OPEN ACCESS
OPUS AUGSBURG

Universität Augsburg
Universitätsbibliothek

SupRB in the context of rule-based machine learning methods: a comparative study

Michael Heider, Helena Stegherr, Roman Sraj, David Pätzel, Jonathan Wurth, Jörg Hähner

Angaben zur Veröffentlichung / Publication details:

Heider, Michael, Helena Stegherr, Roman Sraj, David Pätzel, Jonathan Wurth, and Jörg Hähner. 2023. "SupRB in the context of rule-based machine learning methods: a comparative study." *Applied Soft Computing* 147 (August): 110706. https://doi.org/10.1016/j.asoc.2023.110706.

Nutzungsbedingungen / Terms of use: **CC BY-NC-ND 4.0**

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under these conditions:
CC-BY-NC-ND 4.0: Creative Commons: Namensnennung - Nicht kommerziell - Keine Bearbeitung
Weitere Informationen finden Sie unter: / For more information see:
https://creativecommons.org/licenses/by-nc-nd/4.0/deed.de

SupRB in the Context of Rule-based Machine Learning Methods: A Comparative Study

Michael Heider*, Helena Stegherr, Roman Sraj, David Pätzel, Jonathan Wurth, Jörg Hähner

[a] *Universität Augsburg, Am Technologiezentrum 8, Augsburg, Germany*

Abstract

Comprehending why an artificial intelligence–based agent makes certain predictions is often discussed as one of the central issues that needs to be solved in the near future to make such agents ubiquitously deployable. However, many common black-box machine learning models are hard to analyse. Recently, we proposed a new machine learning algorithm to construct concise sets of rules. This algorithm, SupRB, creates compact, interpretable and transparent models. One key feature that differentiates it from many similar previously proposed systems is that it optimizes the model selection tasks concerning rule discovery and rule set composition separately. Therefore, users can tailor a model's structure more easily towards fulfilling use-case specific explainability requirements. Keeping the model selection tasks separate during optimization allows for more clearly defined goals when compared to state-of-the-art systems and—importantly—facilitates independent rule fitnesses, making the training process easier to explain and the models easier to analyse post-hoc. In this paper, we extend previous benchmarking of SupRB, where it was compared with XCSF, a prominent rule set learning system, to also include Decision Trees (DTs) and Random Forests (RFs) in our analysis. We find that—as expected—SupRB performs comparable to XCSF, while allowing easier control of model structure and showing a substantially smaller sensitivity to random seeds and data splits, and it performs slightly worse than RFs, and better than DTs. Importantly, the overall model size was substantially smaller with SupRB, making it the easiest to interpret system while not falling significantly short in terms of prediction errors.

Keywords: rule-based learning, learning classifier systems, evolutionary machine learning, interpretable models, explainable AI

*Corresponding author

Email addresses: michael.heider@uni-a.de (Michael Heider), helena.stegherr@uni-a.de (Helena Stegherr), roman.sraj@uni-a.de (Roman Sraj), david.paetzel@uni-a.de (David Pätzel), jonathan.wurth@uni-a.de (Jonathan Wurth), joerg.haehner@uni-a.de (Jörg Hähner)

1. Introduction

Decision making agents that utilize machine learning models need to exhibit not only a high degree of accuracy of the models, but also provide sufficiently detailed explanations of the decisions made by them. Limitations in one of these capabilities lead to reduced applicability due to scepticism among the human stakeholders. An example for this are industrial settings, where seasoned machine operators will not accept the recommendations of the model if these go against their experiences and knowledge while not providing sufficient evidence of their validity. Furthermore, models in real-world scenarios are hardly ever perfect in every detail, as learning tasks are often complex and training data may be scarce. This exacerbates the problem of acceptance by the human stakeholders.

This leads to a necessity of increasing the stakeholders' trust in the predictions made by digital agents, as only then their benefits can be exploited comprehensively. Therefore, explanations are needed, not only for the processes involved in producing the models, but also of the entire model itself. In some cases, explainable models are even preferred over ones with better performance but higher complexity. This makes rule-based learners, for example Learning Classifier Systems (LCSs), a good fit for application in this kind of setting, as they facilitate thorough explanations. [1]

LCSs [2] are rule-based learners that utilize a finite set of if-then rules, from which their models are composed. They are considered as inherently transparent and interpretable. In each rule, a simpler and more comprehensive submodel, which is fitted to particular areas of the feature space, is included. Rules apply under certain conditions, which are optimized during training, usually utilizing an evolutionary algorithm for this process. In general, two main styles of LCSs can be distinguished, depending on their approach on rule set generation: while *Pittsburgh-style* systems evolve a population of sets of rules, which have a combined fitness per set, *Michigan-style* systems modify only a single set of rules, with each rule having an individual fitness. As these traditional styles have often been hybridised in the past, we recently proposed a new classification system that comprises four non-overlapping classes [3]. This leads to differences in the evolutionary algorithm's take on the optimization (the parameter space that the evolutionary algorithm operates on as well as its objective function), but not in its general goal of finding an "accurate and maximally general" [4] set of rules. The optimization does usually not include explainability-related requirements as a target when the more common Michigan-style systems are applied, although it can partially be present in the concept of generality. Pittsburgh-style systems, on the other hand, utilize the evolutionary algorithm to target error and rule set size simultaneously; however, they need to optimize both the positioning as well as the selection of rules. To this end, alterations of several rules in the set are performed in each iteration, which can cause beneficial changes to a rule to go unnoticed as they are not reflected in a corresponding change in the fitness of the overall set, e.g. because the effects on the measured fitness caused by different rules may cancel out. These rules might then be discarded, i.e. not included in the next generation, and the suboptimal positioning of rules might not influence the performance of the system, which is a problem when the rule conditions or the training process are required in explanations. In contrast, Michigan-style systems often generate and keep a large set of rules, containing well-performing and suboptimal ones, which is much larger than actually required for the problem at hand. Therefore, additional procedures are applied after training, including *compaction* techniques, that reduce the rules in the population to the most relevant ones, therefore enhancing explainability [5, 6].

The main reason to employ LCS models rather than less transparent models like artificial neural networks is to exploit their inherent properties that aid the explainability of models. However, the design of LCS training algorithms is quite challenging. As already mentioned, they should produce small but nevertheless accurate models of a small number of rules to be easier read by humans (cf. also Section 2 and [7]). For this type of model, no analytical solutions are known, therefore, it can probably only be optimized heuristically, e.g. with evolutionary algorithms. However, there is no global optimum to be found for the creation of individual rules but only for a highly complex model consisting of a previously unknown number of rules (cf. our discussion of the role of the optimizer of an LCS within [3]) where individual rules should cover a diverse option of input space partitions. However, individual rules' decision boundaries also need to be well placed without the option of global optimization. Combined with the multiple objectives introduced by explainability requirements (e.g. following our study in [8]) algorithm design for an LCS becomes a major challenge. The goal of those algorithms is, however, quite clear and—with the increasing number of ML models being deployed as decision makers in everyday life—becoming more and more timely (cf. [9]): create ML models that function accurately at making predictions but can be understood (and investigated for potential biases) by a human supervisor.

A new LCS algorithm named SupRB in which the optimization of rule conditions and of the composition of rules to provide a problem solution is separated was initially described in [10]. With this separation, the rule fitnesses are independent from influences other than their conditions' direct alterations, which increases the *locality* and additionally improves the explainability of these quality parameters. Furthermore, an improvement in terms of explainability is achieved by directly controlling rule set sizes and by the optimizer pushing good rules towards being more specific or more general.

This paper extends the investigations of SupRB's capabilities, described in Section 3, by comparing it to other well-known rule-based learning approaches. These include a modern version of XCSF [11, 12], which is one of the most developed and advanced LCSs, as well as Decision Trees (DTs) and Random Forests (RFs). All systems are evaluated on a variety of different regression datasets (cf. Section 4). This article is an extension of [13] where we explored how SupRB and XCSF relate in terms of performance.

