Structure and Function of the XCS Classifier System

Stewart W. Wilson  
Prediction Dynamics  
Concord, MA  
wilson@prediction-dynamics.com
• Learning machine (program).
• Minimum *a priori*.
• “On-line”.
• Capture regularities in environment.
What does it learn?

To get reinforcements ("rewards", "payoffs")

(Not "supervised" learning—no prescriptive teacher.)
Inputs:
Now binary, e.g., 100101110
—like thresholded sensor values.
Later continuous, e.g., <43.0 92.1 7.4 ... 0.32>

Outputs:
Now discrete decisions or actions,
e.g., 1 or 0 (“yes” or “no”),
“forward”, “back”, “left”, “right”
Later continuous, e.g., “head 34 degrees left”
XCS contains rules (called classifiers), some of which will match the current input. An action is chosen based on the predicted payoffs of the matching rules.

\[ \text{<condition>}:\text{<action>} \Rightarrow \text{<prediction>} \. \]

Example: \(01\#1\#\# : 1 \Rightarrow 943.2\)

Note this rule matches more than one input string:

\[
010100 \\
010110 \\
010101 \\
011111 \\
011100 \\
011100 \\
011101 \\
011110 \\
011111.
\]

This adaptive “rule-based” system contrasts with “PDP” systems such as NNs in which knowledge is distributed.
How does the performance cycle work?

- For each action in [M], classifier predictions $p$ are weighted by fitnesses $F$ to get system’s net prediction in the prediction array.
- Based on the system predictions, an action is chosen and sent to the environment.
- Some reward value is returned.
1. By “updating” the current estimate.

For each classifier $C_j$ in the current $[A]$, 

$$p_j \leftarrow p_j + \alpha(R - p_j),$$

where $R$ is the current reward and $\alpha$ is the learning rate.

This results in $p_j$ being a “recency weighted” average of previous reward values:

$$p_j(t) = \alpha R(t) + \alpha(1-\alpha)R(t-1) + \alpha(1-\alpha)^2R(t-2) + \ldots + (1-\alpha)^t p_j(0).$$

2. And by trying different actions, according to an explore/exploit regime.

A typical regime chooses a random action with probability 0.5.

Exploration (e.g., random choice) is necessary in order to learn anything. But exploitation—picking the highest-prediction action is necessary in order to make best use of what is learned.

There are many possible explore/exploit regimes, including gradual changeover from mostly explore to mostly exploit.
• Usually, the “population” [P] is initially empty. (It can also have random rules, or be seeded.)

• The first few rules come from “covering”: if no existing rule matches the input, a rule is created to match, something like imprinting.

Input: 11000101

Created rule: 1##0010# : 3 => 10
Random #’s and action, low initial prediction.

• But primarily, new rules are derived from existing rules.
Besides its prediction $p_j$, each classifier’s error and fitness are regularly updated.

Error: $\varepsilon_j \leftarrow \varepsilon_j + \alpha(|R - p_j| - \varepsilon_j)$.

Accuracy: $\kappa_j \equiv \varepsilon_j^{-n}$ if $\varepsilon_j > \varepsilon_0$, otherwise $\varepsilon_0^{-n}$

Relative accuracy: $\kappa'_j \equiv \kappa_j / \left( \sum_i \kappa_i \right)$, over [A].

Fitness: $F_j \leftarrow F_j + \alpha(\kappa'_j - F_j)$.

Periodically, a genetic algorithm (GA) takes place in [A].

Two classifiers $C_i$ and $C_j$ are selected with probability proportional to fitness. They are copied to form $C_i'$ and $C_j'$.

With probability $\chi$, $C_i'$ and $C_j'$ are crossed to form $C_i''$ and $C_j''$, e.g.,

$$
\begin{array}{c|c|c|c|c|c}
1 & 0 & # & # & 1 & 1 : 1 \\
# & # & 0 & 0 & 0 & 1 : 1
\end{array}
\Rightarrow
\begin{array}{c|c|c|c|c|c}
1 & 0 & # & # & 1 & 1 : 1 \\
# & 0 & 0 & 0 & 1 & 1 : 1
\end{array}
$$

$C_i''$ and $C_j''$ (or $C_i'$ and $C_j'$ if no crossover occurred), possibly mutated, are added to [P].
Can I see the overall process?

