Mining Oblique Data with XCS

Stewart W. Wilson

The University of Illinois, Urbana-Champaign IL 61801, USA
Prediction Dynamics, Concord MA 01742, USA
wilson@prediction-dynamics.com

Abstract. The classifier system XCS was investigated for data mining applications where the dataset discrimination surface (DS) is generally oblique to the attribute axes. Despite the classifiers' hyper-rectangular predicates, XCS reached 100% performance on synthetic problems with diagonal DS's and, in a train/test experiment, competitive performance on the Wisconsin Breast Cancer dataset. Final classifiers in an extended WBC learning run were interpretable to suggest dependencies on one or a few attributes. For data mining of numeric datasets with partially oblique discrimination surfaces, XCS shows promise from both performance and pattern discovery viewpoints.

1 Introduction

Data mining has been described as "the process of discovering patterns in data" [17]. The discovered patterns are often represented by rules that, given a data instance, can be used to predict an outcome or consequence of interest. Similarly, XCS [13], a learning classifier system, evolves rules (classifiers) through which the system gradually improves its ability to obtain environmental reward. In effect, XCS *mines its environment* for patterns that, expressed in classifiers, allow it to make increasingly remunerative decisions. XCS is potentially applicable to data mining problems because in many environments it evolves accurate, maximally general rules in which the patterns are easily seen.

Some of the relevant XCS work has involved learning Boolean functions [13, 5, 14]. From a data-mining point of view, a Boolean function is a very strongly patterned dataset containing all two-valued attribute strings of a given length. The ability to learn Boolean functions indicates XCS's ability to find highly non-linear patterns among attributes. Saxon and Barry [9] tested XCS on The Monk's Problems [11], a widely used non-Boolean test set for comparing learning algorithms. The domain has six attributes taking up to four nominal values (e.g., the `head_shape` attribute takes values `round`, `square`, and `octagon`). The Monk's 1 problem is a relatively simple concept,

```
(head_shape = body_shape) or (jacket_color = red).
```

The second problem is a more complicated concept not simply described by a disjunction of conjunctions of attribute-value pairs. Monk's 3 is again a simple concept but contains noise in the form of approximately 5% misclassified examples in the training set. Comparing the results with published results for other systems led Saxon and Barry to conclude that on The Monk's Problems XCS "performed at least as well as traditional Machine Learning techniques", and that XCS was "also able to produce and maintain an easily identifiable accurately general set of classifiers representing the concepts within the data sets". Wilson [15] showed that XCS could learn a problem in which there was a Boolean relation among real-valued attributes, showing that XCS could be adapted to take continuous inputs.

Promising as these results are, in all the problems addressed the concept or function sought is essentially "logical". I.e., it can be expressed by a combination of `and`, `or`, and `not` applied to attribute values, simple recodings of them, or, in the case of [15], ranges of the values. XCS appears well suited to "logical" data-mining problems, including realistically large ones (as suggested by preliminary complexity results in [14]). However, in many data-mining problems the function sought is *not* expressible logically. This is because the problem's discrimination surface (DS) is *oblique*—neither parallel nor perpendicular to—the attribute axes. Datasets with oblique DS's form a large class and their instances typically have numeric attributes.

XCS is good at logical problems because the classifier conditions define hyper-rectangles, which can evolve to fit the shape of a logical DS exactly. However, if the DS is oblique, hyper-rectangles can only approximate the shape. Many classifiers may be required, and the shape may be difficult to see in the classifiers. One approach to oblique problems is to modify the syntax of classifier conditions so that oblique DS's can be approximated. The most general method is to use *S-classifiers* [7, 12], where the conditions are Lisp S-expression predicates and can be based on arithmetic function primitives. Our aim in this paper, however, is to apply basic XCS (modified for numeric inputs) to oblique data, using the well-known Wisconsin Breast Cancer dataset [2] as test bed. The WBC dataset offers an opportunity to test XCS's performance and the interpretability of its evolved rules on a realistic, oblique, data-mining problem.

The next section gives a brief introduction to XCS. Section 3 presents modifications needed for the WBC problem. We warm up on some synthetic data in Section 4. Section 5 presents results on the WBC dataset, including a stratified cross-validation (train/test) experiment, and an experiment that reveals patterns. Sections 6 and 7 contain discussion and conclusion.

2 XCS in Brief

The following, drawn from [15], is an abbreviated description of XCS. (For more detail, see [13], [5], and [14]; [3] gives an algorithmic description.) XCS is designed for both single- and multiple-step tasks, but the discussion here applies only to XCS for independent single-step tasks in which an input is presented, the system makes a decision, and the environment provides some reward.

Structurally, each classifier C_j in XCS's population [P] has a *condition*, an *action*, and a set of associated parameters. The condition is a string from {0,1,#}; the action is an integer. The three principal parameters are: (1) *payoff prediction*

p_j, which estimates the payoff the system will receive if C_j matches and its action is chosen by the system; (2) *prediction error* ϵ_j, which estimates the error in p_j with respect to actual payoffs received; and (3) *fitness* F_j, computed as later explained. It is convenient to divide the description of a single operating cycle or time-step into the traditional performance, update (reinforcement), and discovery components.

2.1 Performance

Upon presentation of an input—a binary string—XCS forms a *match set* [M] of classifiers in [P] whose conditions are satisfied by the input. The condition is satisfied if and only if each of its non-# positions is the same as the corresponding bit of the input string.

[If no classifiers match, the input is "covered" by creating matching classifiers with each of the possible actions, and placing them in [M]. Each new classifier's condition is a copy of the input except that #'s ("don't cares") are inserted with probability $P_\#$ per position. Parameters are given initial values. If [P] has reached its allowable maximum N, deletion of classifiers occurs to make room for the new ones (deletion is generally not necessary; initial populations are empty and covering normally occurs only at the very beginning of a run)].

Then, for each action a_k represented in [M], the system computes a fitness-weighted average P_k of the predictions p_j of each classifier in [M] having that action: $P_k = \sum_j F_j p_j / \sum_j F_j$. P_k is termed the *system prediction* for action a_k.

