

# SupRB: A Supervised Rule-based Learning System for Continuous Problems

Michael Heider\*  
University of Augsburg  
Organic Computing Group  
Augsburg, Germany  
michael.heider@informatik.uni-augsburg.de

David Pätzel\*  
University of Augsburg  
Organic Computing Group  
Augsburg, Germany  
david.paetzel@informatik.uni-augsburg.de

Jörg Hähner  
University of Augsburg  
Organic Computing Group  
Augsburg, Germany  
joerg.haehner@informatik.uni-augsburg.de

## ABSTRACT

We propose the SupRB learning system, a new accuracy-based Pittsburgh-style learning classifier system (LCS) for supervised learning on multi-dimensional continuous decision problems. SupRB learns an approximation of a quality function from examples (consisting of situations, choices and associated qualities) and is then able to make an optimal choice as well as predict the quality of a choice in a given situation. One area of application for SupRB is parametrization of industrial machinery. In this field, acceptance of the recommendations of machine learning systems is highly reliant on operators' trust. While an essential and much-researched ingredient for that trust is *prediction quality*, it seems that this alone is not enough. At least as important is a *human-understandable explanation* of the reasoning behind a recommendation. While many state-of-the-art methods such as artificial neural networks fall short of this, LCSs such as SupRB provide human-readable rules that can be understood very easily. The prevalent LCSs are not directly applicable to this problem as they lack support for continuous choices. This paper lays the foundations for SupRB and shows its general applicability on a simplified model of an additive manufacturing problem.

## KEYWORDS

Learning Classifier Systems, Evolutionary Machine Learning, Manufacturing

## 1 INTRODUCTION

Parametrization of industrial machinery is often determined by human operators. These specialists usually obtained most of their expertise through year-long experimental exploration based on prior knowledge about the system or process at play. Transferring that knowledge to other operators with as little loss as possible (e. g. to new colleagues whenever experienced operators retire or to end users of the machinery after commissioning is finished) is a challenge: Humans' ability of exactly attributing parameter choices to the situations that led to them and then communicating this knowledge in a comprehensible manner tends to be rather restricted—which leads to new operators being forced to repeat exploration to learn for themselves. Machine learning (ML) can help with this, for example, by supporting new operators or users with recommendations or simply by recording existing experiences and extracting knowledge to make it available at a later point.

Parts of an operator's knowledge can be seen as a collection of mappings from parametrizations for the machine and variables beyond their influence to an expected process quality resulting from them—abstractly speaking, a collection of if-then rules with outcomes subject to noise. While many ML methods represent knowledge in a less or differently structured way, this is not the case for *learning classifier systems* (LCSs) whose models are collections of human-readable if-then rules constructed using ML techniques and model structure optimizers [12, 31]. This learning scheme is thus suited naturally to incorporate an operator's knowledge as externally specified rules can be included directly. Also, due to their inner structure, LCSs can more easily provide explanations for their predictions. Due to this transparency towards human users, compared to black box systems, an increased trust by operators that contained knowledge and thus recommendations are correct can be expected; which is essential for these system's actual applicability.

This paper proposes the SupRB learning system, a new accuracy-based Pittsburgh-style LCS for supervised learning on continuous multi-dimensional decision problems such as the one of parametrization of industrial machinery. Pittsburgh-style LCSs [26] have a model structure optimizer (in classic Pittsburgh-style systems, a genetic algorithm (GA)) operate on a population of rule collections of variable length each of which represents a potential solution to the learning problem at hand.

This work focuses on solving the problem of parametrization optimization of industrial machinery, which is defined in Section 2. An LCS architecture that solves this problem, SupRB, is introduced in Section 3 along with its first implementation SupRB-1 in Section 4. SupRB-1 is evaluated on different function approximation problems in Section 5. Section 6 gives an account of related research.

## 2 PARAMETRIZATION OPTIMIZATION

Parametrization optimization is the process of finding the best parameter choice, or *parametrization*, for a given system  $S$  with regard to some *quality measure*  $q$ . One such parametrization can be viewed as a vector  $a \in \mathcal{A} \subseteq \mathbb{R}^{D_{\mathcal{A}}}$  where  $\mathcal{A}$  is the parametrization space,  $D_{\mathcal{A}}$  is the number of parameters to be optimized and each component of  $a$  corresponds to one adjustable system parameter for  $S$ . Which parametrization is optimal regarding  $q$  depends on a number of environmental factors (e. g. ambient temperature or humidity) in addition to characteristics of process, machine, material and the part to be produced. For a given system  $S$ , we call one instance of those additional factors a *situation*; situations can again be assumed to be represented by a vector  $x \in \mathcal{X} \subseteq \mathbb{R}^{D_{\mathcal{X}}}$  where  $\mathcal{X}$  is called the situation space and  $D_{\mathcal{X}}$  is the fixed dimensionality of situations

\*These authors contributed equally to the paper.

for  $S$ . Having defined parametrizations and situations, we can now specify the quality measure’s form as

$$q : \{\mathcal{X}, \mathcal{A}\}^T \rightarrow \mathbb{R} \quad (1)$$

where every  $\{x, a\}^T = (x_1, \dots, x_{D_X}, a_1, \dots, a_{D_A})^T \in \{\mathcal{X}, \mathcal{A}\}^T$  is a stacked vector consisting of a situation  $(x_1, \dots, x_{D_X}) \in \mathcal{X}$  and a parametrization  $(a_1, \dots, a_{D_A}) \in \mathcal{A}$ . For readability, we write  $q(x, a)$  instead of  $q(\{x, a\}^T)$ . The target of  $q$  is a single scalar which is possibly derived appropriately from a vector of multiple quality features. We assume that  $q(x, a)$  is at least continuous in  $a$  which we think is realistic in most real-world scenarios:

$$\lim_{a \rightarrow a_0} q(x, a) = q(x, a_0) \quad (2)$$

With the definition for  $q$ , we can now define the optimization problem that describes the search for an optimal parametrizations for a given situation  $x$ :

$$\underset{a}{\text{maximize}} \quad q(x, a) \quad (3)$$

Note that, realistically, neither  $q$  nor its derivative can be assumed to be known (albeit either of those would simplify the problem greatly). Instead, we assume that the only information about  $q$  is a fixed set of examples.

Thus, the learning problem we consider is: Given a fixed set of  $N$  examples for situations and parametrizations  $\{\{x, a\}^T\}$  as well as their respective qualities  $\{q(x, a)\}$ , learn to predict for a given unknown situation  $x \in \mathcal{X}$  a parametrization  $\hat{a}_{\max}(x) \in \mathcal{A}$  for which

$$\hat{a}_{\max}(x) \approx a_{\max}(x) = \underset{a}{\operatorname{argmax}} \quad q(x, a) \quad (4)$$

where  $a_{\max}(x)$  is the actual optimal parametrization in situation  $x$ .