Environment

Match Set [M]

Prediction Array

Action Set [A]

GA

Detectors

Effectors

Update: predictions, errors, fitnesses

Reward

01

"left"

#011 : 01 43 .01 99
11## : 00 32 .13 9
#0## : 11 14 .05 52
001# : 01 27 .24 3
#0#1 : 11 18 .02 92
1#01 : 10 24 .17 15

...etc.

(p)

(p)

(p)

(p)

(p)

(p)

(p)

(p)

(p)

(p)

(p)
They remain in [P], in competition with their offspring.

But two classifiers are deleted from [P] in order to maintain a constant population size.

Deletion is probabilistic, with probability proportional to, e.g.:

- A classifier’s average action set size $a_j$—estimated and updated like the other classifier statistics.

- $a_j/F_j$, if the classifier has been updated enough times, otherwise $a_j/F_{ave}$, where $F_{ave}$ is the mean fitness in [P].

—And other arrangements, all with the aim of balancing resources (classifiers) devoted to each niche ([A]), but also eliminating low fitness classifiers rapidly.
Basic example for illustration: Boolean 6-multiplexer.

\[ 1 0 1 0 0 1 \rightarrow F_6 \rightarrow 0 \]

\[
F_6 = x_0'x_1'x_2 + x_0'x_1x_3 + x_0x_1'x_4 + x_0x_1x_5
\]

\[ l = k + 2^k \quad k > 0 \]

\[
F_{20} = x_0'x_1'x_2'x_3'x_4 + x_0'x_1'x_2'x_3x_5 +
         x_0'x_1'x_2x_3'x_6 + x_0'x_1'x_2x_3x_7 +
         x_0'x_1x_2'x_3'x_8 + x_0'x_1x_2'x_3x_9 +
         x_0'x_1x_2x_3'x_10 + x_0'x_1x_2x_3x_11 +
         x_0x_1'x_2'x_3'x_12 + x_0x_1'x_2'x_3x_13 +
         x_0x_1'x_2x_3'x_14 + x_0x_1'x_2x_3x_15 +
         x_0x_1x_2'x_3'x_16 + x_0x_1x_2'x_3x_17 +
         x_0x_1x_2x_3'x_18 + x_0x_1x_2x_3x_19
\]

\[ 01100010100100001000 \rightarrow 0 \]
What are the results like? — 2
Population at 5,000 problems in descending order of numerosity (first 40 of 77 shown).

