Kernel-based, Ellipsoidal Conditions in the Real-Valued XCS Classifier System

Martin V. Butz
Department of Cognitive Psychology
University of Würzburg
Würzburg, 97070, Germany
mbutz@psychologie.uni-wuerzburg.de

ABSTRACT

Many learning classifier system (LCS) implementations are restricted to the binary problem realm. Recently, the XCS classifier system was enhanced to be able to handle real-valued inputs among others. In the real-valued enhancement, XCSF applies as a function approximation system that partitions the input space in hyperrectangular subspaces specified in the classifiers. This paper changes the classifier conditions to hyperspheres and hyperellipsoids and investigates the consequent performance impact. It is shown that the modifications yield improved performance in continuous functions. Even in discontinuous functions with parallel boundaries, XCS's performance does not degrade. Thus, for the real-valued problem domain, ellipsoidal condition structures can improve XCS's performance. From a more general perspective, this paper shows that XCS is readily applicable in diverse problem domains. To apply the system even more successfully, suitable kernel-based bases need to be found and used as classifier conditions. XCS distributes the available structures over the problem space evolving more specialized structures in more complex problem subspaces.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*connectionism and neural nets*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search; I.5.1 [**Pattern Recognition**]: Models—*fuzzy set; statistical; neural nets*; I.5.0 [**Pattern Recognition**]: General

General Terms

Algorithms, Experimentation, Theory, Performance

Keywords

Learning Classifier Systems, XCS, GAs, Function Approximation, Piece-wise Linear Approximation, Radial Bases

GECCO'05, June 25–29, 2005, Washington, DC, USA.

1. INTRODUCTION

Learning classifier systems (LCSs) [15, 16] have gained increasing interest in the genetic and evolutionary computation literature. Especially the accuracy-based XCS classifier system [22, 23] was successfully applied in many real-world problems [3, 4]. The recent analyses [10, 9, 11] have shown that XCS scales machine learning competitively, that is, it scales polynomial in solution complexity, problem length, reliability, and error tolerance. The results confirm that XCS is a valuable learning system that is able to solve complex problems machine-learning competitively. The advantages of an XCS-based learning approach lie in its online learning capability, its noise robustness, the generality in the learning mechanism, and its continuous adaptivity.

The basic XCS system is only applicable in binary problems. Lanzi has previously experimented with messy coding and S-expressions [17, 18]. Bull and O'Hara introduced full multi-layer perceptron classifiers [7]. Wilson [24, 25] has successfully enhanced the system to the integer- and real-valued problem domains. The resulting system, XCSR, partitions the problem space in hyperrectangular subspaces. In each subspace, XCSR learns a real-valued prediction of the expected reward or value. In this way, XCS evolves a payoff surface that is represented by its population of classifiers.

This paper investigates if the hyperrectangular conditions are suitable bases for partitioning the problem space. It is suggested that a function may be approximated more efficiently if conditions do not specify hyperrectangles but hyperspheres or, more generally, hyperellipsoids. We show that this new representation is in fact equally or more effective in most real-valued functions investigated.

From a machine-learning perspective, we show that XCS's conditions can be any type of kernel-based basis structure. XCS searches a given feature space defined by the available basis structures and it evolves a problem space partitioning that allows a highly accurate classification of the data. Thus, XCS evolves the available features partitioning the problem space in such a way that most accurate classifications or predictions are possible.

The remainder of this work first gives a short introduction to XCSR and investigates its performance in several real-valued functions. Hereby, we point out several representation-based constraints. Next, we introduce spherical conditions, ellipsoidal conditions and general ellipsoidal conditions. The three approaches are evaluated and compared on several real-valued functions. Summary and conclusions put the results in a bigger perspective.

2. XCS FOR REAL-VALUED INPUTS

XCS is a typical Michigan-style learning classifier systems (LCSs) [16, 5]. The following introduction of XCS introduces the enhanced XCS system for real-valued inputs [24, 25, 26, 27]. The consequent system specifies hyperrectangles in the classifier conditions. That is, given a real-valued input vector $s = (s_1, ..., s_n) \in \mathcal{S} \subseteq \Re^n$, a condition in XCS specifies intervals for all n values. A classifier matches if the current problem instance s lies in all n intervals. Since Stone and Bull [20] showed that the center-spread-based condition representation yields an unnecessary learning bias at value boundaries, we use interval coding with lower and upper boundaries. Additionally, we use linear predictions that are dependent on the input x, as introduced in [26, 27]. In the remainder of this work we will refer to this version of a real-valued XCS classifier system as XCSR.

2.1 Basic XCSR System

As in all LCSs, the knowledge in XCSR is represented by a population of classifiers. Each classifier in XCSR has five main components. (1) The condition part C specifies a hyperrectangle by the means of interval encoding, that is, $C = (l_1, u_1, l_2, u_2, ..., l_n, u_n)$. (2) The action part A specifies an available action, that is, $A \in \mathcal{A}$, where $\mathcal{A} = \{A_1, ..., A_m\}$ is the set of all possible actions in the problem. (3) The reward prediction R specifies a linear prediction of the input vector s in the form of a weight vector, that is, $R = (w_0, w_1, ..., w_n)$ where w_0 is the offset weight. (4) The prediction error ε estimates the mean absolute deviation of the reward predictions. (4) The fitness F specifies the relative accuracy of the classifier. The values are iteratively modified and evolved by the means of reinforcement learning (RL) techniques and a GA.

