mathematics

MDPI

Article

NiaAutoARM: Automated Framework for Constructing and Evaluating Association Rule Mining Pipelines

Uroš Mlakar *, Iztok Fister, Jr. and Iztok Fister

Faculty of Electrical Engineering and Computer Science, University of Maribor, Koroška Cesta 46, 2000 Maribor, Slovenia; iztok.fister1@um.si (I.F.J.); iztok.fister@um.si (I.F.)

* Correspondence: uros.mlakar@um.si

Abstract: Numerical Association Rule Mining (NARM), which simultaneously handles both numerical and categorical attributes, is a powerful approach for uncovering meaningful associations in heterogeneous datasets. However, designing effective NARM solutions is a complex task involving multiple sequential steps, such as data preprocessing, algorithm selection, hyper-parameter tuning, and the definition of rule quality metrics, which together form a complete processing pipeline. In this paper, we introduce NiaAutoARM, a novel Automated Machine Learning (AutoML) framework that leverages stochastic population-based metaheuristics to automatically construct full association rule mining pipelines. Extensive experimental evaluation on ten benchmark datasets demonstrated that NiaAutoARM consistently identifies high-quality pipelines, improving both rule accuracy and interpretability compared to baseline configurations. Furthermore, NiaAutoARM achieves superior or comparable performance to the state-of-the-art VARDE algorithm while offering greater flexibility and automation. These results highlight the framework's practical value for automating NARM tasks, reducing the need for manual tuning, and enabling broader adoption of association rule mining in real-world applications.

Keywords: AutoML; association rule mining; numerical association rule mining; pipelines

MSC: 68TXX; 68T42

check for updates

Academic Editor: Janez Žerovnik

Received: 8 May 2025
Revised: 29 May 2025
Accepted: 12 June 2025
Published: 13 June 2025

Citation: Mlakar, U.; Fister, I., Jr.; Fister, I. NiaAutoARM: Automated Framework for Constructing and Evaluating Association Rule Mining Pipelines. *Mathematics* **2025**, *13*, 1957. https://doi.org/10.3390/math13121957

1. Introduction

The design of Machine Learning (ML) pipelines usually demands user interaction to select appropriate preprocessing methods, perform feature engineering, select the most appropriate ML method, and set a combination of hyper-parameters [1]. Therefore, preparing an ML pipeline is complex, and, primarily, it is inappropriate for non-specialists in the data science or artificial intelligence domains [2]. On the other hand, tuning the entire pipeline to produce the best results may also involve a great deal of time for the users, especially if we deal with very complex datasets.

Automated Machine Learning (AutoML) methods have appeared to draw the application of ML methods nearer to the users (in the sense of ML democratization) [2,3]. The main benefit of these methods is searching for the best pipeline in different ML tasks automatically. Until recently, AutoML forms can be found for solving classification problems, neural architecture search, regression problems [4], and reinforcement learning.

Association Rule Mining (ARM) is a ML method for discovering the relationships between items in transaction databases. Bare ARM is limited since it initially operates with a categorical type of attributes only. Recently, Numerical Association Rule Mining (NARM) was proposed, which is a variant of a bare ARM and allows for dealing with

numerical and categorical attributes concurrently. Thus, it removes the bottleneck of the bare ARM. The NARM also delivers several benefits since the results can be more reliable and accurate, and it contains less noise than bare ARM, but the numerical attributes need to be discretized before use. Currently, the problem of NARM is mainly tackled through using population-based meta-heuristics, which can cope large search spaces effectively. (Please note that the acronym ARM is used as a synonym for the acronym NARM in the paper.)

The ARM pipeline (see Figure 1) is far from being uncomplicated since it consists of several components: (1) data preprocessing, (2) mining algorithm selection, (3) hyper-parameter optimization, (4) evaluation metric selection, and (5) evaluation. Each of these components can be implemented using several ML methods.

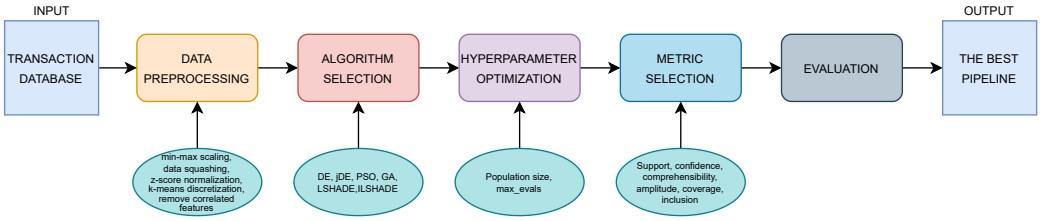


Figure 1. The structure of the basic ARM pipeline.

