Support Vector Regression for Classifier Prediction

Daniele Loiacono[†], Andrea Marelli[†], Pier Luca Lanzi[†*]
[†]Artificial Intelligence and Robotics Laboratory (AIRLab)
Politecnico di Milano. P.za L. da Vinci 32, I-20133, Milano, Italy
[*]Illinois Genetic Algorithm Laboratory (IlliGAL)
University of Illinois at Urbana Champaign, Urbana, IL 61801, USA
loiacono@elet.polimi.it, am673447@lau.polimi.it, lanzi@elet.polimi.it

ABSTRACT

In this paper we introduce XCSF with support vector prediction: the problem of learning the prediction function is solved as a support vector regression problem and each classifier exploits a Support Vector Machine to compute the prediction. In XCSF with support vector prediction, XCSFsvm, the genetic algorithm adapts classifier conditions, classifier actions, and the SVM kernel parameters. We compare XCSF with support vector prediction to XCSF with linear prediction on the approximation of four test functions. Our results suggest that XCSF with support vector prediction compared to XCSF with linear prediction (i) is able to evolve accurate approximations of more difficult functions, (ii) has better generalization capabilities and (iii) learns faster.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Learning Classifier Systems*

General Terms

Algorithms, Performance

Keywords

Learning Classifier Systems , Support Vector Machines, Computed Prediction, XCS, Genetic Algorithms

1. INTRODUCTION

XCSF [26] extends the idea of learning classifier systems through the introduction of a computable classifier prediction. In XCSF classifier prediction is not memorized into a parameter but computed as a linear combination of the current input and a weight vector associated to each classifier. Thus, the original update of the prediction parameter [24] is replaced by the update of the parameters of the prediction function. In [26], the prediction function is defined as the linear combination $\mathbf{s} \cdot \mathbf{w}$, where $\mathbf{s}$ is the current input and $\mathbf{w}$ is the parameters vector associated to each classifier, but more complex functions can be used [10, 11]. As shown in [12] the problem of learning the prediction function is basically an incremental parameters estimation problem that can be solved with many parameters estimation techniques.

In this paper, we show that the problem of learning the prediction function can also be solved as a support vector regression problem [21], i.e., using Support Vector Machines (SVMs) to compute classifier prediction. SVMs are a widely used machine learning technique with both a sound theoretical background and a lot of practical successful applications. We introduce XCSF with support vector prediction, dubbed XCSFsvm, in which classifiers prediction is computed using SVMs. In XCSFsvm classifier prediction is updated with a well known SVM training techniques adapted to the online learning setting used in XCSF. The extension of XCSF with SVMs has two main advantages: (i) it sets the problem of learning the classifier prediction function as a well-posed quadratic optimization problem and (ii) it can tackle non-linear prediction function using the so-called "*kernel trick*" [21]. One of the known drawbacks of SVMs is that they require suitable values for the kernel parameters which are typically set through empirical search and experience. In XCSFsvm, we let evolution do the job, so that the system is in charge of finding the adequate parameters for the SVM kernels. Thus, in XCSFsvm the genetic algorithm works both on the partitioning of the problem space (as it usually happens) *and* on the search for the most suitable SVM kernel parameters. We compared XCSF with our XCSFsvm on four functions inspired to those used by Butz [1, 2]. Our results suggest that XCSFsvm is not only able to evolve accurate approximations of functions that are too difficult for XCSF [26, 9], but it also generalizes significantly better than XCSF.

The paper is organized as follows. In Section 2 we introduce the basics of support vector regression. Then, we show how support vector regression is extended for online learning (Section 3). In Section 4 we describe XCSF and then we show how we extended XCSF by adding support vector regression for classifier prediction (Section 5). The experimental design is described in Section 6 while in Section 7 we report the experimental results.

2. SUPPORT VECTOR REGRESSION

Consider a given set of training samples $\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell)\} \subset X \times \mathbb{R}$, where X is the input space (e.g. $X \equiv \mathbb{R}^3$) and y_i is the target associated to each input $\mathbf{x}_i$. We define $\varepsilon-$SV regression [21] as the problem of finding a function, $f : X \mapsto \mathbb{R}$, such that $f(\mathbf{x}_i)$ has at most a deviation ε from the target y_i for all the training samples. In order to be a good solution, f should be as *flat* as possible [19], i.e. the smoothest function with at most an ε deviation from the targets.

In the simplest case the problem can be solved with a linear function, $f(x) = \mathbf{w} \cdot \mathbf{x} + b$, where $\mathbf{w} \in X$ and $b \in \mathbb{R}$. In this particular case the *flattest* solution is the smallest $\mathbf{w}$ [19], such that

$$|\mathbf{w} \cdot \mathbf{x}_i + b - y_i| \leq \varepsilon, \ \forall i. \tag{1}$$

Unfortunately very often this problem has no solution and thus the $\varepsilon-$SV regression problem is usually defined as

$$\min \frac{1}{2} \| \mathbf{w} \|^2 + C \sum_{i}^{l} (\xi_i + \xi_i^*), \tag{2}$$

subject to ($\forall i = 1 \cdots \ell$):

$$y_i - \mathbf{x}_i \cdot \mathbf{w} - b \leq \varepsilon - \xi_i; \quad \mathbf{x}_i \cdot \mathbf{w} + b - y_i \leq \varepsilon - \xi_i^*, \tag{3}$$

where ξ_i and ξ_i^*, called the *slack* variables, represents at what degree the original problem constraints (Equation 1) are violated by the solution; the positive constant, C, represents a trade-off between the accuracy and the *flatness* of the solution.

The optimization problem defined by Equation 2 and Equation 3 can be often solved more easily in its dual formulation (for the details we refer the reader to [19]).