<table>
<thead>
<tr>
<th>PRED</th>
<th>ERR</th>
<th>FITN</th>
<th>NUM</th>
<th>GEN</th>
<th>ASIZ</th>
<th>EXPER</th>
<th>TST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.</td>
<td>11</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.00</td>
<td>884.</td>
<td>30</td>
</tr>
<tr>
<td>1.</td>
<td>00</td>
<td>1</td>
<td>#</td>
<td>0</td>
<td>0.00</td>
<td>819.</td>
<td>24</td>
</tr>
<tr>
<td>2.</td>
<td>01</td>
<td>#</td>
<td>1</td>
<td>1000.</td>
<td>.00</td>
<td>856.</td>
<td>22</td>
</tr>
<tr>
<td>3.</td>
<td>01</td>
<td>#</td>
<td>1</td>
<td>0</td>
<td>0.00</td>
<td>840.</td>
<td>20</td>
</tr>
<tr>
<td>4.</td>
<td>11</td>
<td>#</td>
<td>1</td>
<td>0</td>
<td>0.00</td>
<td>719.</td>
<td>20</td>
</tr>
<tr>
<td>5.</td>
<td>00</td>
<td>1</td>
<td>#</td>
<td>1</td>
<td>1000.</td>
<td>.00</td>
<td>698.</td>
</tr>
<tr>
<td>6.</td>
<td>01</td>
<td>#</td>
<td>0</td>
<td>1000.</td>
<td>.00</td>
<td>664.</td>
<td>18</td>
</tr>
<tr>
<td>7.</td>
<td>10</td>
<td>#</td>
<td>1</td>
<td>1000.</td>
<td>.00</td>
<td>712.</td>
<td>18</td>
</tr>
<tr>
<td>8.</td>
<td>00</td>
<td>0</td>
<td>#</td>
<td>1</td>
<td>1000.</td>
<td>.00</td>
<td>674.</td>
</tr>
<tr>
<td>9.</td>
<td>10</td>
<td>#</td>
<td>0</td>
<td>1000.</td>
<td>.00</td>
<td>706.</td>
<td>17</td>
</tr>
<tr>
<td>10.</td>
<td>11</td>
<td>#</td>
<td>0</td>
<td>0</td>
<td>1000.</td>
<td>.00</td>
<td>539.</td>
</tr>
<tr>
<td>11.</td>
<td>10</td>
<td>#</td>
<td>1</td>
<td>0</td>
<td>0.00</td>
<td>638.</td>
<td>16</td>
</tr>
<tr>
<td>12.</td>
<td>01</td>
<td>0</td>
<td>#</td>
<td>1</td>
<td>0.00</td>
<td>522.</td>
<td>15</td>
</tr>
<tr>
<td>13.</td>
<td>00</td>
<td>0</td>
<td>#</td>
<td>1</td>
<td>0.00</td>
<td>545.</td>
<td>14</td>
</tr>
<tr>
<td>14.</td>
<td>10</td>
<td>#</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td>425.</td>
<td>12</td>
</tr>
<tr>
<td>15.</td>
<td>11</td>
<td>#</td>
<td>1</td>
<td>1100.</td>
<td>.00</td>
<td>458.</td>
<td>11</td>
</tr>
<tr>
<td>16.</td>
<td>11</td>
<td>1</td>
<td>#</td>
<td>1000.</td>
<td>.00</td>
<td>233.</td>
<td>6</td>
</tr>
<tr>
<td>17.</td>
<td>0#</td>
<td>0</td>
<td>#</td>
<td>1</td>
<td>0</td>
<td>210.</td>
<td>6</td>
</tr>
<tr>
<td>18.</td>
<td>11</td>
<td>0</td>
<td>1</td>
<td>1000.</td>
<td>.00</td>
<td>187.</td>
<td>5</td>
</tr>
<tr>
<td>19.</td>
<td>01</td>
<td>10</td>
<td>#</td>
<td>1</td>
<td>0.00</td>
<td>168.</td>
<td>4</td>
</tr>
<tr>
<td>20.</td>
<td>11</td>
<td>#</td>
<td>0</td>
<td>0</td>
<td>1000.</td>
<td>.00</td>
<td>114.</td>
</tr>
<tr>
<td>21.</td>
<td>10</td>
<td>1</td>
<td>#</td>
<td>0</td>
<td>0</td>
<td>152.</td>
<td>4</td>
</tr>
<tr>
<td>22.</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td>131.</td>
<td>3</td>
</tr>
<tr>
<td>23.</td>
<td>00</td>
<td>0</td>
<td>#</td>
<td>0</td>
<td>1000.</td>
<td>.00</td>
<td>117.</td>
</tr>
<tr>
<td>24.</td>
<td>11</td>
<td>1</td>
<td>#</td>
<td>0</td>
<td>0.00</td>
<td>68.</td>
<td>3</td>
</tr>
<tr>
<td>25.</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1000.</td>
<td>.00</td>
<td>46.</td>
</tr>
<tr>
<td>26.</td>
<td>10</td>
<td>#</td>
<td>1</td>
<td>1000.</td>
<td>.00</td>
<td>81.</td>
<td>3</td>
</tr>
<tr>
<td>27.</td>
<td>#1</td>
<td>0</td>
<td>#</td>
<td>0</td>
<td>0.00</td>
<td>86.</td>
<td>3</td>
</tr>
<tr>
<td>28.</td>
<td>01</td>
<td>10</td>
<td>#</td>
<td>0</td>
<td>1000.</td>
<td>.00</td>
<td>61.</td>
</tr>
<tr>
<td>29.</td>
<td>01</td>
<td>00</td>
<td>#</td>
<td>0</td>
<td>1000.</td>
<td>.00</td>
<td>58.</td>
</tr>
<tr>
<td>30.</td>
<td>10</td>
<td>0</td>
<td>#</td>
<td>0</td>
<td>1</td>
<td>63.</td>
<td>2</td>
</tr>
<tr>
<td>31.</td>
<td>11</td>
<td>0</td>
<td>#</td>
<td>1</td>
<td>1000.</td>
<td>.00</td>
<td>63.</td>
</tr>
<tr>
<td>32.</td>
<td>00</td>
<td>1</td>
<td>#</td>
<td>0</td>
<td>1000.</td>
<td>.00</td>
<td>77.</td>
</tr>
<tr>
<td>33.</td>
<td>10</td>
<td>#</td>
<td>1</td>
<td>0</td>
<td>0.00</td>
<td>93.</td>
<td>2</td>
</tr>
<tr>
<td>34.</td>
<td>11</td>
<td>#</td>
<td>1</td>
<td>1000.</td>
<td>.00</td>
<td>59.</td>
<td>2</td>
</tr>
<tr>
<td>35.</td>
<td>01</td>
<td>#</td>
<td>1</td>
<td>1000.</td>
<td>.00</td>
<td>75.</td>
<td>2</td>
</tr>
<tr>
<td>36.</td>
<td>01</td>
<td>#</td>
<td>0</td>
<td>1</td>
<td>1000.</td>
<td>.00</td>
<td>36.</td>
</tr>
<tr>
<td>37.</td>
<td>11</td>
<td>#</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>92.</td>
<td>2</td>
</tr>
<tr>
<td>38.</td>
<td>10</td>
<td>#</td>
<td>0</td>
<td>1</td>
<td>703.</td>
<td>.31</td>
<td>8</td>
</tr>
<tr>
<td>39.</td>
<td>#1</td>
<td>1</td>
<td>#</td>
<td>0</td>
<td>856.</td>
<td>.22</td>
<td>11</td>
</tr>
</tbody>
</table>