Consequently, composing the ARM pipeline manually requires a great deal of human intervention, and it is potentially a time-consuming task. Therefore, automation of this composing led us to ARM democratization and, consequently, to the new domain of AutoML, i.e., AutoARM.

The data entering the ARM pipeline are in the form of a transaction database; the optional first component of the ARM pipeline is preprocessing, where the data can be preprocessed further using various ML methods. The selection of the proper processing component presents a crucial step, where the most appropriate population-based meta-heuristic nature-inspired (NI) algorithm needs to be determined for ARM. Mainly, the NI algorithms encompasses two classes of population-based algorithms: Evolutionary Algorithms (EAs) [5] and swarm intelligence (SI)-based ones [6].

According to previous studies, no universal, population-based meta-heuristic exists for ARM that can achieve the best results by mining all datasets. This phenomenon is also justified by the No Free Lunch (NFL) theorem of Wolpert and Macready [7]. The next component in the pipeline is the hyper-parameter optimization for the selected population-based meta-heuristic, where the best combination of hyper-parameters is searched for. Finally, the selection of the favorable association rules depends on the composition of the more suitable metrics captured in the fitness function. In our case, the fitness function is represented as a linear combination of several ARM metrics (e.g., support, confidence, amplitude, etc.) weighted with particular weights.

A structured comparison of existing ARM approaches is presented in Table 1, where their level of automation, hyper-parameter tuning capabilities, and optimization techniques were focused on. The table illustrates the diversity in methodological design, ranging from manual, heuristic-based systems to fully automated, data-driven solutions.

To the best knowledge of the authors, no specific AutoML methods exist for constructing the ARM pipelines automatically. Therefore, the contributions of this study are as follows:

- To propose the first AutoARM solution for searching for the best ARM pipeline, where this automatic searching is represented as an optimization problem.
- To dedicate special attention to the preprocessing steps of ARM, which have been neglected slightly in recent research works.
- To implement a new framework called NiaAutoARM v.0.1.1 as a Python package v.3.9.
- To evaluate the proposed framework rigorously on several datasets.

Table 1. Comparison of association rule mining methods.

Method	Type	AutoML	HPT	Optimization
Apriori [8]	Traditional	✗	✗	None
FP-Growth [9]	Traditional	✗	✗	None
MODENAR [10]	NARM	✗	✗	DE
Hybrid DE-SCA [11]	NARM	✗	✗	DE + SCA [1]
NARM-DE [12]	NARM	✗	✗	DE
Multi-objective GA [13]	NARM	✗	✗	GA
BA [14]	NARM	✗	✗	BA
PSO [15]	NARM	✗	✗	PSO
NiaAutoARM	AutoML-NARM	✓	✓	AutoARM pipeline

[1] Sine cosine algorithm.

The structure of the remainder of this paper is as follows: The materials and methods, needed for understanding the observed subjects that follow, are discussed in Section 2. The proposed method for automated ARM is described in Section 3 in detail. The experiments and the obtained results are the subjects of Section 4, where a short discussion of the results is also presented. This paper is then concluded in Section 5 with a summarization of the performed work and an outlining of the potential directions for future work.

2. Materials and Methods

The section highlights the topics necessary for understanding the subjects of this paper. In line with this, the following topics are handled:

- NI meta-heuristics.
- AutoML.
- NiaAML.
- NiaARM.

The mentioned topics are discussed in detail in the remainder of this paper.

2.1. NI Meta-Heuristics

Exact solving of NP-hard optimization problems [16] requires enormous time and space resources. However, in practical applications, exact solutions are often unnecessary as we are typically satisfied with high-quality approximate solutions obtained within a reasonable time. Consequently, interest in approximate, or heuristic, approaches for solving intractable problems has grown significantly, especially following the emergence of nature-inspired (NI) algorithms.

Heuristics solve the optimization problems directly, i.e., on the lower level. The term "meta-heuristic" refers to a higher-level procedure or heuristic in the fields of computer science, mathematical optimization, and engineering, and it is used to search for, find, generate, or select a heuristic that may offer a good solution to an optimization problem, particularly for large problems (i.e., NP-hard problems) or in cases of limited, incomplete, or imperfect information [17].

