Towards Principled Synthetic Benchmarks for Explainable Rule Set Learning Algorithms

David Pätzel
david.paetzel@uni-a.de
University of Augsburg
Germany

Michael Heider
University of Augsburg
Germany

Jörg Hähner
University of Augsburg
Germany

ABSTRACT

A very common and powerful step in the design process of a new learning algorithm or extensions and improvements of existing algorithms is the benchmarking of models produced by said algorithm. We propose a paradigm shift in the benchmarking of explainable if-then-rule–based models like the ones generated by Learning Classifier Systems or Fuzzy Rule-based Systems. The principled method we suggest is based on synthetic data sets being sampled from randomly generated but known processes that have the same overall structure as the models that are being benchmarked (i. e. each process consists of a set of components each of which corresponds to one if-then rule) which is up-to-date not available among the many synthetic data generators. This approach has several benefits over other benchmarks and we expect that it will lead to fresh insights into how algorithms producing such explainable models work and can be improved. We demonstrate its usage by benchmarking the effects of different rule representations in the XCSF classifier system.

CCS CONCEPTS

• **Computing methodologies** → **Rule learning**; • **General and reference** → **Empirical studies**.

KEYWORDS

Rule Set Learning, Learning Classifier Systems, Fuzzy Rule-based Systems, Benchmarking, Evolutionary Machine Learning

ACM Reference Format:
David Pätzel, Michael Heider, and Jörg Hähner. 2023. Towards Principled Synthetic Benchmarks for Explainable Rule Set Learning Algorithms. In *Genetic and Evolutionary Computation Conference Companion (GECCO '23 Companion), July 15–19, 2023, Lisbon, Portugal.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3583133.3596416

1 INTRODUCTION

One family of approaches for building human-comprehensible, explainable machine learning models is based on searching an optimal set of if-then rules which are inherently transparent. Within this work we term these algorithms *Rule Set Learners* (RSLs). RSLs can be roughly categorized into RSLs that utilize heuristics to iteratively generate a set of rules (e. g. the very popular C4.5 algorithm [17]) and RSLs that more directly search the space of possible rule sets using metaheuristics such as evolutionary algorithms. This paper focuses on the second category which we call *Metaheuristic RSLs* (MRSLs). Prominent representatives of MRSL algorithms include *Learning Classifier Systems* (LCSs) [8, 22], *Fuzzy Rule-based Systems* (FRBS) [5] and Ant-Miner [12]. There are MRSL approaches to classification [e. g. 2, 12], regression [e. g. 1, 10, 25], unsupervised learning [e. g. 20] as well as reinforcement learning [e. g. 4].

Since MRSLs build models consisting of if-then rules, they implicitly assume that the data was generated by a process consisting of if-then components [7]. Applying an MRSL to data that was not generated by such a process may still yield good results and many years of successful empirical MRSL research applying methods to data sets whose generating process is unknown (and thus probably does not fulfil above-stated assumption) show that this is indeed the case. Nevertheless, we argue that more insights into MRSLs and how they may be improved can be gained if the data was generated by a process fulfilling said assumption—and even more so, if the process itself was known formally. In particular, the generated models' transparency can be gauged more directly due to the fact that the data-generating process itself is transparent as well (after all, it itself consists of if-then components) and a machine learning expert can properly trace specific issues in the algorithm's optimization process.

We therefore propose to benchmark MRSLs using sets of synthetic data sets. At that, each data set is sampled from a randomly generated but known process that has the same structure as an MRSL model (i. e. it consists of a set of components each of which corresponds to one if-then rule). We then expect, the benchmarked MRSL algorithms to build models that reconstruct this data-generating process—with measurable progress towards that goal.

It should be noted that some of the learning tasks that have been used by MRSL research in the past *are* based on known data-generating processes. Prominent examples from the LCS community for which the optimal set of if-then rules are known include

- the Boolean multiplexer tasks [e. g. 24] which are based on certain functions of the form $\mathbb{B}^n \to \mathbb{B}$,
- the real-valued checkerboard problem [3] as well as
- grid world reinforcement learning tasks [e. g. 24] where one may derive the optimal set of rules by inspection of the environment's dynamics.