In order to extend this approach to non linear regression problems it is possible to introduce a mapping function, $\phi : X \mapsto \mathcal{H}$, that maps the input space X in a suitable *feature space* $\mathcal{H}$ where the regression problem can still be solved by a linear function $f(\mathbf{x}) = \mathbf{w} \cdot \phi(\mathbf{x}) + b$. In practice, in many problems the dimensionality of such a suitable feature space grows exponentially and this approach would quickly become infeasible. Luckily the solution of the optimization problem associated to the regression problem can be expressed solely in terms of the *kernel function*, $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x}) \cdot \phi(\mathbf{x}')$, i.e. the inner product in $\mathcal{H}$. This key observation, usually referred to as the *kernel trick*, allows to solve the regression problem in a feasible way, because a large class of mapping function $\phi(\cdot)$ admits an easy to compute kernel. Accordingly the solution of the regression problem becomes [19]:

$$f(\cdot) = \sum_{i=1}^{\ell} \beta_i k(\mathbf{x}_i, \cdot) + b, \tag{4}$$

where the coefficients β_i and b are obtained solving the dual formulation of the optimization problem defined by Equation 2 and Equation 3. It is worthwile observing that, in the solution reported by Equation 4, many β_i coefficients are usually zero: only few training samples plays a role in the computation of the regression problem solution, i.e. the training samples associated to non zero β_i coefficients. These particular training samples are usually called Support Vectors (SVs) [21].

3. ONLINE LEARNING WITH SUPPORT VECTOR REGRESSION

Solving the $\varepsilon-$SV regression problem introduced in the previous section involves a non-trivial quadratic programming optimization that is, in general, computationally quadratic both in time and memory with respect to the number of training samples. Many efforts have been done for solving efficiently such optimization problem [6, 16] but the proposed algorithms usually require all the training samples at once, i.e. they solve a *batch* learning problem. In this paper instead we only deal with the *online* learning problem, that is at each time step a new training sample is available. A lot of incremental algorithms have been introduced in literature either for computing an approximate solution [7, 22] or the exact solution [14, 15, 4] of the optimization problem. Unfortunately these incremental algorithms are not widely used in practice basically for two reasons [13]: (i) there is not yet any standard available implementation of such algorithms; (ii) they are rather complex and tricky to implement in order to get a good computational performance.

A completely different approach is the one of *chunking methods* [21, 17, 18, 20]. The basic idea of chunking methods is that of solving at each time step a new support vector regression problem from scratch using a small subset of the training data. The selection of such training subset thus a key role in the chunking methods. A common approach [21, 18] is using at each time step, as training set, the last available training sample and the actual set of SVs.

Here we implemented a slightly different version of chunking methods. The main drawback of chunking methods is actually that the complexity of the solution, i.e. the number of SVs, can grow indefinitely. Even if some methods for keeping small the number of SVs have been introduced in the literature [7], this is a serious issue for us both because it results in a significant slowdown of the training process and because our aim is solving many simple local regression problems. To deal with this issue we introduced a superior bound, θ_{SV}, to the number of SVs used. At time step, t, a new training sample $(\mathbf{x}_t, y_t)$ is presented to the Support Vector Machine (SVM) that is updated as described in Algorithm 1. At first the SVM is used to predict the output of the current training sample, `SVM(`$\mathbf{x}_t$`)`; if the predicted output differs more than ε from the target, i.e. $|\text{SVM}(\mathbf{x}_t) - y_t| > \varepsilon$ (line 2, Algorithm 1), a new SVM is trained; the training set, T, is built by adding the latest training sample, $(\mathbf{x}_t, y_t)$, to the actual set of SVs (line 7, Algorithm 1). If the number of SVs is greater than a fixed threshold θ_{SV}, the oldest one is dropped[1]. The resulting data set, T, is used for training a new SVM from scratch with a standard procedure for solving a batch support vector regression problem (line 8, Algorithm 1). In our implementation we used the well known `LIBSVM` [5] library for performing the underlying batch training of SVMs.

Algorithm 1 Incremental update of SVM.

```
 1: procedure UPDATE(SVM,$\mathbf{x}$, $y$)
 2:     if ($|y$- SVM($\mathbf{x}$) $| > \epsilon$) then
 3:         $T \leftarrow$SVM.SVs
 4:         if $|T| > \theta_{SV}$ then
 5:             Remove the oldest SV from T
 6:         end if
 7:         $T \leftarrow T \cup \{(\mathbf{x}, y)\}$
 8:         SVM $\leftarrow$ TRAIN($T$)
 9:     end if
10: end procedure
```

4. DESCRIPTION OF XCSF

In XCSF [26], computed prediction replaces the usual classifier prediction with a parameter vector $\mathbf{w}$ and a prediction function $p(\mathbf{s}_t, \mathbf{w})$, which defines how classifier prediction is computed from the current input $\mathbf{s}_t$ and parameter vector $\mathbf{w}$.

Classifiers consist of a condition and four main parameters. Since as in [26], we focus on function approximation problems, classifiers do not have actions. The condition is represented by a concatenation of interval predicates, $int_i = (l_i, u_i)$, where l_i ("lower") and u_i ("upper") are real values. The four parameters are: the weight vector $\mathbf{w}$, used to compute the classifier prediction as a function of the current input; the prediction error ε, that estimates the error affecting classifier prediction; the fitness F that estimates the accuracy of the classifier prediction; the numerosity *num*, a counter

[1]At each time step, the training set built has at most $\theta_{SV}+1$ training samples, resulting in no more than $\theta_{SV}+1$ SVs. Thus at most one support vector requires to be dropped at the next training step.

used to represent different copies of the same classifier. The weight vector $\mathbf{w}$ has one weight w_i for each possible input, and an additional weight w_0 corresponding to a constant input x_0, that is set as a parameter of XCSF (in this work x_0 is always set to 1).