One of the first meta-heuristic concepts that used NI algorithms was introduced by Grefenstette in [18], who applied a meta-Genetic Algorithm (meta-GA) to control the parameters of another GA. Recently, this approach has become increasingly widespread, especially in the field of Machine Learning (ML), where meta-heuristics are used for setting the hyper-parameters of neural networks (NNs) [19,20].

2.2. AutoML

Using ML methods in practice demands experienced human ML experts, who are typically expensive and hard to find on the market. On the other hand, computing is becoming cheaper day by day. This fact has led to the advent of AutoML, which is capable

of constructing ML pipelines that are of a similar, or even better, quality than those by human experts [2]. Consequently, AutoML enables the so-called democratization of ML. This means that the usage of the ML methods is drawn closer to the user by AutoML; hus, this technology tries to avoid the principle of human-in-the-loop [21].

Automation of ML methods is allowed by AutoML using ML pipelines. Indeed, these pipelines are the control points of the AutoML system. Typically, the ML pipeline consists of the following processing steps:

- Preprocessing.
- Processing with definite ML methods.
- Hyper-parameter optimization.
- Evaluation.

AutoML is, currently, a very studied research area. The recent advances in the field have been summarized in several review papers [1,3,22,23]. There also exist a dozen applications of AutoML [24,25], where the special position is devoted to NiaAML, which is discussed in more detail in the remainder of this section.

2.3. NiaAML

NiaAML is an AutoML method based on stochastic Nia-s for optimization, where the AutoML is modeled as an optimization problem. The first version of NiaAML [26] covers composing classification pipelines, whereas a stochastic Nia searches for the best classification pipeline. The following steps are included in the AutoML pipeline, i.e., automatic feature selection, feature scaling, classifier selection, and hyper-parameter optimization. Each classifier configuration found by the optimizers was tested using cross-validation.

Following NiaAML, the NiaAML2 [27] method was proposed, which eliminates the main weakness of the original NiaAML method, where the hyper-parameters' optimization is performed simultaneously with the construction of the classification pipelines in a single phase. In NiaAML, only one instance of the stochastic algorithm was needed. However, in NiaAML2, the construction of the pipeline and hyper-parameter optimization was divided into two separate phases, where two instances of nature-inspired algorithms were deployed, one after the other, to cover both steps. The first step covers the composition of the classification of the pipeline, while the second is devoted to hyper-parameter optimization.

2.4. NiaARM

NiaARM is a Python framework [28] that implements the ARM algorithm comprehensively [12], where the ARM is modeled as a single objective, continuous optimization problem. The fitness function in NiaARM is defined as a weighted sum of arbitrary evaluation metrics. One of the most vital points of NiaARM is that it is based on the NiaPy framework [29]; thus, different Nia-s can be used in the optimizer role. According to the knowledge of the authors, NiaARM is the only comprehensive framework for NARM as it is where all NARM steps are implemented, i.e., preprocessing, optimization, and visualization. Other benefits of NiaARM are good documentation and the many examples provided by its maintainers.

3. Proposed Framework: NiaAutoARM

The proposed framework NiaAutoARM was mainly inspired by the meta-heuristic concept, where the higher-level meta-heuristic controls the hyper-parameters of the lower-level heuristic. Both algorithms explore implementations from the NiaAML library (Figure 2).

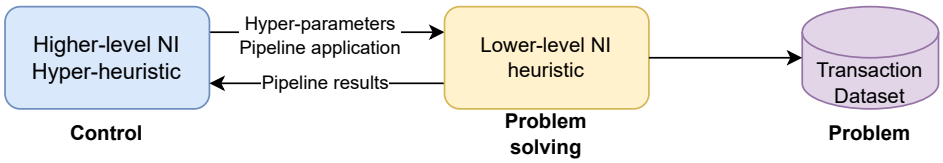


Figure 2. NiaAutoARM framework for automated ARM.

Indeed, the NiaAutoARM higher-level meta-heuristic controls the behavior of the lower-level NI heuristic devoted for problem solving, i.e., ARM. The task of the control meta-heuristic is searching for the optimal hyper-parameter setting of the lower-level heuristic. The hyper-parameter settings direct the ARM pipeline construction. As can be observed from Figure 2, there is two-way communication between the control and the problem heuristics: (1) the pipeline constructed by the higher-level metaheuristic is transmitted to the lower-level heuristic, and (2) the results of the constructed pipeline are transmitted back to the higher-level heuristics that evaluate them in order to specify the best one.

