conditions and actions [11].

1.3.2 Performance Component

At each time step, an input example coming from the training dataset is selected randomly and presented to XCS. The system finds the matching classifiers and proposes a classification. Then, the environment returns a reward that is used by XCS to update the parameters of the contributing rules. In the following we give the details.

At each time step, an input example x is presented to XCS. Given x, the system builds a match set [M], which is formed by all the classifiers in [P] whose conditions are satisfied by the example.

An XCS's run may be started with an empty or incomplete ruleset. Therefore, an input example may not find any matching classifier. In this case the covering operator is triggered, creating new classifiers that match the current sample. Covering may also trigger if the number of actions represented in [M] is less than a threshold θ_{mna}. Then new classifiers are generated with conditions matching the example and classes selected randomly from those not present in [M].

From the resulting match set, a class must be selected and sent to the environment. In exploration mode (i.e., during training), the class is selected randomly so that the system can learn the consequences of all possible classes for each input. The chosen class is used to form the action set [A], which consists of all the classifiers proposing that class. Then, the parameters of these classifiers are updated as described in the next section.

In exploitation mode (i.e., during test) the best class, from those present in [M], is selected to maximize performance. This selection is based on a measure of quality for each class, $P(a)$, which is computed as a fitness-weighted average of the predictions of all classifiers proposing that class. In fact, $P(a)$ estimates the payoff that the system will receive if class a is chosen. The selected class determines the action set [A] as in the case of exploration mode. The difference here is that the classifier's parameters are not updated.

1.3.3 Reinforcement Component

In exploration mode, the class is sent to the environment, which returns a reward r that is used to update the parameters of the classifiers in [A]. First, the prediction of each classifier is updated:

$$p \leftarrow p + \beta(r - p) \tag{1.1}$$

where β ($0 \leq \beta \leq 1$) is the learning rate. Next, the prediction error:

$$\epsilon \leftarrow \epsilon + \beta(|r - p| - \epsilon) \tag{1.2}$$

Then, the accuracy of the classifier is computed as an inverse function of its prediction error:

$$k = \begin{cases} \alpha(\epsilon/\epsilon_0)^{-\nu} & \epsilon \geq \epsilon_0 \\ 1 & \text{otherwise} \end{cases} \tag{1.3}$$

where ϵ_0 ($\epsilon_0 > 0$) determines the threshold error under which a classifier is considered to be accurate. α ($0 < \alpha < 1$) and ν ($\nu > 0$) control the degree of decline in accuracy

if the classifier is inaccurate [9]. Then, XCS computes the classifier's accuracy relative to the accuracies of the classifiers in the action set:

$$k' = \frac{k}{\sum_{cl \in [A]} k_{cl}} \tag{1.4}$$

This value is then used to update the fitness F as follows:

$$F \leftarrow F + \beta(k' - F) \tag{1.5}$$

Thus, fitness estimates the accuracy of the classifier's prediction relative to the accuracies of the classifiers belonging to the same action sets.

The experience parameter exp counts the number of times that a classifier is updated. It is increased by one each time the classifier participates in an action set. It is a measure of the confidence on the classifier's parameters. The action set size parameter as averages the number of classifiers of the action sets where the classifier participates. It is updated whenever the classifier belongs to an action set.

1.3.4 Search Component

The search component in XCS tries to improve the ruleset, by means of a GA. The GA is triggered eventually and takes place in [A]. The GA's trigger mechanism is designed to give balanced resources to the different action sets. That is, the GA is activated when the average time since the last occurrence of the GA in the action set (computed from the classifiers' parameter ts) exceeds a threshold θ_{GA}. If the GA is triggered, then it is applied locally into the current [A]. It selects two parents from [A] with probability proportional to fitness, and gets two offspring by applying crossover with probability χ and mutation with probability μ per allele.

The resulting offspring are introduced into the population. First, the offspring are checked for subsumption with their parents. If one of the parents is experienced, accurate and more general that the offspring, then the offspring is subsumed by its parent. This tends to condense the population towards maximally general classifiers.

If an offspring classifier can not be subsumed by its parents, it is inserted into the population, deleting another classifier if the population is full. Deletion is the mechanism by which useless classifiers are discarded from the population, leaving its place to promising solutions. The classifiers with higher probabilities of being deleted are those that participate in large action sets. Also those classifiers with enough experience and low fitness have higher probabilities of being removed from the population. This biases the search towards highly fit classifiers, and at the same time balances the distribution of classifiers through the feature space.

1.3.5 How XCS Learns the Target Concept

When XCS operates as a pure classifier system, it receives training instances from the dataset, performs classifications, and gets feedback from the environment in the form of rewards. The environment is designed to give a maximum reward if the system predicts the correct class and a minimum reward (usually zero) otherwise. XCS's goal is to maximize rewards, which is internally translated to the compound goal of evolving a *complete, consistent and minimal representation* of the target concept.

XCS learns incrementally. Usually, it starts from an empty population and performs generalizations (in the form of rules) of the input examples to cover the empty regions of the feature space. These rules are incrementally evaluated by the reinforcement component and revised by the search mechanism.

The reinforcement component evaluates the current classifiers so that highly fit classifiers correspond to consistent (accurate) descriptions of the target concept. The fitness of each classifier is based on the accuracy of the reward prediction. Highly fit classifiers are those that accurately predict the environmental reward in all the situations where they match.

The search component is based on a genetic algorithm. The GA is guided by fitness, and since fitness is based on accuracy, the GA will tend to evolve accurate rules. The GA should also favor the maintenance of a diverse set of rules which jointly represent the target concept. This is enforced by the use of niching mechanisms, which try to balance the classifiers' allocation in the different regions of the search space. Niching is implicit in different parts of the GA: a) the GA's triggering mechanism, which tries to balance the application of the GA among all the action sets, b) selection, applied locally to the action sets, c) crossover, performing a kind of restricted mating, and d) the deletion algorithm, which tends to delete resources from the more numerous action sets. The GA also enforces the evolution of maximally general rules which allow more compact representations. This generalization pressure is explained by Wilson's generalization hypothesis [26], which can be summarized as follows: if two classifiers are equally accurate but have different generalizations, then the most general one will participate in more action sets, having more reproductive opportunities and finally displacing the specific classifier. Through the interaction of these components, the GA tries to evolve consistent, complete and minimal representations. For more details, please see [10].