A natural way of measuring improvements on this learning problem is the following: A model can be said to be an improvement over another on a set of situations  $X_{\text{eval}} \subset X$  if the *actual quality* of the *predicted* optimal parametrizations on those situations is closer to the *actual quality* of the *actual* optimal parametrizations. This can be quantified, for example, by using the mean error for an error measure  $L$  on the model’s prediction:

$$\frac{1}{|X_{\text{eval}}|} \sum_{x \in X_{\text{eval}}} L(q(x, a_{\max}(x)), q(x, \hat{a}_{\max}(x))) \quad (5)$$

### 3 AN LCS ARCHITECTURE FOR CONTINUOUS PROBLEMS

This section presents a high-level view of SupRB, the overall LCS architecture we propose, in order to solve parametrization optimization problems which were introduced in the previous section.

#### 3.1 Model structure

Just like other LCSs, SupRB forms a *global* model from a population  $C$  of *local* models, called *classifiers*; in the case of SupRB, the global model is meant to approximate the quality measure  $q$  defined in Section 2. Each classifier is responsible for a subspace of the input space  $\mathcal{X}$ ; which subspace it is for a certain classifier is specified by that classifier’s *condition* which is also sometimes called its *localization*. The set of classifier conditions forms the overall model structure of SupRB; this structure fulfills a similar role as the graph structure of a neural network in that it needs to be chosen carefully in order for the system to perform well. At that, performing well

is not just about approximating  $q$  as close as possible; there are usually additional goals such as being explainable (cf. Section 1) which require the model structure’s complexity to be as low as possible.

#### 3.2 Local models: Classifiers

A classifier  $c$  consists of three main components:

- Some representation of a matching function  $m_c : \mathcal{X} \rightarrow \{\text{T}, \text{F}\}$ . We say that  $c$  *matches* situation  $x$  iff  $m_c(x) = \text{T}$ . Correspondingly, we say that  $c$  *does not match*  $x$  iff  $m_c(x) = \text{F}$ .
- Some local model approximating  $q$  on all  $x \in \mathcal{X}$  which the classifier matches.
- An estimation of the classifier’s goodness-of-fit on the situations it matches (solely used in classifier mixing).

Be aware that the classifiers’ matching functions’ domain is  $\mathcal{X}$  and not  $\{\mathcal{X}, \mathcal{A}\}^T$ . This increases explainability greatly as an ideal partitioning of  $\mathcal{X}$  (total, without overlaps) entails that there is exactly one rule regarding the parametrization for each possible situation. Conversely, if we partitioned in  $\{\mathcal{X}, \mathcal{A}\}^T$  optimally, there would possibly still be multiple rules for a given situation as the system might have partitioned in the dimensions of  $\mathcal{A}$  as well. Since we assume continuity of  $q(x, a)$  regarding  $a$  (see (2)), partitioning in  $\mathcal{A}$  would only be necessary if the local models could not capture  $q$ ’s behaviour in  $\mathcal{A}$ , for example because it is highly multi-modal. In that case, partitioning in  $\mathcal{A}$  might be sensible, as would using more sophisticated local models.

#### 3.3 Epoch-wise training

Training an LCS can generally be divided into two subproblems: For once, the classifiers’ local models need to be trained so that the predictions they make on the subspace they are responsible for are as accurate as possible. Secondly, the overall model structure of the LCS has to be optimized: the classifier’s localizations have to be aligned in such a way that every local model can capture the characteristics of the subspace it is assigned to as well as possible.

Michigan-style LCS such as XCS(F) [6, 37] or ExSTraCS [32] try to solve these problems incrementally by, for each seen example, performing a single update on some of the classifier’s local models and then improving these classifier’s localizations. This approach is especially sensible when learning has to be incremental (e. g. in reinforcement learning settings). However, due to the learning problem being non-incremental (all training data is available from the very start), SupRB can be trained non-incrementally. This means that training can be done in two separate phases that are repeated alternately until overall convergence [8], each phase being responsible for solving one of the subproblems:

- (1) (Re-)train each local classifier model on the data that it (now) matches.
- (2) Optimize the model structure (i. e. the set of classifier conditions), for example using a heuristic such as a GA.

At that, each phase is executed until it converges or some termination criterion, such as a fixed number of updates, is met. It is important to note that during fitting of each phase’s parameters, the parameters of the respective other are considered fixed—otherwise, convergence cannot be guaranteed. If a GA is used for the model

structure optimization, then this GA works on a population of classifier populations—these kind of systems are commonly called Pittsburgh-style LCS. However, since we expect many optimization methods to be applicable to this (see Section 7), an implementation of SupRB does not necessarily contain a GA. Nevertheless, the general SupRB architecture should probably be placed into or close to the Pittsburgh-style category. The *implementation* of SupRB we present in Section 4 is definitely a Pittsburgh-style LCS since it uses a GA to optimize the model structure.

Dividing the learning process into two distinct phases is advantageous. First of all, optimization of the process’s hyperparameters can be done more straightforwardly because hyperparameters are divided into two disjoint sets, one for each of the two phases. Besides that, the learning process is analysed more easily because the overall optimization problem of fitting the model to the data decomposes nicely into the two subproblems solved by the phases [8]—‘nicely’ meaning, that solving the subproblems independently of each other solves the overall problem. For example, if learning does not work and, upon inspection, the classifier weight updates converge correctly and fast enough, it is immediately clear that the model structure optimization is the culprit and corresponding measures can be taken.

### 3.4 Prediction

After training, SupRB can make two kinds of prediction. A *quality prediction* consists of using SupRB’s internal function approximation to predict the quality resulting from a certain parametrization  $a$  given a certain situation  $x$ . To do so, SupRB retrieves all classifiers from the classifier population that match  $x$ , that is, the set

$$M(x) = \{c \in C \mid m_c(x) = T\}. \quad (6)$$

The predictions of these classifiers then need to be *mixed* in order to yield the overall *system prediction* for the inputs,  $\hat{q}(x, a)$ . One way of mixing is a simple sum which is weighted by some accuracy measure  $F_c$  defined for each classifier  $c$ :

$$\hat{q}(x, a) = \sum_{c \in M(x)} F_c \hat{q}_c(x, a) \quad (7)$$

Here,  $\hat{q}_c(x, a)$  denotes the quality value that the local model of  $c$  predicts for parametrization  $a$  in situation  $x$ . It is important that

$$\sum_{M(x)} F_c = 1 \quad (8)$$

or otherwise the classifier’s combined predictions systematically over- or undershoot the actual value as the local models must be trained independently [8], which means that they are unaware of the other local model’s predictions during training.