Another example is the first benchmark task used by Drugowitsch [7] who manually defines three if-then rules to generate one-dimensional data with the purpose of visualizing how the learnt RSL models look like. However, to our knowledge, all existing data

ACM ISBN 979-8-4007-0120-7
https://doi.org/10.1145/3583133.3596416

generators are rather restricted *individual* approaches whereas our proposal is to *automatedly generate as many benchmark tasks as required*. Furthermore, despite restricting ourselves to regression tasks within this paper, the proposed approach can be applied to all kinds of learning tasks (i. e. classification and regression as well as reinforcement and unsupervised learning).

The remainder of this paper is structured as follows: We start by giving in Section 2 an overview of the types of models that MRSL algorithms build and, correspondingly, the structure of the data-generating processes that we aim for. Section 3 then provides an overview of how data-generating processes and data sets can be built. In Section 4, we discuss several benefits the proposed benchmarking approach has. In order to provide a better intuition, we demonstrate in Section 5 how the approach can be applied to a comparison of two algorithm variants. Section 6 concludes our work with a short discussion of possible next research steps.

2 METAHEURISTIC RULE SET LEARNING

To keep this paper mostly self-contained, we start by giving a rough overview of MRSL for supervised learning tasks. This is similar to but slightly extends the treatment of LCS models given by Heider et al. [8].

Supervised learning deals with the task of approximating a process that maps inputs $x \in \mathcal{X} = \mathbb{R}^{\mathcal{D}_\mathcal{X}}$ to outputs $y \in \mathcal{Y}$. At that, this mapping is possibly (or rather, *most certainly* for real-world tasks) non-deterministic (e. g. noisy). The definition of $\mathcal{Y}$ depends on the type of supervised learning task being solved: For example, regression tasks typically specify $\mathcal{Y} = \mathbb{R}$ whereas classification tasks with $\mathcal{D}_\mathcal{Y}$ classes are commonly approached by defining $\mathcal{Y}$ to be the set of one-hot encoded class representations (i. e. $\mathcal{Y} = \{y \mid y \in \{0,1\}^{\mathcal{D}_\mathcal{Y}}, \sum y = 1\} \subset \mathbb{R}^{D_\mathcal{Y}}$).

MRSL assumes that the process to be approximated can be modelled well by a set of *rules* akin to if-then statements[1]; correspondingly, MRSL models contain a *finite set of K rules*, $K \in \mathbb{N}$. The kth of those rules (simply termed *rule k* in the remainder of this paper) contains a *matching function* $m(\psi_k) : \mathcal{X} \to \mathbb{B}$ that corresponds to the rule's if-part and determines for any possible input $x \in \mathcal{X}$ whether or not that rule *matches* that input[2]. At that, the matching functions are members of a parametric family m of functions parametrized by a set of parameters ψ_k.

Rule k further contains a *local model*, a discriminative function $\widehat{f_k}(\theta_k) : \mathcal{X} \to \mathcal{Y}$ which corresponds to the rule's then-part and whose parameters θ_k are trained on the inputs that the rule matches—independently of other rules. MRSL algorithms in general explicitly allow rules in the generated models to overlap which, however, makes it necessary to define a method of mixing the local model outputs of all the rules that match a certain input. This is achieved by the MRSL model's *mixing model* which constitutes the third and final component of the MRSL model. Simple (and most commonly–used) mixing models assign to each rule k a *constant mixing weight* γ_k. How that mixing weight is computed exactly differs between MRSL systems; a simple mental model sufficient for the present work is to assume γ_k to be some sort of *inverse of the rule k's prediction error on the inputs that it matches* (i. e. rules that have a higher error on the inputs that they match contribute less to the overall model's output). The number of rules and the set of matching function parameters $\mathcal{M} = (K, \{\psi_k\}_{k=1}^K)$ form the MRSL model's *model structure* (similar to how the graph of a Neural Network forms that network's model structure). The local model parameters and mixing weights form the set of the MRSL model's *parameters* $\theta = \{\theta_k, \gamma_k\}_{k=1}^K$. Putting this all together, the overall form of an MRSL model is a weighted sum of local model outputs:

$$\widehat{f}_\mathcal{M}(\theta, x) = \sum_{k=1}^{K} m(\psi_k; x)\, \gamma_k\, \widehat{f_k}(\theta_k; x) \tag{1}$$