1.4 Knowledge Representation

A rule in XCS takes the form: *condition* $\rightarrow$ *class*. The condition is a conjunction of tests over the problem features: $t_1 \wedge t_2 \wedge \ldots \wedge t_n$. The representation of each test depends on the type of attribute. Even for some types of attributes, several representations are available. In fact, this is a particularity of the codification of solutions in GAs. GAs are not tied to any specific representation, so that they can be applied to many domains. The only restriction is to adapt the genetic operators to the particular representation so that the search algorithm can explore efficiently.

If the feature is binary (or belongs only to two categories) the test over this feature is usually codified in the ternary representation, which consists of the symbols {0,1,#}. 0 and 1 codify the two categories respectively, while the symbol # codifies the 'don't care' case, which belongs to the case where the feature is found to be irrelevant.

If the feature is categorical, several encodings are available. The enumeration encoding maps an attribute with c possible categories into a binary string of length c, where each bit tests membership to a distinct category. The test is then a disjunction of the membership tests over each category. An irrelevant feature is codified by a string with all bits set to 1. The nominal encoding codifies the test with a single symbol, which can take values from {0,1,2,...,c-1,#}, where c is the number of categories. Again the don't care symbol makes the attribute irrelevant.

In the case of continuous-valued features, a possibility is to discretize the real values into nominal ranges, and then proceed as in the categorical case. However, this can limit the accuracy of the rule since the nominal ranges must be fixed a priori. Another approach is to let the GA find the necessary ranges, by codifying an interval of type $[l_i, u_i]$, where $l_i \leq u_i$. A set of such intervals describes a hyperrectangle in the feature space. For simplicity, the attributes of the dataset examples are usually normalized to the range [0,1].

Other representations have been proposed for XCS, such as messy coding [19], and S-expressions [20]. Focusing on real features within the scope of this paper, we use the hyperrectangle representation for being one of the most used and successful representations (see [4]). The class is codified as an integer.

Genetic Operators

Once the representation is designed, the genetic operators that manipulate representations must be adapted. This affects covering, mutation, and crossover. Covering must be designed to cover training points that are not covered by the current pool of rules. Crossover exploits the potentially good solutions by recombining parts of them. Mutation should give randomness to explore new regions of the rule space.

Covering initializes new rules which cover empty regions of the search space. Given a training example described by its features $x = (x_1, x_2, ..., x_n)$, covering obtains a rule by means of generalizing each attribute with a matching interval. For each attribute x_i, covering creates an interval $[l_i, u_i]$, where $l_i = x_i - rand(r_0)$ and $ui = x_i + rand(r_0)$. $rand(r_0)$ gives a random value between 0 and r_0, where r_0 is a parameter set by the user. Fig. 1(a) gives two examples of rules obtained by covering.

Mutation introduces randomness into the exploration process. It is applied with probability μ per allele, where an allele is each of the hyperrectangle bounds. To mutate an allele, its valued is changed by an amount $\pm rand(m_0)$, where the sign is selected randomly and m_0 is a parameter set by the user (a typical value is 0.1). Figure 1(b) shows the effect of mutation over an individual in a two-feature space. The individual, with interval ranges defined by $([l_1, u_1], [l_2, u_2])$, suffers mutation on l_2, which is decreased by 0.09.

Crossover takes two parent solutions and produces two offspring. Usually two-point crossover is applied. It computes two random cut points on the rule and the subsequences defined by them are interchanged into the offspring. The cut points can occur between intervals as well as within intervals. Fig. 1.2 shows an example of crossover. On the left, there are two parents selected for crossover. On the right, two offspring are obtained by their recombination. This case corresponds to a cut point among the first and the second dimension. Observe that each offspring gets respectively the first interval (i.e., that of the first attribute) from one parent and the second interval from the other parent.

If the recombination operators result in an invalid interval, either exceeding the [0,1] range or violating the condition $l_i \leq u_i$, then a repair process is applied so that the interval is restricted to a valid one.

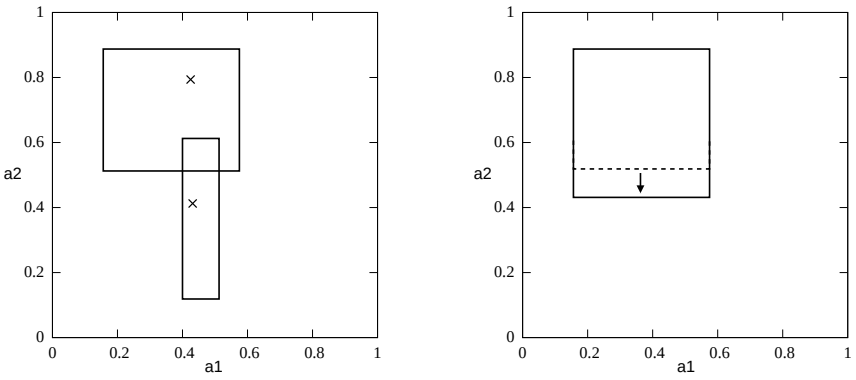

Fig. 1.1. Example of covering and mutation on the hyperrectangle representation. (a) The covering operator is applied to two training points (plotted by a cross) and two rules are obtained. (b) Mutation alters one of dimensions of the hyperrectangle rule.

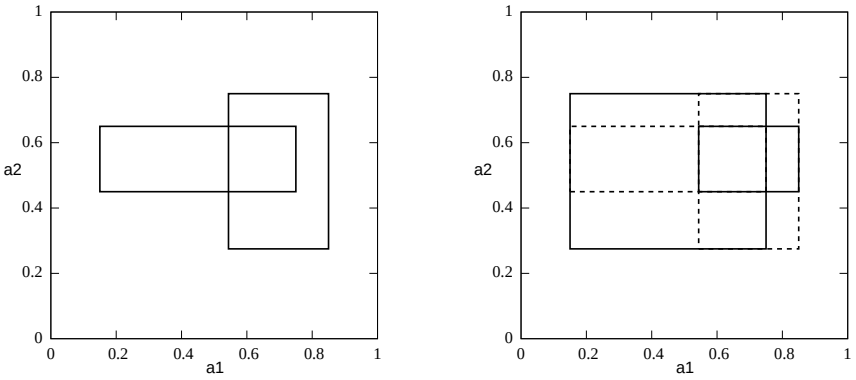

Fig. 1.2. A crossover example. Fig. (a) plots two parent individuals and Fig. (b) plots the offspring resulting from a cut point occurring between the first and the second interval.