Next, XCS chooses an action from those represented in [M] and sends it to the environment. According to the *action-selection* regime in force, the action may be picked randomly or otherwise probabilistically based on the P_k, or it may be picked deterministically—i.e., the action with the highest P_k is chosen. Finally, an *action set* [A] is formed consisting of the subset of [M] having the chosen action.

2.2 Update

In this component, the parameters of the classifiers in [A] are re-estimated according to the *reward* R returned by the environment as a consequence of the system's taking action a_k. First, the predictions are updated: $p_j \leftarrow p_j + \beta(R - p_j)$. Next, the errors: $\epsilon_j \leftarrow \epsilon_j + \beta(|R - p_j|)$. Third, for each C_j, an *accuracy* κ_j is computed: $\kappa_j = 0.1(\epsilon_j/\epsilon_0)^{-n}$, for $\epsilon_j > \epsilon_0$, else 1.0. Then, from the κ_j, each classifier's *relative accuracy* κ'_j is computed: $\kappa'_j = \kappa_j / \sum_j \kappa_j$. Finally the fitnesses F_j are updated according to: $F_j \leftarrow F_j + \beta(\kappa'_j - F_j)$.

2.3 Discovery

On some time-steps, XCS executes a *genetic algorithm* (GA) within [A]. Two classifiers are chosen probabilistically based on their fitnesses and copied. The copies are crossed (two-point crossover) with probability χ, and then mutated with probability μ per allele. The resulting offspring are inserted into [P]; if the population size is already at its maximum value, N, two classifiers are deleted. The probability of deletion of a classifier is determined by Kovacs's method [6] and is designed to preferentially remove low-fitness classifiers that have participated in a threshold number of action sets—that is, have had sufficient time for their parameters to be accurately estimated.

Whether or not to execute the GA on a given time-step is determined as follows. The system keeps a count of the number of time-steps since the beginning of a run. Every time a GA occurs, the classifiers in that [A] are "time-stamped" with the current count. Whenever an [A] is formed, the time-stamp values of its members are averaged and subtracted from the current count; if the difference exceeds a threshold θ_{GA}, a GA takes place.

A *macroclassifier* technique is used to speed processing and provide a more perspicuous view of population contents. Whenever a new classifier is generated by the GA (or covering), [P] is scanned to see if there already exists a classifier with the same condition and action. If so, the *numerosity* parameter of the existing classifier is incremented by one, and the new classifier is discarded. If not, the new classifier is inserted into [P]. The resulting population consists entirely of structurally unique classifiers, each with numerosity ≥ 1. If a classifier is chosen for deletion, its numerosity is decremented by 1, unless the result would be 0, in which case the classifier is removed from [P]. All operations in a population of macroclassifiers are carried out as though the population consisted of conventional classifiers; that is, the numerosity is taken into account. In a macroclassifier population, the sum of numerosities equals N, the traditional population size. [P]'s actual size in macroclassifiers, M, is of interest as a measure of the population's space complexity.

2.4 Overall Picture

XCS's overall objective is to maximize the rate of reward it obtains from the environment. If the environment (or trainer, teacher, etc.) gives high reward for actions (decisions) that it regards as "correct" or "best", then XCS's achieving a high reward rate will be equivalent to learning or solving the problem presented by the environment. This is the framework of *reinforcement learning* [10]. Data mining problems can be cast in terms of reinforcement learning if rewards given the system correlate with the correctness of its decisions.

XCS is designed to evolve classifiers that accurately predict environmental reward in all the situations (inputs) the system might encounter. Having these it can choose the actions that maximize reward. XCS evolves accurate classifiers by a process of trial and error in which, given an input and a match set, a particular action is tried, the reward (if any) is received, and the associated classifiers are updated as previously described. Classifiers that predict accurately tend to be reproduced, while inaccurate classifiers are eventually deleted.

XCS not only evolves accurate classifiers, it evolves ones that are accurate and *maximally general*. That is, if several *different* input situations result in the same reward when a particular action is made by the system, then XCS will tend to evolve a *single* (macro)classifier having that action that matches all of those situations. As a result, the system achieves compactness of representation, and the evolved classifiers display the structure of the environment or problem in a way that is often readily interpretable by users. This important tendency of XCS occurs because if there are two equally accurate classifiers but one matches a subset of the states matched by the other, the more general classifier will win out since it occurs in more action sets and will have more reproductive opportunities [13, 5, 14, 16].

2.5 Subsumption Deletion

The generalization mechanism just mentioned, while powerful, may sometimes not go as far as one would wish. Winning classifiers will be those that match more inputs—environmental *states*—than equally accurate rivals. Usually, a classifier whose condition is formally more general (has more #'s) will match more states. However, in certain environments (termed *sparse*), not all states may actually be present, with the result that some formally more general classifiers, while still perfectly accurate, cannot "drive out" their rivals. The resulting population will be larger than is strictly necessary to solve the problem, and its knowledge will not be as perspicuous. To deal with sparseness and evolve classifiers that are as formally general as possible without sacrificing accuracy, an optional procedure called "subsumption deletion" was introduced [14, 16, 3].

Two forms of subsumption deletion are used. In *GA subsumption*, a new offspring is examined to see if its condition is logically subsumed by the condition of either of its parents. If so, and if the subsuming parent is both highly accurate and sufficiently experienced (its parameters have been updated a threshold number of times), the offspring is not added to the population but the parent's numerosity is incremented. *Action-set subsumption* is different from and independent of GA subsumption. Each action set is searched for the most general classifier that is both accurate and sufficiently experienced. Then all other classifiers in the set are tested against the general one to see if it subsumes them. Any that are subsumed are deleted from the population.

3 XCSI: XCS Modified for Integer Inputs

The attributes of WBC dataset instances are integer-valued, from 1 to 10 inclusive. To handle this format, XCS was modified in its input interface, the mutation operator, covering, and subsumption. For convenience, we call the modified system "XCSI". The changes are analogous to those made for real-valued inputs in XCSR [15]. The classifier condition is changed from a string from {0,1,#} to a concatenation of "interval predicates", $int_i = (l_i, u_i)$, where l_i ("lower") and u_i ("upper") are integers. A classifier matches an input x with attributes x_i if and only if $l_i \leq x_i \leq u_i$ for all x_i. A consequence of using interval predicates is that the number of numerical values or alleles in the condition is twice the number of components in x.