Since one of the major selling points of using MRSL algorithms is the inherent transparency of the resulting models, the *local* models are usually assumed to be *simple* with common choices being constant [e. g. 2, 5, 12, 24] or linear models [e. g. 7, 25]. This can be expressed more generally by stating that the number of parameters of the local models are in $O(\mathcal{D}_\mathcal{X})$ with small constant factors. Assuming local models to be simple and trained independently of each other means that fitting their parameters is straightforward since optimizing the local model parameters (of which there are $O(K\mathcal{D}_\mathcal{X})$ many) is decomposed into K independent optimization problems of $O(\mathcal{D}_\mathcal{X})$ parameters. As was already mentioned above, mixing weights are most commonly implemented as a single constant per rule, an overall of $O(K)$ mixing model parameters, with well-performing heuristics to choose them being available (see [7] for an overview of techniques). This means that most of the effort of training MRSL algorithms stems from searching for well-performing model structures, since model structures typically have $O(K\mathcal{D}_\mathcal{X})$ parameters (i. e. $O(\mathcal{D}_\mathcal{X})$ parameters per each of the K rules) that can, other than the $O(K\mathcal{D}_\mathcal{X})$ local model parameters, *not* be decomposed straightforwardly and need to optimized jointly.

MRSL model structure space (i. e. good values of $\mathcal{M}$) can be searched in many ways; a high-level classification and discussion is given by Heider et al. [8]. Most approaches use some form of metaheuristic such as evolutionary [22, 24] or swarm-based algorithms [12]. For example, the most prominent LCS, XCS [24], and its main derivatives [e. g. 2, 25] use a niching steady-state genetic algorithm (GA) to optimize the set of matching functions.

A common example for an MRSL model family for regression tasks is based on the following choices: Each matching function corresponds to a half-open $\mathcal{D}_\mathcal{X}$-dimensional interval, that is, $\psi_k = \{l_k, u_k\}$ with $l_k, u_k \in \mathcal{X}$ and $m_k(\psi_k; x) = 1$ if $x \in [l_k, u_k)$ and $m_k(\psi_k; x) = 0$ otherwise. Local models are linear regression models including an intercept term, they thus have a vector of parameters $\beta_k \in \mathbb{R}^{\mathcal{D}_\mathcal{X}+1}$. Each local model is further associated with an estimate of the expected noise variance; the most common choice is isotropic normally–distributed noise which induces a parameter $\sigma_k^2 \in \mathbb{R}$ resulting in overall local model parameters $\theta_k = \{\beta_k, \sigma_k^2\}$. Finally, the mixing weight for each rule k is a constant $\gamma_k \in \mathbb{R}$. In total, this model family has

$$K(2\mathcal{D}_\mathcal{X} + (\mathcal{D}_\mathcal{X} + 2) + 1) = 3K + 3K\mathcal{D}_\mathcal{X} \tag{2}$$

[1] Note that while there *are* probabilistic formulations of MRSL models [e. g. 7] these are a minority and we stick for ease of introduction with the more commonly–used discriminative function form.

[2] There are approaches to MRSL that use matching by degree, most notably, FRBS. In that case, matching functions map to a number between 0 and 1, $m_k : \mathcal{X} \to [0, 1]$, however, this distinction is not critical for the treatment given in this paper.

parameters (for each of the K rules: $2\mathcal{D}_\mathcal{X}$ matching function parameters, $\mathcal{D}_\mathcal{X} + 2$ local model parameters, one mixing weight).

3 GENERATING LEARNING TASKS AND DATA SETS FOR BENCHMARKING MRSL

As was already stated in Section 1, we propose to generate benchmark learning tasks based on the assumptions that MRSL models make about the data that they are meant to model. This means essentially that we generate a random MRSL model which can be queried to generate data. In order to generate a benchmark learning task this way, we need to specify the number of rules K that the generating MRSL model should have and for each of the rules, the matching function parameters ψ_k the local model parameters θ_k and the mixing weight parameter γ_k.