Performance Component. At each time step t, XCSF builds a *match set* [M] containing the classifiers in the population [P] whose condition matches the current sensory input $\mathbf{s}_t$; if [M] contains less than θ_{mna} actions, *covering* takes place as in XCSI [25, 26]. The weight vector $\mathbf{w}$ of covering classifiers is initialized with zero values (note that originally [26], the weights were initialized with random values in [-1,1]); all the other parameters are initialized as in XCS (see [3]).

XCSF computes the *system prediction* of the classifiers in [M] which, at time t for input $\mathbf{s}_t$ is defined as,

$$P(\mathbf{s}_t) = \frac{\sum_{cl \in [M]} cl.p(\mathbf{s}_t) \times cl.F}{\sum_{cl \in [M]} cl.F}, \tag{5}$$

where *cl* is a classifier, [M] is the match set, *cl.F* is the fitness of *cl*; $cl.p(\mathbf{s}_t)$ is the prediction of *cl* in state $\mathbf{s}_t$, which is computed as:

$$cl.p(\mathbf{s}_t) = cl.w_0 \times x_0 + \sum_{i>0} cl.w_i \times \mathbf{s}_t(i),$$

where $cl.w_i$ is the weight w_i of cl. Then XCSF performs a dummy action (there are actually no actions involved in function approximation problems [26]) and a reward P is returned to the system.

Reinforcement Component. XCSF uses the incoming reward P to update the parameters of classifiers in [M]. In the original XCSF, the weight vector w is updated using a *modified delta rule* [23]. In this paper, we use the most recent recursive least square update introduced in [9]. After the prediction of the classifiers in [M] is updated according to the target value P. Finally, the prediction error ε and the classifier fitness is updated as in XCS.

Discovery Component. The genetic algorithm in XCSF works as in XCSI [25]. On a regular basis depending on the parameter θ_{ga}, the genetic algorithm is applied to classifiers in [A]. It selects two classifiers with probability *proportional to their fitness*, copies them, and with probability χ performs crossover on the copies; then, with probability μ it mutates each allele. Crossover and mutation work as in XCSI [25, 26].

5. EXTENDING XCSF WITH SVM

It is rather straightforward to extend XCSF with classifier prediction based on support vector regression. In XCSF with support vector prediction, briefly XCSFsvm (i) each classifier cl_k has an associated SVM for regression, $cl_k.\texttt{SVM}(\cdot)$, that is used to compute the prediction; (ii) the update of the classifier parameter vector $\mathbf{w}$ (Section 4) is replaced by the update of $cl_k.\texttt{SVM}$ according to Algorithm 1; (iii) the genetic algorithm is applied both to classifier conditions and to the parameters of the SVM associated to each classifier.

A support vector machine for regression (Section 2) is completely described by the pair $\langle \epsilon, C \rangle$ (see Equation 2 and Equation 3) and by the kernel function $k(\cdot,\cdot)$ used. In our implementation of XCSFsvm, we used the RBF kernel, provided by the LIBSVM [5] library, which is defined as,

$$k(\mathbf{x}, \mathbf{x}') = e^{-\gamma \|\mathbf{x}-\mathbf{x}'\|^2}, \tag{6}$$

where $\gamma \in \mathbb{R}$ is a kernel parameter. The performance of this type of SVM is highly affected by the value of γ which is problem dependent. For this reason, in XCSFsvm the value of γ is not fixed but it is evolved. The parameter ε is set as the error threshold ϵ_0 that is used in XCSF to control classifier accuracy. In XCSFsvm the parameters vector $\mathbf{w}$ used in XCSF [26] is replaced by the kernel parameter γ and a set of, at most, θ_{SV} support vectors. New classifiers are initialized with an empty set of support vectors and with a value of γ uniformly chosen in $[\gamma_{min}, \gamma_{max}]$, where γ_{min} and γ_{max} are system parameters

At each time step, the prediction of each classifier cl_k in [M] is computed as the output of the SVM associated to the classifier, $cl_k.\texttt{SVM}(\mathbf{s}_t)$. Then the system prediction $P(\mathbf{s}_t)$ of the classifiers in [M] is computed. For each classifier cl_k in the match set [M], $cl_k.\texttt{SVM}$ is updated according to the procedure $\texttt{UPDATE}(cl_k.\texttt{SVM}, \mathbf{s}_{t-1}, P)$ described in Algorithm 1. The prediction error ϵ and the fitness are instead updated as usual [3].

The genetic algorithm works as in XCSF [26] for what concerns the classifier condition. In addition in XCSFsvm genetic algorithm is also applied to the kernel parameter γ. With probability χ crossover is applied and the γ values of the offsprings are recombined. In this work we compared two different recombination strategies. The first one simply switches the γ values between the two offsprings, the second one set γ in both the offsprings to the average of γ values of the parents. The resulting versions of XCSFsvm are dubbed respectively XCSFsvm-χ_1 and XCSFsvm-χ_2. Finally, with probability μ, mutation is applied and γ is either increased or decreased of an amount uniformly drawn from the continuous interval $(0, \gamma_0]$. The initialization of the set of SVs in the offsprings is straightforward: each offspring inherits the parent SVs.

Computational Complexity. XCSFsvm requires to store, for each classifiers, the value of γ, the coefficient b, and at most θ_{SV} support vectors with the respective targets y_i and coefficients β_i; thus, in the worst case, the additional memory requirements of XCSFsvm results in a space complexity of order $O(n \cdot \theta_{SV})$, where n is the dimensionality of the in the input space. The update of classifier SVMs is based on the LIBSVM optimized training algorithm that has time complexity of order $O(n \cdot \theta_{SV})$ [5]. Therefore the computational complexity of XCSFsvm is strongly influenced by the choice of the threshold θ_{SV}. In this paper we compared XCSFsvm to the XCSF version with recursive least squares update [9], which complexity is $O(n^2)$ both in time and in space. XCSF can be computationally more expensive than XCSFsvm if $n \gg \theta_{SV}$ but this is not the case of the problems considered in this paper, where $n \leq 4$ and $\theta_{SV} = 10$.