3.1. Higher-Level Meta-Heuristic

Thus, we defined the problem of ARM pipeline construction as a continuous optimization problem. This means that an arbitrary population-based NI meta-heuristic, which works in a continuous search space, can be applied for solving this problem. In the NiaAutoARM higher-level meta-heuristic, each individual in the population of solutions represents one feasible ARM pipeline that is encoded into the representation of an individual:

$$\mathbf{x}_i^{(t)} = \Big\langle \underbrace{x_{i,1}^{(t)}}_{\text{ALGORITHM}} , \underbrace{y_{i,1}^{(t)}, y_{i,2}^{(t)}}_{\text{CONTROL-PARAM}} , \underbrace{p_{i,1}^{(t)}, \ldots, p_{i,P}^{(t)}}_{\text{PREPROCESSING}} , \underbrace{z_{i,1}^{(t)}, \ldots, z_{i,M}^{(t)}}_{\text{METRICS}} , \underbrace{w_{i,1}^{(t)}, \ldots, w_{i,M}^{(t)}}_{\text{METRIC WEIGHTS}} \Big\rangle, \quad (1)$$

where parameter P denotes the number of potential preprocessing methods, and parameter M is the number of potential ARM metrics to be applied. As is evident from Equation (1), each real-valued element of solution in a genotype search space within the interval $[0, 1]$ decodes the particular NiaAutoARM hyper-parameter of the pipeline in a phenotype solution space, as presented in Table 2, and it is determined for each hype-parameter of its corresponding domain values.

Table 2. Hyper-parameters and their domains.

Nr.	Hyper-Parameter	Domain
1	ALGORITHM	{PSO,DE,GA,ILSHADE,LSHADE,jDE}
2	CONTROL-PARAM	{*NP*,*MAXFES*}
3	PREPROCESSING	{MM,ZS,DS,RHC,DK}
4	METRICS	{Supp,Conf,Cover,Amp,Incl,Comp}
5	METRIC-WEIGHTS	$\sum_{i=1}^{M} w_i = 1.0$

As is evident from the table, the ALGORITHM component denotes the specific stochastic NI population-based algorithm, which is chosen from the pool of available algorithms and is typically selected by the user from a NiaPy library to the relative value of $x_{i,1}^{(t)}$ [28]. The CONTROL-PARAM component indicates a magnitude of two algorithm's parameters: the maximum number of individuals NP, and the maximum number of fitness function evaluations $MAXFES$ as a termination condition for the lower-level heuristic. Both values, $y_{i,1}^{(t)}$ and $y_{i,2}^{(t)}$, are mapped in genotype–phenotype mapping to the specific domain of the mentioned parameters, as proposed by Mlakar et al. in [30]. The PREPROCESSING component determines the pool of available preprocessing methods that can be applied to the dataset. On the one hand, if $P = 0$, no preprocessing method is applied; meanwhile, on the

other hand, if $P > 0$ and $p_{i,j}^{(t)} > .5$ for $j = 1, \ldots, P$, then the j-th preprocessing methods from a pool of available ones. For instance, the pool of preprocessing methods in Table 2 consists of the following: "Min_Max normalization" (MM), "Z-Score normalization" (ZS), "Data Squashing" (DS), "Remove Highly Correlated features" (RHC), and "Discretization K-means" (DK). The METRICS component is reserved for the pool of M rule evaluation metrics devoted for estimating the quality of the mined association rules. Additionally, the weights of the metrics are included by the *METRIC_WEIGHTS* component, which weighs the influence of the particular evaluation metric on the appropriate association rule.

Typically, the evaluation metrics illustrated in Table 3 are employed in an NiaAutoARM higher-level meta-heuristic. These metrics were chosen because they reflect both the statistical strength and the practical usefulness of the discovered rules. The framework uses these metrics in the fitness function of the lower-level heuristic. Consequently, this allows for the higher-level meta-heuristic to be directed into the more promising areas of the underlying hyper-parameter's search space while still catering to a dataset-specific context.

Table 3. ARM metrics used for evaluating the mined rules.