A *parametrization choice* (or  $\hat{a}_{\max}$ -prediction) consists of predicting the best parametrization for a given fixed situation  $x_0$ , that is, performing (3) to yield (4)—which is the more central kind of prediction for parametrization optimization. The way of doing this highly depends on the used form of local models. For example, if the local models are polynomial functions of a degree of less than five, an exact analytical solution exists (Abel-Ruffini theorem) as partial derivatives can be used to find the set of local optima  $A_{\text{local}}$  from which SupRB then can retrieve the global optimum by using

its function approximation:

$$\hat{a}_{\max} = \operatorname{argmax}_{a \in A_{\text{local}}} \hat{q}(x_0, a) \quad (9)$$

For other functions, where an exact analytical solution is unknown or impractical, there are other options that range from root-finding algorithms [4] to heuristics such as hill climbing with random restarts [24], genetic algorithms [11] or chemical reaction optimization [15]. Although these non-analytical methods require a comparably larger amount of computation time, they are feasible in the setting SupRB targets: Industrial processes that are being optimized are usually preplanned anyway, which takes a lot longer than any of the heuristics needs to find SupRB’s parametrization choice.

## 4 SUPRB-1: A FIRST IMPLEMENTATION OF SUPRB

While the previous section introduced SupRB’s general architecture, learning process as well as its desired prediction capabilities, we now want to give a detailed account of SupRB-1, a first implementation of that system<sup>1</sup>.

### 4.1 Training and validation sets

SupRB-1 randomly splits the available training data,  $\{X, A\}^T$ , into two disjoint sets of configurable sizes,  $\{X, A\}_{\text{train}}^T \sqcup \{X, A\}_{\text{valid}}^T$ , a training and a validation set. The training set is used exclusively to fit the classifier’s local models to the data they match whereas the validation set is used exclusively to optimize the model structure. This approach is rather simplistic; incorporating more sophisticated sample management (k-fold cross validation etc.) is planned for the future.

To simplify representation and computation, we assume that parametrization and situation values are normalized to  $[-1, 1]^{D_{\mathcal{A}}}$  and  $[-1, 1]^{D_X}$ , respectively; this means that  $\mathcal{A} \simeq \mathcal{A}_{\text{actual}}$  needs to hold where  $\mathcal{A}_{\text{actual}}$  is the *actual* action value that is reported back to an external system. Given the context of optimizing parameters of industrial machinery it is reasonable to assume that upper and lower bounds for  $\mathcal{A}_{\text{actual}}$  exist in all cases which makes this normalization trivial.

### 4.2 Classifiers

Classifier conditions are interval-based using an ordered bound representation [29, 36]; an extension to hyper-ellipsoids [5] is already in the works.

All classifiers’ local models are a simple *linear regression*<sup>2</sup> on a subset of the *second order polynomial features* of the input which is fitted on  $\{X, A\}_{\text{train}}^T$  and  $q_{\text{train}}$ . In order to be able to analytically derive the  $\hat{a}_{\max}$ -prediction, we exclude all combinations of different dimensions of  $\mathcal{A}$  resulting in the following features set:

$$\{x_i x_j, x_i a_k, a_k^2 \mid i, j \in 1, \dots, D_X, k \in 1, \dots, D_{\mathcal{A}}\} \quad (10)$$

<sup>1</sup>Which we will make available in the camera-ready version.

<sup>2</sup>We use the one from the Python library *scikit-learn* [23].

The reasoning behind our choice for second order polynomial features instead of linear models alone is that a linear model’s maximum is always at one of the boundaries of the domain, if they exist.

The classifiers’ goodness-of-fit is measured using a *mean squared error* on  $\{X, A\}_{\text{train}}^T$ . We don’t use a separate validation set for estimating the goodness-of-fit in order to be as sample efficient as possible; in the industrial machinery context this system mainly targets, labeled data sets are comparably small. Nevertheless, a separate validation set for goodness-of-fit estimation would most certainly help the system to evolve better generalizing solutions faster.

### 4.3 GA for optimizing the model structure

We optimize the model structure (the classifier’s localizations) using a simple GA whose population consists of classifier populations. As already stated above, this makes SupRB-1 a Pittsburgh-style LCS.

The GA is *generational* with a configurable number of *elitists* (cf. for example [11]); a steady-state version was implemented as well but in a few short preliminary tests did not perform significantly better (albeit there is still no conclusive answer yet). The GA performs *mutation* and *crossover* on classifier populations.

For a single classifier population, mutation changes the bounds of the interval-based conditions of all classifiers by a normal distribution widened by a step-size  $s$ . Mutation steps are clipped at the minimum lower and maximum upper interval bounds,  $-1$  and  $1$  respectively (this can be disabled via a hyperparameter) in order to keep the hyperrectangle described by the classifier’s condition entirely within  $[-1, 1]^{D_X}$  (confer Section 4.1). Given any bound (lower or upper)  $b \in \mathcal{X}$ , its mutated value is distributed according to

$$\max(1, \min(-1, \{b + s * \mathcal{N}(\mu = 0, \sigma = 1)\})) \quad (11)$$

The step size  $s$  is initially set to one thousandth of the maximum interval width, which is  $\frac{2}{1000}$  in our normalized case. SupRB-1 adapts  $s$  by the well-known one-fifth rule as it is used in [1] with a small update factor of  $F = 1.05$ .

Crossover is done similarly as in [8] but using a normal distribution instead of a uniform one in order to keep offspring sizes closer together and less often generating really small offspring. Given two parents of lengths  $l_1$  and  $l_2$ , a number  $l'_1$  is drawn from  $\mathcal{N}(\mu = l_1 + l_2/2, \sigma = 1)$  repeatedly until  $1 \leq l'_1 \leq l_1 + l_2 - 1$  (the condition ensures that each offspring contains at least one classifier). After that, the classifiers of both parents are shuffled together and divided randomly among two children, one of size  $l'_1$ , the other of size  $l_1 + l_2 - l'_1$ . Performing this kind of crossover too often might be too disruptive making a crossover rate necessary.

SupRB selects parents for crossover using a simple *tournament selection* with tournaments of size 2 and the individual with the highest fitness always winning [21]. We measure fitness relatively between two individuals  $i_1$  and  $i_2$  based on their respective mean squared errors  $e_1$  and  $e_2$  on the validation set  $\{X, A\}_{\text{valid}}^T$  and their respective lengths  $l_1$  and  $l_2$  which is a naïve measure for their model structure’s complexity. Individual  $i_1$  wins the tournament if either of the following is true:

$$e_1 < e_2 \quad \wedge \quad l_1 \leq \frac{e_2}{e_1} l_2 \quad (12)$$

or

$$e_1 \geq e_2 \quad \wedge \quad l_1 \leq k \frac{e_2}{e_1} l_2 \quad (13)$$

where  $k \in [0, 1]$  is a hyperparameter weighing higher solution complexities against lower errors. This means that if  $i_1$ ’s error on  $\{X, A\}_{\text{valid}}^T$  is smaller than  $i_2$ ’s,  $i_1$ ’s complexity is allowed to be up to  $\frac{e_2}{e_1}$  ( $> 1$ ) times larger than the one of  $i_2$ . On the other hand, in order for  $i_1$  to win the tournament with a higher error, it needs to have an at least  $k \frac{e_2}{e_1}$  ( $< 1$ ) times smaller complexity than  $i_2$ .