XCSR is initialized with an empty population. Initial classifiers are generated by a covering mechanism that creates intervals controlled by parameter r_0. The resulting interval size lies between 0 and $2r_0$. Covering is triggered if not all actions are represented in the current match set. Each learning iteration, given the current problem instance $s \in \mathcal{S}$, a match set $[M]$ is formed that contains all classifiers in the current population $[P]$ whose conditions are satisfied by s. Thus, s lies in each of the hyperrectangles of all matching classifiers. $[M]$ can then be used to form a prediction about the class or value of the problem instance. Hereby, XCSR forms a prediction array $P(\mathcal{A})$ of expected payoff values for all possible actions. The array is used to make an action decision. During learning, the action is usually chosen at random. All classifiers in $[M]$ that specify the chosen action A form the action set $[A]$. After action execution, the environment provides scalar feedback $r \in \Re$ of the actual value of the executed action. In this paper we are interested in the predictive capabilities of XCSR. Thus, we focus on function approximation and only one (dummy) action is available.

XCS iteratively learns from the successive problem instances. First, it uses the feedback r to update the reward prediction, prediction error, and fitness estimate of all classifiers in $[A]$. Hereby, we use the usual delta update rule for the weight vector R, that is,

$$\begin{aligned} w_0 &\leftarrow w_0 + \eta(r - ||R(l-s)^*||), \\ w_i &\leftarrow w_i + \eta(r - ||R(l-s)^*||)(s_i - l_i), \end{aligned} \tag{1}$$

where η denotes the learning rate and $||Rs^*||$ the inner product of weight vector R and the difference vector (l-s) that is enhanced by an additional one entry at the first position to account for w_0. Note that we do not apply an additional normalization factor as done elsewhere [27]. Comparisons with the normalization factor yielded hardly any performance differences. Next, we update reward prediction error.

$$\varepsilon \leftarrow \varepsilon + \beta(|r - P(A)| - \varepsilon) \tag{2}$$

Finally, fitness is updated first determining the accuracy of the classifier:

$$\begin{aligned} \kappa &= \begin{cases} 1 & \text{if } \varepsilon < \varepsilon_0 \\ \alpha \left(\frac{\varepsilon}{\varepsilon_0}\right)^{-\nu} & \text{otherwise} \end{cases}, \\ \kappa' &= \frac{\kappa \cdot num}{\sum_{cl \in [A]} cl.\kappa \cdot cl.num}, \\ F &\leftarrow F + \beta(\kappa' - F), \end{aligned} \tag{3}$$

where ε_0 is the minimum error threshold, α an error offset, and ν the degree of the polynomial. After rule evaluation and possible GA invocation, the next iteration starts.

XCS applies a GA for rule evolution. The GA selects two parental classifiers from the current action set $[A]$ using set-size relative tournament selection [13]. Two offspring are generated by applying crossover and mutation to the two selected parents. We apply a relative real-valued mutation instead of the fixed interval mutation previously used. The relative real-valued mutation moves the center of the interval uniformly randomly within its current interval. Additionally, it increases or decreases (equally likely) the size of the current interval by zero to fifty percent chosen uniformly randomly. The relative real mutation showed to improve performance throughout the experiments. Especially if the population is initialized very general, performance improved much faster and converged more stably. Parents remain in the population, competing with their offspring. The population of classifiers $[P]$ is of fixed size N. Excess classifiers are deleted from $[P]$ with probability proportional to an estimate of the size of the action sets that the classifiers occur in. If the classifier is sufficiently experienced and its fitness F is significantly lower than the average fitness of classifiers in $[P]$, its deletion probability is further increased.

2.2 Learning Bias in XCS

The evolutionary algorithm is the main component that searches for better problem space partitions with respect to the achieved predictive accuracy. GA selection propagates currently most accurate classifiers. Mutation searches in the neighborhood for better space partitions. Crossover combines previously successful sub-partitions. Selection in action sets in combination with deletion in the whole population causes a generalization pressure [12]. Moreover, the mechanism has a niching effect [8] biasing the evolutionary process towards evolving a classifier population that covers the whole problem space.

While the evolutionary mechanism is designed to evolve partitions in which linear approximations are maximally accurate, the gradient descent-based methods expressed in equations 1, 2, 3 estimate the suitability of the current partitions. Thus, XCS applies a distributed, local search mechanism combining evolutionary techniques with gradient-descent learning techniques to find a global problem solution. As a whole, XCS strives to evolve a complete, maximally accurate, and maximally general problem solution.

2.3 Performance of XCSR

Before we introduce the applied system modifications we first evaluate XCSR's performance on three challenging two dimensional functions:

$$f_1(x,y) = ((x*3)\%3)/3 + ((y*3)\%3)/3 \quad (4)$$

$$f_2(x,y) = (((x+y)*2)\%4)/6 \quad (5)$$

$$f_3(x,y) = \sin(2\Pi(x+y)), \quad (6)$$

where the % is the modulo operator. All functions in this paper are defined for values between zero and one in all their n dimensions. Figure 1 shows the three functions. Function f_1 is an axis-parallel step function. It is expected to be learned quite effectively by XCSR since the steps are rectangular so that it is possible to represent each step with one classifier. Function f_2 is an axis-diagonal step function. XCS faces oblique boundaries so that the approximation is expected to be significantly harder. Finally, function f_3 is an axis-diagonal, continuous sinusoidal function. Additional challenges due to the curvature are expected.

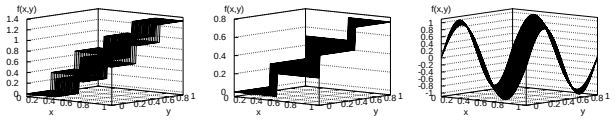


Figure 1: The Axis-parallel step function f_1, the axis-diagonal step function f_2 and the sinusoidal function f_3.