Crossover (two-point) operates in direct analogy to crossover in XCS. A crossover point can occur between any two alleles, i.e., within an interval predicate or between predicates, and also at the ends of the condition (the action is not involved in crossover). Mutation, however, is different. Preliminary experiments were done, and the best method appears to be to mutate an allele by adding an amount $\pm rand(m_0)$, where m_0 is a fixed integer, $rand$ picks an integer uniform randomly from $(0, m_0]$, and the sign is chosen uniform randomly. If a new value of l_i is less than the minimum possible input value, in this case 1, the new value is set to 1. If the new value is greater than u_i, it is set equal to u_i. A corresponding rule holds for mutations of u_i.

The condition of a "covering" classifier has components $l_0, u_0, ..., l_n, u_n$, where each $l_i = x_i - rand_1(r_0)$, but limited to the minimum possible input value, and each $u_i = x_i + rand_1(r_0)$, limited to the maximum possible input value; $rand_1$ picks a random integer from $[0, r_0]$, with r_0 a fixed integer.

Subsumption of one classifier by another occurs if every interval predicate in the first classifier's condition subsumes the corresponding predicate in the second classifier's condition. An interval predicate subsumes another one if its l_i is less than or equal to that of the other and its u_i is greater than or equal to that of the other. For purposes of action-set subsumption, a classifier is more general than another if its *generality* is greater. Generality is defined as the sum of the widths $u_i - l_i + 1$ of the interval predicates, all divided by the maximum possible value of this sum.

4 Experiments on Synthetic Oblique Data

To get a feel for its behavior, we began by applying XCSI to some synthetic oblique data. In the first experiment the problem had just two dimensions, in order to give a clear picture of the resulting classifiers, In the other experiment, the problem had the same dimensionality as the WBC dataset—nine. In both cases the DS (discrimination surface) was diagonal to all the axes, i.e., maximally oblique.

4.1 Experiment 1—Two Dimensions

A dataset, "Random-Data2", was constructed as follows. Random vectors (x_1, x_2) were created, with each x_i a random integer from (1, 10). The correct outcome o for each vector was chosen according to

$$o = \texttt{ if } x_1 + x_2 \geq 11 \texttt{ then } 1 \texttt{ else } 0.$$

Each vector and its outcome constituted an instance of Random-Data2. The dataset had 683 such instances (comparable to the size of the WBC dataset), and the attribute range from 1 through 10 corresponded to the WBC's attribute range. Note that the DS for this problem is indeed diagonal to the x_1 and x_2 axes.

In the experiment, XCSI operated in a *pure explore* mode in which on each time-step the system was given a randomly selected instance from Random-Data2, it carried out the performance, update, and discovery procedures as explained in Sections 2 and 3, but its action selection rule was to choose its output action (its decision as to the correct outcome) randomly. If the action was correct, XCSI received a reward of 1000; if incorrect, 0. Both forms of subsumption deletion were enabled.

To determine whether the system was actually learning something, the explore steps alternated with *test* steps in which XCSI deliberately chose as its decision the action with the maximum system prediction. During test steps the update, discovery, and subsumption components were disabled.

The following parameters of XCSI were the same for all experiments in this paper: $\beta = 0.2$, $\epsilon_0 = 1$, $n = 5$, $\theta_{GA} = 48$, $\chi = 0.8$, $\mu = 0.04$, and $m_0 = 2$. Also the same were the experience threshold for deletion and for GA subsumption, $\theta_{del} = 50$ (*experience* is the number of times a classifier has been in an updated action set) and a separate experience threshold for action-set subsumption, $\theta_{as} = 1000$. The space of parameter settings is obviously large, and a number of experiments preceded those reported here. Principally, it was found that ϵ_0 should be quite small (to push toward high accuracy) and θ_{as} should be quite large (to ensure that a classifier's accuracy is reliable before allowing it to delete another classifier in action-set subsumption). Parameters for this experiment that were not the same through all experiments were $N = 800$ and $r_0 = 4$.

Figure 1 shows results for a typical run of XCSI on Random-Data2. The quantities are all moving averages over the past 50 problems (instances seen); values are plotted every 100 problems. Individual plots appear in the top-to-bottom order shown in the legend.

Approximately 100,000 explore problems (instances) were required for the system to reach 100% performance. However, 98% performance (one error in 50 problems) was reached quickly, so most of the time was needed to get the last few instances—usually just one or two—right. "Generality" is a fitness-weighted average of classifier generality. It climbs slowly and seems to level off at around 0.5. "Popsize/800" is the number of macroclassifiers divided by N, the total numerosity. It falls quickly to approximately 0.08. The actual number of classifiers at the end was 63. Finally, "System Error" is the difference (divided by 1000) between the system prediction for the chosen action and the actual reward received. It falls quickly to a very small value.

Figure 2 depicts 29 representative classifiers from the 63 in the final population of Experiment 1, ordered by descending prediction. A graphic notation is used to represent the conditions. In each interval predicate (shown between "|"s), the range of input values accepted is indicated by a sequence of `O`'s. Thus the first classifier accepts (matches) input vectors for which $7 \leq x_1 \leq 10$ and $4 \leq x_2 \leq 10$.

