Estimating Classifier Generalization and Action's Effect: A Minimalist Approach

Pier Luca Lanzi

Artificial Intelligence and Robotics Laboratory
Dipartimento di Elettronica e Informazione
Politecnico di Milano
pierluca.lanzi@polimi.it

Abstract. We present an online technique to estimate the generality of classifiers conditions. We show that this technique can be extended to gather *some* basic information about the effect of classifier actions in the environment. The approach we present is *minimalist* in that it is aimed at obtaining as much information as possible from online experience, with as few modifications as possible to the classifier structure. Because of its plainness, the method we propose can be applied virtually to any classifier system model.

1 Introduction

Learning classifier systems are online techniques which exploit *reinforcement learning* and *evolutionary computation* to learn how to solve problems. Learning is achieved through the evolution of a set (or *population*) of condition-action-payoff rules (the *classifiers*) which represent the solution. As all the other *reinforcement learning* techniques, a learning classifier system neither assumes any knowledge about space of possible input configurations that will be encountered, nor any knowledge about the possible effect that its actions will have in the environment.

When the solutions evolved by a classifier system are analyzed, one of the main focus is the degree of generalization achieved. Classifiers that apply in many situations are studied because they provide some high level knowledge about the target problem; classifiers that apply in few situations are studied because they provide insight about some specific aspect of the target problem. In principle, since no assumptions are made about the input space, it should not be possible to know which situations classifiers have been applied to (unless we can keep track of the situations encountered as done in [3]). In practice, the analysis techniques usually applied assume that the problem space is known completely (e.g., [9]). However, this assumption becomes easily infeasible in large applications such as real-world robotics and analysis of huge amount of data. Thus it would be useful to have a technique capable of providing some information about the classifier generality which (i) does not assume that the input space is known completely, and (ii) does not store large amounts of additional information.

E. Cantú-Paz et al. (Eds.): GECCO 2003, LNCS 2724, pp. 1894–1904, 2003.

In this paper, we present a very simple technique to estimate the degree of generality of classifier conditions. The technique does not require any prior knowledge about the input space and requires only limited amount of additional information. The technique works online while learning is in progress and classifiers are used. It consists of collecting elementary statistics about the sensory inputs for classifiers that are currently matching. We show that the same technique can be extended to collect some elementary information about the effect of classifier actions. The approach we present is *minimalist* in that it is aimed at obtaining as much information as possible from online experience, with as few modifications as possible to the classifier structure. The method is far from being as powerful as those specifically designed for anticipatory behavior [1]. Nevertheless, the results produced might be still interesting. Most important, because of its plainness the method we propose can be applied virtually to any classifier system model.

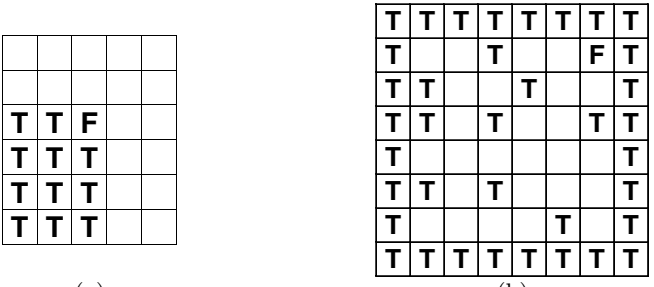
2 Motivation

In learning classifier systems, it is usually difficult to distinguish between classifiers that are *syntactically general* and classifier that are *actually general*, unless some knowledge about the problem space is available. Let us illustrate this with a simple example. Suppose that we are dealing with the most usual learning classifier system model; conditions are represented as strings of fixed length L in the alphabet $\{0, 1, \#\}$; the symbol $\#$, called *don't care*, can match both 1s and 0s; actions are represented as binary strings of fixed length; the problem we are trying to solve involves binary sensory inputs made of 16 bits, and allows eight actions encoded with a binary string of three bits. Suppose that the system has evolved a classifier with condition `################1` and action `111`. This classifier seems very general: it contains 15 don't care symbols and therefore *in principle* it can match 2^{15} different situations. However, it might be also the case that there is only one input configuration with a one in the last position (e.g., `0000000000000001`), so that the classifier is not general but very specific instead.