We find that, as expected, SupRB performs competitively to XCSF with respect to prediction error across datasets, while producing substantially more compact models directly. SupRB achieves better expected errors than DT while still constructing smaller models and worse errors than RF, albeit, with vastly smaller models than RF.

2. Related Work

A well-known representative of LCSs is the XCS Classifier System (XCS), which can be utilized to solve all the major learning tasks through its many derivatives and extensions [2]. To compare it to SupRB and other rule-based learning systems on a set of supervised regression tasks with real-valued domains, a specific derivative and configuration is required. XCSF provides the basis for this, being designed specifically for supervised function approximation by replacing the constant predicted payoff with a linear function [11]. Additionally, the binary matching function is replaced with interval-based ones to enable applicability on real-valued problem domains [14]. There are more elaborate variants for replacing matching functions and linear models, which aim at improving the performance, but, at the same time, reduce the overall model transparency [15, 16].

Rule-based machine learning (RBML) systems and, therefore, also LCSs, are considered transparent or intepretable by design due to their natural relation to human behaviour. This means they do not necessitate the use of complex post-hoc methods, e.g. visualisation or model transformation, to be explainable to humans. There are, however, factors that can reduce the capabilities of these models, even when they are inherently transparent. These include for LCS, among others, the applied encodings, the number of rules in the model and the complexity resulting from the complicated matching functions or submodels in the individual rules. [7]

While it is common to control these limitations in LCSs by design, it is also possible to utilize specialised post-hoc methods. Especially visualisation techniques for classifiers, but also other strategies, aim at improving the model's interpretability [17, 18, 19]. Nevertheless, controlling interpretability by design can be advantageous, as it does not require to devise designated post-hoc methods or adapt existing ones to the specific needs of the problem at hand, as well as the model itself, which requires not only time but also sufficient expertise. The design must therefore focus on understandable matching functions and predictive submodels, but still keep the predictive power acceptable. Other factors, for example problem-dependent complex variables/features, can reduce the interpretability and are additionally hard to influence, which is why a thoroughly thought through design is important.

The interpretability of LCS and of rule-based learning systems in general is also highly dependent on the resulting rule set sizes. For example, visual inspections are much easier for smaller sets and the extensiveness of subsequent analysis is reduced. For LCSs, the control of the rule set size is different for Pittsburgh-style and Michigan-style systems. In Pittsburgh-style LCSs, the optimization algorithm's fitness function often combines several objectives, most commonly the accuracy and the number of rules, therefore enabling rule set size control. In GAssist [20], for example, the accuracy and the minimum description length are combined into a single objective, while applying an additional penalty for rule sets with too few rules. For Michigan-style systems, large populations are generally beneficial during the training process and therefore they do not control the rule set size using the fitness function. However, during training, two rules can be merged if one fully encloses the other, a mechanism which is called subsumption. Additionally, compaction can be invoked as a post-hoc method to decrease the number of rules in the set [14, 6]. It removes redundant rules, i.e. ones whose deletion does not decrease prediction accuracy. Compaction methods are, however, commonly designed for classification tasks only.

In contrast to LCSs, Decision Trees (DTs)—the most well known RBML method—are composed of a set of non-overlapping rules, whereas in LCSs rules may overlap which enables a higher degree of generalization. Models based on multiple DTs, such as Random Forests (RFs) using bagging or XGBoost using boosting, are typically less interpretable than DTs and even LCSs. LCSs are build from overlapping rules, while RFs and XGBoost models are build from aggregations of trees which are in turn build from non-overlapping rules. For an extensive discussion on how the concepts of bagging, boosting, and mixing relate to and differ from LCSs see [3].

3. The Supervised Rule-based Learning System

The Supervised Rule-based Learning System (SupRB) is a new type of LCS, with alternating phases of rule discovery and solution composition [10]. In the first phase, rule conditions are optimized independently of other rules. Thereby, a pool of diverse rules with convenient localizations is discovered. The subsequent second phase aims at composing an accurate but small solution to the learning task. To this end, the system utilizes an optimization algorithm to select a subset of all available rules, i.e. all rules that are in the pool after rule discovery. Therefore, SupRB diverges from other LCSs, as the *model selection* objectives—identifying a number of well positioned rules and choosing a set of these for the resulting model—are separated (cf. Figure 1). Note that the quality of the positioning of rules depends on a tradeoff between the local prediction error and the matched volume. Altogether, this enables the prediction of any input with minimal error, but the smallest possible set of rules, maintaining high transparency and interpretability. It is, however, difficult to determine beforehand how many rules are required to compose a good solution. Therefore, the two phases are applied in turns until a termination criterion is reached, i.e. a specified number of iterations (cf. Figure 2 and Algorithm 1). The alternating phases provide a unique advantage: subsequent rule discovery stages can be guided towards less explored regions, which were up until then only covered by no or only imperfectly placed rules. To achieve this, SupRB incorporates in-sample performance information from the last solution composition phase. Note, however, that a rule, once it has been put into the pool, is fixed and will never be removed during the training, which stands in contrast to Pittsburgh-style LCSs.

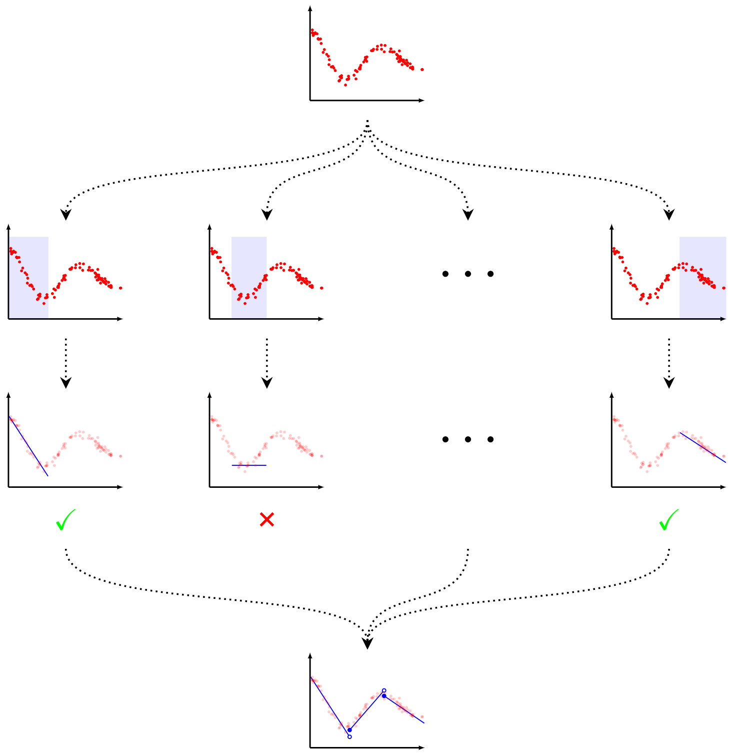


Figure 1: Extracting a rule-based model from data by discovering appropriate areas of the input space to place linear models in, creating individual rules. Afterwards, a subset of these rules is selected (denoted by cross and checkmark) to compose the output model from. In the example, we utilize three rules and use a mixing model where only the fittest rule per area is used for predictions.

SupRB mainly aims at providing insights into model predictions. Therefore, attaining simplicity and interpretability of its model is essential [10] which led to the following choices being made:

1. Rules' conditions use interval-based matching: A rule k applies for example x iff $x_i \in [l_{k,i}, u_{k,i}]\,\forall i$ with l being the lower and u the upper bounds.
2. Rules' submodels $f_k(x)$ are linear. They are fit using linear least squares with a l2-norm regularization (Ridge Regression) on the subsample matched by the respective rule.
3. When mixing multiple rules to make a prediction, a rule's experience (the number of examples matched during training and therefore included in fitting the submodel) and in-sample error are used in a weighted sum.