At that, we suggest to set the number of rules K manually since that is the main parameter for influencing the generated learning task's difficulty (an optimal learnt model will have to have that many rules; which defines the number of parameters that the metaheuristic searching model structure space has to optimize) and to then draw the remaining parameters at random from suitable probability distributions. Assuming we have done that, we have generated a learning task from which we can sample a single data point as follows:

(1) Draw a random input $x \in \mathcal{X}$ (e. g. uniformly at random).
(2) For each of the rules k, draw an output value from the rule's output distribution (i. e. for the normal-noise linear models, draw a noise value from $\mathcal{N}(0, \sigma_k^2)$ and add that to the output of the linear model $\widehat{f_k}(\theta_k; x)$).
(3) Compute Eq. (1) to obtain the output $y \in \mathcal{Y}$.

4 BENEFITS OF THIS APPROACH

The presented approach has several benefits over benchmarking based on data sets with unknown data-generating processes, some of which we now discuss shortly.

Learning tasks as well as data can be generated in large quantities which enables robust statistical conclusions to be drawn about the performance of a certain algorithm under benchmark. For example, we could draw 50 such learning tasks for different sizes of K and for each of them a large test data set.

The difficulty of the generated learning tasks is easily configurable. For one, we are able to specify how many rules optimal solutions should have (with more rules being more difficult for the model structure search to optimize due to the additional parameters that need to be optimized). In addition, we can enforce certain other properties as well—for example, by manipulating the probability distributions that parameters are drawn from (e. g. having more or less steep local linear models). Note further that it is not necessary to specify exactly the properties we want before generating learning tasks. Instead, we can simply generate many tasks and then select ones we find interesting, for example by filtering out too easy tasks that can be approximated well by a single linear model.

Finally, since we know the ground truth (i. e. not just the labels of the data set but the actual data-generating model) and that ground truth has a model form similar to the model we're trying to fit, we can compute additional statistics that let us reason more directly about the algorithm being analysed. For example, we may compute how close the metaheuristic model structure search's solutions (e. g. matching function parameters) actually are to the corresponding parameters of the data-generating process and based on that trace and explain specific issues in the algorithm's optimization process.

5 DEMONSTRATION

Let's assume that we want to investigate the effects of different representations of interval-based matching functions (see Section 2) on the XCSF algorithm[3] [25] which is a prominent MRSL for regression tasks. There are many ways to encode such intervals for performing evolutionary search with two of the main ideas being the *unordered-bound* and *centre-spread* representations [6, 19].

Unordered-bound representation means that the $\mathcal{D}_\mathcal{X}$-dimensional interval $[l, u)$ is encoded as a vector of reals $(b_{1,1}, b_{1,2}, b_{2,1}, b_{2,2}, \ldots, b_{\mathcal{D}_\mathcal{X},1}, b_{\mathcal{D}_\mathcal{X},2})$ where each pair $(b_{j,1}, b_{j,2})$ corresponds to a one-dimensional interval $[l_j, u_j) = [\min(b_{j,1}, b_{j,2}), \max(b_{j,1}, b_{j,2}))$. This implies that the interval bounds are sorted whenever the phenotype is computed. The centre-spread representation on the other hand represents the $\mathcal{D}_\mathcal{X}$-dimensional interval $[l, u)$ as a vector of reals $(c_1, s_1, c_2, s_2, \ldots, c_{\mathcal{D}_\mathcal{X}}, s_{\mathcal{D}_\mathcal{X}})$ where each pair (c_j, s_j) corresponds to a one-dimensional interval $[c_j - s_j, c_j + s_j)$, that is, c_j is the jth interval's *centre* and s_j is its *spread*.

Assume that we are particularly interested in general statements about whether XCSF using unordered-bound representations performs better than XCSF using centre-spread representations. Note that investigations like this have already been the topic of several publications [e. g. 6, 19] but for the sake of demonstration we'll assume for now that we are not aware of these. Note further that, for either of the two, several sub-options are possible (especially with respect to the exact effects of the evolutionary operators used); we simplify this by stating that we use Preen's XCSF Python library [13] which supports both of these representations out-of-the-box and publish our experimentation scripts [14] which contain the parametrization. In the remainder of this paper, we abbreviate *XCSF with unordered-bound representation as implemented by Preen's library* as UBR and *XCSF with centre-spread representation as implemented by Preen's library* as CSR.