Predictive performance of XCSR is shown in figure 2. All experiments herein are averaged over 20 experiments. If not stated differently, parameters were set as follows: $N = 6400$, $\beta = \eta = 0.5$, $\alpha = 1$, $\varepsilon_0 = .01$, $\nu = 5$, $\theta_{GA} = 50$, $\chi = 1.0$, $\mu = 0.05$, $r_0 = 1$, $\theta_{del} = 20$, $\delta = 0.1$, $\theta_{sub} = 20$. GA subsumption was applied. Uniform crossover was applied. Note that we start with a fairly general population due to the high covering value r_0. We also apply a rather high learning rate that showed to slightly speed-up performance due to faster evolving approximations. If the problems were noisy problems it would be necessary to decrease the learning rate. The parameter values are nearly identical to the values chosen in Wilson's work [27].

Figure 2 confirms the expected results. Function f_1 is learned best resulting in a low error of about .007—significantly below the error threshold ε_0. The axis-diagonal step function is initially slightly easier to approximate due to its smaller y-value range. While learning proceeds the population size rises to a higher level and the prediction just reaches .01. Due to the obliqueness of the function, the hyperrectangular conditions make it harder to evolve an effective space partitioning. However, the oblique structure is still easier to represent than the sinusoidal function. In f_3, XCSR does only reach an error level of .011. Learning takes longer and the consequent population size is much higher.

Function approximation performance is shown in figure 3a, b, c (resolution 50×50). In the parallel-step function, XCS approximates the steps accurately. No severe under- or overestimations occur except at the boundaries. The condition structure is the main obstacle in functions f_2 and f_3 since the steps in the diagonal-step function are oblique and the

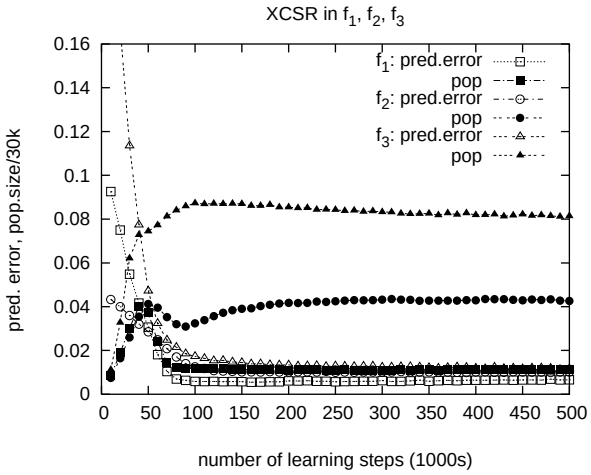


Figure 2: XCSR performs well in the axis-parallel function f_1. Performance is worse in the axis-diagonal function. In the sinusoidal function, XCSR does hardly reach the targeted level of 1% error.

sinusoidal function has additionally a strong curvature. Several approaches are imaginable to improve performance of XCSR. If the typicality of the function was known, then performance could certainly be improved. For example, in the case of the diagonal step function, an enhancement with diagonal boundaries should strongly improve performance. It should also be helpful in the sinusoidal function. However, it cannot be expected that such boundary types are known beforehand.

A more general approach lies in a boundary approximation by the means of circular or ellipsoidal structures. Although these structures cannot be expected to yield perfect approximations, the piecewise linear approximations can be expected to suitably overlap to represent a more smooth approximation surface. Disruptive effects in the corners of hyperrectangles are expected to be prevented. The next section introduces such hyperspherical and hyperellipsoidal condition structures to XCSR.

3. ELLIPSOIDAL CONDITIONS

This section introduces the new spherical and ellipsoidal condition structures to XCSR. The condition parts of the classifiers in XCSR are changed. We start with conditions represented by hyperspheres and then proceed to the more general ellipsoids. All enhancements are related to radial bases structures since we use a Gaussian to represent the structures.

3.1 Hyperspheres

The conditions are changed to hyperspheres in that a condition part is now represented by a center point and a deviation, that is, $C = (m_1, m_2, ..., m_n, \sigma)$. A classifier then is active if the current problem instance lies within a certain range of the specified hypersphere. We use the Gaussian kernel function to determine the activity of a classifier:

$$cl.ac = \exp\left(-\frac{||s-m||^2}{2\sigma^2}\right), \quad (7)$$

where $m = (m_1, ..., m_n)$. To determine if a classifier is ac-

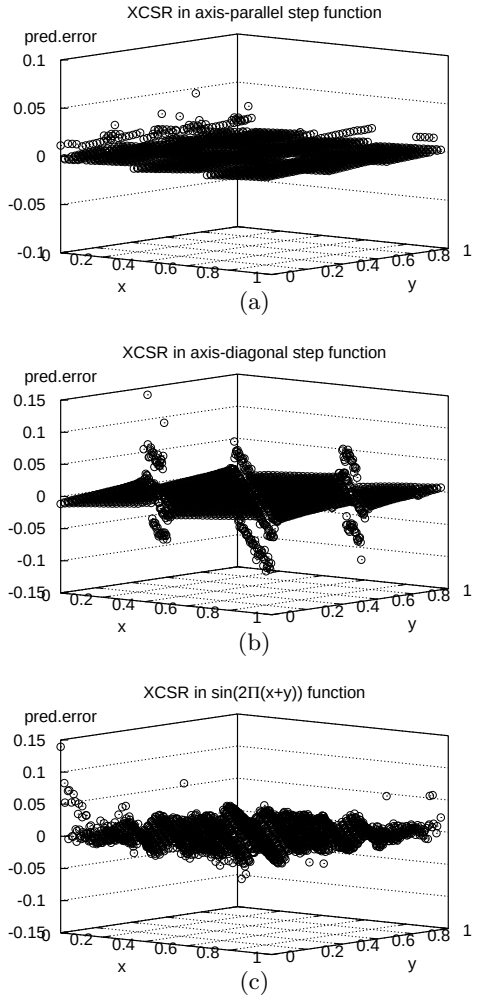


Figure 3: Average approximation error in various function approximate tasks. (a) The boundaries of the axis-parallel function are well-approximated by XCSR. (b) In the case of oblique steps, XCSR misses some of the boundary points. (c) In the sinusoidal function, XCSR under- or overestimates. Corners and areas with strong curvature are affected most severely.

tive, its activity $cl.ac$ needs to lie above a threshold θ_m. The smaller θ_m, the larger the receptive field and thus the probability of matching of a condition structure. In a sense, θ_m is the pendant to σ. Since σ is evolved, θ_m can be fixed. It is set to $\theta_m = .7$ throughout the experiments. In general, it should be noted that spheres could be defined slightly more effectively simply using the l^2 norm. The chosen radial bases conditions, though, allow the immediate enhancement to fuzzy conditions. Moreover, they emphasize the potential of substituting the structure with any other kernel-based basis structure.

Covering, mutation and crossover need to be adjusted. When creating a covering classifier, the center is set directly to the current values (that is, $m = s$) and the deviation σ is set uniformly randomly between zero and r_0 (zero excluded). For mutation, we define a relative mutation similar to the one introduced to XCSR with hyperrectangles above. Each attribute in the condition part is mutated with a probability μ. If an attribute of the center is mutated, the new value m_i' is set to a value uniformly randomly chosen in the interval the classifier applies in, that is, $|m_i - m_i'| \leq \sigma\sqrt{-2\log\theta_m}$. The standard deviation σ is either increased or decreased (equally likely) between zero and 50% chosen uniformly randomly. If the standard deviation is larger than a deviation necessary to contain the whole problem space, it is set to that value, that is:

$$\sigma \leq \frac{\sqrt{\sum_{i=1}^{n}(u_i^* - l_i^*)^2}}{\sqrt{-2\log\theta_m}}, \tag{8}$$

where u_i^* and l_i^* denote the maximum upper and lower values of each dimension, respectively. As crossover operator we chose uniform crossover since no dependencies between the dimensions are known. A classifier is considered as more general if its hypersphere completely contains the other hypersphere.

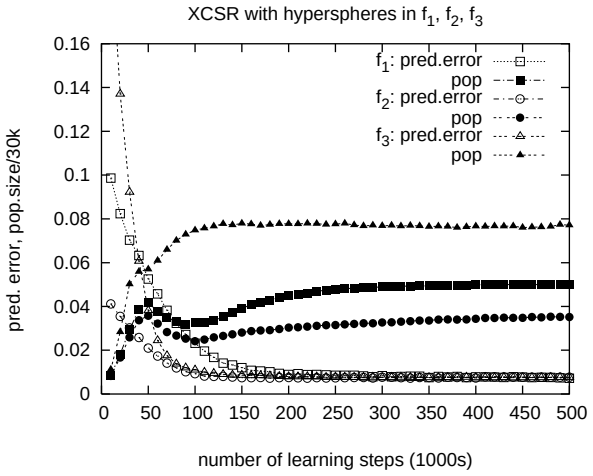
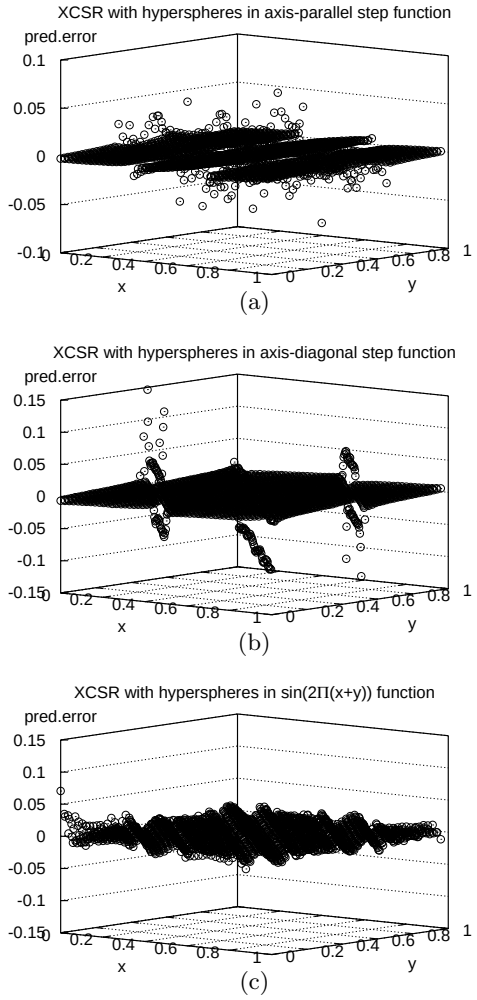


Figure 4: While the axis-parallel step function is harder to learn for XCSR with hyperspheres, the axis-diagonal as well as the sinusoidal functions are approximated easier.