6. DESIGN OF EXPERIMENTS

In this work, we followed the standard experimental design used in the literature [24]. Each experiment consists of a number of problems that the system must solve. Each problem is either a *learning* problem or a *test* problem. Classifiers parameters are always updated. The genetic algorithm is enabled only during *learning*, and it is turned off during *test*. The covering operator is always enabled, but operates only if needed. Learning problems and test problems alternate.

6.1 Test Functions

We tested XCSF with support vector prediction, XCSFsvm, on the approximation of the four real functions reported in Table 1, that are a generalization of the ones introduced in [1]. The functions are continuous; F_1, F_2, and F_4 are normalized so that their range is $[-1, 1]$; the range of F_3 is $[0, 1]$. Figure 1 shows the plots of the four functions when only two input variables are used. The function

$$\begin{aligned}
F_1(x_1,\cdots,x_n) &= \frac{1}{n}\sum_{i=1}^{n}\sin(2\pi x_i)\\
F_2(x_1,\cdots,x_n) &= \sin\left(2\pi\sum_{i=1}^{n}x_i\right)\\
F_3(x_1,\cdots,x_n) &= \frac{1}{\sqrt{2}}\left|\sin\left(2\pi\sum_{i=1}^{n}x_i\right)+\cos\left(2\pi\sum_{i=1}^{n}x_i\right)\right|\\
F_4(x_1,\cdots,x_n) &= \sin\left(2\pi\sum_{i=1}^{n-1}x_i+\sin(x_n)\right)
\end{aligned}$$

Table 1: The four test functions used in this paper; $x \in [0,1]$, $n \geq 2$.

F_1 is the simplest one since it is additively separable. Accordingly, it is easily approximated by XCSF (see for instance [1]). This does not hold in F_2 which is therefore more difficult to approximate with XCSF [1]. Function F_3 and F_4 present additional difficulties: the former is not continuously differentiable and the second one has a continuously changing slope. All the statistics reported in this paper are averages over 10 experiments.

6.2 Statistical Analysis

To analyze the results reported in this paper, for each experiments and for every tested setting, we collected (i) the average of the absolute system error in the first 50000 test problems, to measure the convergence speed; (ii) the average of the absolute system error in the last 50000 test problems, to measure the accuracy of the evolved solutions; (iii) a fitness-weighted average of the classifiers generality (i.e. the hypervolume identified by the classifier conditions in the input space) in the final populations, to measure the level of generalization achieved. On these data, we applied a one-way analysis of variance (ANOVA) [8] to test whether there was some statistically significant differences; we also applied four *post hoc tests* [8], (Tukey HSD, Scheffé, Bonferroni, and Student-Neumann-Keuls) to find which settings performed significantly different.

7. EXPERIMENTAL RESULTS

We compared XCSF with the two models of XCSFsvm, XCSFsvm-χ_1 and XCSFsvm-χ_2, on four test functions (Table 1) using different number of inputs. Our goal was threefold. First, we wanted to test how the choice of the parameter γ would affect the performance of the XCSFsvm approaches. In addition, we also wanted to test whether the XCSFsvm models could evolve the adequate value of the kernel parameter γ. Then, we wanted to study how the performance of XCSF and of the XCSFsvm models would be influenced by the problem size.

7.1 Fixed γ vs. Evolved γ

In this first set of experiments we analyzed how the choice of the kernel parameter, γ, affects the system performances. For this purpose we used a slightly modified version of XCSFsvm in which the value of γ is initialized to a given fixed value and is neither mutated nor recombined. Then we compared XCSFsvm with different values of γ on the four functions in Table 1 with two input variables. We also compared such XCSFsvm model that keeps a fixed value of γ with XCSFsvm-χ_1 and XCSFsvm-χ_2, where the value of γ is randomly initialized and can be both mutated and recombined. All the experiments were performed with the following parameters setting: $N = 6400$; $\eta = 0.5$; $\beta = 0.5$; $\alpha = 0.1$; $\nu = 5$; $\chi = 1.0$, $\mu = 0.05$, $\epsilon_0 = 0.05$; $\theta_{del} = 20$; $\theta_{GA} = 20$; $\delta = 0.1$; GA-subsumption is on with $\theta_{sub} = 50$; while action-set subsumption is off; $r_0 = 1.0$, $m_0 = 0.5$; in XCSFsvm-χ_1 and XCSFsvm-χ_2 we used $C = 30$, $\gamma_{min} = 0$, $\gamma_{max} = 100$, and $\gamma_0 = 10$. For each function we tested the following values of γ: (i) $\gamma = 0.05$, (ii) $\gamma = 100$ and (iii) the best value of γ experimentally found. It is worthwile observing that the best value of γ is different for each function tested. Figure 2a compares performance of XCSFsvm-χ_1, XCSFsvm-χ_2, and XCSFsvm with $\gamma \in \{0.5, 10, 100\}$ on the approximation of function $F_1(x_1, x_2)$. All the systems compared are able to evolve an accurate approximation of the target function, but XCSFsvm with $\gamma = 10$ (the best value empirically found) learns faster than all the other settings; however XCSFsvm-χ_1 and XCSFsvm-χ_2 learn faster than XCSFsvm with $\gamma = 0.5$ and $\gamma = 100$. Figure 2b compares the performance of XCSFsvm-χ_1, XCSFsvm-χ_2, and XCSFsvm with $\gamma \in \{0.5, 32, 100\}$ on the approximation of function $F_2(x_1, x_2)$. On this function XCSFsvm-χ_1 and XCSFsvm-χ_2 learns even faster than XCSFsvm with $\gamma = 32$ (the best value empirically found). The results on the more complex functions, $F_3(x_1, x_2)$ and $F_4(x_1, x_2)$, show (Figure 2c and Figure 2d) that XCSFsvm-χ_1 and XCSFsvm-χ_2 learn roughly as fast as XCSFsvm with the best fixed value of γ. Overall our finding suggest that evolving γ not only does not slow down the learning in XCSFsvm but can even result in a faster learning, suggesting that the most suitable value of γ may change either during the learning process or in different subspace of the input space.