We now start by generating learning tasks (our data generation code can be found online [15]) as described in Section 3 based on the MRSL model family example given at the end of Section 2. Assume that we are interested in analysing performance of models that are human-comprehensible [9] and that we thus create tasks that can be solved optimally with not too many rules while at the same time varying the number of rules required. Therefore, we choose learning tasks with three different numbers of components: $K \in \{5, 10, 20\}$. We further want to investigate different input space dimensions and choose $\mathcal{D}_\mathcal{X} \in \{1, 3, 5, 10, 20\}$ and for each of those a somewhat sensible number of training examples N (given in Table 1). Note that all of these choices are rather arbitrary and more thought would need to be given to them if one were to conduct an in-depth study rather than a demonstration.

For each of the 15 combinations of K and $\mathcal{D}_\mathcal{X}$, we now generate 10 learning tasks: For each learning task,

[3] How the XCSF algorithm works is not relevant for the following discussion; we refer the interested reader to the given reference.

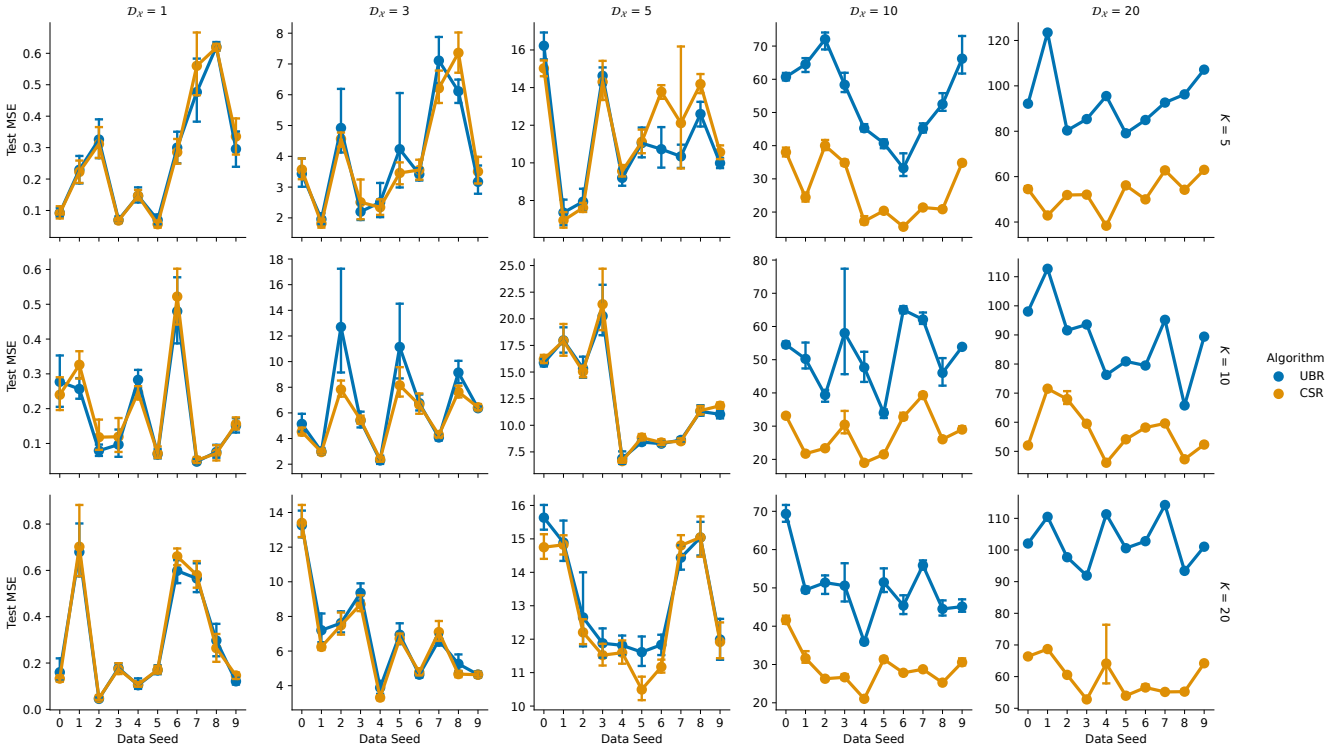

Figure 1: For each generated learning task, the mean of the test MSEs achieved by the 20 runs performed for UBR and CSR. Error bars are non-parametric 95 % confidence interval (estimated from 1000 bootstrap samples).