Some information about the situations that classifiers match can also be useful to analyze the results produced through symbolic representations. For instance, [5] reports the following example of symbolic classifier condition expressed with a Lisp syntax:

```
(AND(AND(AND(NOT(EQ(A2)(A1)))(NOT(EQ(A2)(A1))))
(AND(AND(GT(A5)(1))(4))(NOT(EQ(A2)(A1)))))(NOT(EQ(A2)(A1))))
```

which involves five integer attributes, `A1` ... `A5`. `AND`, `OR`, and `NOT` represent the corresponding Boolean functions; `GT` is the relation *greater-than*, and `EQ` is the relation *equal-to*. Note that, although the above condition is quite simple, it is still difficult to estimate whether the condition is general. Of course, we might simplify it with some tools, but the resulting condition might still be too complex to be analyzed. Thus, we believe that the analysis of large rulesets



(a) (b)

Fig. 1. The `Woods1` environment (a) and the `Maze4` environment (b)

evolved through symbolic representations would be made easier if some support to evaluate the degree of generalization of classifiers could be available.

3 Experimental Design

To test the methods presented in this paper, we implemented them on an available implementation of XCS [6] and applied our extended version of XCS to Boolean multiplexer and to woods environments.

Experiments. All the experiments reported were performed following the standard procedure used in the literature [8]. Each experiment consists of a number of problems that XCS must solve. When XCS solves the problem correctly, it receives a constant reward equal to 1000; otherwise it always receives a constant reward equal to 0.

Boolean Multiplexer. Boolean multiplexer are defined for l Boolean variables (i.e., bits) where $l = k+2^k$: the first k variables $(x_0 \dots x_{k-1})$ represent an address which indexes into the remaining 2^k variables $(y_0 \dots y_{2^k-1})$; the function returns the value of the indexed variable. For example, consider the multiplexer with six variables $\mathtt{mp}_6(x_0, x_1, y_0, y_1, y_2, y_3)$ ($\mathtt{mp}_6 : \{0,1\}^6 \to \{0,1\}$) defined as follows:

$$\mathtt{mp}_6(x_0, x_1, y_0, y_1, y_2, y_3) = \overline{x_0}\ \overline{x_1} y_0 + \overline{x_0}\ x_1 y_1 + x_0\ \overline{x_1} y_2 + x_0 x_1 y_3$$

The product corresponds to the logical *and*, the sum to the logical *or*, and the overline corresponds to the logical not. The system has to learn how to represent the Boolean multiplexer from a set of examples. For each problem, the system receives as input an assignment of the input variables $(x_0, x_1, y_0, y_1, y_2, y_3)$; the system has to answer with the corresponding truth value of the function (0 or 1); if the answer is correct the system is rewarded with 1000, otherwise 0.

Woods Environments. These are simple grid worlds like those depicted in Fig. 1, which contain obstacles (`T`), free positions (`.`), and food (`F`). There are eight sensors, one for each possible adjacent cell. Each sensor is encoded with two bits: `10` indicates the presence of an obstacle `T`; `11` indicates a goal `F`; `00`

indicates a free cell. Classifiers conditions are 16 bits long (2 bits $\times$ 8 cells). There are eight possible actions, encoded with three bits. The system goal is to learn how to reach food position from any free position; the system is always in one free position and it can move in any surrounding free position; when the system reaches a food position (F) the problem ends and the systems is rewarded with 1000; in all the other cases the system receives zero reward.

4 Estimating Rule Generality

The method we propose is aimed at collecting as much information as possible regarding the classifier generality, making as few modifications as possible to the classifier system structure. The method is quite simple. To the classifier structure, we add (i) an array *in* to estimate the averages of the sensory inputs matched by the classifier; (ii) an array ε_{in} to estimate the error affecting the values in the array *in*. The size of *in* and ε_{in} is the number of sensory inputs (e.g., in the first example discussed it would be 16; in the second example it would be six). At the beginning of an experiment, the values of *in* and ε_{in} of classifiers in the rulebase are initialized with a predefined value (0 in our experiments, but a random value would work as well). When classifiers are matched against the current sensory inputs s_t, for every classifier cl that matches s_t, the array *in* and ε_{in} are updated as follows:

$$\forall cl \text{ that matches } s_t \; \forall k :$$

$$cl.in[k] \leftarrow cl.in[k] + \beta_e(s_t[k] - cl.in[k]) \quad (1)$$

$$cl.\varepsilon_{in}[k] \leftarrow cl.\varepsilon_{in}[k] + \beta_e(|s_t[k] - cl.in[k]| - cl.\varepsilon_{in}[k]) \quad (2)$$

where $s_t[k]$ represent the k-th sensory input of s_t; $cl.in[k]$ represents the k-th element of the array *in* of classifier cl; likewise $cl.\varepsilon_{in}[k]$ represents the k-th element of the array ε_{in} of classifier cl; β_e ($0 < \beta_e < 1$) is the learning rate. Equation 1 tries to estimate the average of all the input values that classifier cl matches. Given the average input values stored in $cl.in[k]$, Equation 2 builds an estimate of the error $cl.\varepsilon_{in}[k]$ affecting $cl.in[k]$. The reader familiar with Wilson's XCS [8] will recognize in equations 1 and 2 the update of the classifier prediction parameter p, and classifier error parameter ε used in XCS.