### 4.4 Initialization

The GA’s population of classifier populations is initialized by randomly generating a number of individuals of a user-specified fixed size. We experimented with initializing randomly-sized individuals up to an upper bound to have a higher chance of finding the ‘correct’ individuals’ size early on. However, that did not (yet) work out—it seems that an initially larger overall amount of classifiers is more important than finding the correct solution size quickly.

Classifiers for an individual are generated randomly by sampling the bounds of their match function’s intervals uniformly from  $\mathcal{X}$ . Although this is one of the least sophisticated methods and results in a high chance of initial overlaps and unmatched examples, it seemed to result in far larger overall stability when compared to initializing classifier populations with evenly spaced individuals. The reason is probably the greater genetic diversity in the system.

At the end of initialization, all classifiers of all classifier populations are fitted to the examples from  $\{X, A\}_{\text{train}}^T$  that they match once.

### 4.5 Fitting classifiers

SupRB uses the most simple linear regression model from *scikit-learn*<sup>3</sup> as the local model for each classifier. These models provide a builtin means of fitting them to data which we use with standard parametrization, which minimizes the  $L^2$ -norm by Ordinary Least Squares.

### 4.6 Prediction

Due to the simplicity of the classifier’s local models, SupRB-1 can easily perform the two kinds of prediction the SupRB architecture postulates. To predict a quality value given a situation and an action,  $\{x, a\}^T$ , the linear regression models of all classifiers  $c$  that match  $x$  are queried for their respective predictions  $\hat{q}_c(\{x, a\}^T)$ . These predictions are then mixed with weights based on the classifiers’ normalized goodness-of-fit  $g$  which we calculate from their mean squared error on  $\{X, A\}_{\text{train}}^T$ . The unnormalized goodness-of-fit of a single classifier is:

$$g'_c = \frac{1}{e_c + 1} \quad (14)$$

where  $E = \sum_{c' \in \mathcal{M}(x)} e_{c'} + 1$  is the sum of the errors of all classifiers matching  $x$  and serves as normalization term for the error. Note the—for now—naïve addition of 1 to all errors in order to avoid zero terms. We further have to normalize  $g'$  again, in order to fulfill (8) yielding

$$g_c = \frac{g'_c}{G'} \quad (15)$$

<sup>3</sup>sklearn.linear\_model.LinearRegression

where  $G' = \sum_{c' \in M(x)} g_{c'}$  is the sum of the unnormalized goodness-of-fit values of the classifiers matching  $x$ . Substituting into (7) results in the following mixing model:

$$\hat{q}(x, a) = \sum_{c \in M(x)} g_c \hat{q}_c(x, a) \quad (16)$$

An parametrization choice  $\hat{a}_{\max}(x)$  for a given situation  $x$  again results from mixing each matching classifier  $c$ 's prediction. At that,  $\hat{a}_{\max_c}(x)$  can be derived analytically based on the implicit parabolooids that performing a linear regression on a second order polynomial feature space yields. For example,

- if the paraboloid opens downwards, the parametrization choice is the position of the vertex in  $\mathcal{A}$  whereas
- if the paraboloid opens upwards, it is one of the points in

$$\{(a_1, \dots, a_{D_{\mathcal{A}}}) \mid a_1, \dots, a_{D_{\mathcal{A}}} \in \{-1, 1\}\}. \quad (17)$$

The parametrization choices of the matching classifiers are then mixed using the same procedure based on their respective mean squared errors on  $\{X, A\}_{\text{train}}^T$  which gives

$$\hat{a}_{\max}(x) = \sum_{c \in M(x)} g_c \hat{a}_{\max_c}(x). \quad (18)$$

#### 4.7 Summary of hyperparameters

We now want to give a quick overview of all the hyperparameters introduced so far and a discussion of their impact.

Test set size is the percentage of the training data used exclusively for evaluating the individuals' fitness (see Section 4.1). This value is only really critical whenever there is only little data available as in that case, a trade-off has to be made between giving more data to the process of fitting local model predictions versus the problem structure optimization. By incorporating more sophisticated training data organisation techniques, however, this hyperparameter loses some if not most of its impact.

The size of the initial individuals corresponds to the initial number of classifiers in the system (see Section 4.4). Having enough genetic diversity from the start is extremely relevant, so a higher value is generally better. However, higher values naturally tend to increase the time until a compact solution is found.

The GA's population size is a less sensitive hyperparameter as long as there exist enough classifiers at the very start. Higher values allow the system to explore more search space in the same number of generations while generations themselves need more computation time. A similar argument goes for the number of elitists: While it is certainly important to have some amount of elitism for most problems in order to not accidentally forget good solutions, the actual amount of elitists in the population seems not to be that relevant.

Last, but not least,  $k$  (see (12) and (13)) is the most impactful hyperparameter as it directly interferes with the used fitness measure and thus with the evolutionary pressures within the system. A higher value (closer to 1) emphasizes the generation of less complex solutions while allowing for a higher error. In the long-term, this value should probably made dependant on the dimensionalities of the input space,  $D_X$  and  $D_{\mathcal{A}}$ , because in higher-dimensional spaces, a slight deviation from the intended target can lead to comparably larger errors than in spaces of lower dimensionality. We expect this

effect to lead to different behaviour for the same  $k$  on problems that only differ in their dimensionality but not in their general characteristics. However, we defer a closer look at this to future work.

The other hyperparameters have not that high of an impact and are discussed more in-depth in the publications referenced at their first mention. Table 1 gives a quick reference of all hyperparameters, their expected impact as well as a proposed default value.

## 5 EVALUATION

We evaluated SupRB-1 on two computable problems which are discussed together with the obtained results in the following sections.