1.5 Evolving Class Boundaries: Two Case Studies

We study XCS's behavior in two artificial problems: the checkerboard problem (depicted in Fig. 3(a)) and the fourclass problem (Fig. 3(b)). The checkerboard problem is designed to test XCS on a case of multiple distributed classification regions. It has

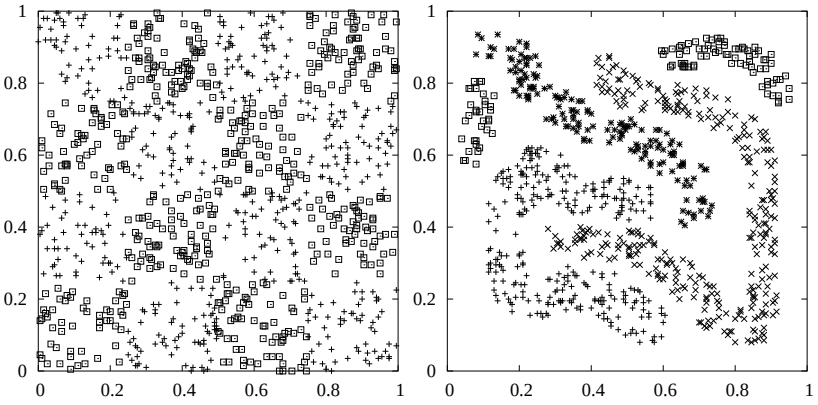

Fig. 1.3. Distribution of training points in the checkerboard problem (a) and the fourclass problem (b).

two classes alternating as in a checkerboard. The fourclass problem is designed to test XCS in problems with multiple classes and curved boundaries. Fig. 1.3 shows the distribution of the training points in each problem. Each point is plotted with a different symbol depending on the class to which it belongs. We analyze XCS's behavior in these problems, and compare its performance with a nearest neighbor (NN) classifier. We aim to show that classification problems may present different degrees of difficulty to different classifiers. We restrict the analysis to two-feature problems so that we can have a graphical representation of the results of each classifier.

To analyze the classification boundaries evolved by each classifier, we train a classifier with the training points depicted in Fig. 1.3. Then we test the classifier with a dense dataset which samples the feature space with 10000 points distributed uniformly. XCS is run with the following parameter settings (see [11] for the terminology): $reward = 1000/0$, $N = 6400$, explore trials $= 200\,000$, $\theta_{mna} =$ number of actions, $\beta = 0.2$, $\epsilon_0 = 1.0$, $\alpha = 0.1$, $\nu = 5$, $\theta_{GA} = 25$, $\chi = 0.8$, $\mu = 0.04$, $\theta_{del} = 20$, $\delta = 0.1$, $doGASubsumption = yes$, $doActionSetSubsumption = no$, $\theta_{sub} = 30$, $r_0 = 0.6$, and $m_0 = 0.1$. The NN is designed with neighborhood 1, and Euclidian distance.

Figs. 4(a) and 4(c) show the classification boundaries obtained by XCS. In the checkerboard problem, XCS has evolved an accurate representation of the feature space. The boundaries almost correspond to the true boundaries of the problem. This is a case where the hyperrectangle representation fits very well, which coupled with the learning mechanisms of XCS, allow XCS to extract a good knowledge representation. In the fourclass problem, XCS approximates the curved boundaries by partially overlapping several hyperrectangles. The resulting boundaries are less natural than the original training set, due to this knowledge representation. The generalization mechanisms of XCS result in a complete coverage of the feature space, although there are no representative training points in all the feature space. This

means that rules tend to expand as much as possible until they reach the boundaries with points belonging to different classes. Figures 4(b) and 4(d) show the same test performed on a nearest neighbor classifier, whose representation based on the Voronoi cells is more suitable to the fourclass problem but less appropriate for the checkerboard problem. The result is that classification accuracy in both classifiers is different; in the checkerboard problem, XCS's error is 0.6%, while NN's error is 0.7%; in the fourclass problem, XCS's error is 1.9% and NN's error is 0.06%.

The classifier's behavior depends on the geometrical complexity of boundaries and the capability of the knowledge representation to approximate these boundaries. In XCS, as also happens with most of the classifiers, the error rate depends on both the knowledge representation and the ability of the search mechanisms to evolve it. Although a knowledge representation may fit perfectly, the algorithms of XCS may not find the appropriate rules. This especially tends to happen with imbalanced problems, i.e., when there are regions of the search space with very few examples. The generalization algorithms of XCS may mask these regions by overgeneral rules (see [4]).

We emphasize the need to characterize XCS's behavior on computable measures of problem complexity and relate the differences between classifiers to these measures. The study performed in the next section takes this approach. We remark that the study is tied to XCS using the hyperrectangle representation, so we include both the limitations of the search algorithms coupled with the hyperrectangle constraints.

1.6 How XCS Adapts to Data Complexity

We study how XCS's behavior depends on data complexity. First, we aim to relate XCS's performance to measures of problem complexity and identify easy and difficult domains for XCS. Such an study could serve to give an expectation of accuracy for XCS given a classification problem with computed complexity measures. We also want to establish the relation between XCS's performance and that of other classifiers in the complexity measurement space. The final aim is to identify areas of the measurement space where XCS excels among other classifiers. Thus, given a problem with its complexity characterization we could either recommend XCS as a suitable classifier or discard XCS in favor of other better approaches.