5 Experiments on Rule Generality

To test the proposed method for estimating classifier generality, we implemented it on an available XCS implementation [6] and applied our XCS extension to the 6-multiplexer with the following parameters (see [2] for details): $N = 400$, $\beta = 0.2$, $\alpha = 0.1$, $\epsilon_0 = 0.01$, $\nu = 5$, $\theta_{GA} = 25$, $\chi = 0.8$, $\mu = 0.04$, $\theta_{del} = 20$, $\delta = 0.1$, $\theta_{sub} = 20$, $P_{\#} = 0.6$, $\theta_{mna} = 2$, *doGASubsumption*= 1, *doActionSetSubsumption*= 0 (i.e., action set subsumtption is not used). The learning rate β_e is 0.001. Table 1 reports an example of population that was evolved after 20000 learning problems; for the sake of brevity, only classifiers with non null

Table 1. Classifiers with non-zero prediction evolved by XCS for the 6-mpultiplexer. For every classifier we report: a unique identifier, the classifier condition and the classifier action separated by ":" , the prediction p, the prediction error ε, and the fitness F. The array in and ε_{in} are reported as a sequence of $in[k] \pm \varepsilon_{in}[k]$ for $k \in \{0 \dots 5\}$.

23331	`11###1:1`	$p = 1000\ \varepsilon = 0.00\ F = 1.00$
		$\langle 1.00 \pm 0.00, 1.00 \pm 0.00, 0.53 \pm 0.50, 0.48 \pm 0.50, 0.47 \pm 0.50, 1.00 \pm 0.00 \rangle$
3485	`000###:0`	$p = 1000\ \varepsilon = 0.00\ F = 1.00$
		$\langle 0.00 \pm 0.00, 0.00 \pm 0.00, 0.00 \pm 0.00, 0.48 \pm 0.50, 0.49 \pm 0.50, 0.52 \pm 0.50 \rangle$
7810	`11###0:0`	$p = 1000\ \varepsilon = 0.00\ F = 1.00$
		$\langle 1.00 \pm 0.00, 1.00 \pm 0.00, 0.55 \pm 0.49, 0.47 \pm 0.50, 0.47 \pm 0.50, 0.00 \pm 0.00 \rangle$
8807	`01#0##:0`	$p = 1000\ \varepsilon = 0.00\ F = 1.00$
		$\langle 0.00 \pm 0.00, 1.00 \pm 0.00, 0.54 \pm 0.50, 0.00 \pm 0.00, 0.47 \pm 0.50, 0.53 \pm 0.50 \rangle$
20897	`01#1##:1`	$p = 1000\ \varepsilon = 0.00\ F = 1.00$
		$\langle 0.00 \pm 0.00, 1.00 \pm 0.00, 0.46 \pm 0.50, 1.00 \pm 0.00, 0.50 \pm 0.50, 0.46 \pm 0.50 \rangle$
373	`10##1#:1`	$p = 1000\ \varepsilon = 0.00\ F = 1.00$
		$\langle 1.00 \pm 0.00, 0.00 \pm 0.00, 0.48 \pm 0.50, 0.50 \pm 0.50, 1.00 \pm 0.00, 0.50 \pm 0.50 \rangle$
7723	`001###:1`	$p = 1000\ \varepsilon = 0.00\ F = 1.00$
		$\langle 0.00 \pm 0.00, 0.00 \pm 0.00, 1.00 \pm 0.00, 0.53 \pm 0.50, 0.50 \pm 0.50, 0.47 \pm 0.50 \rangle$
6108	`10##0#:0`	$p = 1000\ \varepsilon = 0.00\ F = 1.00$
		$\langle 1.00 \pm 0.00, 0.00 \pm 0.00, 0.51 \pm 0.50, 0.52 \pm 0.50, 0.00 \pm 0.00, 0.52 \pm 0.5 \rangle$

prediction are reported. For each classifier, on the first line, we report: a unique identifier, the classifier condition and the classifier action separated by a":", the prediction p, the prediction error ε, the fitness F; on the second line, for each input $s[k]$, we report the interval $in[k] \pm \varepsilon_{in}[k]$ which provides an estimate of the values of the k-th input that the classifier matches.