Rule discovery (RD), as displayed in Algorithm 2, can be accomplished by many different methods. While we use an evolution strategy (ES) in this paper, we also have examined different variations of novelty search [21]. A single rule discovery phase generates multiple rules to be added. Therefore, multiple runs

Algorithm 1 SupRB's main loop [13]

```
pool ← ∅
elitist ← ∅
for i ← 1, n_iter do
    pool ← pool ∪ DISCOVER RULES(elitist)
    elitist ← COMPOSE SOLUTION(pool, elitist)
end for
return elitist
```

Algorithm 2 SupRB's Rule Discovery [13]

```
procedure DISCOVER RULES(elitist)
    rules ← ∅
    for i ← 1, n_rules do                          ▷ (1, λ)-ES for each new rule
        candidate, proponent ← INIT RULE(elitist)
        repeat
            children ← ∅
            for k ← 1, λ do
                children ← children ∪ MUTATE(proponent)
            end for
            proponent ← child with highest fitness
            if candidate's fitness < proponent's fitness then
                candidate ← proponent
                j ← 0
            else
                j ← j + 1
            end if
        until j = δ
        rules ← rules ∪ candidate
    end for
    return rules
end procedure
```

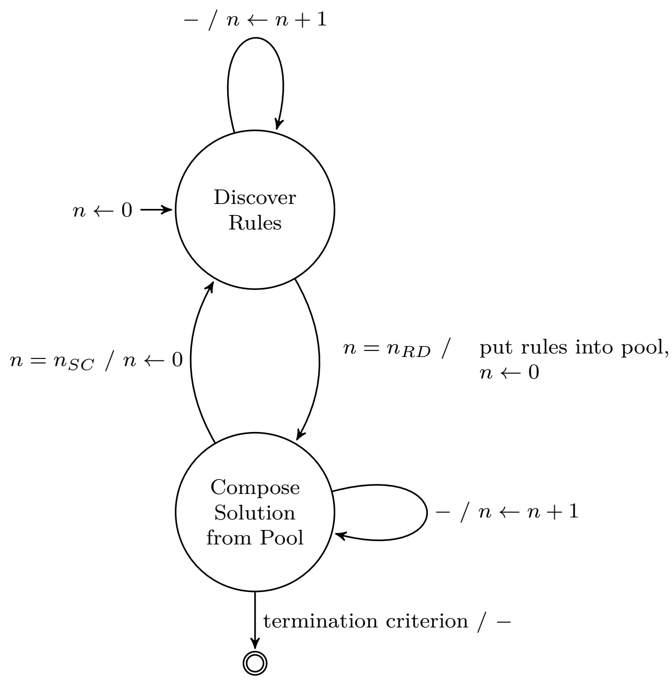


Figure 2: Rule discovery and solution composition alternation in SupRB [10]. n_{SC} is the number of iterations for the solution composing optimizer and n_{RD} is the number of iterations performed for rule discovery.

of the $(1, \lambda)$-ES are performed independently, enabling parallelization. For each of those, an initial candidate, i.e. parent rule, is placed around a training example which is chosen by applying roulette-wheel selection based on the in-sample error of the current *elitist* (or intermediate solution). Thus, higher probabilities of being selected are assigned to examples where the prediction exhibited a high error. The next step is the creation of λ children by repeatedly applying the non-adaptive mutation operator. It increases the distance of the parent's lower and upper bounds to the center by adding values sampled from a halfnormal distribution to each. The fittest child individual—based on its in-sample error and the matched feature space volume—is selected as the new parent. Additionally, if its fitness is higher than that of the current candidate, it replaces this individual. In rule discovery, the fitness is calculated as

$$F(o_1, o_2) = \frac{(1+\alpha^2) \cdot o_1 \cdot o_2}{\alpha^2 \cdot o_1 + o_2}\,, \tag{1}$$

with

$$o_1 = \mathrm{PACC} = \exp(-\mathrm{MSE} \cdot \beta)\,, \tag{2}$$

and

$$o_2 = V = \prod_i \frac{u_i - l_i}{\min_{x \in \mathcal{X}} x_i - \max_{x \in \mathcal{X}} x_i}\,. \tag{3}$$

This metric (cf. eq. (1)) was adapted from [22], where the authors utilized it to combine two objectives for a feature selection problem. The Pseudo-Accuracy (PACC), eq. (2), compresses the Mean Squared Error (MSE) of a rule's prediction into a (0, 1] range, while the volume share $V \in [0, 1]$ (cf. eq. (3)) of its bounds is used as a generality measure. The parameter β controls the slope of the PACC and α weighs the importance of the objective o_1 and o_2 against each other. Multiple values for β were tested and we found $\beta \in \{1, 2\}$ to be suitable defaults and selected a β of 1 for this article's runs. For α, 0.05 can be used in many problems but, ultimately, the value should always depend on the model size requirements, which are task dependent. The optimization process terminates if the candidate was not replaced within the last δ generations, adding this specific elitist to the pool. Overall, the process of discovering and adding new rules is repeated until a predefined number of rules is reached. Note that the optimizer does not aim at finding a single globally optimal rule—as would be the case in most optimization scenarios—but the goal is to find many optimally placed rules that allow a more accurate prediction for all inputs than a trivial model, i.e. simply returning the mean of all data, would produce. To this end, independent evolution is advantageous.

Algorithm 3 SupRB's Solution Composition [13]

```
procedure COMPOSE SOLUTION(pool, elitist)
    population ← elitist
    for i ← 1, pop_size do
        population ← population ∪ INIT SOLUTION()
    end for
    for i ← 1, generations do
        elitists ← SELECT ELITISTS(population)
        parents ← TOURNAMENT SELECTION(population)
        children ← CROSSOVER(parents)          ▷ 90% probability n-point
        population ← MUTATE(children)          ▷ probabilistic bitflip
        population ← population ∪ elitists
    end for
    return best solution from population
end procedure
```

The second phase, solution composition (SC), utilizes a genetic algorithm (GA) to select the subset of rules from the pool to form a new solution. Naturally, this phase can also utilize any optimizer, and in a previous experiment in [23] we compared several options and found that the GA is an appropriate choice. For indicating whether a rule from the pool is included in the solution, a bit string representation is used. Pairs of parents are selected with tournament selection before being recombined utilizing n-point crossover, where the crossover probability has a default setting of 90%. The resulting children are mutated by probabilistically flipping bits, controlled by the mutation rate. For the next generation, the children and some of the fittest parents (*elitism*) are kept. While the number of elitists depends on the population size, we found that, in our experiments, 5 or 6 elitists are most suitable for a population with 32 individuals. The fitness is also calculated based on the metric in eq. (1). In this case, the objectives are the solution's in-sample mean squared error and the number of rules selected (the *complexity*). Altogether, each individual corresponds to a subset of the pool. In contrast to other rule-based learning systems and attributable to the two phases of rule discovery and solution composition, rules in the pool remain unchanged by the optimizer of the SD phase and only rules within the pool can form a solution candidate.

In general, SupRB is intended—and therefore conceptualised and designed—as a regressor. This is reflected in the system's description, as well as the following experimental setup and evaluation. Nevertheless, adapting the system so that it is able to solve classification problems is quite straightforward: An appropriate classifier, e.g. a constant model, logistic regression, or any more complex model (depending on the explainability requirements), can replace the linear submodels. In addition, the fitness function needs to be based on accuracy instead of PACC and MSE.

4. Evaluation

To evaluate SupRB, we compare it to a recent implementation of XCSF[1] [11, 12] with hyperrectangular conditions and linear submodels (trained by recursive least squares updates [24]). This configuration very closely corresponds to the conditions and submodels used in SupRB. While we want to specifically acknowledge that some better-performing conditions, e.g. hyperellipsoids [25], have been proposed for XCSF, we consider them substantially less interpretable for average users, which worsens with increasing input space dimension. We expect XCSF to perform somewhat on-par with SupRB. Usually, errors should be lower while the number of rules in the model (after model compression/compaction) is higher. Additionally[2], we compare SupRB with Decision Trees (DTs) and Random Forests (RFs)[3]. As LCSs can be considered as one option to generalize DTs (by dropping the assumption of non-overlapping rules [cf. 3]) a comparison to them is very intuitive. We hypothesize that DTs show worse errors than the other ML approaches while producing somewhat easier-to-interpret models. Their interpretability is strongly supported by rules not overlapping. However, this usually also makes for less precisely approximated functions (assuming equal number of rules before mixing), increasing the error of the overall model. RFs combine a multitude of trees which, at first glance, might look similar to LCSs since these trees are mixed together in order to produce predictions, whereas, in LCSs, *rules* are mixed. As each individual

[1] https://github.com/rpreen/xcsf https://doi.org/10.5281/zenodo.5806708

[2] The comparison to XCSF has already been part of [13], although we performed a different statistical analysis there.