1.6.1 Analysis Procedure

We characterize the complexity of a classification problem by a set of measures that describe different aspects of boundary complexity. We rely on the study by Ho and Basu [17] where a suite of metrics is proposed and analyzed as measurements of problem complexity. These metrics are found to quantify complexity of problems so that easy problems (such as linearly separable problems) and difficult problems (such as random labeling problems) represent two extremes of the complexity space, with different problems spanning through these extremes. From this study, we select seven metrics representative of the most relevant aspects of complexity. These are enumerated in Table 1.1. They describe different geometrical distributions of class boundaries, such as `boundary`, `intra-inter`, `nonlin-NN`, `nonlin-LP`, and `pretop`, as well as the discriminant power of attributes (`fisher`). We include the ratio of

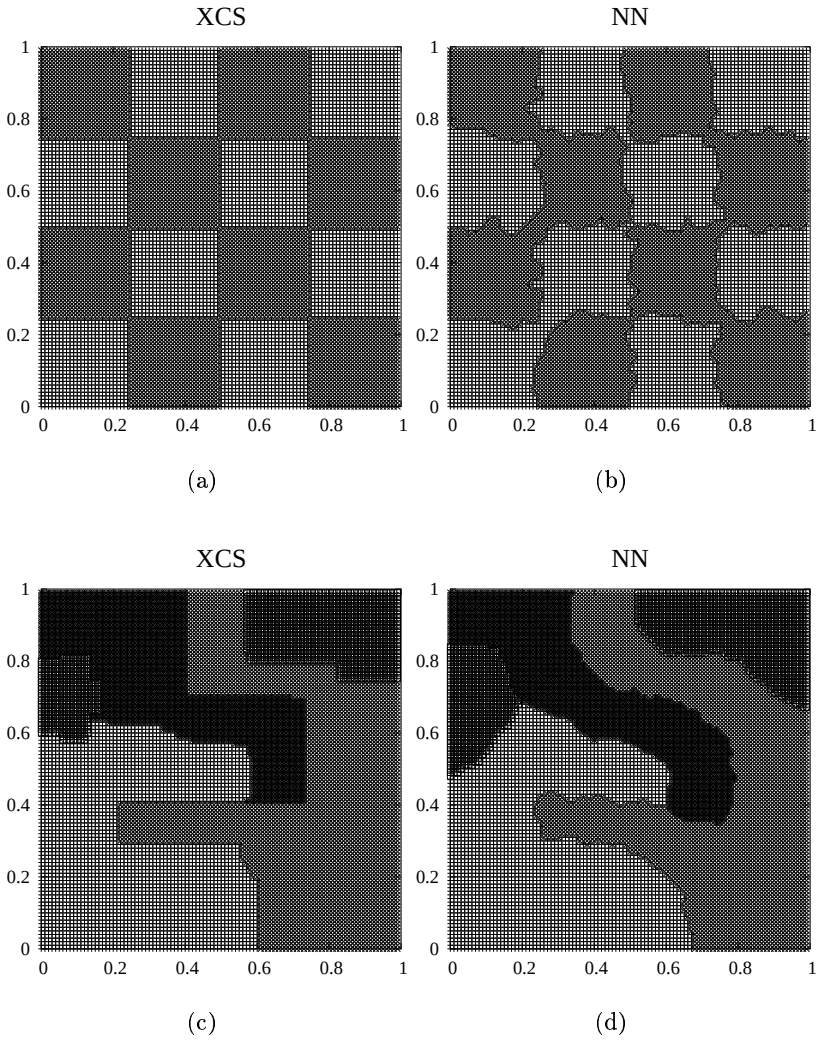

Fig. 1.4. Boundaries evolved by (a) XCS and (b) NN in the checkerboard problem, and (c) XCS and (d) NN in the fourclass problem.

Table 1.1. Complexity metrics used in this study

Measure	Description
`boundary`	percentage of points on boundary estimated by an MST
`intra-inter`	ratio of average intra-inter class nearest neighbor distances
`nonlin-NN`	nonlinearity of nearest neighbor
`nonlin-LP`	nonlinearity of linear classifier
`pretop`	percentage of points with maximal adherence subset retained
`fisher`	maximum Fisher's discriminant ratio
`npts-ndim`	ratio of the number of points to the number of dimensions

the number of points to the number of dimensions (`npts-ndim`) as an estimation of sparsity. All these metrics are computed from the available training sets, so that they give measurements of the apparent complexity of problems.

We evaluate XCS on a set of 392 two-class problems. These problems are generated from pairwise comparisons of 14 problems from the UCI repository [7] containing at least 500 points with no missing values. These are *abalone*, *car*, *german*, *kr-vs-kp*, *letter*, *lrs*, *nursery*, *pima*, *segmentation*, *splice*, *tic-tac-toe*, *vehicle*, *wdbc*, and *yeast*. Their pairwise comparisons result in 844 two-class problems, 452 of which are discarded for being linearly separable problems. The remaining 392 are used as our testbed. All the categorical values are translated into numerical values. Therefore, XCS is using only the hyperrectangle representation.

We measure the relation between XCS's error and data complexity, which is characterized by a set of seven metrics. To estimate the classifier's error, we use a ten-pass two-fold crossvalidation test. The detailed steps are as follows:

1. Each dataset is randomly permuted ten times.
2. Each time, the dataset is divided in two disjoints sets. Then the classifier is trained in each of these two sets and tested on the other one. The error rate for this particular permutation is estimated as the sum of the errors on each half, divided by the dataset size.
3. Thus, for each dataset there are ten error estimates, one for each permutation. The final XCS's error on the dataset is the average of these ten error rates.

1.6.2 XCS's Error and Data Complexity

Fig. 1.5 plots XCS's error related to each of the complexity measures. The y axis depicts the error of XCS for a given problem, while the x axis is one of the complexity metrics.

We observe a clear dependency (almost a linear correlation) of XCS's error rate with respect to the percentage of points in boundary. Since this behavior is also observed in other classifiers (not shown for brevity), it seems that the percentage of points of boundary is a good measure for data complexity. Nevertheless, there are some exceptions to this behavior where XCS performs reasonably well despite a high number of points in boundary. These cases are *car (acc vs good)*, *kr-vs-kp (no-win vs won)*, *nursery (priority vs spec_prior)*, and *tic-tac-toe (neg vs pos)*. As shown in Table 1.2 these cases belong to very low nonlinearities. This suggests that the combined effect of different measures may be necessary to explain data complexity.