Since in Boolean multiplexer all the possible input configurations are considered, we should expect that (i) every position k of the classifier condition containing a don't care, would correspond to a $in[k]$ equal to 0.5, and to a $\varepsilon_{in}[k]$ equal to 0.5 (i.e., 0.5 ± 0.5); (ii) every position k of the classifier condition containing a 1 or a 0, would correspond to a $in[k]$ equal to 1 or 0 respectively, and to a $\varepsilon_{in}[k]$ equal to 0.0 (i.e., 0.5 ± 0.5). Table 1 confirms these expectations. All the interval corresponding to a constant input (0 or 1) have an error estimate (ε_{in}) equal to 0; all the interval corresponding to a don't care have values very near to 0.5. Of course, since in and ε_{in} are estimates they are affected by errors mainly due to the order in which the inputs appeared and to the value of β_e used. For instance, in classifier `23331`, the interval corresponding to the first don't care is 0.53 ± 0.50, instead of 0.50 ± 0.50. Probably, we could obtain better approximations by tuning β_e or by using an adaptive value for β_e. On the other hand, some experiments we performed with more sophisticated techniques did not result in relevant improvements of the results; while we find that the estimates developed through our elementary technique are quite satisfactory. In fact, in Table 1 all the values of in and ε_{in} turn out to be correct when approximated to the first decimal digit.

The technique appears even more interesting if we apply it to a version of XCS involving symbolic expressions (e.g., [7]). We apply XCS with symbolic expression to the 6-multiplexer with the same parameters used in the previous experiments. In one of the final populations appears the following classifier:

```
((((NOT(NOT(NOT(( X0 OR Y0 )AND(NOT Y0 )))))OR((NOT((NOT( X0 AND Y1 )) AND((NOT X1
) AND( Y3 AND X0 )))) OR(NOT((( Y0 OR Y0 )OR( Y3 OR Y1 ))AND((NOT Y3) AND((NOT X1
)AND ( Y3 AND X0 )))))))AND((NOT X0 )AND(NOT(NOT( Y1 AND X1 )))))AND(( X1 AND(((( Y1
AND ( Y1 AND X1 ))AND Y1 )AND( Y1 AND(NOT X0 ))) AND((((NOT( Y0 OR X0 )) OR(((NOT X0
)AND Y0 )OR( Y1 AND Y0 )))AND(NOT(NOT( Y1 AND X1 ))))AND( Y1 AND(NOT X0 )))))AND X1
)):1
```

$p = 1000\ \varepsilon = 0\ F = 1$

$\langle 0.00 \pm 0.00, 1.00 \pm 0.00, 0.55 \pm 0.50, 1.00 \pm 0.00, 0.46 \pm 0.50, 0.51 \pm 0.50 \rangle$

where `AND`, `OR`, and `NOT` represent the corresponding Boolean operators; `X0`, `X1`, `Y0`, ... , `Y3` represent the six problem variables. Note that it is almost impossible to evaluate the generality of the condition (unless we simplify it), however the intervals suggest that the above classifier matches situations in which $x_0 = 0$, $x_1 = 1$, and $y_1 = 1$ while all the other variables can take any value between 0 and 1. When we simplify the condition with a tool for logic synthesis, we find that the condition corresponds to the expression $\overline{x_0}\ x_1\ y_1$, confirming what suggested by the intervals represented with in and ε_{in}.

6 Estimating Actions' Effect

The same principle used to estimate rule generality can be applied to provide some *elementary* information about the effect of classifier actions. The classifier structure is further extended, adding (i) an array ef to estimate the average values of the sensory inputs that the system will receive if the classifier action is performed; (ii) an array ε_{ef} to estimate the error affecting the values in the array ef. As in the previous case, the size of ef and ε_{ef} is the number of sensory inputs. Consider the (quite typical) performance cycle of a classifier system:

```
1.  repeat
2.      consider the current sensory input s_t ;
3.      build the set [M] of classifiers that match s_t ;
4.      select an action  a_t from those in [M];
5.      store the classifiers in [M] with action  a_t into [A];
6.      perform action  a_t;
7.      get the reward  r_t;
8.      get the next sensory input  s_{t+1} ;
9.      distribute the reward  r_t to classifiers;
10. until stop criterion met;
```

Although the structure still resembles that of XCS, we left out some typical details of XCS (e.g., the genetic algorithm and the distribution of the reward in

$[A]$) to keep it as much general as possible. We can extend the cycle above by adding a step, between line `8` and line `9`, in which for every classifier cl in `[A]`, the array ef and the array ε_{ef} are updated with s_{t+1} as follows:

$$\forall cl \in [A]\ \forall k :$$
$$cl.ef[k] \leftarrow cl.ef[k] + \beta_e(s_{t+1}[k] - cl.ef[k]) \quad (3)$$
$$cl.\varepsilon_{ef}[k] \leftarrow cl.\varepsilon_{ef}[k] + \beta_e(|s_{t+1}[k] - cl.ef[k]| - cl.\varepsilon_{ef}[k]) \quad (4)$$

where $s_{t+1}[k]$ represents the k-th input of state s_{t+1}, i.e., the effect that performing action a_t in state s_t has on the k-th sensory input; as before, $cl.ef[k]$ represents the k-th element of the array ef of classifier cl; likewise $cl.\varepsilon_{ef}[k]$ represents the k-th element of the array ε_{ef} of classifier cl. Equation 3 estimates the average effect that the execution of classifier cl will have on the next k-th sensory input (i.e., on $s_{t+1}[k]$); equation 4, estimates the error affecting $cl.ef[k]$.