[3] We use them as implemented in the Python package *scikit-learn* [26] v1.0.1.

tree that is part of the RF consists of rules that fully cover the input space, predictions are performed by mixing multiple rules (exactly one rule per tree since there are no overlapping rules within each tree). However, due to how RF are trained the training steps that lead to these rules are much harder to trace. Furthermore, RFs typically use large numbers of trees, resulting in mixing many more rules than we expect to see mixed for each prediction in SupRB. We hypothesize that their errors will be lower than SupRB's due to higher model complexities being allowed and, probably, also lower than XCSF's. The interpretation of the models generated by RFs, on the other hand, can be expected to be more difficult due to the high number of interacting trees.

4.1. Experiment Design

SupRB is implemented[4] in Python 3.9, adhering to *scikit-learn* [26] conventions. We standardize the target and transform input features into the range $[-1, 1]$. These transformations improve SupRB's training process as they remove the need to tune error coefficients in fitness calculations, i.e. balancing complexity versus error for each dataset, and help prevent rules from being placed in regions where no samples exist (and therefore none would be matched). While other optimizers have been proposed for rule discovery (RD) [21] and solution composition (SC) [23] in SupRB, the results have largely been inconclusive, with ES and GA, respectively, probably being a good (or even the best choice) for the optimizers. Therefore, we follow the setup of [13] and use these optimizers as well as the remaining general configuration: We assume that for the datasets under investigation discovering a total of 128 rules is sufficient to compose good models from. Hence, 32 cycles of alternating rule discovery and solution composition are performed, generating 4 rules per cycle. For the ES we selected a λ (size of the offspring population) of 20. The GA performs 32 iterations with a population size of 32. In general, all of these parameters can be chosen quite freely as long as they are "large enough", with larger values being neither considerably beneficial nor detrimental to model performance but may impose additional runtime cost during training. To tune some of the more sensitive parameters, we perform a hyperparameter search using a Tree-structured Parzen Estimator in the Optuna framework [27] that optimizes average solution fitness on 4-fold cross validation. We tune the datasets independently for 1000 iterations or a maximum of 1080 core hours per tuning process. For XCSF we follow the same process, selecting typical default values[5] [12] and tuning the remaining parameters independently on the datasets. This same setup is also used for DT and RF.

The final evaluation, for which we report results in Section 4.2, uses 8-split Monte-Carlo cross-validation, each with 25 % of samples reserved as a validation set. Each learning algorithm is evaluated with 8 different random seeds for each 8-split cross-validation, resulting in a total of 64 runs per algorithm per dataset.

[4]https://github.com/heidmic/suprb
[5]https://github.com/rpreen/xcsf/wiki/Python-Library-Usage

Table 1: Overview of the experimental test data results of 64 runs per dataset rounded to two decimals. The columns give the mean error and its standard deviation over the runs per algorithm. The errors are calculated in a standardized target space, meaning the data was transformed to zero mean and unit variance beforehand. If the data is normally distributed and sample size is large enough, then a very trivial model that predicts the mean of the output training data for any input is able to achieve an MSE of 1 in standardized target space. Highlighted in bold are the models where the mean performance was stronger than the other models'.

	CCPP	ASN	CS	EEC
SupRB	0.07±0.00	0.16±0.02	0.14±0.03	0.03±0.01
XCSF	0.06±0.00	0.12±0.16	0.17±0.13	**0.02±0.02**
DT	0.06±0.00	0.27±0.03	0.17±0.03	0.08±0.02
RF	**0.04±0.00**	**0.06±0.01**	**0.09±0.02**	0.04±0.01

We evaluate the four approaches on the same datasets as in [13] which are taken from the UCI Machine Learning Repository [28]. The Combined Cycle Power Plant (CCPP) [29, 30] dataset shows an almost linear relation between features and targets and can be acceptably accurately predicted using a single rule. Airfoil Self-Noise (ASN) [31] and Concrete Strength (CS) [32] are both highly non-linear and will likely need more rules to predict the target sufficiently. The CS dataset has more input features than ASN but is easier to predict overall. Energy Efficiency Cooling (EEC) [33] is another rather linear dataset, but has a much higher input features to samples ratio compared to CCPP. It should similarly be possible to model it using only few rules.

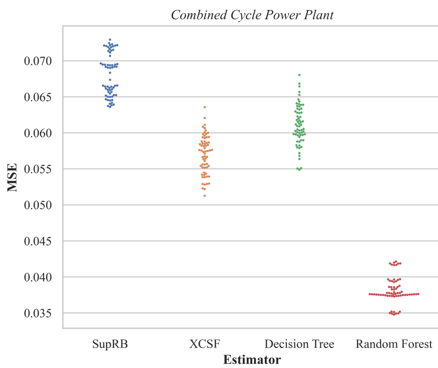
4.2. Results

In this section, we present and discuss the results of our experiments. We first focus on MSEs, then discuss solution complexities, and close with a rigorous statistical analysis.

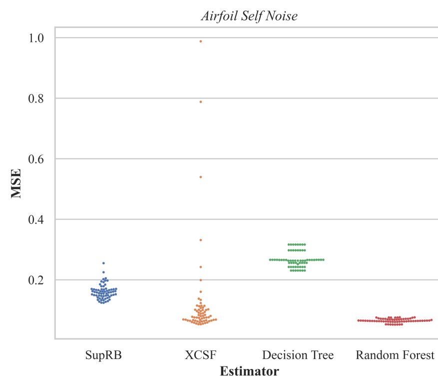
4.2.1. Mean Squared Errors

In our experiments, we find that XCSF performs slighly better than SupRB, which in turn performs slightly better than DT with regards to mean squared errors (MSEs). RF is competitive or better on all datasets and does show the lowest variance over the different runs. This follows our previous hypotheses. While RF is performing worse than XCSF and SupRB on EEC and optically vastly better than all on CCPP (cf. Figure 3a), we want to stress the very low difference in actual value of the runs on these two datasets, which is likely not practically relevant. Plots of all runs are found in Figure 3, while we report mean (across runs) MSEs and their standard deviations in Table 1[6]. The presented

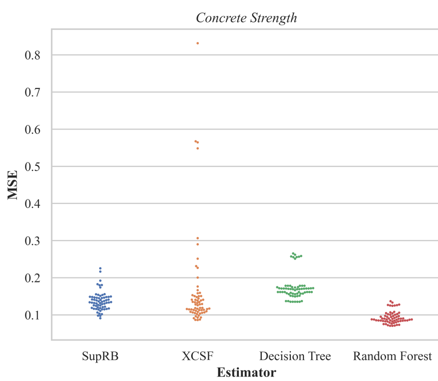
[6]Note that the presented values marginally differ from those in [13]. We did adjust some of the available hyperparameter ranges for our hyperparameter tuning. For SupRB, the differences were very small, only visible on ASN with the given two decimal precision. However,



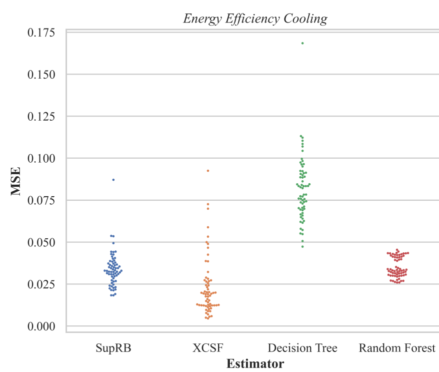
(a) Distribution of runs on CCPP



(b) Distribution of runs on ASN



(c) Distribution of runs on CS



(d) Distribution of runs on EEC

Figure 3: Distribution of runs' errors. All datasets are standardized with unit variance, therefore, a trivial model should at least achieve an error of 1.0. Note that the plots are on different scales, reflecting the varying difficulty of the learning tasks.

results are based on the standardized (zero mean and unit variance) versions of the targets of the respective datasets, facilitating an easier across-dataset comparison. However, comparison with results from other publications that used the same datasets might be more difficult since often times metrics are reported on non-standardized-outputs scales as they are made with different intentions. As we do not attempt to find the best solution to those learning tasks but want to contrast SupRB to three other RBML techniques, we deem this an appropriate compromise.