- we draw uniformly at random and independently valid values for $\{\psi_k\}$, $\{\theta_k\}$ and $\{\gamma_k\}$. These define the learning task's data-generating model.
- we use these parameters to generate a training set of size N and a test set of size $10N$ by drawing input data points uniformly at random from the input space and based on those sampling an output data point from the model.

This yields a total of 150 pairs of training and test data sets[4]. Table 1 displays training data set sizes as well as mean (over the ten learning tasks) goodness-of-fit measured by *mean squared error* (MSE) on the respective test data set (called *test MSE* in the following) of a linear model fitted to the respective training data as well as of the data-generating model[5] itself.

For each learning task, we fit each of UBR and CSR to the training data set and then compute the test MSEs. Since XCSF is a stochastic algorithm, we have to control for the associated inherent variance and perform 20 independent repetitions (20 different random seeds) of this procedure. The means over the 20 test MSEs per task and representation are displayed in Figure 1.

We can see that for input dimension 1, UBR and CSR appear to perform very similarly whereas for input dimensions 3 and 5 some differences are visually noticeable with UBR maybe performing slightly worse on some of the generated tasks. For input dimensions 10 and 20, the difference in test MSE performance seems to increase even further—it is visually unambiguous that UBR has a much higher mean test MSE than CSR on the tasks considered[6]. Nevertheless, neither UBR nor CSR come even close to the MSEs achieved by the data-generating model itself (Table 1) despite being

		Lin. Models' MMSE			Gen. Models' MMSE		
	K	5	10	20	5	10	20
$\mathcal{D}_\mathcal{X}$	N						
1	300	4.23	1.93	1.00	0.01	0.01	0.00
3	500	10.10	10.38	10.00	0.01	0.01	0.01
5	1000	16.99	15.54	15.97	0.01	0.02	0.01
10	2000	30.96	28.59	29.24	0.02	0.01	0.02
20	5000	54.14	56.66	58.40	0.02	0.02	0.01

Table 1: Statistics of the XCSF study learning tasks showing for each input space dimension $\mathcal{D}_\mathcal{X}$ and number of components K the number N of training data points generated as well as the mean (over the 10 generated tasks) test MSE achieved by a single linear regression model fitted to the training data and by the data-generating model itself.

[4]Data sets can be found online [16].
[5]Note that the data-generating model's MSE is not zero due to the rules' noise terms.
[6]Performing proper statistical tests to confirm this is out of the scope of the present paper where we focus on how to generate data to perform such tests rather than solving the question of when to use UBR or CSR.

allowed to use at least ten times the number of rules[7] to model the data. Also, CSR's MSEs are not even much lower than the MSEs achieved by the overly simple linear model (the order of magnitude is given by the means over the 10 data sets per combination of $\mathcal{D}_\mathcal{X}$ and K provided in Table 1). This indicates that these kinds of learning tasks may actually be quite challenging for XCSF[8] and additional effort such as parameter tuning and further investigation may be necessary to achieve an acceptable fit. In addition, UBR's MSEs are considerably worse than the MSEs achieved by singular linear models which hints at further problems such as *cover-delete-cycles* [e. g. 18] caused by misconfiguration; and, indeed, when looking at XCSF's internal rule *experience* parameters, cover-delete-cycles seem likely to be the case since for a majority of rules experience values are rather small for UBR (but not for CSR). This is likely due to, in our experiments, initial rule sizes being too small for UBR for higher input dimensions; yet, further investigation and resolution of this issue is out of the scope of this demonstration.