Metric	**Evaluation Functions**
Support	$Supp(X \implies Y) = \frac{\lvert t_i \lvert t_i \in X \wedge t_i \in Y \rvert}{N}$
Confidence	$Conf(X \implies Y) = \frac{Supp(X \cup Y)}{Supp(X)}$
Coverage	$Cover(X \implies Y) = \frac{\lvert t_i \lvert t_i \in Y \rvert}{M}$
Amplitude	$Amp(X \implies Y) = \frac{Supp(X \cap Y)}{Supp(X)} - \frac{Supp(Y)}{N}$
Inclusion	$Incl(X \implies Y) = \frac{Supp(X \cap Y)}{Supp(X)}$
Comprehensibility	$Comp(X \implies Y) = \frac{Supp(X \cap Y)}{Supp(Y)}$

Although the quality of the mined association rules is calculated in the lower-level algorithm using the weighted linear combination of the ARM metrics, the higher-level meta-heuristic estimates the quality of the pipeline due to the fairness using the fitness function as follows:

$$f(\mathbf{x}_i^{(t)}) = \frac{\alpha \cdot supp(X \implies Y) + \beta \cdot conf(X \implies Y)}{\alpha + \beta}, \tag{2}$$

where α and β designate the impact of the definite ARM metric on the quality of the solution. It is discarded if no rules are produced or the pipeline fails to decode to the solution space.

The pseudo-code of the proposed NiaAutoARM higher-level meta-heuristic for constructing the classification pipelines is presented in Algorithm 1, from which it can be observed that the higher-level meta-heuristic starts with a random initialization of the population (function INITIALIZE_REAL-VALUED_VECTORS_RANDOMLY in line 1). After evaluation regarding Equation (2) and determining the best solution (function EVAL_AND_SELECT_THE_BEST in Line 2), the evolution cycle was started (Lines 3–15), and it was terminated using function TERMINATION_CONDITION_NOT_MET. Within the evolution cycle, each individual $\mathbf{x}_i$ in the population $\mathbf{P}$ (Lines 4–14) is, at first, modified (function MODIFY_USING_NI_ALGORITHMS in Line 5). This modification results in the production of a trial solution $\mathbf{x}_{trial}$. Next, both the trial and target solutions are mapped to the phenotype solution space, producing the trial *pipeline* and target *cur_pipeline* (and also the current best) solutions (Lines 6 and 7). If the fitness function value of the trial pipeline is better that of the current best evaluated using EVAL function (Line 8), the target solution becomes a trial one (Line 9). Finally, if the trial pipeline is even better than the global best pipeline, *best_pipeline* (Line 11), the global best pipeline becomes the trial pipeline (Line 12).

Algorithm 1 A pseudo-code of the NiaAutoARM higher-level meta-heuristic.

```
P ← INITIALIZE_REAL-VALUED_VECTORS_RANDOMLY(x_i)
best_pipeline ← EVAL_AND_SELECT_THE_BEST(P)
while TERMINATION_CONDITION_NOT_MET do
    for each x_i ∈ P do
        x_trial ← MODIFY_USING_NI_ALGORITHM(x_i)
        pipeline ← CONSTRUCT_PIPELINE(x_trial)
        cur_pipeline ← CONSTRUCT_PIPELINE(x_i)
        if EVAL(pipeline) ≥ EVAL(cur_pipeline) then
            x_i ← x_trial                          ▷ Replace the worse individual
        end if
        if EVAL(pipeline) ≥ EVAL(best_pipeline) then
            best_pipeline ← pipeline
        end if
    end for
end while
return best_pipeline
```

3.2. Lower-Level Heuristics

The NiaAutoARM lower-level heuristic can be any NI algorithm from the Niapy library. The library contains implementations of NI algorithms, which can be used for solving the ARM problem. The lower-level heuristic is controlled via the hyper-parameters, like the algorithm's parameters, preprocessing methods, and orders for constructing the fitness function. It is devoted to solving the problem and returning the corresponding results.

Because the design and implementation of the lower-level heuristic algorithms are described in the corresponding documentation of the Niapy library in detail, we focused only on the construction of the fitness function, which is defined as follows:

$$f(\mathbf{x}) = \sum_{i=1}^{M} w_i \cdot *z_i(\mathbf{x}), \tag{3}$$

where the variable w_i denotes the weight of the corresponding ARM metric, and $*z_i(\mathbf{x})$ is a pointer to the function for calculating the corresponding ARM metric. Please note that the sum of all weights should be one, in other words $\sum_{i=1}^{M} w_i = 1.0$.