What are the results like?—3
Can you show the evolution of a rule?

Action sets $[A]$ for input 101001 and action 0 at several epochs.

<table>
<thead>
<tr>
<th>PRED</th>
<th>ERR</th>
<th>FITN</th>
<th>NUM</th>
<th>GEN</th>
<th>ASIZ</th>
<th>EXPER</th>
<th>TST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>#0</td>
<td>#0</td>
<td>1</td>
<td>1</td>
<td>.50</td>
<td>10.6</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>1#</td>
<td>0#</td>
<td>0</td>
<td>1</td>
<td>.67</td>
<td>10.9</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>1#</td>
<td>#0</td>
<td>6</td>
<td>1</td>
<td>.50</td>
<td>22.0</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>1#</td>
<td>0#</td>
<td>8</td>
<td>2</td>
<td>.33</td>
<td>14.3</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>1#</td>
<td>1#</td>
<td>0</td>
<td>2</td>
<td>.33</td>
<td>18.4</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PRED</th>
<th>ERR</th>
<th>FITN</th>
<th>NUM</th>
<th>GEN</th>
<th>ASIZ</th>
<th>EXPER</th>
<th>TST</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1#</td>
<td>1#</td>
<td>2</td>
<td>5</td>
<td>.33</td>
<td>18.4</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>1#</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1000</td>
<td>.000</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>1#</td>
<td>0#</td>
<td>0</td>
<td>0</td>
<td>1000</td>
<td>.000</td>
<td>2</td>
</tr>
</tbody>
</table>

Can you show the evolution of a rule?
Consider two classifiers C1 and C2 having the same action, and let C2 be a generalization of C1. That is, C2 can be obtained from C1 by changing some non-# alleles in the condition to #’s. Suppose that C1 and C2 are equally accurate. They will therefore have the same fitness. However, note that, since it is more general, C2 will occur in more action sets than C1. What does this mean? Since the GA acts in the action sets, C2 will have more reproductive opportunities than C1. This edge in reproductive opportunities will cause C2 to gradually drive C1 out of the population.

Example:

\[
C1: \quad 1 \ 0 \ # \ 0 \ 0 \ 1 : \ 0 \ \Rightarrow \quad 1000 \quad .001 \quad 920
\]
\[
C2: \quad 1 \ 0 \ # \ # \ 0 \ # : \ 0 \ \Rightarrow \quad 1000 \quad .001 \quad 920
\]

C2 has equal fitness but more reproductive opportunities than C1.

C2 will “drive out” C1.
Does XCS scale up?
20m ~5x harder than 11m
11m ~5x harder than 6m.

⇒ $D = cg^p$,

where $D$ = “difficulty”, here learning time,
  $g$ = number of maximal generalizations,
  $p$ = a power, about 2.3
  $c$ = a constant about 3.2

Thus “$D$ is polynomial in $g$”.