7.2 Two variables

In this set of experiments we applied XCSF, XCSFsvm-χ_1, and XCSFsvm-χ_2 to the approximation of the four functions in Table 1 when $n = 2$. The parameters are set as in the previous experiments. For each function and for each version of XCSF we compared, Table 2a reports the average absolute system error in the first 50000 test problems. The data in Table 2a show that XCSFsvm-$\chi_1$and XCSFsvm-χ_2 learn faster than XCSF and after the same number of learning problems reach a smaller prediction error. We applied a *one-way* analysis of variance (ANOVA [8]) to test whether the differences reported in Table 2a are statistically significant. Then, we applied the typical post-hoc procedures (SNK, Tukey, Scheffé, and Bonferroni) to analyze the differences among the three systems The analysis of variance shows that the differences in Table 2a are statistically significant at the 99.99% confidence level; the subsequent post-hoc procedures cluster the three versions of XCSF into two groups with similar (not significantly different) performances. The first group contains only XCSF, the second contains both XCSFsvm-χ_1 and XCSFsvm-χ_2. This suggests that the difference between XCSF and XCSFsvm is statistically significant, i.e., XCSFsvm models are faster than XCSF with a probability of the 99.99%. However, the two versions of XCSFsvm perform similarly, the difference in their performance is not statistically significant. The average prediction error over the last 50000 test problems is reported in Table 2b. All the three systems learn an accurate solution for the four functions. The statistical analysis shows that XCSF is generally more accurate than XCSFsvm-χ_1 and XCSFsvm-χ_2 (the difference is statistically significant at the 99.99% confidence). But this is not surprising since, as Table 2c demonstrates, XCSF evolves solutions that are more specific and therefore tend to be more accurate. XCSFsvm models evolve accurate solutions (the final prediction error is smaller than the target threshold ϵ_0) that are significantly more general than those evolved by XCSF. The ANOVA shows that the differences in classifier generality reported

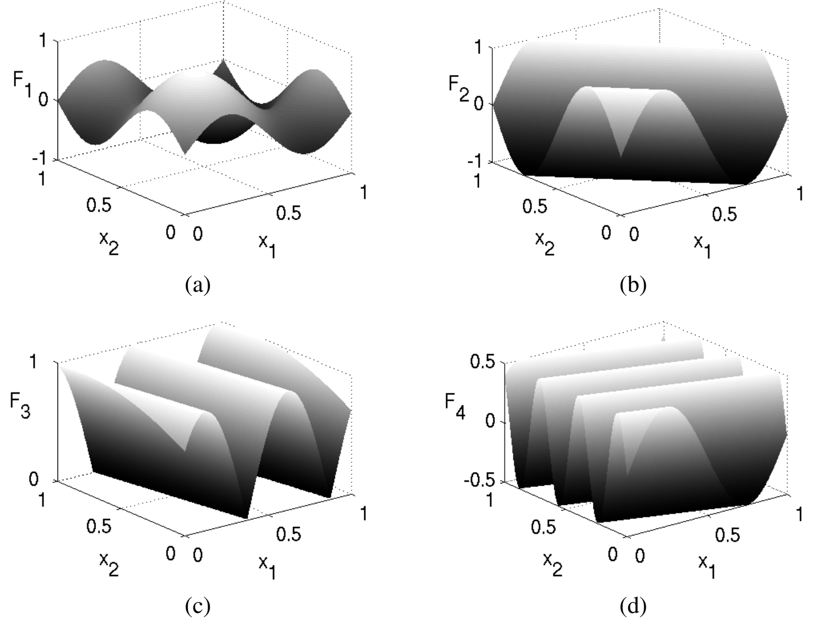

Figure 1: Test functions with two input variables: (a) $F_1(x_1, x_2)$**, (b)** $F_2(x_1, x_2)$**, (c)** $F_3(x_1, x_2)$**, (d)** $F_4(x_1, x_2)$

in Table 2c are statistically significant at the 99.99% confidence level. The post-hoc procedures cluster the three versions into three groups, one for each version of XCSF, i.e., all the three versions perform significantly differently. Thus, from the data in Table 2c, XCSFsvm-χ_1 evolves classifiers that are on the average more general than those evolved by XCSFsvm-χ_2 and XCSF.

Overall, the experiments with the functions of two variables show that (i) XCSFsvm models converges faster than XCSF to accurate solutions; (ii) at the end of the runs, the solutions evolved by XCSF are generally more accurate than those evolved by XCSFsvm models, however, they are also the less general: XCSFsvm-χ_1 and XCSFsvm-χ_2 evolve solutions that are significantly more general than those evolved by XCSF; (iii) although the solutions evolved by XCSFsvm-χ_1 and by XCSFsvm-χ_2 are equally accurate, XCSFsvm-χ_1 generalizes better than XCSFsvm-χ_2. This suggests that the recombination strategy of XCSFsvm-χ_1 may be more effective than the one used in XCSFsvm-χ_2, allowing for a more reliable generalization than XCSFsvm-χ_2 despite to an equivalent learning speed and accuracy.