Table 1.2. Four easy problems for XCS despite having moderate boundary values. The table shows the values of the complexity measures for each of the problems.

	car	kr-vs-kp	nursery	tic-tac-toe
	acc-good	nowin-won	pr-sp	neg-pos
`boundary`	33.11	20.49	22.90	32.99
`intra-inter`	0.87	0.71	0.96	0.96
`nonlin-NN`	0.00	0.00	0.00	0.00
`nonlin-LP`	0.93	0.79	5.14	1.67
`pretop`	100.00	100.00	100.00	100.00
`fisher`	0.47	0.54	0.38	0.28
`npts`	453	3196	8310	958
`ndim`	21	73	27	27
`npts-ndim`	21.57	43.78	307.78	35.48
XCS's error	1.96	4.82	1.19	2.00

Other metrics are also relevant for XCS's performance. These are the intra-interclass NN distances ratio and the nonlinearities. A high value of intra-interclass ratio means that the classes are very dispersed with respect to the class groupings. Also the nonlinearities impose a degree of difficulty for XCS. If the nonlinearity is high, it probably means that the classes are very interleaved. In both cases, the complex distribution of class groupings makes XCS to evolve a high number of small rules, i.e., specialized rules with few possible generalizations, producing higher classification errors.

The remaining metrics do not influence XCS's error in the same way as before. For example, the highest XCS's error rates correspond to high percentages of retained adherence subsets (`pretop`), but the converse is not true; a high `pretop` value does no imply necessarily a high error. On the contrary, low values on the `pretop` measure always give low XCS's error rates.

The error rate of XCS neither depends directly on the ratio between the number of points and the number of dimensions of the dataset. We can observe only that there are some problems where the XCS's error rate is high (greater than 40%) which correspond to a ratio `npts-ndim` below 50%. In fact, the ratio of the number of points to the number of dimensions is a rough estimate of the sparsity of the training set, so it is difficult to relate XCS's error to the training set sparsity.

High values of the maximum Fisher's discriminant ratio indicate that there is an attribute discriminating fairly well. The higher this value, the easier the problem. This is consistent with our results with XCS. Observe that high values of this metric (greater than 3) always correspond to low error rates. The converse is not necessarily true. A low value of `fisher` does not lead necessarily to high error rates. However, note that the highest error rates belong all to low `fisher` values.

Trying to identify easy and difficult domains for XCS, we have classified our current set of problems in four types: the most difficult problems (XCS's error $\geq$ 45%), difficult problems (XCS's error $\geq$ 40%), easy problems (XCS's error $\leq$ 10%) and the easiest problems (XCS's error $\leq$ 5%). Table 1.3 gives the mean and standard deviation of the complexity metrics for these types of problems. Note that if we move from difficult problems to easy problems, the percentage of points in boundary decreases dramatically, as well as the nonlinearities. Also the intra-interclass NN

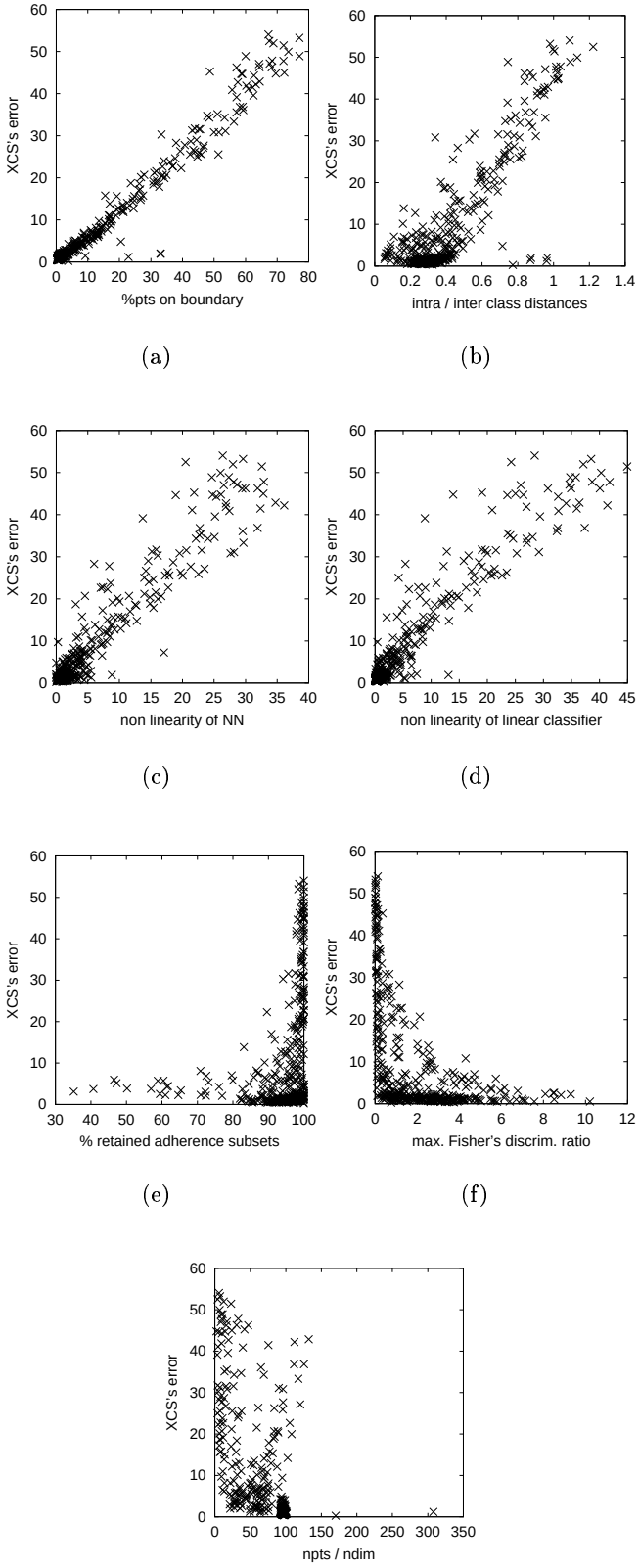

Fig. 1.5. Relation between XCS's error and data complexity. The y axis shows the error of XCS, and the x axis shows respectively the following complexity metrics: (a) the percentage of points in boundary, (b) the ratio of intra-interclass nearest neighbor distances, (c) the nonlinearity of the nearest neighbor, (d) the nonlinearity of the linear classifier, (e) the percentage of retained adherence subsets, (f) the maximum Fisher's discriminant ratio, and (g) the ratio of the number of points to the number of dimensions.