Analysing the distributions of errors more closely, we find that the algorithms do show different behaviours. Clearly, runs are not actually normally distributed for any algorithm. Presumably, the fact that small real-world datasets are used in this analysis has a influence on this behaviour, but the algorithms do also not converge towards the exact same model per split. No clear relations between random seeds or data splits and errors can be seen from the experiment data itself.

SupRB, XCSF, and DT all did show at least one outlier, although, for XCSF these were much further away and substantially more frequent. A cover-delete-cycle, a known issue in Michigan-style LCSs where experienced rules are removed from the population by error and new rules are only shortly part of the population, seems to have affected many of XCSF's runs. In Michigan-style LCSs, covering is a rule generating mechanism that is triggered when few rules match a given input and then creates a new rule that is guaranteed to match this input. The deletion mechanism of XCSF randomly, but fitness-proportionately, removes rules when the population has reached its hyperparameter-imposed maximum population size. During tuning, we allowed our tuner to choose both the number of training steps and the maximum population size (among the many other parameters of XCSF). Regardless, we find that post-training population sizes almost always reach the population size limit (not accounting for numerosity). The tuner seems to have found settings that counteract the cover-delete-cycle somewhat well. We find that XCSF's hyperparameter tuning chose much larger populations than what would be expected following the typical rule of thumb of using ten times as many rules as should be needed for a good problem solution (based on domain knowledge or prior modelling experience) [4]. Upon further investigation, we also found that rules contained in the final models were usually only discovered late during training, although the system error had remained stagnant (at least not showing practically significant changes) for quite some time beforehand. Subsumption in the EA—a typical method to combat cover-delete-cycles—was utilized. This important mechanism checks whether a newly discovered rule is fully contained within a parent rule, in which case, the new rule is not added to the population and the respective parent's numerosity parameter gets increased by one. A numerosity of n means that this rule is in the population n times and thus counts as n simple rules (with

---

XCSF did perform slightly better, falling less frequently into cover-delete cycles (a common problem with XCS-based systems, cf. [34]).

Table 2: Overview of the solution complexities (number of rules in the solution proposed by SupRB, the final macro-classifier count in an XCSF population after compaction and the number of leaf nodes in a DT, respectively) across 64 runs per dataset rounded to two decimals. We report mean and standard deviation. Note that we omit RF here, which used 194, 228, 91, and 122 trees of—on average—similar sizes to the individual DT models, respectively, and is therefore vastly out of scale in terms of number of rules.

	CCPP	ASN	CS	EEC
SupRB	2.97±0.67	32.67±2.8	28.47±2.68	11.84±1.85
XCSF	1922.71±390.7	1617.58±413.12	481.68±336.33	707.94±282.16
DT	556.44±18.38	67.92±2.59	110.62±6.37	26.50±0.71

a numerosity of 1; also known as micro-classifiers rather than macro-classifiers which have a numerosity > 1) towards the population size limit. Numerosity is also used in the fitness-proportionate delete mechanism. A rule selected for deletion gets its numerosity reduced by one and is only fully removed if that parameter hits 0. Theoretically, subsumption is thus decreasing the number of macro-classifiers in the population. Regardless, it seems that in our runs cover-delete-cycles could not be prevented as we found most rules contained in the final model have only been part of the population for relatively short amounts of time and mostly had numerosities of 1.

Overall, SupRB's training seems to converge more reliably then the other LCS, XCSF. Some variance based on random seeds is strongly expected, as this is a common occurrence in (evolutionary) stochastic optimization, cf. e.g. [35]. RF in particular suffers less from this phenomenon, however, overall variance, especially within one data split, is not substantially different from SupRB (and DT) when practical significance gets taken into account as well.

4.2.2. Model Complexity

A main concern of SupRB is to produce (better) explainable models while maintaining competitive results in terms of error. Figure 3 and Table 1 demonstrated that SupRB is able to learn complex datasets comparably well to established methods. Table 2 gives the model sizes[7] of SupRB, XCSF, and DT. We report their mean over 64 runs and their standard deviation. Model size (number of rules) is one of the key parameters that hinders interpretability and therefore explainability of RBML models [7]. Due to the frequent cover-delete-cycles, discussed extensively in the previous section, XCSF's trained models are usually very close to the population size limit set during tuning. We then perform compaction (cf. Section 2) to remove as many rules as possible, which resulted in about 10% of rules being removed. However, the final model size still far exceeds the one of SupRB. Admittedly, XCSF training does not ex-

[7]Note that the presented values differ from those in [13]. These differences are the result of adjusting some of the available hyperparameter ranges for our tuning. For SupRB, this results in larger models, while, for XCSF, it reduced model size compared with [13].

plicitly promote small macro-classifier counts; however, many mechanisms are present that should reduce it (in particular: set subsumption, EA subsumption, and, most importantly, compaction). DT did show a tendency towards surprisingly large and complex trees for CCP. Its models for ASN and EEC were only twice the size of SupRBs, which is easily explained by LCSs ability to have overlapping rules which DTs lack. However, the errors of DT on these two datasets were also notably higher than SupRB's. Rule overlaps could explain the roughly four times as large model for CS as well, where DT performed only slightly worse than SupRB in terms of error. For the training of DTs we used a slightly modified version of the well-known CART algorithm, as implemented in scikit-learn [36]. This algorithm controls tree size on the basis of some configurable hyperparameters like maximum depth and minimal sample count in a leaf node and does also feature a pruning mechanism that balances prediction error with tree complexity directly. For RF, we tuned the number of trees to be bagged, which resulted in 194, 228, 91, and 122 trees for CCPP, ASN, CS, and EEC, respectively. The individual trees were of roughly the same size as the ones used by DT, making RF's models substantially larger than those of the other three algorithms'. Models of that size are theoretically interpretable but practically it is not realistic to gain substantial insights in an appropriate time. SupRBs models can, however, be read rather easily and analysed even without very deep LCS knowledge. Our results on the complexity metric, fit with our hypotheses made at the beginning of Section 4.

4.2.3. Model Inspection

Table 3: Exemplary rule generated by SupRB on CS dataset. The target is the concrete compressive strength in MPa. The original space intervals denote the area matched by the rule in terms of the original variable scales, while the intervals in feature spaces are scaled into $[-1, 1]$ and help perceiving rule generality at a glance. Coefficients denote the weight vector used for the linear model. This rule was also presented in our previous study [13].

	Original Space	Feature Space σ	
input variable	interval	interval	coef
Cement [kg/m^3]	$[104.72, 516.78]$	$[-0.99, 0.89]$	2.38
Blast Furnace Slag [kg/m^3]	$[0, 359.40]$	$[-1.00, 1.00]$	2.29
Fly Ash [kg/m^3]	$[13.45, 200]$	$[-0.87, 1.00]$	0.68
Water [kg/m^3]	$[122.64, 244.80]$	$[-0.99, 0.96]$	-1.26
Superplasticizer [kg/m^3]	$[6.02, 24.80]$	$[-0.63, 0.54]$	-0.67
Coarse Aggregate [kg/m^3]	$[950.16, 1145]$	$[-0.13, 1.00]$	0.71
Fine Aggregate [kg/m^3]	$[756.14, 992.60]$	$[-0.19, 1.00]$	0.60
Age [days]	$[18.36, 365]$	$[-0.90, 1.00]$	2.07
		intercept$_\sigma$ = 3.9160	
In-sample MSE$_{\text{orig}}$ 1.5310	In-sample MSE$_\sigma$ 0.0917	Experience 84	

An example rule extracted from a model trained on the CS dataset is presented in Table 3. The data set consists of eight input features (and a target). For each dimension of the input, Table 3 gives the name (and SI unit), what portion of that dimension is matched, and how influential the value in a dimension is towards our target (denoted by the coefficient of the localized linear model). For the matching interval (we always match hyperrectangular subspaces), Table 3 contains the absolute (original/unscaled) and the normalized (min-max-scaled features) bounds. The normalized bounds are what SupRB operated on and make it easier to grasp what part of the populated (with samples) space is being matched, whereas the original space representation helps at understanding proportions and relates stronger to the real world. While we think that both representations are important for XAI, we can assume that algorithmic behaviour is better understood/analysed on the normalized space while the original space is more helpful for domain experts tasked to operate with the rule's predictions. Additionally, Table 3 gives the constant intercept of the linear submodel and the in-sample (encountered during training) error of the submodel, as well as, how many examples from the training data were matched and used for its training.