7 Experiments on Actions' Effect

We finally test the proposed techniques approach by applying our modified version of XCS to two grid environments: `Woods1` and `Maze4`. The experiments are conducted as follows. First, we apply XCS to the two environments to develop some very small (i.e., very general) solutions. To achieve this we use both action-set subsumption and an additional condensation phase (see [2,5] for details). The parameters are set as in the previous experiments except: $N = 1600$, $\gamma = .7$, *doActionSetSubsumption*$= 1$ (i.e., AS-subsumption was used), $\theta_{nma} = 8$. For `Woods1`, the smallest solution consisted of 33 classifiers; for `Maze4` consisted of 101 classifiers.

As a second step, we compute the exact average values that our method should be able to estimate. For every classifier cl evolved, we consider: (i) the input set I_{cl} of all the sensory inputs that cl matches; (ii) the effect set E_{cl} of all the sensory inputs that appears next to the execution of cl. Given I_{cl} and E_{cl}, for each input k we compute the average of the input values appearing in position k, $ai[k]$, the average of the input values appearing in position k after the execution of cl, $ae[k]$, and the average errors affecting $ai[k]$ and $ae[k]$, i.e., $\varepsilon_{ai}[k]$ and $\varepsilon_{ae}[k]$. More formally:

$$cl.ai[k] = (\sum_{s \in I_{cl}} s[k])/|I_{cl}| \quad (5)$$

$$cl.\varepsilon_{ai}[k] = (\sum_{s \in I_{cl}} |cl.ai[k] - s[k]|)/|I_{cl}| \quad (6)$$

$$cl.ae[k] = (\sum_{s \in E_{cl}} s[k])/|E_{cl}| \quad (7)$$

$$cl.\varepsilon_{ae}[k] = (\sum_{s \in E_{cl}} |cl.ae[k] - s[k]|)/|E_{cl}| \quad (8)$$

The values of $cl.ai[k]$, $cl.\varepsilon_{ai}[k]$, $cl.ae[k]$, and $cl.\varepsilon_{ae}[k]$ are the average of input values for sensor k and the average effect for sensor k, i.e., what our technique should be able to estimate. Thus they serve as reference to compare the quality of the estimates developed through our approach.

Then we apply our version of XCS to `Woods1`. Table 2 reports three classifiers evolved for `Woods1`; due to space constraints, it is not possible to consider here all the 33 classifiers. For each classifier we report on the first line: a unique identifier, the condition and the action separated by a "`:`", the prediction p, the error ε, the fitness F. On the next lines, we report the list I of inputs that the classifier matches and the corresponding list E of inputs that appear after as effect of the classifier execution. Then, we report: (i) the array of average input values, which contains, for every k, the intervals $cl.ai[k] \pm cl.\varepsilon_{ai}[k]$ (Equation 5 and 6); (ii) the array of average effect values, which contains, for every k, the intervals $cl.ae[k] \pm cl.\varepsilon_{ae}[k]$ (Equation 7 and 8), (iii) the array representing the estimated classifier input, which contains, for every k, the intervals $cl.in[k] \pm cl.\varepsilon_{in}[k]$ (Equation 1 and 2); and (iv) the array representing the estimated classifier effect, which contains, for every k, the intervals $cl.ef[k] \pm cl.\varepsilon_{ef}[k]$ (Equation 3 and 4).

The first classifier in Table 2 (`3112676`) is the most general classifiers found; it matches all the possible sensory inputs of `Woods1`, nevertheless is very accurate and has an high fitness. Note that, since classifier `3112677` is very general, the estimate of the matching sensory inputs and the estimate of the classifier effect provide only limited information. The computed values of ai and ε_{ai} suggest that three of the sensory inputs (the second, the third, and the fifth) will be always 0; the computed values of ae and ε_{ae} suggest that no matter in which situation the classifier is applied, in the next state, nine of the inputs will be always zero. If we now consider the estimated input in and the estimated effect ef for the classifier, we find that our technique was successful in finding the same information. All the positions in ai and ae that correspond to zero values, corresponds to zero also in in and ef. While the estimated non zero values of in and ef are quite near to the corresponding computed values.