Notice that the first classifier is always correct in its prediction: all accepted inputs satisfy $x_1 + x_2 \geq 11$, the classifier advocates action 1, and the reward received is indeed 1000. Furthermore, the classifier is maximally general: neither

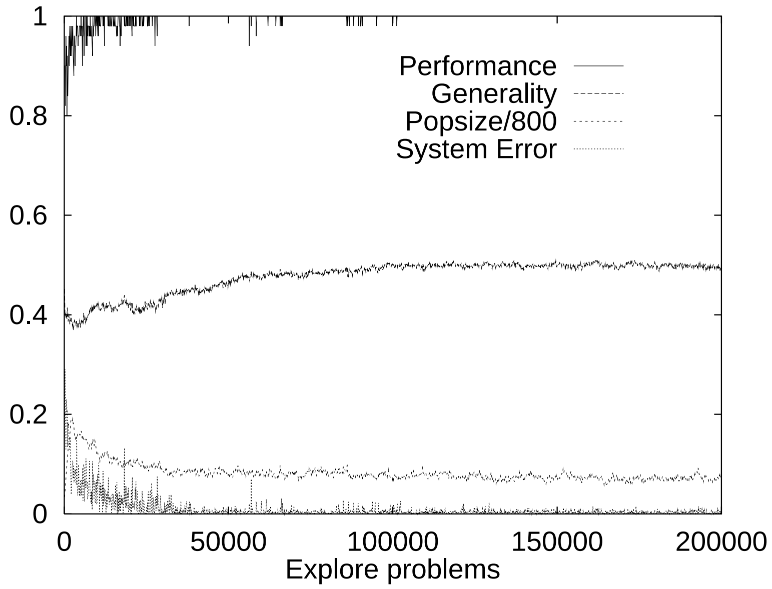

Fig. 1. Performance (fraction correct), generality, population size (/800), and system error vs. number of explore problems in Experiment 1.

of its predicates can be expanded without causing an error. For the second classifier, all accepted inputs satisfy $x_1 + x_2 < 11$, it advocates action 0, and it is completely accurate predicting a reward of 1000. It, too, is maximally general. In fact all classifiers in the first group are accurate (error 0.000) and maximally general except no. 12 which is merely accurate.

The second group represents the 13 classifiers that were not accurate. Note that this is reflected in their very small fitnesses (all fitness values are shown multiplied by 1000). Because system predictions are a fitness-weighted sum (Section 2.1), low-fitness classifiers have minimal effect on the system, and are soon deleted. They represent GA offspring that turned out not to be accurate.

The classifier conditions in the first and third groups can be thought of as "boxes"—hyper-rectangles in 2-d—fitting into the space on either side of the line defined by the dataset's DS. The fit is awkward: apart from nos. 12 and 55, each box is as big as it can be without causing an error, but they overlap in such a way that none subsumes any other and all survive. In effect, the boxes are trying to approximate the diagonally divided input space but can't do it elegantly because they are rectangular. Despite this frustration, the system did its job: of the 63 classifiers in the set, all with substantial fitness (and thus able to affect the system prediction) were accurate, and all of these except nos. 12 and 55 were maximally general.

Still, this experiment's input space had only two dimensions. Despite the system's high performance and relatively small population (space complexity) in this case, what can be expected for more realistic data mining problems such

<table>
<tr><th></th><th>CONDITION</th><th>ACT</th><th>PRED</th><th>ERR</th><th>FITN</th><th>NUM</th><th>GEN</th><th>EXPER</th></tr>
<tr><td>0.</td><td>|......0000|...0000000|</td><td>1</td><td>1000.</td><td>.000</td><td>282.</td><td>15</td><td>.55</td><td>29435</td></tr>
<tr><td>1.</td><td>|000000....|0000......|</td><td>0</td><td>1000.</td><td>.000</td><td>359.</td><td>26</td><td>.50</td><td>22546</td></tr>
<tr><td>2.</td><td>|0000......|000000....|</td><td>0</td><td>1000.</td><td>.000</td><td>153.</td><td>13</td><td>.50</td><td>22780</td></tr>
<tr><td>3.</td><td>|00000.....|00000.....|</td><td>0</td><td>1000.</td><td>.000</td><td>358.</td><td>19</td><td>.50</td><td>23150</td></tr>
<tr><td>4.</td><td>|0000000...|000.......|</td><td>0</td><td>1000.</td><td>.000</td><td>295.</td><td>19</td><td>.50</td><td>18197</td></tr>
<tr><td>5.</td><td>|....000000|.....00000|</td><td>1</td><td>1000.</td><td>.000</td><td>150.</td><td>12</td><td>.55</td><td>29394</td></tr>
<tr><td>6.</td><td>|000000000.|0.........|</td><td>0</td><td>1000.</td><td>.000</td><td>435.</td><td>28</td><td>.50</td><td>7924</td></tr>
<tr><td>7.</td><td>|.......000|..00000000|</td><td>1</td><td>1000.</td><td>.000</td><td>250.</td><td>23</td><td>.55</td><td>26022</td></tr>
<tr><td>8.</td><td>|00000000..|00........|</td><td>0</td><td>1000.</td><td>.000</td><td>236.</td><td>16</td><td>.50</td><td>12647</td></tr>
<tr><td>9.</td><td>|........00|.000000000|</td><td>1</td><td>1000.</td><td>.000</td><td>229.</td><td>29</td><td>.55</td><td>17336</td></tr>
<tr><td>10.</td><td>|.........0|0000000000|</td><td>1</td><td>1000.</td><td>.000</td><td>429.</td><td>22</td><td>.55</td><td>9401</td></tr>
<tr><td>11.</td><td>|..00000000|.......000|</td><td>1</td><td>1000.</td><td>.000</td><td>213.</td><td>10</td><td>.55</td><td>19615</td></tr>
<tr><td>12.</td><td>|0.........|......000.|</td><td>0</td><td>1000.</td><td>.000</td><td>515.</td><td>29</td><td>.20</td><td>1658</td></tr>
<tr><td>13.</td><td>|00........|00000000..|</td><td>0</td><td>1000.</td><td>.000</td><td>501.</td><td>24</td><td>.50</td><td>12346</td></tr>
<tr><td>25.</td><td>|0000000...|0000......|</td><td>0</td><td>991.</td><td>.032</td><td>0.</td><td>1</td><td>.55</td><td>46</td></tr>
<tr><td>26.</td><td>|0000......|0000000...|</td><td>0</td><td>990.</td><td>.037</td><td>0.</td><td>1</td><td>.55</td><td>41</td></tr>
<tr><td>27.</td><td>|000000....|00000.....|</td><td>0</td><td>966.</td><td>.081</td><td>5.</td><td>3</td><td>.55</td><td>80</td></tr>
<tr><td>28.</td><td>|00........|000000000.|</td><td>0</td><td>930.</td><td>.135</td><td>0.</td><td>1</td><td>.55</td><td>51</td></tr>
<tr><td>29.</td><td>|000.......|00000000..|</td><td>0</td><td>800.</td><td>.161</td><td>0.</td><td>2</td><td>.55</td><td>39</td></tr>
<tr><td>30.</td><td>|...0000000|...0000000|</td><td>1</td><td>754.</td><td>.308</td><td>3.</td><td>1</td><td>.70</td><td>16</td></tr>
<tr><td>47.</td><td>|00000.....|00000.....|</td><td>1</td><td>0.</td><td>.000</td><td>202.</td><td>15</td><td>.50</td><td>21652</td></tr>
<tr><td>48.</td><td>|........00|.000000000|</td><td>0</td><td>0.</td><td>.000</td><td>232.</td><td>20</td><td>.55</td><td>17411</td></tr>
<tr><td>49.</td><td>|.........0|0000000000|</td><td>0</td><td>0.</td><td>.000</td><td>284.</td><td>23</td><td>.55</td><td>10208</td></tr>
<tr><td>50.</td><td>|....000000|.....00000|</td><td>0</td><td>0.</td><td>.000</td><td>407.</td><td>32</td><td>.55</td><td>27947</td></tr>
<tr><td>51.</td><td>|..00000000|.......000|</td><td>0</td><td>0.</td><td>.000</td><td>345.</td><td>8</td><td>.55</td><td>21179</td></tr>
<tr><td>52.</td><td>|000.......|0000000...|</td><td>1</td><td>0.</td><td>.000</td><td>251.</td><td>27</td><td>.50</td><td>18306</td></tr>
<tr><td>53.</td><td>|0000000...|000.......|</td><td>1</td><td>0.</td><td>.000</td><td>188.</td><td>20</td><td>.50</td><td>16057</td></tr>
<tr><td>54.</td><td>|.....00000|....000000|</td><td>0</td><td>0.</td><td>.000</td><td>234.</td><td>20</td><td>.55</td><td>25685</td></tr>
<tr><td>55.</td><td>|0.........|......000.|</td><td>1</td><td>0.</td><td>.000</td><td>556.</td><td>28</td><td>.20</td><td>1735</td></tr>
</table>