Since we have the ideal set of rules available (i. e. the data-generating model), we were able to also inspect the matching function parameters discovered by XCSF and compare them to the parameters of the data-generating models' matching function parameters (i. e. the ψ_k). We noticed that there is quite some discrepancy here (even for the lower input dimension tasks where XCSF's performance is in a somewhat acceptable range) which definitely warrants a closer inspection of XCSF's modelling capabilities as well as its evolutionary optimization process.

Learning tasks being of different difficulty for different combinations of $\mathcal{D}_\mathcal{X}$ and K is expected since these parameters specify the number of parameters of the model generating the data (see Section 3) and thus the number of parameters that a learning algorithm has to fit; this effect is clearly visible from the differing scales of the test MSEs in Figure 1. However, we can also see from that same figure that there are considerable differences in test MSE between different learning tasks *of the same combination* of $\mathcal{D}_\mathcal{X}$ and K. This indicates that even for fixed $\mathcal{D}_\mathcal{X}$ and K, the generated learning tasks are of significantly different difficulty and suggests that a more thorough investigation is called for.

6 CONCLUSION AND FUTURE WORK

We have presented a principled method to benchmark explainable if-then-rule–based models like the ones generated by metaheuristic Rule Set Learners (MRSLs) such as Learning Classifier Systems or Fuzzy Rule-based System. The approach is based on generating and sampling from random data-generating processes that have the same overall structure as the models being tested. This allows for new ways of investigating MRSLs to be developed since it provides large amounts of data, configurability and a clear target for metaheuristic model structure search in the form of a known data-generating process. As a demonstration, we conducted a preliminary study that compared the effects of different rule encodings on the XCSF classifier system. We were able to show that our approach works and can lead to fresh insights.

We are currently in the process of reviewing principled statistical tools that leverage the benefits of the proposed approach. Aside from this we are looking into how learnt rule-based models should be scored if the data-generating process is known.

Further research is necessary, for example, into the differences within the sets of generated learning tasks and whether the distributions that the data-generating model parameters are drawn from require tuning (e. g. considering component overlap or overall linearity). This includes an investigation of the difficulty of the generated learning tasks (e. g. by baselining with different kinds of machine learning algorithms) as well as the differences in difficulty for a fixed combination of $\mathcal{D}_\mathcal{X}$ and K. We expect that work on this will also exhibit ways to meaningfully filter the set of learning tasks with respect to certain characteristics.

[7] We set XCSF's population size to 200; a common rule of thumb is "ten times the number of expected rules" which we fulfil with that number.

[8] Note that we also preliminarily fitted (with minimal manual tuning) a few other kinds of learning algorithms (e. g. random forests, multi-layer perceptrons) to some of the $\mathcal{D}_\mathcal{X} = 20$ data and that on first glance the resulting models did not perform significantly better than the linear regression model either.

REFERENCES

[1] Rafael Alcala, María José Gacto, and Francisco Herrera. 2011. A Fast and Scalable Multiobjective Genetic Fuzzy System for Linguistic Fuzzy Modeling in High-Dimensional Regression Problems. *IEEE Transactions on Fuzzy Systems* 19, 4 (2011), 666–681. https://doi.org/10.1109/TFUZZ.2011.2131657

[2] Ester Bernadó-Mansilla and Josep M. Garrell-Guiu. 2003. Accuracy-Based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks. *Evolutionary Computation* 11, 3 (09 2003), 209–238. https://doi.org/10.1162/106365603322365289 arXiv:https://direct.mit.edu/evco/article-pdf/11/3/209/1493332/1063656033223652 89.pdf

[3] Larry Bull and Jacob Hurst. 2003. A neural learning classifier system with self-adaptive constructivism. In *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, Vol. 2. 991–997 Vol. 2.

[4] Martin V. Butz and Wolfgang Stolzmann. 2002. An Algorithmic Description of ACS2. In *Advances in Learning Classifier Systems*, Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 211–229.