7.3 Three variables

We repeated the same set of experiments increasing the number of variables up to three. The parameters are set exactly as in previous experiments, except for the population size N which in this case is set to 12800 in accordance with the settings in [2].

As the number of variables increases, the problems become more challenging so that XCSF cannot reach an accurate solutions for some of the functions. In contrast, XCSFsvm models always reach accurate solutions. Table 3b reports the prediction error in the last 50000 test problems for all the versions of XCSF. As it can be noted, XCSF model cannot evolve an accurate approximation for the most difficult functions, $F_3(x_1, x_2, x_3)$ and $F_4(x_1, x_2, x_3)$: the final prediction error is higher than the target threshold $\epsilon_0 = 0.05$. It can be noted that the solutions for the function F_1 with three variables evolved by XCSFsvm-χ_1 and XCSFsvm-χ_2 are actually more accurate than those evolved with only two variables: the errors in Table 3 are smaller than those in Table 2. On the other hand, even in the simple function F_1 the performance of XCSF decreases as the number of variables increases: the error in Table 3 for F_1 is larger than that in Table 2. These results suggest that, even in a linearly separable functions, XCSFsvm is able to scale up better than XCSF. The statistical analysis of the data in Table 3 shows that the differences between XCSF and the two XCSFsvm models are significantly different at the 99.99%, while the differences between XCSFsvm-χ_1 and XCSFsvm-χ_2 are not. XCSFsvm-χ_1 and XCSFsvm-χ_2 learn significantly faster than XCSF though these differences are statistically significant at the 99.99% confidence only in $F_2(x_1, x_2, x_3)$. Table 3c reports the average generality of the classifier in the final populations. Again, the data show that, when XCSF is able to reach an accurate solution, it still evolves solutions that are less general classifier than those evolved by XCSFsvm models. As in the previous experiments, XCSFsvm-χ_1 generalizes better than XCSFsvm-χ_2. In fact, the statistical analysis shows that at the 99.99% confidence level, the populations evolved by XCSFsvm-χ_1 are significantly more general than those evolved by XCSFsvm-χ_2.

7.4 Four variables

Finally, we applied XCSF, XCSFsvm-χ_1, and XCSFsvm-χ_2 to F_1 and F_2 with four variables. The parameters are set as in the previous experiment except for the population size N which is set to 51400 for F_1 and to 102800 for F_2. The results reported in Table 4 confirm what previously found. In the simple F_1, all the three models find an accurate solution but the solutions evolved by XCSFsvm-χ_1 and XCSFsvm-χ_2 are significantly more accurate than those evolved by XCSF. In the more difficult F_2, XCSF cannot reach an accurate solution, below the target error ϵ_0, and only XCSFsvm-χ_1 and XCSFsvm-χ_2 succeed in the task. As in all the previous experiments, in F_1, all the three systems find accurate solutions but those evolved by the XCSFsvm models are significantly more accurate than those evolved by XCSF. As before, the data of the average prediction error during the initial 100000 test problems

(Table 4a) shows that XCSFsvm-χ_1 and XCSFsvm-χ_2 learn faster than XCSF, though this difference is statistically significant only in function F_1. The previous findings on the generalization are also confirmed. The solutions evolved by XCSFsvm-χ_1 are more general than those evolved by XCSFsvm-χ_2 and by XCSF, though the difference is significant only in function F_1.

8. CONCLUSIONS

We have introduced XCSF with support vector prediction, briefly XCSFsvm, in which classifier prediction is computed using a Support Vector Machine. We proposed two XCSFsvm models, dubbed XCSFsvm-χ_1 and XCSFsvm-χ_2, for evolving the kernel parameter of classifier SVMs. The results presented suggest that XCSFsvm-χ_1 and XCSFsvm-χ_2 effectively adapt the kernel parameter and are able to learn an accurate solution for the target problem as fast as XCSFsvm with the best kernel parameter empirically determined. We have also compared XCSFsvm-χ_1 and XCSFsvm-χ_2 with XCSF on the approximation of four functions that are a generalization of the ones introduced in [1]. Our results suggest that the proposed XCSFsvm models are not only able to evolve accurate solutions for functions too difficult for XCSF but also learn significantly faster than XCSF, exploiting better generalization capabilities.

9. REFERENCES

[1] Martin V. Butz. Kernel-based, ellipsoidal conditions in the real-valued XCS classifier system. In *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, volume 2, pages 1835–1842, Washington DC, USA, 25-29 June 2005. ACM Press.

[2] Martin V. Butz, Pier-Luca Lanzi, and Stewart W. Wilson. Function approximation with XCS: Hyperellipsoidal conditions, recursive least squares, and compaction. (Available on request from `www.illigal.org/butz`), 2007.

[3] Martin V. Butz and Stewart W. Wilson. An algorithmic description of XCS. *Journal of Soft Computing*, 6(3–4):144–153, 2002.

[4] Gert Cauwenberghs and Tomaso Poggio. Incremental and decremental support vector machine learning. In *NIPS*, pages 409–415, 2000.

[5] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines. *Software available at* `http://www.csie.ntu.edu.tw/∽cjlin/libsvm`, 2001.

[6] Ronan Collobert and Samy Bengio. Svmtorch: support vector machines for large-scale regression problems. *J. Mach. Learn. Res.*, 1:143–160, 2001.

[7] Yaakov Engel, Shie Mannor, and Ron Meir. Sparse online greedy support vector regression. In *ECML '02: Proceedings of the 13th European Conference on Machine Learning*, pages 84–96, London, UK, 2002. Springer-Verlag.

[8] S. A. Glantz and B. K. Slinker. *Primer of Applied Regression & Analysis of Variance*. McGraw Hill, 2001. second edition.