Some interesting information can be inferred from the non zero values. For instance, the values of $ae[8] = 0.06$ and $\varepsilon_{ae}[8] = 0.12$ suggests that on the subsequent state, the ninth bit will be very likely to be 0. Again, the values estimated through our approach for the same input are quite similar: the values of $ef[8] = 0.04$ and $\varepsilon_{ef}[8] = 0.8$. Note that, in `Woods1` sensory inputs are not visited uniformly, accordingly the estimates evolved through our technique (e.g., in) tend to differ from the computed averages (e.g., ai), since the latter assume a uniform distribution of the inputs.

The second classifier (`3112681`) matches fewer situations, therefore its estimate of classifier input in and its estimate of classifier effect ef provide more information. For instance, the arrays in and ε_{in} of classifier `3112681` suggest that most of the sensory inputs matched do not change; likewise the values in ef and ε_{ef}. Again, we note that for non constant values, the estimates developed

Table 2. Three classifiers payoff evolved by our extended XCS for `Woods1`. For every classifier we report: a unique identifier, the classifier condition and the classifier action separated by ":" , the prediction p, the prediction error ε, and the fitness F, the set I of matched inputs, the set E of inputs that arrive after the classifier execution, the computed average values of the inputs $ai \pm \varepsilon_{ai}$, and of the effects $af\varepsilon_{af}$, the *estimate* input and effect values, $in \pm \varepsilon_{in}$ and $ef \pm \varepsilon_{ef}$.

3112676 ################1:001 $p = 490$ $\varepsilon = 0.00$ $F = 1.00$

input set I	effect set E
1010000000000000	1010000000000000
0000001010000000	1010000000000010
1010000000000010	1010000000000010
0000001110100000	1000000000000010
1000000000000010	0000000000001010
0000000011100000	0000000000000010
0000000000000010	0010100000000000
0000000000110000	0010000000000000
0000000000101100	0000001000000000
0000000000101011	0000101000000000
0000000000001010	0010101000000000
0010000000000000	0010000000000000
0000001000000000	1010000000000000
0000101000000000	0000001010000000
0010101000000000	0010101000000000
0010100000000000	0010100000000000

$ai \pm \varepsilon_{ai}$ $\langle .19 \pm .30, .00 \pm .00, .31 \pm .43, .00 \pm .00, .19 \pm .30, .00 \pm .00, .31 \pm .43, .06 \pm .12,$
$.19 \pm .30, .06 \pm .12, .31 \pm .43, .06 \pm .12, .19 \pm .30, .06 \pm .12, .31 \pm .43, .06 \pm .12\rangle$
$ae \pm \varepsilon_{ae}$ $\langle .31 \pm .43, .00 \pm .00, .62 \pm .47, .00 \pm .00, .31 \pm .43, .00 \pm .00, .31 \pm .43, .00 \pm .00,$
$.06 \pm .12, .00 \pm .00, .00 \pm .00, .00 \pm .00, .06 \pm .12, .00 \pm .00, .31 \pm .43, .00 \pm .00\rangle$

$in \pm \varepsilon_{in}$ $\langle .21 \pm .33, .00 \pm .00, .37 \pm .45, .00 \pm .00, .20 \pm .32, .00 \pm .00, .24 \pm .38, .05 \pm .10,$
$.18 \pm .29, .07 \pm .12, .31 \pm .41, .05 \pm .10, .21 \pm .32, .04 \pm .09, .38 \pm .47, .09 \pm .14\rangle$
$ie \pm \varepsilon_{ie}$ $\langle .34 \pm .45, .00 \pm .00, .71 \pm .43, .00 \pm .00, .36 \pm .45, .00 \pm .00, .25 \pm .39, .00 \pm .00,$
$.04 \pm .08, .00 \pm .00, .00 \pm .00, .00 \pm .00, .09 \pm .14, .00 \pm .00, .32 \pm .42, .00 \pm .00\rangle$

3112681 #############1###:000 $p = 700$ $\varepsilon = 0.00$ $F = 1.00$

input set I	effect set E
0000000000101100	0000000000110000
0000000000101011	0000000000101100
0000000000001010	0000000000101011

$ai \pm \varepsilon_{ai}$ $\langle .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00$
$.00 \pm .00, .00 \pm .00, .67 \pm .44, .00 \pm .00, 1.00 \pm .00, .33 \pm .44, .67 \pm .44, .33 \pm .44\rangle$
$ae \pm \varepsilon_{ae}$ $\langle .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00$
$.00 \pm .00, .00 \pm .00, 1.00 \pm .00, .33 \pm .44, .67 \pm .44, .33 \pm .44, .33 \pm .44, .33 \pm .44\rangle$