Fig. 2. Some classifiers from Experiment 1. Condition predicates shown graphically (see text). Also shown: ACTion, PREDiction, ERRor, FITNess, NUMerosity, GENerality, and EXPERience of each classifier.

as the WBC with its nine dimensions? Will XCSI's "boxes" approximation be adequate, and what insight will it give into the problem's structure? As a first step in answering these questions, we tested XCSI on a nine-dimensional problem with a "diagonal" DS.

4.2 Experiment 2—Nine Dimensions

A second random dataset, "Random-Data9", was constructed like Random-Data2, except that there were nine dimensions and the expression determining the outcomes was

$$o = \texttt{ if } x_1 + ... + x_9 \geq 50 \texttt{ then } 1 \texttt{ else } 0.$$

The dataset contained 683 examples, 361 of which had outcome 1.

Experiment 2 was conducted like Experiment 1 except that $N = 3200$ and $r_0 = 6$. Larger values for these parameters are generally needed as the input space becomes larger. In general, they are chosen empirically such that the total number of classifiers does not exceed approximately $0.75N$ and any initial covering ceases quickly.

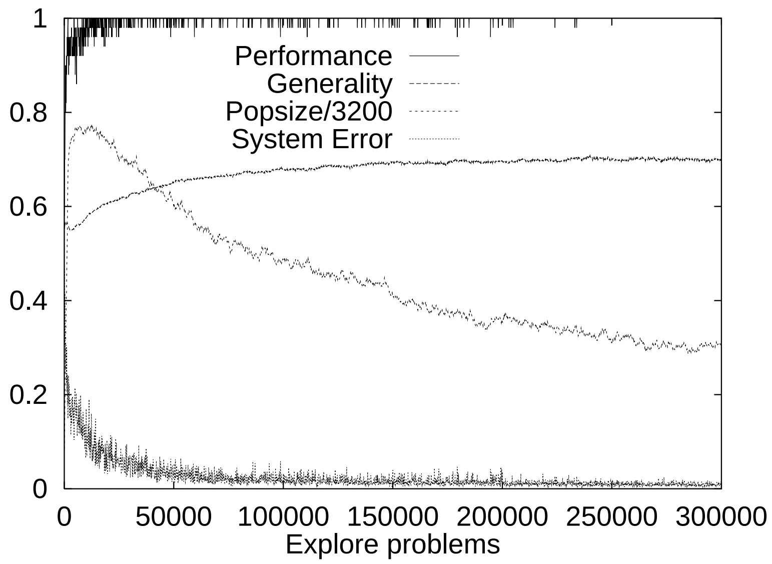

Fig. 3. Performance (fraction correct), generality, population size (/3200), and system error vs. number of explore problems in Experiment 2

Results are presented in Figure 3, showing a typical run. In comparison with Experiment 1, performance took somewhat longer to reach 100%, system error fell more slowly, and the maximum and final population sizes were substantially larger. On the other hand, the input space is exponentially larger—10^9 vs. 10^2—whereas the results differ by a factor of 10 or less. Why the 9-d problem is not harder is not clear at present. It may have to do with the input space being only sparsely occupied by dataset instances, in comparison with the much fuller coverage in the 2-d case. Sparse coverage makes the DS irregular and thus less strictly diagonal; this may aid the classifiers in approximating it.

The 974 classifiers of the final population were examined, but examples are not shown here. As in Experiment 1, roughly 80% of the classifiers were accurate, with the remaining inaccurate ones having very low fitness. Because of the dimensionality and sparseness of the dataset, and unlike Experiment 1, it was not readily possible to judge if the accurate classifiers were maximally general. The accurate classifiers showed no clear pattern in their predicates, as would be expected from the obliqueness of the formula for creating the dataset, together with the sparseness. Nevertheless, as seen in Figure 3, the dataset was learned. It appears that XCSI can deal with nine-dimensional oblique data, at least from the point of view of performance.