[9] Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and Dave E. Goldberg. Generalization in the XCSF classifier system: Analysis, improvement, and extension. Technical Report 2005012, Illinois Genetic Algorithms Laboratory – University of Illinois at Urbana-Champaign, 2005.

[10] Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. Extending XCSF beyond linear approximation. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1827–1834, New York, NY, USA, 2005. ACM Press.

[11] Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. XCS with computed prediction for the learning of boolean functions. In *Proceedings of the IEEE Congress on Evolutionary Computation – CEC-2005*, pages 588–595, Edinburgh, UK, September 2005. IEEE.

[12] Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. Prediction update algorithms for XCSF: RLS, kalman filter, and gain adaptation. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1505–1512, New York, NY, USA, 2006. ACM Press.

[13] Pavel Laskov. Incremental support vector learning: Analysis, implementation and applications. *Journal of machine learning research*, 7:1909, 2006.

[14] Junshui Ma, James Theiler, and Simon Perkins. Accurate on-line support vector regression. *Neural Comput.*, 15(11):2683–2703, 2003.

[15] Mario Martin. On-line support vector machines for function approximation. Technical report, Universitat Politecnica de Catalunya, 2002.

[16] John C. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in kernel methods: support vector learning*, pages 185–208, Cambridge, MA, USA, 1999. MIT Press.

[17] Liva Ralaivola and Florence d'Alché Buc. Incremental support vector machine learning: A local approach. *Lecture Notes in Computer Science*, 2130:322–329, 2001.

[18] R. Rosipal and M. Gorilami. An adaptive support vector regression filter: A signal detection application. In *Artificial Neural Networks, 1999. ICANN 99. Ninth International Conference on (Conf. Publ. No. 470)*, volume 2, pages 603–607vol.2, 7-10 Sept. 1999.

[19] Alex J. Smola and Bernhard Schoelkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, 2004.

[20] Nadeem Ahmed Syed, Huan Liu, and Kah Kay Sung. Handling concept drifts in incremental learning with support vector machines. In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 317–321, New York, NY, USA, 1999. ACM Press.

[21] Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[22] S. Vijayakumar and S. Wu. Sequential support vector classifiers and regression. In *Int. Conf. Soft Computing*, pages 610–619, 1999.

[23] B. Widrow and M. E. Hoff. *Adaptive Switching Circuits*, chapter Neurocomputing: Foundation of Research, pages 126–134. The MIT Press, Cambridge, 1988.

[24] Stewart W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995. http://prediction-dynamics.com/.

[25] Stewart W. Wilson. Mining Oblique Data with XCS. volume 1996 of *Lecture notes in Computer Science*, pages 158–174. Springer-Verlag, April 2001.

[26] Stewart W. Wilson. Classifiers that approximate functions. *Journal of Natural Computating*, 1(2-3):211–234, 2002.

f	XCSF	XCSFsvm-χ_1	XCSFsvm-χ_2
$F_1(x_1, x_2)$	0.0940 ± 0.0132	0.0414 ± 0.0113	0.0450 ± 0.0113
$F_2(x_1, x_2)$	0.2408 ± 0.0878	0.0606 ± 0.0192	0.0703 ± 0.0207
$F_3(x_1, x_2)$	0.2289 ± 0.0134	0.1069 ± 0.0309	0.0813 ± 0.0114
$F_4(x_1, x_2)$	0.3807 ± 0.0575	0.1455 ± 0.0476	0.1293 ± 0.0322

(a)

f	XCSF	XCSFsvm-χ_1	XCSFsvm-χ_2
$F_1(x_1, x_2)$	0.0106 ± 0.0003	0.0183 ± 0.0006	0.0186 ± 0.0008
$F_2(x_1, x_2)$	0.0143 ± 0.0002	0.0187 ± 0.0028	0.0169 ± 0.0026
$F_3(x_1, x_2)$	0.0148 ± 0.0002	0.0211 ± 0.0005	0.0214 ± 0.0004
$F_4(x_1, x_2)$	0.0178 ± 0.0002	0.0141 ± 0.0005	0.0147 ± 0.0004

(b)

f	XCSF	XCSFsvm-χ_1	XCSFsvm-χ_2
$F_1(x_1, x_2)$	0.0340 ± 0.0004	0.1032 ± 0.0265	0.0868 ± 0.0216
$F_2(x_1, x_2)$	0.0130 ± 0.0002	0.0620 ± 0.0061	0.0455 ± 0.0118
$F_3(x_1, x_2)$	0.0093 ± 0.0001	0.0182 ± 0.0004	0.0151 ± 0.0003
$F_4(x_1, x_2)$	0.0077 ± 0.0001	0.0267 ± 0.0004	0.0227 ± 0.0006

(c)

Table 2: XCSF, XCSFsvm-χ_1, and XCSFsvm-χ_2 applied to two input variables functions: (a) absolute system error average in the first 50000 test problems, (b) absolute system error average in the last 50000 test problems, and (c) fitness-weighted generality of classifiers evolved by the systems. Curves are averages of 10 runs.

f	XCSF	XCSFsvm-χ_1	XCSFsvm-χ_2
$F_1(x_1, x_2, x_3)$	0.1036 ± 0.0116	0.0637 ± 0.0145	0.0796 ± 0.0262
$F_2(x_1, x_2, x_3)$	0.4315 ± 0.0585	0.2843 ± 0.0652	0.2659 ± 0.0484
$F_3(x_1, x_2, x_3)$	0.2668 ± 0.0012	0.2527 ± 0.0103	0.2460 ± 0.0124
$F_4(x_1, x_2, x_3)$	0.5122 ± 0.0430	0.4485 ± 0.0485	0.4422 ± 0.0403