During training, the rule in Table 3 matched 84 examples (its experience is 84) and it did match another 31 examples from our test set. The model we extracted it from consisted of a total of 23 rules, with experiences of 7 to 240. Rules were either rather general or rather specific. The presented rule is more general than many others. Close inspection reveals that it matches most of the available input space for 5 out of CS's 8 dimensions. It is even maximally general for the "Blast Furnace Slag" input variable. In Table 3, we give the matching function's intervals both in terms of the original (feature) space of dataset itself and in the interval-scaled ($[-1, 1]$) features we used (cf. Section 4.1). The scaled intervals clearly make it easier to spot the generality of the rule, however, a domain expert may prefer to also see the absolute values and the unit they are measured in. For model inspection, we assume that most users will prefer the (transformed) feature space. For the "Superplasticizer" input variable, the rule matches slightly more than half of the possible values and is roughly positioned around the center of the range of values, while for "Coarse Aggregate" and "Fine Aggregate", the rule matches the 55 to 60 % of the input range, oriented towards higher values. The linear submodel's parameters are also based on the transformed feature space and the standardized version of the target (in this case, the concrete's compressive strength). While higher concentrations of "Cement", "Blash Furnace Slag", and "Age" of the mixture influence its compressive strength positively, high concentrations of "Superplasticizer" and "Water" have negative effects on this target. The other three variables have lower effects on the target but all seem to positively influence compressive strength of the concrete mixture in the matched areas. Note that this last distinction is very important and actually the great strength of a rule set learning approach. We are able to assign different parts of the inputs space different coefficients for the same features and keep that in an easily readable format. One of the examples matched by the rule is $x = (190.3, 0, 125.2, 161.9, 9.9, 1088.1, 802.6, 56)^T$ with $y = 38.56$. For this example, the rule proposes $\hat{y} = 38.39$. The prediction is clearly close to the ground truth. Providing this example to a user with domain experience can help them assess the model's predictive power. Moreover, the user can then extrapolate to other examples that would fit in the provided matching intervals, increasing their trust over other models, where such extrapolation is not possible from the models' structure. The rule performed slightly below average compared to the other rules in this solution (mean in-sample MSE of 0.0751) on its training data, although this should be viewed critically as more specific rules tend to be able to fit their (noisy and non-linear) input data more easily. We find that rule inspection can provide critical insights into the model's predictive process and is easy to do due to the rule's design and the overall small number of rules per model.

4.2.4. Statistical Analysis

In this section, we compare SupRB with the three established RBML algorithms using a Bayesian model comparison approach. For this, we apply Corani and Benavoli's Bayesian correlated t-test [37] to determine probability distributions over the differences in performance between the algorithms. Despite their remaining popularity, we deliberately avoid the use of null-hypothesis significance testing due to their many flaws and possible pitfalls—cf. e.g. [38]. The resulting posteriors (given in Figures 4, 5, and 6 for MSEs including 99 % high posterior density intervals (HPDIs)) are the *distribution of the difference between the considered metric for SupRB and the considered metric for the other algorithm* (practically, this equates to values above zero indicating SupRB having a higher (worse) metric value than the other algorithm).

Figure 4 compares SupRB's MSEs with those of XCSF. We see that in 99% of runs, we should expect SupRB to perform better on CS and worse on the other three datasets. For CCPP and EEC, the actual difference in value is very small at about 1% of a standard deviation of the respective target. This difference is also reached fairly consistently. For ASN and CS, the distributions are substantially wider although the expected differences are also clearly below 5% of a standard deviation. The practical significance is likely quite low and we assume that in terms of error XCSF and SupRB should be considered to perform somewhat equally or at least that no clear decision can be made. In terms of model complexity, the statistical analysis is very clear that SupRB performs considerably better.

In Figure 5, we compare SupRB's MSEs with those of DT. DT might perform very slightly better on CCPP, although this difference would very clearly not be practically significant. On the other three datasets, SupRB consistently performs better with a very small distribution of differences. At least on ASN, this difference is practically significant at about 11% of a standard deviation of this dataset's target. As with XCSF, the model complexity analysis very clearly shows SupRB to return models with substantially less rules.

The distributions in Figure 6 show the expected distribution of differences in MSE between SupRB and RF. RF performs better than SupRB on CCPP, ASN, and CS and does so quite consistently. For EEC, the test expects SupRB to

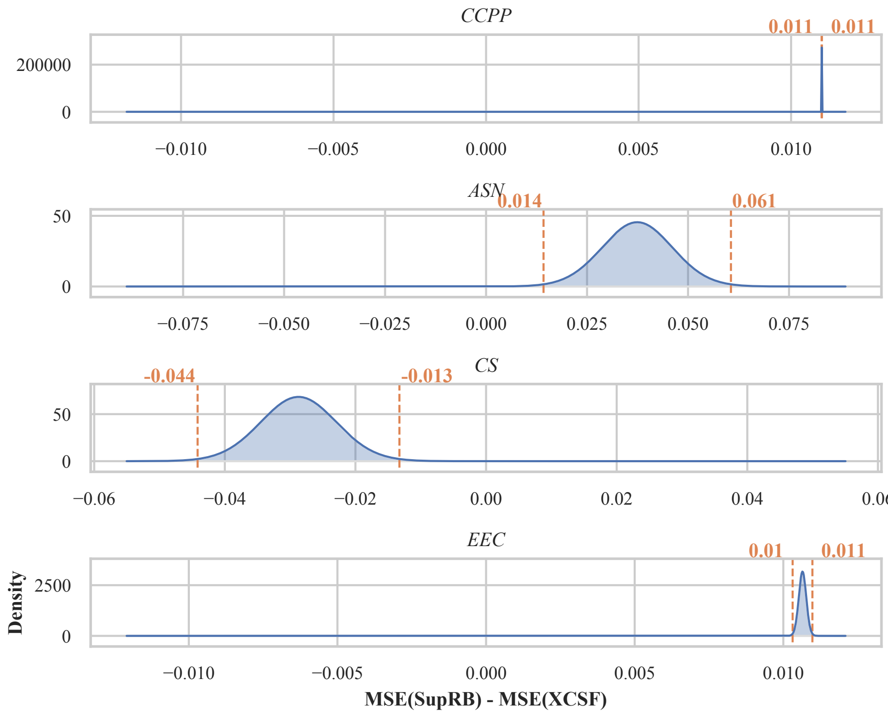


Figure 4: Density plot of the posterior distribution obtained from Corani and Benavoli's Bayesian correlated t-test [37] applied to the difference in MSE between SupRB and XCSF. Orange dashed lines and numbers indicate the 99 % HPDI (i.e. 99 % of probability mass lies within these bounds). HPDI bounds rounded to two significant figures.

perform very slightly better, however, this is clearly not a practically significant difference.

Overall, we can conclude from our tests that, on MSEs, SupRB is probably outperformed by RF or performs equally to it, outperforms DT, and is very slightly outperformed by XCSF in most cases. This fits well with our hypotheses outlined at the beginning of Section 4. In terms of model complexities, SupRB does manage to produce substantially smaller and more interpretable models without loosing too much predictive power.