Performance of XCSR with hyperspheres is shown in Figure 4. In the axis-parallel step function, XCSR with hyperspheres reaches a comparable performance level. However, since hyperspheres are not well-suited to approximate axis-parallel boundaries, the condition structure complicates the task so that more learning time is required and more classifiers are needed for an equally accurate approximation. The roles are reversed in the axis-diagonal step function: With hyperspheres, XCSR needs less classifiers to reach a better performance than with hyperrectangles. Also learning is faster. Similar performance differences are observable in the sinusoidal function. The final population size is again smaller and learning proceeds faster. Moreover, the final approximation is more accurate when hyperspheres are applied. In both cases, the unnatural space partitioning of an oblique function via hyperrectangles (1) requires more specialized final classifiers and (2) causes unsuitable approximations particularly in the corners of the rectangles.

Figures 5a, b, c show the three-dimensional error surfaces when spheroidal conditions are used in XCSR. XCSR



(a) (b) (c)

Figure 5: (a) While the axis-parallel steps are slightly harder to approximate with hyperspherical conditions. (b) In the case of oblique steps, hyperspherical conditions allow a better approximation. (c) The sinusoidal function can be better approximated with hyperspherical conditions.

does not approximate the axis-parallel steps completely accurately. However, the surfaces inside the steps are approximated more accurately so that the performance average over the whole problem reaches a similar level. The axis-diagonal step function as well as the sinusoidal function are both better approximated when hyperspherical conditions are used. In the case of the axis-diagonal step function, the spherical condition structure is slightly recognizable since the errors at the steps are slightly bend indicating that a circle tends to overlap with the boundary. The performance error remains smaller since the overlap is less severe than in the case of hyperrectangles.

Besides the performance curves it is interesting to investigate how XCSR evolves the generality of its conditions. We expect that XCSR evolves more specialized conditions in regions in which the curvature of the function is highest since in those regions linear predictions are hardest to fit. Figure 6 confirms this suspicion in the sinusoidal f_3

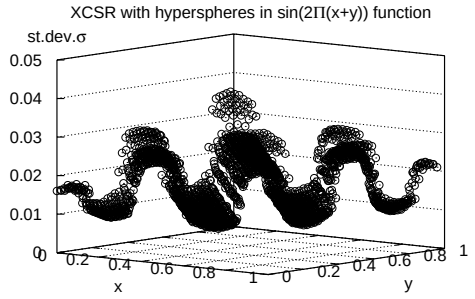


Figure 6: The standard deviations of classifier conditions correlate with the curvature of the approximated sinusoidal function f_3. Worst approximations are encountered at the boundaries of the problem space.

function. The standard deviation (or generality) of the condition parts of the matching classifiers directly depends on the curvature of the underlying function. The outlayers are located at the top and bottom corner of the space (that is, around 1,0 and 0,1). Since the occurrence in this region is small, the classifiers tend to cover larger regions.

The reader might have recognized by now that there is quite a severe limitation in the simple sinusoidal conditions. If the complexity of the target function differs in different dimensions, XCSR with the simple hyperspheres cannot be expected to learn very well since hyperspheres are equally spaced in all dimensions. Take as an example the following function:

$$f_4(x_1, x_2, x_3, x_4) = x_2 + x_4 + 2\Pi \sin(x_1) + 2\Pi \sin(x_3) \quad (9)$$

This four-dimensional function is easily linearly approximated in the x_2 and x_4 dimensions. Dimensions one and three are much harder to approximate. Figure 7 shows performance of XCSR with hyperrectangles and hyperspheres. Clearly, the hyperrectangular encoding outperforms the hyperspheroidal encoding. To represent the problem effectively, the space should only be partitioned in dimensions one and three. The additional partitioning in the other two dimensions forced by the spheroidal encoding hinders the evolution of an effective space partitioning. The next section solves this problem introducing hyperellipsoidal conditions.

3.2 Hyperellipsoids

While hyperspheres have a common radius, hyperellipsoids have different radii (or deviations) for different axis. A hypersphere is a special case of an hyperellipsoid. To represent hyperellipsoids with an XCS condition, we need to redefine the condition part as follows:

$$C = (m_1, m_2, ..., m_n, \sigma_1, ..., \sigma_n) \quad (10)$$

The condition is now represented by a center point and deviations in all n dimensions. The activity of a classifier given the current problem instance s is now defined as:

$$cl.ac = \exp\left(-\sum_{i=1}^{n} \frac{(s_i - m_i)^2}{2\sigma_i^2}\right), \quad (11)$$

effectively dividing in each dimension the squared distance from the center by twice the variance in that dimension.

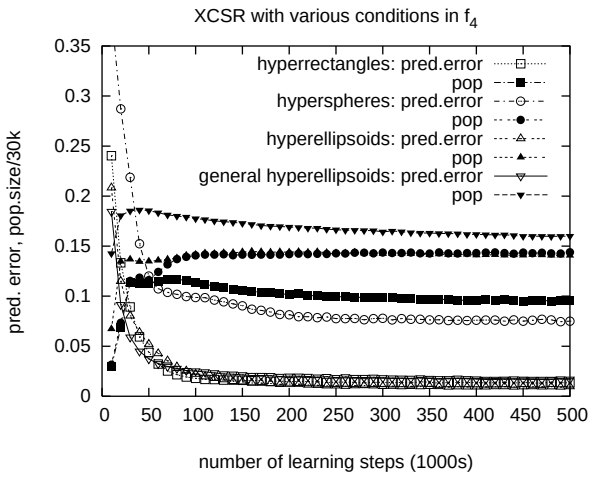


Figure 7: In the four-dimensional function f_4, spheroidal conditions are outperformed by hyperrectangular ones. Hyperellipsoidal conditions alleviate the restriction.