### 5.1 Frog Problem

The 2-dimensional frog problem [38] was already used in the evaluation of systems with similar goals as SupRB, namely GCS [39], XCSFCA [30] and XCSRCA [14] (cf. Section 6 for more information on these systems), which is why it was chosen for this work as well. Essentially a reinforcement learning problem with episode length 1 and continuous states, actions and rewards, achieving maximal performance constitutes choosing an action which is equal to the situation.

$$P(x, a) = \begin{cases} x + a, & \text{if } x + a \leq 1 \\ 2 - (x + a), & \text{if } x + a \geq 1 \end{cases} \quad (19)$$

We trained and evaluated SupRB-1 with standard parameter settings (cf. Table 1) and  $k = 0.1$ . After 100 generations with only 50 training and 50 validation examples, the fitness elitist was able to consistently (averaged over 30 runs) reach an MSE of less than 0.05 on choosing the optimal action when given states from a holdout evaluation set. Regarding predicting the quality of a state-action pair the MSE was below 0.03 on the same data. The fitness elitist of the final generation contained 36 classifiers.

GCS, XCSFCA and XCSRCA all evaluate the function 100,000 times, which is considerably more than SupRB-1's 100 evaluations which took place to generate the training data. However, this direct comparison is slightly unfair as the three other systems could probably also have used a sample far smaller than 100,000 with a similar training procedure: They showed system errors below 0.05 after only 10,000 evaluated samples. Nevertheless, given the few examples required for training SupRB-1, a high sample efficiency is very likely. GCS stagnates at an error of 0.05 with about 1400 classifiers, while XCSFCA achieves an error below 0.01 after 30,000 samples with 740 classifiers whereas XCSRCA solves the problem perfectly after 18,000 samples using about 740 classifiers [14]. It should be noted that SupRB-1 achieves a much greater rule compactness, while definitely performing worse in terms of overall function approximation error.

It can be assumed that the higher function approximation error originates in the fact that SupRB-1 does not partition the search space in  $\mathcal{A}$  (see Section 3.2) and therefore has to fit the non-continuous function with parabolooids, which can not achieve a perfect approximation performance.

### 5.2 AM-Gauss

The AM-Gauss problem is a simplified model of an FDM-based additive manufacturing (AM) process's part quality and was created

hyperparameter	impact	default
validation set size	usually low	0.5
individuals' initial sizes	medium to high	30
GA population size	usually low	30
number of elitists	depends on GA population size	1
$k$ (fitness parameter)	high	too problem dependent
$F$ (one-fifth rule parameter)	low	1.05
crossover rate	medium	0.9 (more results pending)

**Table 1: Overview of hyperparameters of SupRB-1 and their default value.**

using expert knowledge. The process itself consists of material (usually thermoplastic polymers) being melted and then extruded to gradually construct a part whose quality depends on a number of factors such as the temperature to which the material is heated. For a given material (one dimension of  $\mathcal{X}$ ), the resulting part quality varies at increasing temperatures: Up until the melting point any resulting part's quality can be expected to be zero as no part can be produced at all. With a further increase in temperature, quality tends to increase as well at a rate depending on material properties up until some—unknown—point where the material becomes too soft to remain in shape which degrades part quality. At even higher temperatures, material might simply fail to successfully construct the part at all at which point quality can effectively be treated as zero again. This relationship of material, temperature and resulting quality can be simplified to a Gaussian function.

The FDM-based AM process we consider contains five continuous (obviously a simplification by itself) situation dimensions: Material, printer, room temperature, humidity and the kind of part to produce. These situations interact with six continuous parameters: Extrusion temperature, print bed temperature, cooling fan speed, extruder movement speed, material retraction speed and retraction distance (the first four parameters are rather self explanatory; the latter two come into effect whenever the extruder can not construct the part using continuous movement and has to move without extruding material). Assuming that every combination of situation dimensions and parameters can be modeled by a Gaussian function as motivated above leads to the following overall model for the quality function:

$$q(y) = q \left( \begin{pmatrix} x_1 \\ \vdots \\ x_5 \\ a_1 \\ \vdots \\ a_6 \end{pmatrix} \right) = \sum_{\substack{j \in \{1, \dots, 11\}, \\ k \in \{1, \dots, 11\}, \\ k \neq j}} \exp \left( - \left( \begin{pmatrix} y_j \\ y_k \end{pmatrix} - s \right)^T P_{j,k} \left( \begin{pmatrix} y_j \\ y_k \end{pmatrix} - s \right) \right) \quad (20)$$

Here, the  $P_{j,k}$ 's are randomly generated positive semi-definite matrices in  $\mathbb{R}^{2 \times 2}$  with eigenvalues in  $[0, 30]$  (ensures sensible scaling) and  $s$  is a randomly generated vector in  $[-1, 1]^2$  representing the shift of the Gaussian function. We did not include noise in our model, however, an evaluation on more realistic noisy environments is already planned.

We generated 30 such functions from consecutive random seeds and used these to create 30 sets of training data for SupRB-1. These

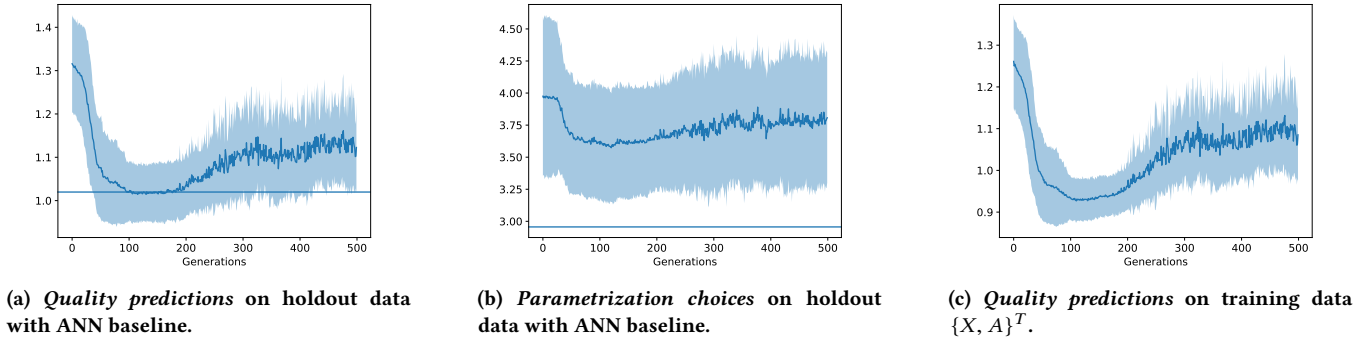
sets each contained 2000 examples for training (1000 training and 1000 validation examples) and 1000 examples we held out for evaluation. On each of those data sets SupRB-1 was run once for 500 generations using all standard parameters but initial individual sizes of 50 and  $k = 10^{-6}$  and then evaluated; the results are shown in Figures 1 and 2. Note that, having 30 different functions to test SupRB-1 on leads to a better estimate of its general performance at the cost of having a higher variance of results than when repeatedly testing on a single function.