5 Experiments on The WBC Dataset

We used the "original" Wisconsin Breast Cancer Database which was donated to the UCI Repository by Prof. Olvi Mangasarian and contains 699 cases (instances) collected over time by Dr. William H. Wolberg. For concreteness, the nine attributes of each instance are: Clump Thickness, Uniformity of Cell Size, Uniformity of Cell Shape, Marginal Adhesion, Single Epithelial Cell Size, Bare Nuclei, Bland Chromatin, Normal Nucleoli, and Mitoses. Each attribute has a value between 1 and 10 inclusive. Sixteen instances contain an attribute whose value is unknown. The outcome distribution is 458 Benign (65.5%), 241 Malignant (34.5%). The following is a small sample of the raw data.

```
1070935,3,1,1,1,1,1,2,1,1,2
1071760,2,1,1,1,2,1,3,1,1,2
1072179,10,7,7,3,8,5,7,4,3,4
1074610,2,1,1,2,2,1,3,1,1,2
1075123,3,1,2,1,2,1,2,1,1,2
1079304,2,1,1,1,2,1,2,1,1,2
1080185,10,10,10,8,6,1,8,9,1,4
1081791,6,2,1,1,1,1,7,1,1,2
1084584,5,4,4,9,2,10,5,6,1,4
1091262,2,5,3,3,6,7,7,5,1,4
1096800,6,6,6,9,6,?,7,8,1,2
```

The first number is a label, the next nine are the attribute values, and the last is the outcome, 2 for Benign and 4 for Malignant. Clearly, Malignant is associated with larger attribute values. But the precise borderline (DS) is not obvious, nor is it clear whether certain attributes are more important than others. The obliqueness of the data is reflected in the fact that a summation of attribute values or “weight of evidence” appears broadly to be decisive. On an earlier (smaller) version of the data, Wolberg and Mangasarian [18] used a statistical method with multiple hyperplanes to obtain accuracies as high as 95.9%. The UCI Repository states ”highest reported accuracy” on the current dataset to be 94%.

XCSI will first be applied to the WBC dataset in a “train/test” experiment using a well-known procedure for evaluating learning algorithms for data mining. After that, we will present an experiment in which XCSI extracted some of the patterns contained in the data.

5.1 Experiment 3—WBC Train/Test

A practical learning algorithm for data mining must of course be able to categorize instances correctly that it hasn't seen before. In the experiments of Section 4, XCSI learned a dataset perfectly, but was not tested on further examples. In this experiment, we applied XCSI to the WBC data in a stratified tenfold cross-validation procedure in which the system learned on part of the data and was tested on the remainder.

According to Witten and Frank [17], “Tenfold cross-validation is the standard way of measuring the error rate of a learning scheme on a particular dataset; for reliable results, ten times tenfold cross-validation.” We followed the procedure as they explain it, adding stratification, but left the 10 times repeat for later work.

Briefly, the procedure is as follows. The dataset is divided into 10 parts called “folds”. The system is then tested on each fold, after being trained on the balance of the dataset. Then the results on the 10 test folds are averaged giving the final score. The folds are made as equal in size as possible, given the size of the actual dataset. “Stratification” means that in each fold the possible outcomes have approximately the same prevalences as in the whole dataset.

The folds were created from the dataset by random selection without replacement. To get stratification in a fold, it was actually made up of two parts, one containing instances randomly selected without replacement from the remaining 0-outcomes in the dataset, the other from the remaining 1-outcomes. Then the two fold parts were shuffled randomly together. The actual numbers of outcomes selected for each fold were given by a predetermined pair of numbers. For eight of the ten folds, the pair was (46, 24) for 0-outcomes and 1-outcomes, respectively. For the last two folds, the pairs were (45, 24) and (45, 25), thus accounting for all the instances as evenly as possible. (The WBC outcome notation of 2 and 4 was replaced by 0 and 1.)

Parameters for the experiment were the same as previously described, except $N = 6400$ and $r_0 = 4$. If an attribute value was missing, it was assumed to match. Calling each fold an “Out” set and the corresponding balance of the dataset an “In” set, XCSI learned on each of In sets for 50,000 explore problems, then was tested on the associated Out set. Each learning run reached 100% performance by 40,000 problems, with no mistakes after that, except for three runs that reached 100% at approximately 49,500, 49,000, and 40,100 problems. Testing consisted of a simple sweep through the Out set with XCSI in test mode. The test results are shown in Table 1.

Correct	Incorrect	Not Matched	Fraction Correct
68	2	0	0.9714
69	1	0	0.9857
65	5	0	0.9286
66	4	0	0.9429
65	3	2	0.9286
64	3	3	0.9143
70	0	0	1.0000
69	1	0	0.9857
65	3	1	0.9420
67	2	1	0.9571
		MEAN $\Rightarrow$	0.9556

Table 1. Results of stratified tenfold cross-validation test on WBC dataset. Each line has results for one fold. Last line is mean of fold scores.

With a mean performance of about 95.5%, XCSI appears on this problem to be in the same league as the best of other approaches. Of course, the present experiments are exploratory, and further testing is called for on the WBC data—using different fold sets—and on other oblique data problems.

5.2 Experiment 4—Discovering Patterns

As XCSI grinds through thousands of data instances, its generalization and subsumption mechanisms evolve increasingly general, accurate classifiers. On Boolean functions or The Monk's Problems the result is a small set of classifiers in which the problem structure is evident. On oblique data, however, the discrimination surface must be approximated by hyper-rectangles defined by the classifier conditions. To the extent the data are truly oblique, the evolved classifiers may reveal little about the DS. In Experiment 1, for instance, no individual classifier in Figure 2 offers any hint as to the shape of the DS, even though each one having high fitness is accurate and maximally general. On the other hand, if the data while largely oblique contains some "logical" structure—i.e., structure that hyper-rectangles can more directly approximate—we should be able to see it in the evolved classifiers.