5. Conclusion

This article extended the presentation of the Supervised Rule-based Learning System (SupRB) and added further comparative benchmarks over a previous less extensive version [13]. We demonstrated how SupRB, a novel rule-based machine learning (RBML) algorithm that uses two separate optimizers to place and select rules, ranks in terms of compact rule sets and prediction error when compared to three well-established RBML algorithms.

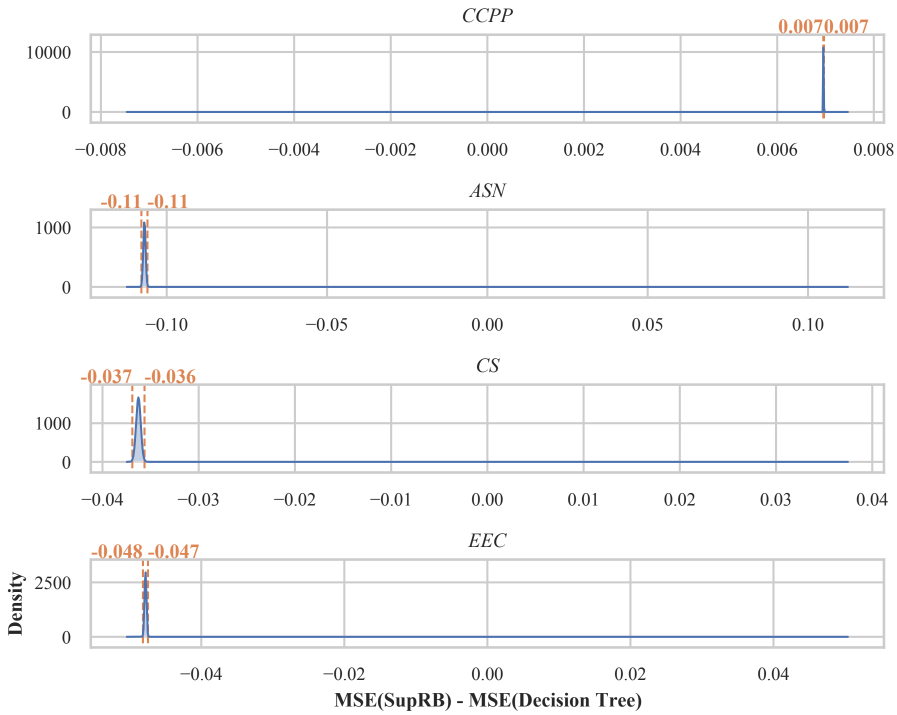


Figure 5: Density plot of the posterior distribution like Figure 4 but between SupRB and DT.

In this comparison, we benchmarked SupRB, Decision Trees (DTs), Random Forests (RFs), and the XCSF classifier systems (XCSF) on four real-world regression datasets with different learning task complexities and dimensionalities. As both SupRB and XCSF should be considered Learning Classifier Systems, XCSF produces the most similar types of machine learning models. For both algorithms, we limited rules to use hyperrectangular conditions and linear submodels, as the main advantage of these types of models (and motivation to use them) over other models is their inherent interpretability and transparency. We tuned the more sensitive hyperparameters of each algorithm for every dataset independently and then performed a total of 64 (8 random seeds and 8-fold Monte-Carlo cross-validation with 25% test data) runs per algorithm per dataset.

We expected SupRB and XCSF to perform similarly in terms of errors, with smaller models for SupRB. Furthermore, we assumed DT to perform worse than SupRB (maybe better regarding interpretability) and RF to perform better than SupRB in terms of error and substantially worse on model complexity. We found those hypotheses to largely hold. While there is some variation across datasets, general tendencies are quite clear. We did perform an extensive statistical analysis using Bayesian correlated t-tests to critically question these

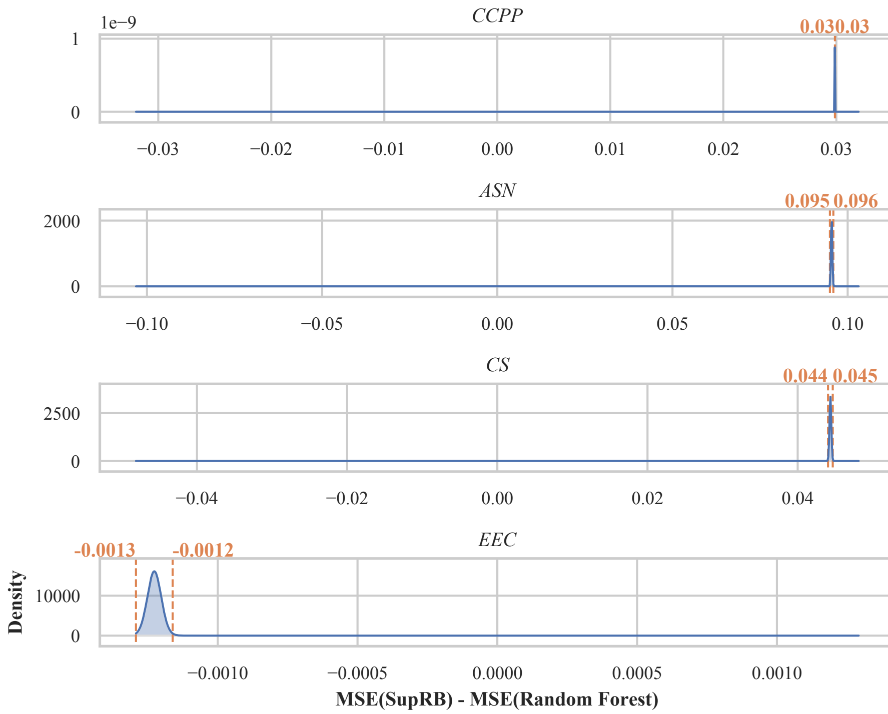


Figure 6: Density plot of the posterior distribution like Figure 4 but between SupRB and RF.

findings. The results of these tests did support the hypotheses and are in-part (where they were most interesting) presented in this article. Overall, we conclude that SupRB is a promising new RBML algorithm for creating predictive models with compact sets of human-readable rules.

It is reasonable to assume that SupRB could construct less erroneous models if the pressure to evolve small rule sets was lower. However, as explainability is the main reason to use RBML models, rather than e.g. neural networks, we think that our current models strike a well-acceptable balance. XCSF's models have been severely more complex while only being slightly better with respect to prediction error. Likely, XCSF, RF, and SupRB find themselves at different points on the Pareto front between error and complexity, whereas DT is probably Pareto-dominated. Therefore, a reasonable step for future work would be to investigate whether other multi-objective (separate objectives rather than a combined objective) optimization in SupRB (potentially reaching further than mere complexity based on use case–specific requirements) might be worthwhile. Other than this, there are of course many options to modify the basic system to potentially improve its performance, runtime, or model size. Some of these, e.g. [23, 21], have already been presented elsewhere. For rule discovery, MAP-Elites [39] and other quality-diversity techniques might also be worth investigating as they have been successfully used for DTs [40]. An important step is to actually deploy such models in use cases where interactions with human domain experts occur regularly to test whether they actually fulfil the explainable AI role for which they have been designed, e.g. following the outline of [8].

Acknowledgement

Partial financial support for this work was received from the *Bavarian Ministry of Economic Affairs, Regional Development and Energy* and the *Deutsche Forschungsgemeinschaft* (DFG).

References

[1] M. Heider, R. Nordsieck, J. Hähner, Learning Classifier Systems for Self-Explaining Socio-Technical-Systems, in: A. Stein, S. Tomforde, J. Botev, P. Lewis (Eds.), Proceedings of LIFELIKE 2021 co-located with 2021 Conference on Artificial Life (ALIFE 2021), 2021. URL: http://ceur-ws.org/Vol-3007/.

[2] R. J. Urbanowicz, J. H. Moore, Learning Classifier Systems: A Complete Introduction, Review, and Roadmap, Journal of Artificial Evolution and Applications (2009).