Table 1.3. Four groups of problems, classified according to XCS's error rates. For each group, we show the mean and standard deviation of each complexity metric.

	$error \geq 45\%$		$error \geq 40\%$		$error \leq 10\%$		$error \leq 5\%$	
	mean	std	mean	std	mean	std	mean	std
`boundary`	67.13	7.29	65.01	6.72	4.03	5.12	2.56	3.94
`intra-inter`	0.99	0.11	0.98	0.097	0.30	0.12	0.30	0.12
`nonlin-NN`	27.78	3.48	27.80	4.23	1.88	1.80	1.48	1.21
`nonlin-LP`	34.38	7.00	32.14	7.65	1.42	1.80	1.02	1.45
`pretop`	99.62	0.57	99.51	0.72	92.05	10.32	92.69	9.49
`fisher`	0.06	0.089	0.06	0.075	2.69	1.90	2.82	1.92
`npts`	213.31	240.79	288.86	347.36	1268.57	685.80	1411.58	642.56
`ndim`	10.88	3.50	10.79	2.99	14.56	4.78	15.42	4.65
`npts-ndim`	17.69	13.43	26.12	31.79	83.10	27.81	89.26	24.09
#Datasets	16		28		281		238	

distances decreases in the easy problems. The maximum Fisher's discriminant ratio tends to be higher for low error rates. Similarly, the ratio of the number of points to the number of dimensions is higher for the easiest problems. The percentage of retained adherence subsets is very similar in the three types of problems, although a bit higher for the most difficult problems.

In summary, the highest error rates correspond to problems with high percentage of points in the boundary between classes, high percentage of retained adherence subsets, high training set sparsity, high values of intra-interclass distances, high nonlinearities of NN and LP, and low Fisher values. The easiest problems correspond to small percentage of points in boundary, low nonlinearities (both NN and LP), low values of intra-interclass NN distances, and a varied range over percentage of adherence subsets, `fisher`, and `npts-ndim` values.

1.6.3 On the Domain of Competence of XCS

The last section identified easy and difficult domains for XCS. Here we want to analyze whether other classifiers can perform better or worse than XCS in the current set of problems and identify where these cases are located in the complexity measurement space.

We have chosen an initial set of five classifiers:

- a nearest neighbor classifier (`nn`), with neighborhood set to 1 and Euclidian distance [1].
- a linear classifier (`lc`) computed by linear programming using the AMPL software [24]. It separates the classes by linear boundaries.
- a decision tree (`odt`) using oblique hyperplanes [22]. The hyperplanes are derived using a simplified Fisher's method, as described in [16].
- a subspace decision forest (`pdfc`), which trains oblique trees on sampled feature subsets and combines them by averaging the posterior leaf probabilities [16].
- a subsample decision forest (`bdfc`), also known as bagged decision trees, which trains oblique trees on sampled training subsets and then combines the result by averaging the posterior leaf probabilities [8].

Decision forests belong to the category of classifier ensemble methods. They are known to outperform decision trees in a varied range of domains. Their comparison with XCS aims to identify the relation between the behavior of classifier combination methods and XCS.

In [5] pairwise comparisons of XCS with each classifier allowed to identify regions of the measurement space where XCS was better, equivalent and worse than each particular classifier. Here we take a different approach; we analyze for each problem which is the best classifier and the worst classifier (from those mentioned above including XCS) and compare XCS with their results. This tells us where XCS excels among the classifiers and where XCS is worst. The results are obviously tied to the particular set of classifiers; as a future work, we plan to add other well-known classifiers, such as neural networks, support vector machines, boosting ensembles and stochastic discrimination.

The methodology is the following:

1. For each problem and each method, we estimate the error rate by a ten-pass two-fold crossvalidation test, as explained in section 1.6.1.
2. For each problem, we consider the classifier with the lowest mean error. Then, we span the ten error estimates of the best method, and compare all other classifiers with these values by means of a paired t-test with a 95% confidence level.
3. The same procedure is used to find the *worst* method of each problem and test the remaining methods against it.

Fig. 1.6 shows where XCS performs equivalently to the best classifier (marked by a circle), equivalently to the worst classifier (marked by a cross), and the remaining cases (denoted by a small plus sign). The plots show XCS's error against selected projections of the measurement space. Fig. 6(a) shows XCS's error against the percentage of points in boundary, plotted in a logarithmic scale. Observe that for very low boundary values, XCS is in the average methods. For larger values, a range of problems correspond to a higher proportion of XCS performing as the best classifier. And while the `boundary` metric is increasing, the percentage of problems where XCS is best diminishes while the problems where XCS is worst increase. The problems where XCS is best also correspond to low nonlinearities (Fig. 6(c)) and low ratio of intra-interclass NN distances (Fig. 6(b)). The `fisher` metric is higher where XCS is best (Fig. 6(d)), while the sparsity of the training set (`npts-ndim`) tends to be smaller (Fig. 6(e)). Fig. 6(f) shows XCS's performance in a projection of two combined metrics: the percentage of points in boundary vs the percentage of retained adherence subsets. This plot separates more clearly the three types of problems: problems where XCS performs in the average are located in `boundary` values under 2% and high `pretop` values. In these cases, the nearest neighbor was shown to perform better than XCS [5]. There is another range of problems for which XCS is the best method that are mainly located in `boundary` values between 2% and 20%, with a varied range of `pretop`. Finally, for `boundary` values higher than 20% and high `pretop` values, XCS is the worst method or equivalent to the worst. The plot also reveals gaps in the measurement space. We are currently investigating if they correspond to constraints imposed by the current pool of datasets or they reflect some geometrical and topological constraints tied to our complexity measurement space. Table 1.4 complements these observations by averaging the complexity measurements in the three types of problems.