An oblique dataset like WBC can be tested for logical structure by continuing the evolution far past the point of 100% performance. Evolutionary search does not stop if the GA is running and further generalization of existing classifiers is possible. We can tell that the search continues as long as population generality is rising and population size is falling. In Experiment 4 we carried a learning

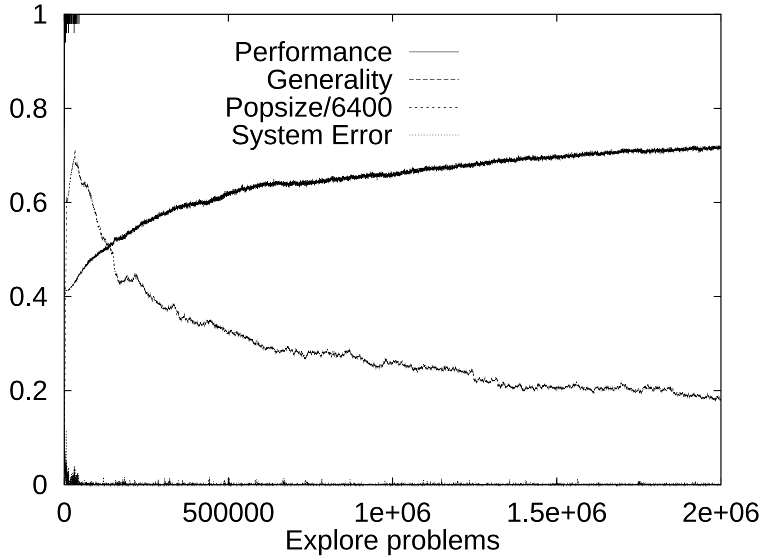

Fig. 4. Performance (fraction correct), generality, population size (/6400), and system error vs. number of explore problems in Experiment 4.

run on the entire WBC dataset out to 2,000,000 time-steps (instances seen) and examined the resulting classifiers. Graphs of the experiment are shown in Figure 4. Notice that 100% performance is reached by roughly 50,000 problems, but generality and population size are still changing at 2,000,000 problems.

Figure 5 shows the 27 most experienced classifiers in the final population (their experiences ranged from 50,810 to 414,215). These are rules that make no mistakes in predicting the payoff for any WBC dataset instance that they match. Because of their large experiences, it is quite safe to assume that they are generalized up to the point where expanding any of the interval predicates, even slightly, would cause an error. In short they are never contradicted by the dataset, and they are maximally general.

High experience means a classifier matches a large fraction of the dataset. The classifiers in Figure 5 should offer some insight about the dataset's structure and by extension, the domain from which the data were drawn. For example, No. 3 appears to say, "Regardless of the values of the remaining attributes, if attribute 1 is 7 or above and attribute 2 is 5 or above, the case is malignant." In medical terms, this would be, "If clump thickness is 7 or above and uniformity of cell size is 5 or above, malignancy is indicated."

The statement is an hypothesis derived from the data, certainly not a proven fact. In fuller terms, it would be: "To the extent the dataset accurately samples the universe of breast cancer cases, it is hypothesized that no case fitting the ranges of No. 3 will not be malignant. Furthermore, there are cases that fit No. 3 except for clump thickness less than 7 or uniformity of cell size less than 5

```
 0. |0000000...|0000000000|0000000000|00........|00........|0000000000|0000000000|0000000000|0000000000|  0 1000
 1. |0000000000|0.........|0000000000|0000000000|0000000000|0000......|0000000000|0000000000|0000000000|  1    0
 2. |00000000..|0000000000|0000000000|0000000000|0000000000|0000......|00........|0000000000|0000000000|  0 1000
 3. |......0000|....000000|0000000000|0000000000|0000000000|0000000000|0000000000|0000000000|0000000000|  1 1000
 4. |......0000|....000000|0000000000|0000000000|0000000000|0000000000|0000000000|0000000000|0000000000|  0    0
 5. |0000000000|0000000000|......0000|.000000000|0000000000|0000000000|0000000000|..00000000|0000000000|  0    0
 6. |0000000000|....000000|0000000000|0000000000|0000000000|0000000000|0000000000|0000000000|.000000000|  0    0
 7. |0000000000|0000000000|0000......|0000000000|0000000000|0000......|000.......|0.........|0000000000|  1    0
 8. |0000000000|....000000|0000000000|0000000000|0000000000|0000000000|0000000000|0000000000|.000000000|  1 1000
 9. |........00|0000000000|0000000000|0000000000|0000000000|0000000000|0000000000|0000000000|0000000000|  1 1000
10. |........00|0000000000|0000000000|0000000000|0000000000|0000000000|0000000000|0000000000|0000000000|  0    0
11. |0000000000|0000000000|.......000|.000000000|0000000000|0000000000|0000000000|0000000000|0000000000|  1 1000
12. |0000000000|0000000000|0000......|0000000000|0000000000|00........|0000000000|00........|0000000000|  0 1000
13. |0000000000|0000000000|0000000000|0000000000|0000000000|0000000000|0000000000|.........0|0000000000|  1 1000
14. |0000000000|0000000000|0000000000|0000000000|0000000000|0000000000|0000000000|.........0|0000000000|  0    0
15. |......0000|0000000000|0000000000|0000000000|0000000000|.......000|0000000000|0000000000|0000000000|  1 1000
16. |......0000|0000000000|0000000000|0000000000|0000000000|.......000|0000000000|0000000000|0000000000|  0    0
17. |0000000000|0000000000|..00000000|.....00000|0000000000|0000000000|0000000000|0000000...|0000000000|  0    0
18. |0000000000|0000000000|0000......|0000000000|0000000000|00........|0000000000|00........|0000000000|  1    0
19. |0000000000|0000000000|0000000000|0000000000|0000000000|.....00000|....000000|0000000...|0000000000|  0    0
20. |0000000...|0000000000|0000000000|00........|00........|0000000000|0000000000|0000000000|0000000000|  1    0
21. |0000000000|0000000000|0000000000|0000000000|0000000000|.....00000|....000000|0000000...|0000000000|  1 1000
22. |00000.....|0000000000|.....00000|.000000000|0000000000|0000000000|0000000000|0000000000|0000000000|  0    0
23. |00000.....|0000000000|.....00000|.000000000|0000000000|0000000000|0000000000|0000000000|0000000000|  1 1000
24. |.....00000|0000000000|0000000000|0000......|0000000000|....000000|0000000000|0000000000|0000000000|  1 1000
25. |00000000..|0000000000|0000000000|0000000000|0000000000|0000......|00........|0000000000|0000000000|  1    0
26. |0000000000|0000000000|0000000000|0000000000|0000000000|0000000000|.......000|0000000000|0000000000|  1 1000
```

Fig. 5. The most experienced classifiers from the final population of Experiment 4.

that are benign." Thus the classifier makes an hypothesis about cases that fit its condition, and a weaker hypothese about cases that nearly fit its condition.