$in \pm \varepsilon_{in}$ $\langle .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00$
$.00 \pm .00, .00 \pm .00, .63 \pm .45, .00 \pm .00, 1.00 \pm .00, .27 \pm .40, .73 \pm .40, .37 \pm .46\rangle$
$ef \pm \varepsilon_{ef}$ $\langle .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00$
$.00 \pm .00, .00 \pm .00, 1.00 \pm .00, .25 \pm .38, .75 \pm .38, .27 \pm .40, .49 \pm .49, .49 \pm .49\rangle$

3112677 ################1:111 $p = 1000$ $\varepsilon = 0.00$ $F = 1.00$

input set I	effect set E
0000000000101011	0000000010101000

$ai \pm \varepsilon_{ai}$ $\langle .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00$
$.00 \pm .00, .00 \pm .00, 1.00 \pm .00, .00 \pm .00, 1.00 \pm .00, .00 \pm .00, 1.00 \pm .00, 1.00 \pm .00\rangle$
$ae \pm \varepsilon_{ae}$ $\langle .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00$
$1.00 \pm .00, .00 \pm .00, 1.00 \pm .00, .00 \pm .00, 1.00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00\rangle$

$in \pm \varepsilon_{in}$ $\langle .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00$
$.00 \pm .00, .00 \pm .00, 1.00 \pm .00, .00 \pm .00, 1.00 \pm .00, .00 \pm .00, 1.00 \pm .00, 1.00 \pm .00\rangle$
$ef \pm \varepsilon_{ef}$ $\langle .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00$
$1.00 \pm .00, .00 \pm .00, 1.00 \pm .00, .00 \pm .00, 1.00 \pm .00, .00 \pm .00, .00 \pm .00, .00 \pm .00, \rangle$

---

through our technique are quite reasonable, also considering that the distribution of inputs is not considered in the computation of *ai* and *ae*.

The last classifier (3112677) is the one we discussed in the first example. Although it might appear quite general, it matches only one possible situation, accordingly its estimates are exact. To provide a rough evaluation of the estimates developed with our technique we computed the average error between the computed values for inputs and effect (ai, ε_{ai}, ae, ε_{ae}) and the corresponding estimated values (in, ε_{in}, ef, ε_{ef}). For this purpose we applied our modified version of XCS to Woods1 ten times. For each run we computed:

$$\texttt{error}(in) = \frac{\sum_{cl \in P} \sum_k |cl.ai[k] - cl.in[k]|}{|P| \times n} \tag{9}$$

$$\texttt{error}(\varepsilon_{in}) = \frac{\sum_{cl \in P} \sum_k |cl.\varepsilon_{ai}[k] - cl.\varepsilon_{in}[k]|}{|P| \times n} \tag{10}$$

$$\texttt{error}(ef) = \frac{\sum_{cl \in P} \sum_k |cl.ae[k] - cl.ef[k]|}{|P| \times n} \tag{11}$$

$$\texttt{error}(\varepsilon_{ef}) = \frac{\sum_{cl \in P} \sum_k |cl.\varepsilon_{ae}[k] - cl.\varepsilon_{ef}[k]|}{|P| \times n} \tag{12}$$

where P represents the final population evolved; n is the number of inputs (16 in Woods1). Over ten runs in Woods1, the average error(in) is 0.0054, the average error(ε_{in}) is 0.0035, the average error(ef) is 0.0075, the average error(ε_{ef}) is 0.0041. We applied the same technique to Maze4 with a population size $N = 2000$ for ten runs, measuring an average error(in) of 0.0058, an average error(ε_{in}) of 0.0024, an average error(ef) of 0.0057, an average error(ε_{ef}) of 0.0028. Since Maze4 does not allow many generalization, the prediction of the classifier effect is a little bit more accurate than that obtained for Woods1, in fact in Maze4 the values of error(ef) and error(ε_{ef}) are a little bit smaller. Note also that, these errors measures the difference between *our estimated values* and the *best values* that can be obtained following our approach. Thus, even if the errors are small, still the accuracy of the prediction built is limited by the approach we follow which is less powerful than those methods specifically design for anticipatory behavior.

As the short example presented and the few statistics collected suggest, the method we propose provides limited information about classifiers that apply in many situations; while it provides more precise information on classifier that apply in fewer situations. In either cases, the method appears to be able to provide reliable information about classifier generality and some interesting information about the effect of classifier activation.

8 Extensions and Limitations

The proposed approach is very simple, therefore it is opened to critics and extensions.