We compare SupRB-1's results with those achieved by a two-layer fully connected *artificial neural network* (ANN) trained on identical data and functions. We performed simple automated architecture optimization on the ANN in terms of error during validation, determining optimal architecture for the given problems at 512 and 8 hidden cells respectively, while using ReLu activation functions twice; model complexity was not factored into the architecture optimization strategy. We show the results of this architecture on the holdout datasets as a baseline.

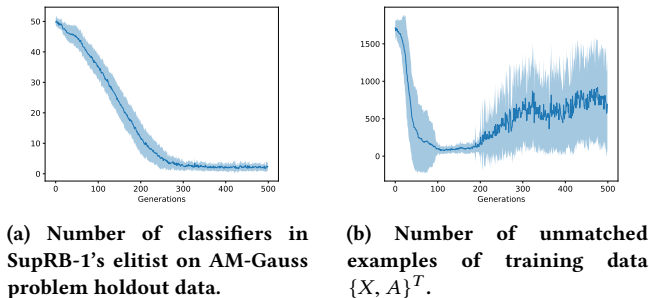
It can be seen in Figure 1a that SupRB-1's quality predictions' RMSE on holdout data improves rapidly over the first 50 generations and then seems to converge at around 1.02 in generation 100 which is on par with the ANN baseline. At around generation 200, however, the error starts to increase again and later fluctuates around a value of 1.1. The same behaviour can be observed on parametrization choices' RMSE on holdout data (Figure 1b) although the baseline is missed on that metric. The same can be seen, however, when looking at the quality prediction's RMSE on the training data (Figure 1c); this means that the problem can be detected and averted during training especially since the number of examples that are not matched by any classifier increases in a similar manner (Figure 2b).

When looking at the number of classifiers in SupRB-1's elitist (Figure 2a), a steady decrease up to a convergence at only 2-4 classifiers can be observed. By construction it is highly unlikely that the AM-Gauss problem can be solved satisfyingly by this few local models. We tried to alleviate that problem by during mutation adding a random classifier with a probability of 0.5 (this is also used for the shown runs)—but to no avail. It can be seen that, between generations 100 and 200, the number of classifiers lies between 13 and 35 which seems to be the sweet spot with the used hyperparameters.

The fact that after finding that sweet spot model complexity still decreases leads us to believe that there is an issue with SupRB-1's fitness measure. And indeed: It accepts individuals with slightly worse error in favour of a lower complexity (see (13)), which, when applied repeatedly can result in classifier population deterioration



**Figure 1: Root mean squared error (with standard deviation (SD)) of different metrics on SupRB-1’s elitist’s performance, averaged over 30 random AM-Gauss problems.**



**Figure 2: Simplistic complexity and knowledge gap measurements with SD of SupRB-1’s elitist.**

such as the one observed. This problem can easily be fixed by making (13) dependent on the error of the best individual ever seen.

Besides, due to the  $k$  hyperparameter being more delicate than expected (see Section 4.7) we assume that the value we determined for these runs was by far not optimal.

## 6 RELATED WORK

The SupRB learning system is inspired by previous research work in the field of learning classifier systems. LCS have been applied to a diverse field of problems resulting in a diverse field of applications, e. g. function approximation [33, 37], complex multiplexers [32], robot kinematics [20, 27]. LCS research can be further divided into Michigan- and Pittsburgh-style systems, with Michigan-style systems featuring a GA operating on the level of individual rules, where the entirety of rules represents a solution to the problem, while in Pittsburgh-style (also abbreviated as Pitt-style) systems the GA operates on sets of rules, where each set represents a complete solution to the problem.

The most famous and well researched family of LCS stem from Wilson’s XCS classifier system [34] following the Michigan-style. XCSR [35] expanded rule representations and therefore input options from ternary to continuous by using interval predicates; a representation used by many following systems such as XCSF [37], BioHEL [3] and of course SupRB-1. XCSF is an extension for XCS to

perform function approximation by replacing the constant payoff prediction with a local linear model, thus performing a piecewise-linear approximation of the overall function. The linear local models have subsequently been replaced by more complex models, such as higher order polynomials [17], interpolation [28] or neural networks [16].

Pittsburgh-style systems perform well when following a supervised learning setup and have usually been applied to classification/data mining problems. GALE [18] performed data mining for various classification tasks such as the detection of breast cancer, solving multiplexers and the classification of irises. NAX [19] has been applied to the diagnosis of prostate cancer without human input. GAssist [2, 9] was built for supervised learning of classification tasks and uses a standard GA to evolve rules basing on GABIL [7]. As typical for Pittsburgh-style LCS, individuals consist of a set of rules that represent a complete problem solution of variable length. The solution returned at the end of training is the highest fitness individual. The basic system uses discrete inputs and continuous inputs get discretised into dynamically generated micro-intervals. Not covered samples get predicted as a default class whose samples were not used for training.

A more recent example of Pittsburgh-style systems for classification on discretised data is EDARIC [25]. It is designed to deal with both over-fitting and class-imbalance by evolving populations for each class separately and using ensemble techniques for unknown samples. Generalisation is achieved by starting from maximally specific rules and gradually deleting constraints on less relevant input attributes. It was shown to perform well when compared to XCS, decision trees and GAssist for a number of classification datasets.

BioHEL [3, 9], a descendent of GAssist leaving the traditional Pittsburgh-style behind, is using an iterative rule learning approach to learn continuous and discrete attributes for bioinformatic datasets. It uses XCSR’s hyper-rectangle representation for continuous inputs and GABILs representation for discrete inputs. Fitness is based on GAssist’s fitness function with the addition of including coverage of rules. It uses the default classification mechanism of GAssist.

In reinforcement learning real world applications can often not be represented by discrete actions. Thus, the field of continuous

actions in XCS has found much research. While the problem described in section 1 is not understood to be in a reinforcement learning context and would only be of a single step nature, the optimal parametrization follows a similar design principle to multi-dimensional continuous actions. Wilson proposed three architectures [39]: IAL, a second XCSF interpolating the choices of the decision making XCSF, CAC, an actor-critic approach, and GCS, where a continuous action is aggregated with the input and a function of both is learned using XCSF. In GCS the optimal action is the action maximizing the learned function. The general approach of GCS is thus related to SupRB with the important distinction that GCS matches on both action and state.