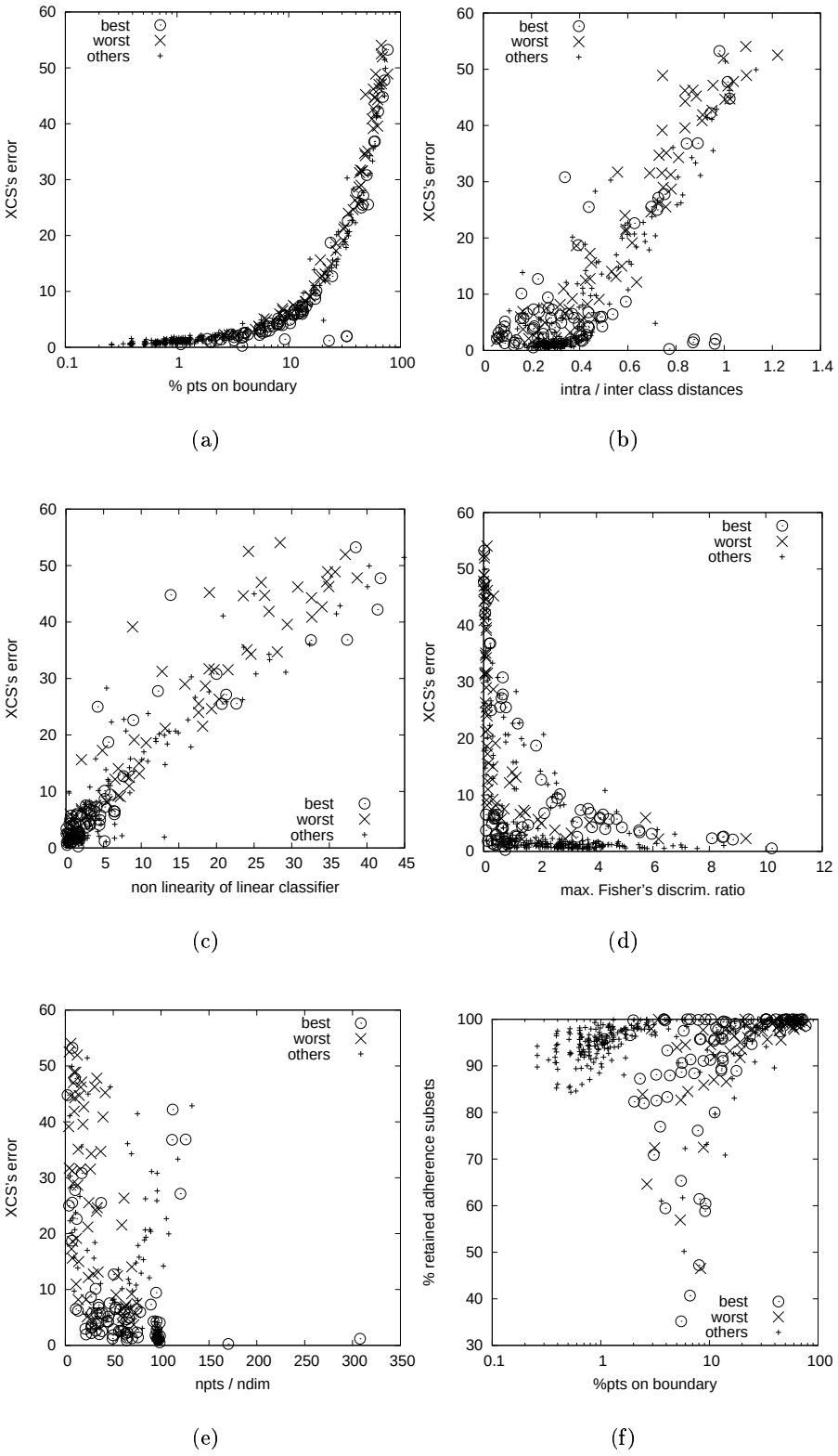

Fig. 1.6. Distribution of problems where XCS is the best method (plotted with a $\odot$), the worst method (plotted with a $\times$), and the remaining problems (plotted with a small $+$).

Table 1.4. Mean and standard deviation of complexity metrics for problems where XCS performs as the best classifier, as an "average" classifier, and as the worst classifier. Last row shows the percentage of problems in each case.

Metric	best		average		worst	
	mean	*std*	*mean*	*std*	*mean*	*std*
`boundary`	17.53	19.17	9.21	16.30	33.79	22.72
`intra-inter`	0.40	0.28	0.37	0.18	0.60	0.29
`nonlin-NN`	6.14	8.50	4.36	7.04	13.64	10.03
`nonlin-LP`	6.26	10.02	4.07	7.83	14.89	12.10
`pretop`	89.90	14.76	94.85	6.12	94.12	10.40
`fisher`	2.11	2.45	2.39	1.69	0.86	1.64
`npts-ndim`	57.66	45.16	84.88	25.16	33.38	25.22
%Problems	19%		64%		17%	

1.7 Conclusions

XCS is an evolutionary learning classifier system which evolves a set of rules describing the target concept. Rules are incrementally evaluated by means of a reinforcement learning scheme and improved through a search mechanism based on a genetic algorithm. Through an appropriate balance of generalization and specialization pressures, rules cover the feature space approximating the class boundaries. The quality of the ruleset approximation will depend on the geometrical distribution of these boundaries. Thus we studied to what degree XCS's performance depends on it. Using computable measures of data complexity, we identified that XCS's error is low for very compact classes, with few interleaving, which is characterized by low percentage of points in boundary, low nonlinearities and low nearest neighbor distances with points of the same class related to points of the other classes. Problems with a dominant discriminating feature tend to be easier. Moving through the complexity axis, XCS's performance becomes increasingly worse for higher points in the class boundaries, higher nonlinearities and higher intra-interclass nearest neighbor distances. The maximum Fisher's discriminating ratio and percentage of adherence subsets are not significant in setting a complex problem for XCS.

We centered our study on XCS, for being one of the best representatives of evolutionary learning classifier systems. However, there are other types of evolutionary classifiers, such as those based on the Pittsburgh approach, which evolve a population of rulesets. Usually Pittsburgh-type classifiers tend to evolve a low number of rules. Problems that require high number of rules will be difficult for them, since the search space becomes extremely high. Large rulesets will be needed for dispersed classes, i.e., for high percentage of points in the boundary, high nonlinearities and high intra-interclass NN distances. We hypothesize that in these cases Pittsburgh classifiers will perform poorly, even worse than XCS, while can offer good approximations for easier problems. We believe that the current study on evolutionary learning and data complexity can be much enhanced considering other types of evolutionary classifiers.