(a)

f	XCSF	XCSFsvm-χ_1	XCSFsvm-χ_2
$F_1(x_1, x_2, x_3)$	0.0167 ± 0.0003	0.0124 ± 0.0004	0.0148 ± 0.0008
$F_2(x_1, x_2, x_3)$	0.0494 ± 0.0011	0.0207 ± 0.0011	0.0201 ± 0.0006
$F_3(x_1, x_2, x_3)$	0.1195 ± 0.0347	0.0426 ± 0.0022	0.0412 ± 0.0016
$F_4(x_1, x_2, x_3)$	0.0836 ± 0.0024	0.0318 ± 0.0007	0.0338 ± 0.0020

(b)

f	XCSF	XCSFsvm-χ_1	XCSFsvm-χ_2
$F_1(x_1, x_2, x_3)$	0.0128 ± 0.0001	0.0312 ± 0.0022	0.0169 ± 0.0049
$F_2(x_1, x_2, x_3)$	0.0021 ± 0.0001	0.0037 ± 0.0002	0.0032 ± 0.0003
$F_3(x_1, x_2, x_3)$	0.0021 ± 0.0008	0.0020 ± 0.0001	0.0019 ± 0.0001
$F_4(x_1, x_2, x_3)$	0.0017 ± 0.0001	0.0019 ± 0.0001	0.0017 ± 0.0001

(c)

Table 3: XCSF, XCSFsvm-χ_1, and XCSFsvm-χ_2 applied to three input variables functions: (a) absolute system error average in the first 50000 test problems, (b) absolute system error average in the last 50000 test problems, and (c) fitness-weighted generality of classifiers evolved by the systems.

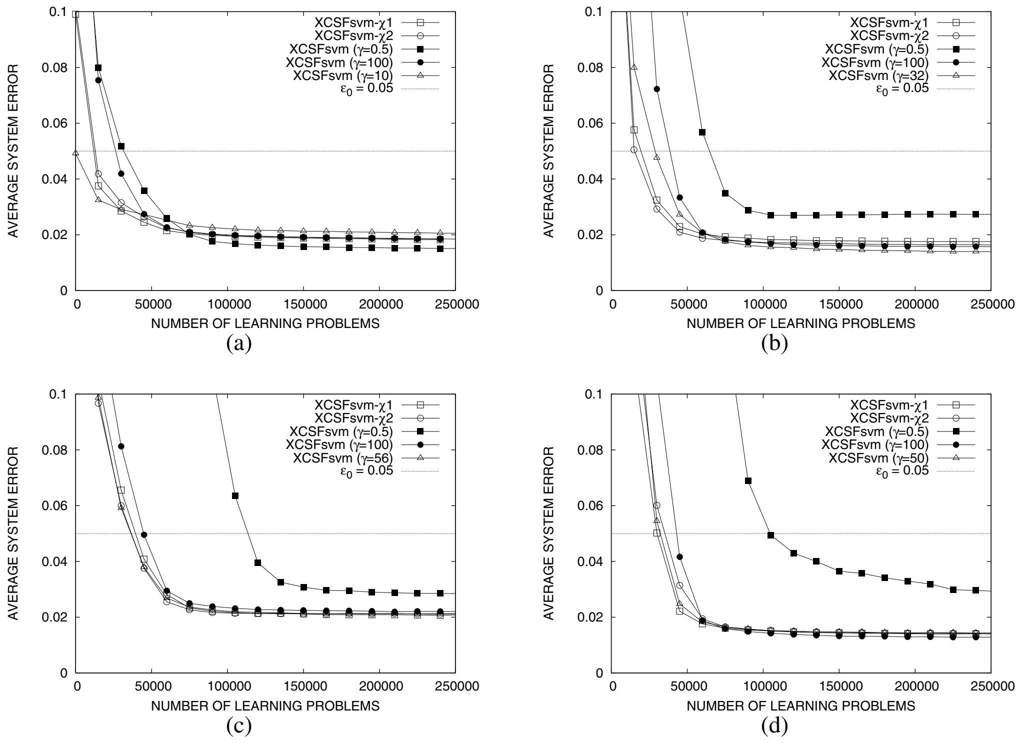

Figure 2: Performance of XCSFsvm-χ_1, XCSFsvm-χ_2, and XCSFsvm with a not evolved value of γ on (a) $F_1(x_1, x_2)$, (b) $F_2(x_1, x_2)$, (c) $F_3(x_1, x_2)$, and (d) $F_4(x_1, x_2)$.

f	XCSF	XCSFsvm-χ_1	XCSFsvm-χ_2
$F_1(x_1, x_2, x_3, x_4)$	0.0994 ± 0.0068	0.0651 ± 0.0062	0.0873 ± 0.0108
$F_2(x_1, x_2, x_3, x_4)$	0.5065 ± 0.0259	0.4596 ± 0.0579	0.4935 ± 0.0186

(a)

f	XCSF	XCSFsvm-χ_1	XCSFsvm-χ_2
$F_1(x_1, x_2, x_3, x_4)$	0.0180 ± 0.0001	0.0123 ± 0.0002	0.0152 ± 0.0010
$F_2(x_1, x_2, x_3, x_4)$	0.0834 ± 0.0006	0.0425 ± 0.0081	0.0434 ± 0.0014

(b)

f	XCSF	XCSFsvm-χ_1	XCSFsvm-χ_2
$F_1(x_1, x_2, x_3, x_4)$	0.0049 ± 0.0000	0.0063 ± 0.0002	0.0036 ± 0.0009
$F_2(x_1, x_2, x_3, x_4)$	0.0004 ± 0.0000	0.0004 ± 0.0001	0.0003 ± 0.0000

(c)

Table 4: XCSF, XCSFsvm-χ_1, and XCSFsvm-χ_2 applied to four input variables functions: (a) absolute system error average in the first 100000 test problems, (b) absolute system error average in the last 100000 test problems, and (c) fitness-weighted generality of classifiers evolved by the systems.