Tran et al. introduce XCSFCA [30] as another way to deal with continuous actions by computing the action directly from the input. XCSFCA approximates a function  $(\mathcal{X}, (\mathcal{X} \rightarrow \mathcal{A})) \rightarrow \mathbb{R}$  which, due to currying, corresponds to  $\mathcal{X} \rightarrow ((\mathcal{X} \rightarrow \mathcal{A}) \rightarrow \mathbb{R})$ . Since  $\mathcal{A}$  is never a domain, XCSFCA only learns exactly one (the best regarding the quality measure) action  $\hat{a}_{\max}(x)$  for each  $x \in \mathcal{X}$ . That optimal  $\hat{a}_{\max}(x)$  is modelled using a linear model which is optimized separately using (1+1)-ES. SupRB instead learns  $(\mathcal{X}, \mathcal{A}) \rightarrow \mathbb{R}$  which can be written as  $\mathcal{X} \rightarrow (\mathcal{A} \rightarrow \mathbb{R})$ . Thus it is able to approximate the quality  $q(x, a)$  of every possible actions  $a \in \mathcal{A}$ , which is far more informative than only getting the best possible action, as the resulting action quality function  $\hat{q}$  could be analysed at will afterwards. The same argument can be made for all systems using computed actions. Besides the above, the structure of Tran et al.’s system is deliberately close to the one of XCS(F).

Howard et. al. [13] expanded on the idea of computed actions in XCSF by using a neural network to determine both matching and actions from the given inputs. Iqbal et al. [14] also dealt with continuous actions by computing them in XCSRCA, where the action is represented by a code fragment of a two branches deep binary tree that is evolved when creating new classifiers, similar to genetic programming. Naqvi and Browne [22] incorporated this approach to solve symbolic regression problems.

Hashemnia et al. [10] incorporate continuous actions into XCSR to balance an unmanned bicycle in simulation. However, to choose an action to execute they discretise the actions, determine a discrete set by a fitness-weighted roulette wheel selection mechanism and choose the continuous action of the fittest classifier within the determined set.

## 7 FUTURE WORK

Given that SupRB-1 was deliberately designed to implement SupRB while being as simplistic as possible there are numerous additions that can and will be made. Some of them are already in the works, such as an expansion to hyper-ellipsoid conditions [5] and testing of different polynomial and non-polynomial local models such as sine, exponential and radial basis functions (e. g. similar to the Gaussians already used as a testbed for SupRB-1). An expansion to neural networks seems plausible as well [16], although, in order to keep the degree of explainability high, they have to be kept as simplistic as possible.

Further, an investigation of a heterogeneous model landscape seems desirable, as some parts of a function might be harder to approximate even with very specific classifier conditions while others

can be described linearly with ease. SupRB keeps model complexity low while keeping performance high (see Section 3.1), we can thus expect that simpler models will be chosen where appropriate, for example, by including model type into the mutation operator.

In SupRB-1, the model structure is evolved by a GA, however, other algorithms such as CRO [15] are capable of improving an underlying model structure. Although this would arguably make SupRB no longer a strict representative of Pittsburgh-style LCS, as this name is strongly linked to GAs, an investigation seems appropriate. Both improvement techniques for GAs (e. g. n-point-crossover, different crossover and mutation rates) and different model structure optimizers should thus be investigated. As explainability is a key feature of SupRB, we will compare it with other machine learning techniques commonly seen as explainable such as decision trees.

In SupRB-1 the search space is not partitioned in  $\mathcal{A}$  to increase explainability, as understanding a singular function is much easier than understanding a combination—possibly including mixing models—of multiple heterogeneous functions. Note, that understanding a single function even of low order polynomials is non-trivial. A hierarchical approach where multiple classifiers will be located within a classifier matching a situation will be investigated in terms of performance and critically evaluated with regard to explainability.

Finally, SupRB-1 should be applied to real industrial datasets.

## 8 CONCLUSIONS

We introduced the SupRB learning system, a general accuracy-based Pittsburgh-style LCS architecture for supervised learning on continuous multi-dimensional decision problems. We laid the ground work for further investigation of this system by clearly defining parametrization optimization, the task SupRB is primarily meant to perform, describing the overall architecture and providing a first, deliberately simplistic, implementation (SupRB-1) of it. Said implementation was evaluated on a problem from the continuous-action LCS literature as well as on an abstract, simplified model of an industrial FDM manufacturing process. SupRB-1 has some shortcomings but these can be attributed to its simplicity. The overall approach shows a lot of prospect.

## ACKNOWLEDGMENTS

This work was in part supported by the German Federal Ministry for Economic Affairs and Energy (BMWi).

## REFERENCES

- [1] Anne Auger. 2009. Benchmarking the (1+1) Evolution Strategy with One-fifth Success Rule on the BBOB-2009 Function Testbed. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers (GECCO '09)*. Association for Computing Machinery, New York, NY, USA, 2447–2452. <https://doi.org/10.1145/1570256.1570342>
- [2] Jaume Bacardit. 2004. *Pittsburgh Genetic-based Machine Learning in the Data Mining Era: Representations, Generalization, and Run-time*. Ph.D. Dissertation. Universitat Ramon Llull.
- [3] Jaume Bacardit, Edmund K. Burke, and Natalio Krasnogor. 2009. Improving the Scalability of Rule-based Evolutionary Learning. *Memetic Computing* 1, 1 (01 Mar 2009), 55–67. <https://doi.org/10.1007/s12293-008-0005-4>
- [4] Richard P. Brent. 1973. *Algorithms for Minimization Without Derivatives*. Prentice-Hall.
- [5] Martin V. Butz. 2005. Kernel-based, Ellipsoidal Conditions in the Real-valued XCS Classifier System. In *Proceedings of the 7th Annual Conference on Genetic and*