[5] Oscar Cordón. 2011. A historical review of evolutionary learning methods for Mamdani-type fuzzy rule-based systems: Designing interpretable genetic fuzzy systems. *International Journal of Approximate Reasoning* 52, 6 (2011), 894–913. https://doi.org/10.1016/j.ijar.2011.03.004

[6] Hai H. Dam, Hussein A. Abbass, and Chris Lokan. 2005. Be Real! XCS with Continuous-Valued Inputs. In *Proceedings of the 7th Annual Workshop on Genetic and Evolutionary Computation* (Washington, D.C.) *(GECCO '05)*. Association for Computing Machinery, New York, NY, USA, 85–87. https://doi.org/10.1145/1102256.1102274

[7] Jan Drugowitsch. 2008. *Design and Analysis of Learning Classifier Systems - A Probabilistic Approach*. Studies in Computational Intelligence, Vol. 139. Springer Berlin Heidelberg, Berlin, Heidelberg.

[8] Michael Heider, David Pätzel, Helena Stegherr, and Jörg Hähner. 2023. *A Metaheuristic Perspective on Learning Classifier Systems*. Springer Nature Singapore, Singapore, 73–98. https://doi.org/10.1007/978-981-19-3888-7_3

[9] Michael Heider, Helena Stegherr, Richard Nordsieck, and Jörg Hähner. 2022. Learning Classifier Systems for Self-Explaining Socio-Technical-Systems. arXiv:2207.02300 [cs.HC]

[10] Michael Heider, Helena Stegherr, Jonathan Wurth, Roman Sraj, and Jörg Hähner. 2022. Separating Rule Discovery and Global Solution Composition in a Learning Classifier System. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (Boston, Massachusetts) *(GECCO '22)*. Association for Computing Machinery, New York, NY, USA, 248–251. https://doi.org/10.1145/3520304.3529014

[11] John D. Hunter. 2007. Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering* 9, 3 (2007), 90–95. https://doi.org/10.1109/MCSE.2007.55

[12] Rafael S. Parpinelli, Heitor S. Lopes, and Alex A. Freitas. 2002. An Ant Colony Algorithm for Classification Rule Discovery. In *Data Mining*. IGI Global, 191–208. https://doi.org/10.4018/978-1-930708-25-9.ch010

[13] Richard J. Preen and David Pätzel. 2022. *XCSF*. https://doi.org/10.5281/zenodo.7139881

[14] David Pätzel. 2023. *dpaetzel/run-rsl-bench: v1.0.1*. https://doi.org/10.5281/zenodo.7915821

[15] David Pätzel. 2023. *dpaetzel/syn-rsl-benchs: v1.0.0*. https://doi.org/10.5281/zenodo.7919420

[16] David Pätzel. 2023. *Synthetic Datasets from the 2023 ECXAI Workshop Paper on Principled Benchmarking for Rule Set Learning Algorithms*. https://doi.org/10.

5281/zenodo.7919492

[17] J. Ross Quinlan. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[18] Patrick O. Stalph, Xavier Llorà, David E. Goldberg, and Martin V. Butz. 2012. Resource management and scalability of the XCSF learning classifier system. *Theoretical Computer Science* 425 (2012), 126–141.

[19] Christopher Stone and Larry Bull. 2003. For Real! XCS with Continuous-Valued Inputs. *Evolutionary Computation* 11, 3 (Sep 2003), 299–336.

[20] Kreangsak Tamee, Larry Bull, and Ouen Pinngern. 2007. Towards Clustering with XCS. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation* (London, England) *(GECCO '07)*. Association for Computing Machinery, New York, NY, USA, 1854–1860. https://doi.org/10.1145/1276958.1277326

[21] Ole Tange. 2020. *GNU Parallel 20200822 ('Beirut')*. https://doi.org/10.5281/zenodo.3996295

[22] Ryan J. Urbanowicz and Will N. Browne. 2017. *Introduction to Learning Classifier Systems*. Springer. https://doi.org/10.1007/978-3-662-55007-6

[23] Michael L. Waskom. 2021. seaborn: statistical data visualization. *Journal of Open Source Software* 6, 60 (2021), 3021. https://doi.org/10.21105/joss.03021

[24] Stewart W. Wilson. 1995. Classifier Fitness Based on Accuracy. *Evolutionary Computation* 3, 2 (1995), 149–175.

[25] Stewart W. Wilson. 2002. Classifiers that approximate functions. *Natural Computing* 1, 2 (01 6 2002), 211–234. https://doi.org/10.1023/A:1016535925043