Figure 5 includes some classifiers in which a single attribute is decisive. For example No. 9 says, "If clump thickness is 9 or above, then malignant." No. 26 says, "If bland chromatin is 8 or greater, then malignant." Some are very close to depending on a single attribute. No. 11 says, "If uniformity of cell shape is 8 or above and marginal adhesion is not 1, then malignant. Similarly for Nos. 6 and 8. Furthermore, rules dependent on just two attributes can be identified, for instance Nos. 1 (which indicates benign), 5, and 15. In all of these we have omitted the associated weaker hypothesis.

6 Discussion

XCSI's good performance on both the synthetic oblique problems and the WBC dataset suggests that the apparent mismatch between the obliqueness of the discrimination surface and XCSI's hyper-rectangular predicates is not a fundamental limitation. The boxy predicates might have been expected to lead to *overfitting*, i.e., excessively detailed learning of the training data causing poor performance on new instances. But test performance on the WBC data was at a competitive level, suggesting that overfitting was not in fact a problem. Still, it is not clear why this pleasant outcome should have occurred. Perhaps there is something about the "boxes" approximation that actually tends to prevent overfitting.

The emergence of some pattern information in the WBC data suggests that the set is not totally oblique, but instead contains regions where there is a strong dependence on one or a few attributes. XCSI seems to zero in on such patterns. Hopefully, the patterns will make sense to clinicians or practitioners, aiding confidence in the system's predictions, and providing suggestive domain hypotheses.

Beyond the patterns shown in Figure 5, the final population contained of course a great many others having a more intricate structure. Some of these classifiers are concerned with the region close to the DS, where discrimination is complex. The patterns of Figure 5, though they cover huge subspaces, may not in fact apply in "close cases" near the DS. There, however, the match set will often indicate the degree of uncertainty of the decision, because it will contain high fitness classifiers that vote strongly each way. A measure of match set agreement could assist practitioners in the close cases. Holmes [4] used this sort of technique for evaluating risk of disease.

Next steps for research on XCS and oblique data should include further fold tests on WBC and tests on additional representative datasets. Classifier patterns such as those seen in Figure 5 should be shown to domain practitioners for plausibility. Analysis is needed of the boxes approximation, including the result that the learning complexity of XCSI on oblique data is apparently far below exponential. The conditions of the classifiers of evolved populations contain a great deal of information about the structure of the dataset, but only a fraction of it is directly interpretable as in Section 5.2. It would be desirable to find algorithms for extracting all the classifiers' implications as rules of thumb and in other representations.

7 Conclusion

XCS, suitably modified for numeric inputs, appears to have considerable potential for mining oblique datasets. On tests so far, performance was high, and simple as well as more complex patterns describing the structure of the data emerged. Further investigation on a wide range of problems may show that systems based on XCS are a very good choice for high performance combined with clear pattern discovery.

Acknowledgment

The author acknowledges helpful conversations with John H. Holmes and Pier Luca Lanzi, and support from the Automated Learning Group at the National Center for Supercomputer Applications in Urbana-Champaign.

References

1. 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). Morgan Kaufmann: San Francisco, CA, 1999.
2. C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998. http://www.ics.uci.edu/~mlearn/MLRepository.html.
3. Martin V. Butz and Stewart W. Wilson. An algorithmic description of XCS. This volume.
4. John H. Holmes. Discovering Risk of Disease with a Learning Classifier System. In Thomas Bäck, editor, *Proceedings of the 7th International Conference on Genetic Algorithms (ICGA97)*. Morgan Kaufmann: San Francisco CA, 1997.
5. Tim Kovacs. XCS classifier system reliably evolves accurate, complete, and minimal representations for boolean functions. In Roy, Chawdhry, and Pant, editors, *Soft Computing in Engineering Design and Manufacturing*, pages 59–68. Springer-Verlag, London, 1997.
6. Tim Kovacs. Deletion schemes for classifier systems. In Banzhaf et al. [1], pages 329–336.
7. Pier Luca Lanzi. Extending the representation of classifier conditions. Part II: From messy coding to S-expressions. In Banzhaf et al. [1], pages 345–352.
8. Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors. *Learning Classifier Systems: From Foundations to Applications*, volume 1813 of *LNAI*. Springer-Verlag, Berlin, 2000.
9. Shaun Saxon and Alwyn Barry. XCS and The Monk's Problems. In Lanzi et al. [8], pages 223–242.
10. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press/Bradford Books, Cambridge, MA, 1998.
11. Sebastian B. Thrun et al. The monk's problems: A performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, Pittsburgh, PA, 1991.
12. Stewart W. Wilson. ZCS: A zeroth level classifier system. *Evolutionary Computation*, 2(1):1–18, 1994.
13. Stewart W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.
14. Stewart W. Wilson. Generalization in the XCS classifier system. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 665–674. Morgan Kaufmann: San Francisco, CA, 1998.
15. Stewart W. Wilson. Get Real! XCS with Continuous-Valued Inputs. In Lanzi et al. [8], pages 209–220.
16. Stewart W. Wilson. State of XCS Classifier System Research. In Lanzi et al. [8], pages 63–82.
17. Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, San Francisco, CA, 2000.
18. W. H. Wolberg and O. L. Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proceedings of the National Academy of Sciences*, 87:9193–9196, 1990.