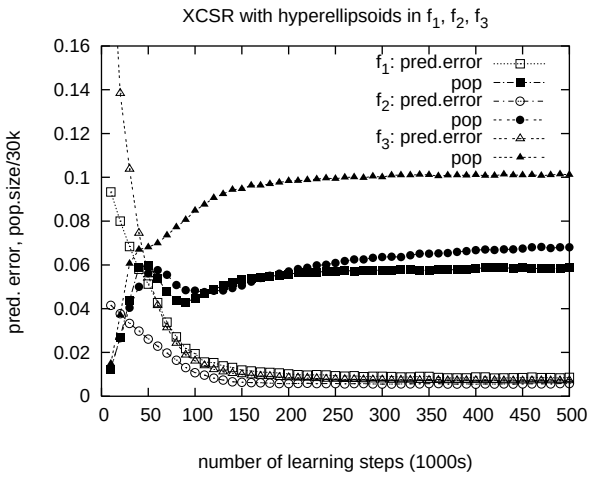


Figure 8: Hyperellipsoidal conditions yield similar performance to hyperspherical conditions in the first three test functions.

As in the case of hyperspheres, a classifier is active, if its activity $cl.ac$ lies above the threshold θ_m.

Again, we need to adjust covering, mutation and crossover. In covering, the center of the condition is again set to the current value, that is, $m = s$. The deviations σ_i are each chosen independently, uniformly randomly between zero and r_0 (zero excluded). Mutation is done as in the case of hyperspheres only that for each dimension each separate σ_i is considered. If any σ_i is larger than the deviation necessary to contain the whole problem dimension, it is set to that value, that is:

$$\forall i : \sigma_i \leq \frac{u_i^* - l_i^*}{\sqrt{-2 \log \theta_m}} \tag{12}$$

The crossover operator is uniform crossover over all $2n$ parameters of a condition part. Finally, a classifier is regarded as more general if it completely contains the other classifier in all n dimensions considered separately.

Performance in the test functions f_1, f_2, f_3 (Figure 8) is similar to that of the hyperspherical ones. The population sizes converge to a higher level, though, due to the additional variability in the conditions. Also the three dimensional error surfaces do not differ significantly (not shown).

Performance in the crucial f_4 function, however, is strongly improved. Figure 7 shows that hyperellipsoids enable XCSR to reach a similar performance level to the representation with hyperrectangles. Note that XCSR with hyperellipsoids however does not beat the performance of XCSR with hyperrectangles in f_4. This can be explained by the independence of the four dimensions: the hyperrectangular encoding basically approximates the underlying function evolving linear approximations for each dimension. The hyperrectangular form causes the independence assumption in the four dimensions. However, if there are dependences like in the diagonal-step function or the sinusoidal function, the assumption is violated and the approximation suffers. Spherical or the more general ellipsoidal conditions alleviate this independence assumption.

Nonetheless, also the hyperspherical conditions obey the dimensionality in that the axes of the hyperspheres coincide with the dimensional axes. The independence assumption still exists—albeit in a less severe form. The next section introduces general hyperellipsoidal conditions in which the independence assumption is completely abolished.

3.3 General Hyperellipsoids

To create an axis-independent condition structure, we now introduce general hyperellipsoidal conditions to XCSR. The next condition part does now not only consist of a center and the length of each axis, but additionally considers interactions between the axis using a full matrix. The new condition part is defined as:

$$C = (m_1, m_2, ..., m_n, \sigma_{1,1}, \sigma_{1,2}..., \sigma_{n,n-1}\sigma_{n,n}) \tag{13}$$

The condition is represented by a center point and a matrix. Note that we do not restrict the matrix in any way fully handing over the responsibility of evolving proper matrices to the evolutionary component. The new activity of a classifier is now defined as:

$$cl.ac = \exp\left(-\frac{\sum_{i=1}^{n}(\sum_{j=1}^{n}(s_j - m_j)\sigma_{ij})^2}{2}\right), \tag{14}$$

effectively multiplying the difference vector $s - m$ with the matrix, which effectively specifies the inverse covariance matrix. The matrix multiplication allows any rotations in the n-dimensional space as well as any possible squeezing and stretching. The hyperellipsoids of the previous section are a special case of these general hyperellipsoids. When only the matrix diagonal contains non-zero values, the condition is equivalent to a condition in the previous section with equal center and inverse deviation values ($\sigma_i = 1/\sigma_{ii}$). Classifier activity is again controlled by threshold θ_m.

In covering, the center of the hyperellipsoid is set to the current value. Only the diagonal entries in the variance matrix are initialized to the squared inverse of the uniformly randomly chosen number between zero and one. All other matrix entries are set to zero. In this way, covering creates condition parts that are on average identical to the ones created by the covering mechanism in the previous (restricted) hyperellipsoids. Mutation is similarly adjusted in that each

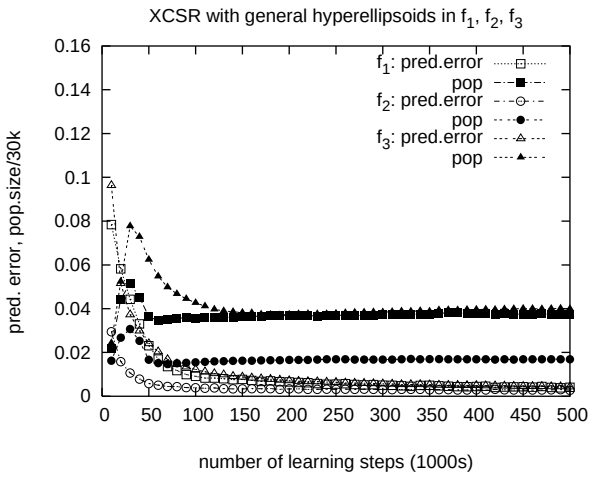


Figure 9: General hyperellipsoidal conditions further improve performance compared to (restricted) hyperellipsoidal conditions in the first three test functions. Also population sizes are smaller.