We also studied the domain of competence of XCS, by comparing its performance with that of other classifiers: a nearest neighbor, a linear classifier, an oblique tree and two types of decision forests. XCS is the best classifier for moderate percentage

of points in boundary. For very low boundaries, XCS is overcome by the nearest neighbor. High number of points in boundary, high nonlinearities, and high intra-interclass distances, where XCS's error is high, mainly correspond to cases where XCS is one the worst performing classifiers. Nevertheless there are few problems placed in this measurement region where XCS performs reasonably well, indicating that the measures may not suffice to discriminate these cases. The sparsity of the training set may be an important factor to help discriminate between these cases, although we cannot compute the true sparsity of the real-world datasets. The number of points to the number of dimensions has demonstrated to be a rough estimate of the true sparsity.

The current study has estimated the domain of competence of XCS, leaving many opened questions related to the other classifiers' behavior, such as: What is the domain of competence of the rest of classifiers? Do classifiers perform similarly or are some classifiers significantly dominant from others? Are there any problems where several classifiers can be applied? The next chapter addresses these questions by enhancing the current study to the domain of competence of the remaining classifiers.

Acknowledgement. Ester and Albert acknowledge the support of Enginyeria i Arquitectura La Salle, Ramon Llull University, as well as the support of Ministerio de Ciencia y Tecnología under Grant TIC2002-04036-C05-03 and Generalitat de Catalunya under Grant 2002SGR-00155.

References

1. D.W. Aha, D. Kibler, and M.K. Albert. Instance-based Learning Algorithms. *Machine Learning, Vol. 6*, pages 37–66, 1991.
2. Jaume Bacardit. *Pittsburgh genetic-based machine learning in the data mining era: representations, generalization and run-time.* PhD thesis, Enginyeria i Arquitectura La Salle, Ramon Llull University, Barcelona, Spain, 2004.
3. Jaume Bacardit and Martin V. Butz. Data Mining in Learning Classifier Systems: Comparing XCS with GAssist. In *Seventh International Workshop on Learning Classifier Systems (IWLCS-2004)*, 2004.
4. Ester Bernadó-Mansilla and Josep M. Garrell Guiu. Accuracy-Based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks. *Evolutionary Computation*, 11(3):209–238, 2003.
5. Ester Bernadó-Mansilla and Tin K. Ho. Domain of Competence of XCS Classifier System in Complexity Measurement Space. *IEEE Transactions on Evolutionary Computation*, 9(1):82–104, 2005.
6. Ester Bernadó-Mansilla, Xavier Llorà Fàbrega, and Josep 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 Computer Science*, pages 115–132. Springer, 2002.
7. C.L. Blake and C.J. Merz. UCI Repository of machine learning databases, [http://www.ics.uci.edu/~mlearn/MLRepository.html]. University of California, Irvine, Department of Information and Computer Sciences, 1998.

8. L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.
9. Martin V. Butz and Martin Pelikan. Analyzing the Evolutionary Pressures in XCS. In L. Spector, E.D. Goodman, A. Wu, W.B. Langdon, H.M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M.H. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001)*, pages 935–942. San Francisco, CA: Morgan Kaufmann, 2001.
10. M.V. Butz. *Rule-based Evolutionary Online Learning Systems: Learning Bounds, Classification, and Prediction.* PhD thesis, University of Illinois, 2004.
11. 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-Verlag Berlin Heidelberg, 2001.
12. E. Cantú-Paz and C. Kamath. Inducing oblique decision trees with evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 7(1):54–68, 2003.
13. Kenneth A. De Jong, William M. Spears, and Diana F. Gordon. Using Genetic Algorithms for Concept Learning. *Genetic Algorithms for Machine Learning (John J.Grefenstette editor), A Special Issue of Machine Learning, 13, 2-3*, pages 161–188, 1993.
14. David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, Inc., 1989.
15. David E. Goldberg. *The Design of Innovation. Lessons from and for Competent Genetic Algorithms.* Kluwer Academic Publishers, 2002.
16. Tin K. Ho. The random subspace method for constructing decision forests. *IEEE Transcations on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.
17. Tin K. Ho and M. Basu. Complexity Measures of Supervised Classification Problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3):289–300, March, 2002.
18. John H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.
19. Pier Luca Lanzi. Extending the Representation of Classifier Conditions. Part I: From Binary to Messy Coding. 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 337–344. Morgan Kaufmann, 1999.
20. Pier Luca Lanzi. Extending the Representation of Classifier Conditions. Part II: From Messy Coding to S-Expressions. 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 345–352. Morgan Kaufmann, 1999.
21. Xavier Llorà and Josep M. Garrell Guiu. Co-evolving Different Knowledge Representations with fine-grained Parallel Learning Classifier Systems. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2002)*, pages 934–941. Morgan Kaufmann, 2002.
22. S. Murthy, S. Kasif, and S. Salzberg. A System for Induction of Oblique Decision Trees. *Journal of Artificial Intelligence Research*, 2(1):1–32, 1994.
23. J.D. Schaffer. Combinations of genetic algorithms with neural netwoks or fuzzy systems. In J.M. Zurada, R.J. Marks, J. II Marks, and C.J. Robinson, edi-

tors, *Computational Intelligence Imitating Life*, pages 371–382. New York: IEEE Press, 1994.
24. F.W. Smith. Pattern classifier design by linear programming. *IEEE Transactions on Computers*, C-17:367–372, 1968.
25. Christopher Stone and Larry Bull. For Real! XCS with Continuous-Valued Inputs. *Evolutionary Computation*, 11(3):299–336, 2003.
26. Stewart W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.
27. Stewart 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*. San Francisco, CA: Morgan Kaufmann, 1998.
28. Stewart W. Wilson. Get Real! XCS with Continuous-Valued Inputs. In L. Booker, S. Forrest, M. Mitchell, and R Riolo, editors, *Festschrift in Honor of John H. Holland*, pages 111–121. Center for the Study of Complex Systems, University of Michigan, 1999.
29. X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

Index