- Evolutionary Computation (GECCO '05)*. Association for Computing Machinery, New York, NY, USA, 1835–1842. <https://doi.org/10.1145/1068009.1068320>
- [6] Martin V. Butz and Stewart W. Wilson. 2002. An Algorithmic Description of XCS. *Soft Computing* 6, 3 (Jun 2002), 144–153. <https://doi.org/10.1007/s005000100111>
- [7] Kenneth A. DeJong and William M. Spears. 1991. Learning Concept Classification Rules Using Genetic Algorithms. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, vol. 2. Morgan Kaufmann Publishers Inc., 651–656.
- [8] Jan Drugowitsch. 2007. *Learning Classifier Systems from First Principles: A Probabilistic Reformulation of Learning Classifier Systems from the Perspective of Machine Learning*. Ph.D. Dissertation. University of Bath (United Kingdom).
- [9] Maria A. Franco, Natalio Krasnogor, and Jaume Bacardit. 2013. GAssist vs. BioHEL: Critical Assessment of Two Paradigms of Genetics-based Machine Learning. *Soft Computing* 17, 6 (01 Jun 2013), 953–981. <https://doi.org/10.1007/s00500-013-1016-8>
- [10] Saeed Hashemnia, Masoud Shariat Panahi, and Mohammad Mahjoob. 2018. Continuous-action XCSR with Dynamic Reward Assignment Dedicated to Control of Black-Box Mechanical Systems. *Asian Journal of Control* 20, 1 (2018), 356–369. <https://doi.org/10.1002/asjc.1659>
- [11] John H. Holland. 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, USA. second edition, 1992.
- [12] John H. Holland. 1976. Adaptation. In *Progress in Theoretical Biology*. Vol. 4. Academic Press, New York, 263–293.
- [13] Gerard D. Howard, Larry Bull, and Pier-Luca Lanzi. 2009. Towards Continuous Actions in Continuous Space and Time Using Self-Adaptive Constructivism in Neural XCSF. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO '09)*. Association for Computing Machinery, New York, NY, USA, 1219–1226. <https://doi.org/10.1145/1569901.1570065>
- [14] Muhammad Iqbal, Will N. Browne, and Mengjie Zhang. 2012. XCSR with Computed Continuous Action. In *AI 2012: Advances in Artificial Intelligence*, Michael Thielscher and Dongmo Zhang (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 350–361.
- [15] Albert Y. S. Lam and Victor O. K. Li. 2010. Chemical-reaction-inspired Metaheuristic for Optimization. *IEEE Transactions on Evolutionary Computation* 14, 3 (June 2010), 381–399. <https://doi.org/10.1109/TEVC.2009.2033580>
- [16] Pier-Luca Lanzi and Daniele Loiacono. 2006. XCSF with Neural Prediction. In *2006 IEEE International Conference on Evolutionary Computation*. 2270–2276. <https://doi.org/10.1109/CEC.2006.1688588>
- [17] Pier-Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. 2005. Extending XCSF beyond Linear Approximation. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation (GECCO '05)*. Association for Computing Machinery, New York, NY, USA, 1827–1834. <https://doi.org/10.1145/1068009.1068319>
- [18] Xavier Llorà and Josep M. Garrell. 2001. Knowledge-Independent Data Mining with Fine-Grained Parallel Evolutionary Algorithms. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation (GECCO '01)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 461–468.
- [19] Xavier Llorà, Rohith Reddy, Brian Matesic, and Rohit Bhargava. 2007. Towards Better than Human Capability in Diagnosing Prostate Cancer Using Infrared Spectroscopic Imaging. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO '07)*. Association for Computing Machinery, New York, NY, USA, 2098–2105. <https://doi.org/10.1145/1276958.1277366>
- [20] Didier Marin, Jérémie Decock, Lionel Rigoux, and Olivier Sigaud. 2011. Learning Cost-Efficient Control Policies with XCSF: Generalization Capabilities and Further Improvement. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO '11)*. Association for Computing Machinery, New York, NY, USA, 1235–1242. <https://doi.org/10.1145/2001576.2001743>
- [21] Brad L. Miller, David E. Goldberg, et al. 1995. Genetic Algorithms, Tournament Selection, and the Effects of Noise. *Complex systems* 9, 3 (1995), 193–212.
- [22] Syed S. Naqvi and Will N. Browne. 2016. Adapting Learning Classifier Systems to Symbolic Regression. In *2016 IEEE Congress on Evolutionary Computation (CEC)*. 2209–2216. <https://doi.org/10.1109/CEC.2016.7744061>
- [23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
- [24] Stuart Russell and Peter Norvig. 2009. *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall Press, USA.
- [25] Shubhra K. K. Santu, Mustafizur Rahman, Monirul Islam, and Kazuyuki Murase. 2014. Towards Better Generalization in Pittsburgh Learning Classifier Systems. In *2014 IEEE Congress on Evolutionary Computation (CEC)*. 1666–1673. <https://doi.org/10.1109/CEC.2014.6900388>
- [26] Stephen F. Smith. 1980. *A Learning System Based on Genetic Adaptive Algorithms*. Ph.D. Dissertation. USA. AAI8112638.
- [27] Patrick O. Stalph and Martin V. Butz. 2012. Learning Local Linear Jacobians for Flexible and Adaptive Robot Arm Control. *Genetic Programming and Evolvable Machines* 13, 2 (01 Jun 2012), 137–157. <https://doi.org/10.1007/s10710-011-9147-0>
- [28] Anthony Stein, Simon Menssen, and Jörg Hähner. 2018. What about Interpolation? A Radial Basis Function Approach to Classifier Prediction Modeling in XCSF. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '18)*. Association for Computing Machinery, New York, NY, USA, 537–544. <https://doi.org/10.1145/3205455.3205599>
- [29] Christopher Stone and Larry Bull. 2003. For Real! XCS with Continuous-Valued Inputs. *Evolutionary Computation* 11, 3 (Sep 2003), 299–336.
- [30] Trung Tran, Cédric Sanza, Yves Duthen, and Thuc Nguyen. 2007. XCSF with Computed Continuous Action. *Proceedings of GECCO 2007: Genetic and Evolutionary Computation Conference*, 1861–1869. <https://doi.org/10.1145/1276958.1277327>
- [31] Ryan J. Urbanowicz and Jason H. Moore. 2009. Learning Classifier Systems: A Complete Introduction, Review, and Roadmap. *J. Artif. Evol. App.* 2009, Article Article 1 (Jan. 2009), 25 pages.
- [32] Ryan J. Urbanowicz and Jason H. Moore. 2015. ExSTraCS 2.0: Description and Evaluation of a Scalable Learning Classifier System. *Evolutionary Intelligence* 8, 2 (01 Sep 2015), 89–116. <https://doi.org/10.1007/s12065-015-0128-8>
- [33] Ryan J. Urbanowicz, Niranjana Ramanand, and Jason Moore. 2015. Continuous Endpoint Data Mining with ExSTraCS: A Supervised Learning Classifier System. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO Companion '15)*. Association for Computing Machinery, New York, NY, USA, 1029–1036. <https://doi.org/10.1145/2739482.2768453>
- [34] Stewart W. Wilson. 1995. Classifier Fitness Based on Accuracy. *Evolutionary Computation* 3, 2 (1995), 149–175.
- [35] Stewart W. Wilson. 2000. Get Real! XCS with Continuous-Valued Inputs. In *Learning Classifier Systems*, Pier-Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 209–219.
- [36] Stewart W. Wilson. 2001. Mining Oblique Data with XCS. In *Advances in Learning Classifier Systems*, Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 158–174.
- [37] Stewart W. Wilson. 2002. Classifiers That Approximate Functions. *Natural Computing* 1, 2 (01 Jun 2002), 211–234. <https://doi.org/10.1023/A:1016535925043>
- [38] Stewart W. Wilson. 2004. Classifier Systems for Continuous Payoff Environments. In *Genetic and Evolutionary Computation – GECCO 2004*, Kalyanmoy Deb (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 824–835.
- [39] Stewart W. Wilson. 2007. Three Architectures for Continuous Action. In *Proceedings of the 2003-2005 International Conference on Learning Classifier Systems (IWCLS '03-05)*. Springer-Verlag, Berlin, Heidelberg, 239–257.