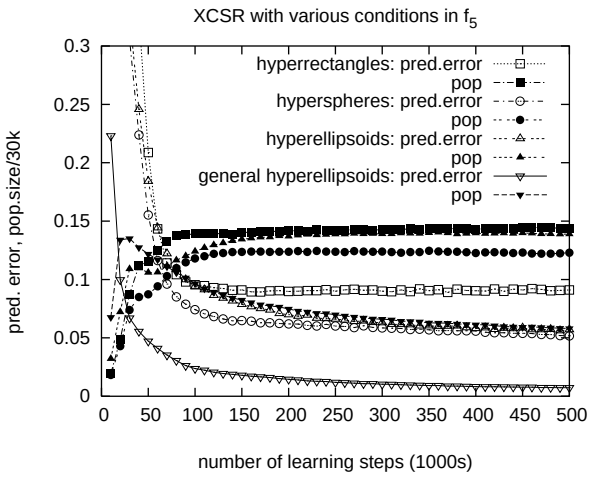


Figure 10: In the three-dimensional sinusoidal function, only the general hyperellipsoidal conditions enable accurate and effective function approximation with XCSR.

matrix entry is mutated separately maximally decreasing (increasing) the value by 50%. If the value was set to zero, it is initialized to a randomly chosen value as in covering for the diagonal matrix entries. The values of the matrix entries are unrestricted. Uniform crossover is applied to all $n + n^2$ condition part values. It is hard to determine exactly if a condition is more general than another condition. We chose to use the approximation that a condition is more general if its interval contains the interval of the other classifier plus the distance to the other classifier in the direction of that other classifier.

Performance of XCSR with general hyperellipsoidal conditions in functions f_1, f_2, f_3 is shown in Figure 9. Function approximation is improved in all three cases and also population sizes reach a smaller level. XCSR is able to exploit the additional freedom in the hyperellipsoidal structures turning and tweaking them to the most effective subspace approximations. In the f_4 problem, XCSR's performance with general hyperellipsoids stays slightly above the performance with restricted hyperellipsoids indicating that the abolished independence assumption between the axis makes the problem harder (Figure 7).

Most obvious becomes the power of general hyperellipsoidal conditions in the three dimensional sinusoidal function:

$$f_5(x_1, x_2, x_3) = \sin(2\Pi(x_1 + x_2 + x_3)) \qquad (15)$$

This function is not only more difficult in the additional dimension in comparison with the sinusoidal function f_3, but it also contains three full sinuses instead of two. Figure 10 shows that only XCSR with general hyperellipsoidal conditions reaches an error of less than .01. Moreover, the decrease in population size confirms that XCSR finds a suitable ellipsoidal shape for the condition part. The shape spreads throughout the population, which results in the observable population size decrease. With hyperspheres or (restricted) hyperellipsoids, performance is worse than with general hyperellipsoids but clearly beats the hyperrectangular approximation.

4. SUMMARY AND CONCLUSIONS

XCSR traditionally used hyperrectangular conditions to partition the problem space. This paper has shown that XCSR can also effectively learn when hyperspherical or hyperellipsoidal conditions are evolved.

The performance comparisons in various function approximation tasks confirmed the interdependence of condition structure and function type. XCSR with hyperrectangles implicitly assumes that the $n-$dimensions contribute to the outcome of the function independently. If this was the case and the fact was known, however, it might be better to apply n independent XCSR modules and combine the predicted outcome—as has been done for example for hidden Markov models [14].

If dimensional dependencies are unknown, it is advantageous to apply a more general condition structure. Hyperspheres make the approximation of diagonal problem boundaries more effective. Hyperellipsoids further facilitate the approximation of unequally structured dimensions. General hyperellipsoids, whose orientation is not bounded by the axis, improve the capabilities of the former further in functions in which the shape of the partitions are most suited for linear predictions when rotating and squeezing the hyperellipsoidal structures.

The paper also confirmed that XCSR is a very general and flexible learning system. The condition structures can be easily exchanged. Enhanced expressibility allows more accurate and effective function approximation as long as the function is sufficiently difficult (nonlinear). Thus, XCSR's learning success strongly depends on the chosen space partitions representable in the classifier conditions. The more the partition structures are suited to separate the nonlinearities in the problem, the more effective XCS will learn.

Several enhancements building on this work come to mind that are discussed in the following.

The current XCSR system with hyperellipsoids may be improved in its covering and mutation operators. Mutation could be improved by constraining mutation to have predictable effects on the condition structure, such as direct

rotation or explicit enlargements in all dimensions of the hyperellipsoid. Currently, mutation had to detect such options implicitly—albeit with actual surprising success.

The enhancement to a fuzzy XCS version is imminent and can be readily integrated into the ellipsoidal framework. Preliminary experiments in this respect, however, did not yield any further improvement in performance.

The performance comparisons showed that more general condition structures can delay learning progress. Thus, it might be interesting to allow the evolution of several condition structures in parallel. An initial restricted competition using additional niching techniques, as done in [1] for GAssist, may prevent the expectable premature convergence to one (possibly sub-optimal) structure.