What is $D$ with respect to $l$, string length?

For the multiplexers, $l = k + 2^k$,
or $l \to 2^k$ for large $k$.

But $g = 4 \cdot 2^k$, thus $l \sim g$,
So that “$D$ is polynomial in $l$” (not exponential).
What about deferred reward?

Apply ideas from multi-step reinforcement learning.

Need the *action-value* of each action in each state.

What is the action-value of a state more than one step from reward?

Intuitive sketch:

\[
p_j \leftarrow p_j + \alpha \left[ (r_{imm} + \gamma \max_{a'} P(x',a')) - p_j \right]
\]

where \(p_j\) is the prediction of a classifier in the current action set \([A]\),

\(x'\) and \(a'\) are the next state and possible actions,

\(P(x',a')\) is a system prediction at the next state,

and \(r_{imm}\) is the current external reward.
• Previous action set \([A]_{-1}\) is saved and updates are done there, using the current prediction array for “next state” system predictions.

• On the last step of a problem, updates occur in \([A]\).
What are the results like?—

- Animat senses the 8 adjacent cells.
  
  \[
  F \ b \ b \\
  O * b \\
  Q \ b \ b 
  \]

- Coding of each object:
  
  \[
  F = 110 \text{“food1”} \\
  G = 111 \text{“food2”} \\
  O = 010 \text{“rock1”} \\
  Q = 011 \text{“rock2”} \\
  b = 000 \text{“blank”} 
  \]

- “Sense vector” for above situation: 00000000000000011010110

- A matching classifier: ####0#00####00001##101## : 7
Two generalizations discovered by XCS in Woods1.

(Food = 11 = “Tasty”, “Opaque”
Rock = 01 = “Blend”, “Opaque”
Blank = 00 = “Blend”, “Clear”)

<table>
<thead>
<tr>
<th>Action</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear W</td>
<td>504</td>
</tr>
<tr>
<td>Opaque W</td>
<td>710</td>
</tr>
</tbody>
</table>
Inputs:
\[ \langle\langle x_1 \pm \Delta x_1, ..., x_n \pm \Delta x_n\rangle\rangle : \langle\text{action}\rangle \Rightarrow p \]

Actions:
\[ \langle\langle x_1 \pm \Delta x_1, ..., x_n \pm \Delta x_n\rangle\rangle : \langle a \pm \Delta a\rangle \Rightarrow p \]
— and combine matching rules à la fuzzy logic, perhaps.

Time:
\[ \langle\langle x_1 \pm \Delta x_1, ..., x_n \pm \Delta x_n\rangle\rangle : \langle a \pm \Delta a\rangle \Rightarrow \frac{dp}{dt} \]
— action selection based on steepest ascent of \( p \).
Example (McCallum’s Maze):

* Aliased states. Optimal action not determinable from current sensory input.

Approaches:

- “History window” — remember previous inputs
- Search for correlation with past input events
- Adaptive internal state:

```
Environmental condition   External action   Prediction
  ###  ##  #1  ##  0#  ###  ###  ###  #  : 1  0  ⇒  504
```

<table>
<thead>
<tr>
<th>Environmental condition</th>
<th>External action</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td># # # # # # 1 # # 0 # # # # # # # # #</td>
<td>1 0</td>
<td>504</td>
</tr>
</tbody>
</table>
Example: “if \( x > y \) for any \( x \) and \( y \), and action \( a \) is taken, payoff is predicted to be \( p \).”

Cannot be represented using a single classifier with traditional conjunctive condition, since it’s a relation.

However, it can be represented using an “s-classifier”:

\((> x \ y) \ : \ <\text{action } a> \Rightarrow p\)

e.g., a classifier whose condition is a Lisp s-expression.

With appropriate elementary functions, s-classifiers can encode an almost unlimited variety of conditions.

They can be evolved using techniques of genetic programming.
Rule-based, not PDP (“parallel distributed processing”)

- Structure is created as needed
- Learning may often be faster because classifiers are inherently non-linear
- Learning complexity may be less than most PDPs
- Classifiers can keep and use statistics; difficult in a network
- Hierarchy and reasoning may be easier, since knowledge is in subroutine-like packages