[3] M. Heider, D. Pätzel, H. Stegherr, J. Hähner, A Metaheuristic Perspective on Learning Classifier Systems, Springer Nature Singapore, Singapore, 2023, pp. 73–98. doi:10.1007/978-981-19-3888-7_3.

[4] R. J. Urbanowicz, W. N. Browne, Introduction to Learning Classifier Systems, 1st ed., Springer Publishing Company, Incorporated, 2017.

[5] J. Tan, J. Moore, R. Urbanowicz, Rapid Rule Compaction Strategies for Global Knowledge Discovery in a Supervised Learning Classifier System, in: ECAL 2013: The Twelfth European Conference on Artificial Life, MIT Press, 2013, pp. 110–117. doi:10.7551/978-0-262-31709-2-ch017.

[6] Y. Liu, W. N. Browne, B. Xue, A Comparison of Learning Classifier Systems' Rule Compaction Algorithms for Knowledge Visualization, ACM Transactions on Evolutionary Learning and Optimization 1 (2021) 10:1–10:38. doi:10/gn8gjt.

[7] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, F. Herrera, Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI, Information Fusion 58 (2020) 82–115. doi:10.1016/j.inffus.2019.12.012.

[8] M. Heider, H. Stegherr, R. Nordsieck, J. Hähner, Assessing model requirements for explainable AI: A template and exemplary case study, Journal of Artificial Life (2023). In press.

[9] J. Bacardit, A. E. I. Brownlee, S. Cagnoni, G. Iacca, J. McCall, D. Walker, The intersection of evolutionary computation and explainable ai, GECCO '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 1757–1762. doi:10.1145/3520304.3533974.

[10] M. Heider, H. Stegherr, J. Wurth, R. Sraj, J. Hähner, Separating Rule Discovery and Global Solution Composition in a Learning Classifier System, in: Genetic and Evolutionary Computation Conference Companion (GECCO '22 Companion), 2022. doi:10.1145/3520304.3529014.

[11] S. W. Wilson, Classifiers that approximate functions, Natural Computing 1 (2002) 211–234. doi:10.1023/a:1016535925043.

[12] R. J. Preen, D. Pätzel, XCSF, 2021. URL: https://github.com/rpreen/xcsf. doi:10.5281/zenodo.5806708.

[13] M. Heider, H. Stegherr, J. Wurth, R. Sraj, J. Hähner, Investigating the Impact of Independent Rule Fitnesses in a Learning Classifier System, in: M. Mernik, T. Eftimov, M. Črepinšek (Eds.), Bioinspired Optimization Methods and Their Applications, Springer International Publishing, Cham, 2022, pp. 142–156.

[14] S. W. Wilson, Compact Rulesets from XCSI, in: Advances in Learning Classifier Systems, Springer Berlin Heidelberg, 2002, pp. 197–208. doi:10.1007/3-540-48104-4_12.

[15] L. Bull, T. O'Hara, Accuracy-Based Neuro and Neuro-Fuzzy Classifier Systems, in: Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation, GECCO'02, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002, p. 905–911.

[16] P. L. Lanzi, D. Loiacono, XCSF with Neural Prediction, in: 2006 IEEE International Conference on Evolutionary Computation, 2006, pp. 2270–2276. doi:10.1109/CEC.2006.1688588.

[17] R. J. Urbanowicz, A. Granizo-Mackenzie, J. H. Moore, An Analysis Pipeline with Statistical and Visualization-guided Knowledge Discovery for Michigan-style Learning Classifier Systems, IEEE Computational Intelligence Magazine 7 (2012) 35–45. doi:10.1109/MCI.2012.2215124.

[18] Y. Liu, W. N. Browne, B. Xue, Absumption to Complement Subsumption in Learning Classifier Systems, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 410–418. doi:10.1145/3321707.3321719.

[19] Y. Liu, W. N. Browne, B. Xue, Visualizations for rule-based machine learning, Natural Computing (2021). doi:10.1007/s11047-020-09840-0.

[20] J. Bacardit, Pittsburgh genetics-based machine learning in the data mining era: representations, generalization, and run-time, Ph.D. thesis, PhD thesis, Ramon Llull University, Barcelona, 2004.

[21] M. Heider, H. Stegherr, D. Pätzel, R. Sraj, J. Wurth, B. Volger, J. Hähner., Approaches for Rule Discovery in a Learning Classifier System, in: Proceedings of the 14th International Joint Conference on Computational Intelligence - ECTA„ INSTICC, SciTePress, Setúbal, Portugal, 2022, pp. 39–49. doi:10.5220/0011542000003332.

[22] Q. Wu, Z. Ma, J. Fan, G. Xu, Y. Shen, A Feature Selection Method Based on Hybrid Improved Binary Quantum Particle Swarm Optimization, IEEE Access 7 (2019) 80588–80601. doi:10/gnxcfb.

[23] J. Wurth, M. Heider, H. Stegherr, R. Sraj, J. Hähner, Comparing different metaheuristics for model selection in a supervised learning classifier system, in: Genetic and Evolutionary Computation Conference Companion (GECCO '22 Companion), 2022. doi:10.1145/3520304.3529015.

[24] P. L. Lanzi, D. Loiacono, S. W. Wilson, D. E. Goldberg, Prediction update algorithms for XCSF: RLS, kalman filter, and gain adaptation, in: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, GECCO '06, Association for Computing Machinery, New York, NY, USA, 2006, p. 1505–1512. doi:10.1145/1143997.1144243.

[25] M. V. Butz, 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, 2005, pp. 1835–1842. doi:10.1145/1068009.1068320.

[26] 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, É. Duchesnay, Scikit-learn: Machine Learning in Python, The Journal of Machine Learning Research 12 (2011) 2825–2830.

[27] T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, Optuna: A Next-generation Hyperparameter Optimization Framework, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 2623–2631. doi:10/gf7mzz.

[28] D. Dua, C. Graff, UCI machine learning repository, 2017. URL: http://archive.ics.uci.edu/ml.

[29] H. Kaya, P. Tüfekci, Local and Global Learning Methods for Predicting Power of a Combined Gas & Steam Turbine, 2012.

[30] P. Tüfekci, Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods, International Journal of Electrical Power & Energy Systems 60 (2014) 126–140. doi:10/gn9s2h.

[31] T. Brooks, D. Pope, M. Marcolini, Airfoil Self-Noise and Prediction (1989).

[32] I.-C. Yeh, Modeling of Strength of High-Performance Concrete Using Artificial Neural Networks, Cement and Concrete Research 28 (1998) 1797–1808. doi:10/dxm5c2.

[33] A. Tsanas, A. Xifara, Accurate Quantitative Estimation of Energy Performance of Residential Buildings Using Statistical Machine Learning Tools, Energy and Buildings 49 (2012) 560–567. doi:10/gg5vzx.

[34] P. O. Stalph, M. V. Butz, Current xcsf capabilities and challenges, in: J. Bacardit, W. N. Browne, J. Drugowitsch, E. Bernadó-Mansilla, M. V. Butz (Eds.), Learning Classifier Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 57–69.

[35] D. Vermetten, H. Wang, M. López-Ibañez, C. Doerr, T. Bäck, Analyzing the impact of undersampling on the benchmarking and configuration of evolutionary algorithms, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 867–875. doi:10.1145/3512290.3528799.

[36] 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, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[37] G. Corani, A. Benavoli, A bayesian approach for comparing cross-validated algorithms on multiple data sets, Machine Learning 100 (2015) 285–304. doi:10.1007/s10994-015-5486-z.

[38] A. Benavoli, G. Corani, J. Demšar, M. Zaffalon, Time for a change: A tutorial for comparing multiple classifiers through bayesian analysis, J. Mach. Learn. Res. 18 (2017) 2653–2688.

[39] J. Mouret, J. Clune, Illuminating search spaces by mapping elites, CoRR (2015). URL: http://arxiv.org/abs/1504.04909. arXiv:1504.04909.

[40] A. Ferigo, L. L. Custode, G. Iacca, Quality diversity evolutionary learning of decision trees, in: Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, SAC '23, Association for Computing Machinery, New York, NY, USA, 2023, p. 425–432. doi:10.1145/3555776.3577591.