XCSR should be applied to other domains with other kernel-based bases. The more suitable the chosen basis structure for the problem at hand, the more accurate and quickly XCSR will learn. The recent book on kernel methods can provide a solid basis for successful XCSR applications to new problem domains [19].

XCSR may be applied to other predictive problems, such as reinforcement learning problems or predictive control problems, in which the prediction is not an actual function value but a consequent sensory input or sensory change [2]. Albeit in a different way than Holland's original vision [16], once such applications are successfully carried through, the enhanced XCS system may be more similar to an actual part of a cognitive system than ever before.

Finally, it should be mentioned that this study has been carried through with XCSR. Due to the generally similar principles of other LCSs, the results should carry over to other LCS frameworks, such as ZCS [21, 6]. Additionally necessary modifications and adjustments remain to be investigated.

Acknowledgments

This work was sponsored by the German research foundation (DFG) under grant DFG HO1301/4-3 as well as from the European commission contract no. FP6-511931.

5. REFERENCES

[1] J. Bacardit, D. E. Goldberg, and M. V. Butz. Improving the performance of a Pittsburgh learning classifier system using a default rule, 2004. http://www.psychologie.uni-wuerzburg.de/IWLCS/.

[2] J. S. Barlow. *The Cerebellum and Adaptive Control*. Cambridge University Press, 2002.

[3] E. Bernadó, X. Llorà, and J. M. Garrell. XCS and GALE: A comparative study of two learning classifier systems and six other learning algorithms on classification tasks. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Advances in Learning Classifier Systems (LNAI 2321)*, pages 115–132. Springer-Verlag, Berlin Heidelberg, 2002.

[4] E. Bernadó-Mansilla and J. M. Garrell-Guiu. Accuracy-based learning classifier systems: Models, analysis, and applications to classification tasks. *Evolutionary Computation*, 11:209–238, 2003.

[5] L. B. Booker, D. E. Goldberg, and J. H. Holland. Classifier systems and genetic algorithms. *Artificial Intelligence*, 40:235–282, 1989.

[6] L. Bull and J. Hurst. ZCS redux. *Evolutionary Computation*, 10(2):185–205, 2002.

[7] L. Bull and T. O'Hara. Accuracy-based neuro and neuro-fuzzy classifier systems. *Proceedings of the Fourth Genetic and Evolutionary Computation Conference (GECCO-2002)*, pages 905–911, 2002.

[8] M. V. Butz, D. E. Goldberg, P. L. Lanzi, and K. Sastry. Bounding the population size to ensure niche support in XCS. IlliGAL report 2004033, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, 2004.

[9] M. V. Butz, D. E. Goldberg, and K. Tharakunnel. Analysis and improvement of fitness exploitation in XCS: Bounding models, tournament selection, and bilateral accuracy. *Evolutionary Computation*, 11:239–277, 2003.

[10] M. V. Butz, T. Kovacs, P. L. Lanzi, and S. W. Wilson. How XCS evolves accurate classifiers. *Proceedings of the Third Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 927–934, 2001.

[11] M. V. Butz, T. Kovacs, P. L. Lanzi, and S. W. Wilson. Toward a theory of generalization and learning in XCS. *IEEE Transactions on Evolutionary Computation*, 8:28–46, 2004.

[12] M. V. Butz and M. Pelikan. Analyzing the evolutionary pressures in XCS. *Proceedings of the Third Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 935–942, 2001.

[13] M. V. Butz, K. Sastry, and D. E. Goldberg. Tournament selection in XCS. *Proceedings of the Fifth Genetic and Evolutionary Computation Conference (GECCO-2003)*, pages 1857–1869, 2003.

[14] Z. Ghahramani and M. I. Jordan. Factorial hidden Markov models. *Machine Learning*, 29:245–273, 1997.

[15] J. H. Holland. Adaptation. In R. Rosen and F. Snell, editors, *Progress in theoretical biology*, volume 4, pages 263–293. Academic Press, New York, 1976.

[16] J. H. Holland and J. S. Reitman. Cognitive systems based on adaptive algorithms. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern directed inference systems*, pages 313–329. Academic Press, New York, 1978.

[17] P. L. Lanzi. Extending the Representation of Classifier Conditions Part I: From Binary to Messy Coding. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, pages 337–344, 1999.

[18] P. L. Lanzi. Extending the Representation of Classifier Conditions Part II: From Messy Coding to S-Expressions. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, pages 345–352, 1999.

[19] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, Cambridge, UK, 2004.

[20] C. Stone and L. Bull. For real! XCS with continuous-valued inputs. *Evolutionary Computation*, 11:299–336, 2003.

[21] S. W. Wilson. ZCS: A zeroth level classifier system. *Evolutionary Computation*, 2:1–18, 1994.

[22] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[23] S. W. Wilson. Generalization in the XCS classifier system. *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 665–674, 1998.

[24] S. W. Wilson. Get real! XCS with continuous-valued inputs. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Learning classifier systems: From foundations to applications (LNAI 1813)*, pages 209–219. Springer-Verlag, Berlin Heidelberg, 2000.

[25] S. W. Wilson. Function approximation with a classifier system. *Proceedings of the Third Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 974–981, 2001.

[26] S. W. Wilson. Mining oblique data with XCS. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Advances in learning classifier systems: Third international workshop, IWLCS 2000 (LNAI 1996)*, pages 158–174. Springer-Verlag, Berlin Heidelberg, 2001.

[27] S. W. Wilson. Classifiers that approximate functions. *Natural Computing*, 1:211–234, 2002.