Classifier Generality. The first limitation regards the estimates of classifier generality. Our approach, for every input k, develops an interval $in[k] \pm \varepsilon_{in}[k]$ that estimates the range of values that appears at corresponding input. The larger the range, the more general the classifier appears to be with respect to that specific input. Accordingly, we might be tempted to use these intervals to build a subsumption relation for classifier representations that usually do not allow it. Lanzi [4] noted that with symbolic conditions subsumption operators are computationally infeasible. In Sect. 7 we showed that the intervals $in[k] \pm \varepsilon_{in}[k]$ might provide some indication about the classifier generality. Thus, we might try to define a subsumption relation for symbolic conditions using our technique.

Unfortunately this is not possible. As a counter example consider the two conditions defined over a variable `X`$\in \{0 \ldots 9\}$: (i) `(2<X) AND (X<6)`; (ii) `(X<3) AND (X>5)`. Both conditions have the same average input value, $in = 4$, while the associated error ε_{in} is 0.66 for condition (i), 3 for condition (ii). Comparing the two intervals we should argue that condition (ii) is more general than condition (i) since the estimated input interval of condition (ii) subsumes the estimated input interval for condition. But the two conditions form a partition of the input domain and they cannot be compared through an inclusion operator. Thus we cannot define subsumption operators based on the estimated intervals. These in fact are developed from linear estimators, while symbolic conditions usually express non linear relations

Classifier Effect. Our approach tend to provide information on the classifier effect in terms of input values that the system will experience after the classifier activation. Other anticipatory techniques like ACS [1] provides information in term of what will change in the current sensory input after the classifier execution. It is straightforward to extend our technique to provide such kind of information. Given the current input s_t and the next input s_{t+1} we define an array $\Delta[k]$ as follows: $\Delta[k] = 1$ if $s_t[k] \neq s_{t+1}[k]$, 0 otherwise. The array Δ represents the difference that occurred to the current state s_t after the classifier execution. The value $\Delta[k]$ can now be used in Equation 3 and Equation 4 instead of $s_{t+1}[k]$ so that *ef* will now estimate the *changes* in the environment state rather than the next state.

In addition, note that in our approach we also take into account the final state, corresponding to the position with food. Classifier `3112677` in Table 2 has reward 1000 meaning that performing action `111` in state `0000000000101011` will take the agent to the end position, corresponding to state `0000000010101000`. This information might be exploited to perform some *very elementary* planning (as done with Anticipatory Classifier Systems), although, given the basic information provided by the approach is not clear at the moment the true potential of the approach. Alternatively, we might eliminate the effect estimate from classifiers whose action takes the agent to a final state. In this case, planning should focus on reaching of classifiers with a null effect.

9 Summary

We presented a very basic method to estimate the degree of generality of classifier conditions. The method was also applied to obtain *basic* information about the effect of classifier execution. We showed some example of information that we could extract from some very elementary problems. Since the method does not provide an exact anticipation, it is currently difficult to provide some performance metrics to estimate its effectiveness. On the other hand, because of its simplicity, the method can be added virtually on any classifier system model. The code to experiment the approach discussed here is available under GNU Public License at the `xcslib` Web site [6].

References

1. Martin Butz. *Anticipatory Learning Classifier Systems*. Kluwer, 2002.
2. Martin V. Butz and Stewart W. Wilson. An algorithmic description of xcs. *Journal of Soft Computing*, 6(3–4):144–153, 2002.
3. Pierre Géard and Olivier Sigaud. Yacs: Combining dynamic programming with generalization in classifier systems. In Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, *IWLCS*, volume 1996 of *Lecture Notes in Computer Science*, pages 52–69. Springer, 2001.
4. Pier Luca Lanzi. Extending the Representation of Classifier Conditions Part II: From Messy Coding to S-Expressions. In Wolfgang Banzhaf, Jason Daida, Agoston 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.
5. Pier Luca Lanzi. Mining interesting knowledge from data with the xcs classifier system. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 958–965, San Francisco, CA 94104, USA, 7-11 July 2001. Morgan Kaufmann.
6. Pier Luca Lanzi. The xcs library. `http://xcslib.sourceforge.net`, 2002.
7. Pier Luca Lanzi and Alessandro Perrucci. Extending the Representation of Classifier Conditions Part II: From Messy Coding to S-Expressions. In Wolfgang Banzhaf, Jason Daida, Agoston 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, Orlando (FL), July 1999. Morgan Kaufmann.
8. Stewart W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995. http://prediction-dynamics.com/.
9. Stewart W. Wilson. Compact rulesets from xcsi. In Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, *IWLCS*, volume 2321 of *Lecture Notes in Computer Science*, pages 197–210. Springer, 2002.