3.3. An Example of Genotype–Phenotype Mapping

An example of decoding an ARM pipeline to the solution space is illustrated in Figure 3, where the parameters are set as $P = 1$ and $M = 6$. Let us suppose that the domains of hyper-parameters are given in accordance with Table 2, and the individual in genotype space is defined as that presented in Table 3.

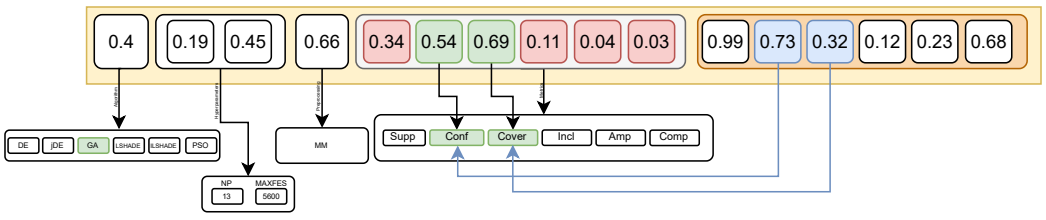


Figure 3. An example of the genotype–phenotype mapping within the ARM pipeline construction.

Then, the higher-level meta-heuristic algorithm transmits the hyper-parameters to the lower-level heuristic algorithm via the following program call:

$$\underbrace{*Alg[\Gamma(x_{i,1})]}_{\text{Algorithm call}}(\underbrace{P, M}_{\text{Param}}, \underbrace{\Gamma(y_{i,1})}_{\text{NP}}, \underbrace{\Gamma(y_{i,2})}_{\text{MAXFES}}, \underbrace{\Gamma(\mathbf{Prep}, \mathbf{p})}_{\text{Preprocess}}, \underbrace{\Gamma(\mathbf{Metr}, \mathbf{z})}_{\text{Metrics}}), \underbrace{\Gamma(\mathbf{Metr}, \mathbf{w})}_{\text{Weights}})), \tag{4}$$

where the function Γ denotes the mapping of genotype values to the phenotype values. Let us mention that the scalar values of 'Algorithm call', NP, and MAXFES are decoded by mapping their values from the interval [0, 1] to the domain values in the solution space. On the other hand, the preprocessing methods and ARM metrics represent sets, where each member is taken from the sets **Prep** and **Metr** according to the probability 0.5 based on the values of the vectors **p** and **z**. Interestingly, the weight vector can be treated either statically or adaptively with respect to setting the parameter *weight_adaptation*. When the parameter is set as **true**, the adapted values from vector **w** indicate an impact of a definite ARM metric in the linear combination of ARM metrics within the fitness function. If this parameter is set to **false**, the values are fixed to the value 1.0.

As a result of the pipeline application, the support and confidence of the best association rule are returned to the higher-level meta-heuristic.

4. Results

The primary goal of the experiments was to evaluate whether NiaAutoARM can find an optimal pipeline for solving various ARM problems automatically. A series of experiments utilized the most common ARM publicly available datasets to justify this hypothesis.

The UCI ML datasets, listed in Table 4, were used for evaluating the performance of the proposed method [31]. Each database is characterized by the number of transactions, number of attributes, and their types, which can be either categorical (discrete) or numerical (real). These datasets were selected since they vary in terms of the number of transactions, the types of attributes, and the total number of attributes they contain. They are also commonly used within the ARM literature [30], making them appropriate benchmarks for evaluating the generalizability of the proposed NiaAutoARM framework. It is worth mentioning that the proposed method automatically determines the most suitable preprocessing algorithm as a part of its process; therefore, no manual preprocessing was applied to the original datasets.

Table 4. The evaluation datasets used in the experiments.

Dataset	Nr. of Inst.	Nr. of Attr.	Attr. Type [D/N]
Abalone	4177	9	DN
Balance scale	625	5	DN
Basketball	96	5	N
Bolts	40	8	N
Buying	100	40	N
German	1000	20	DN
House	22,784	17	N
Ionosphere	351	35	DN
Quake	2178	4	N
Wine	178	14	N

In our experiments, we used two NI algorithms for the ARM pipeline optimization as the higher-level meta-heuristics, namely the DE and the PSO. Both have appeared in several recent studies in the ARM domain in either original or hybridized form [32–34]. To ensure a fair comparison, the most important parameters of both algorithms were set equally. Therefore, the population size was set to $NP = 30$, and the maximum number of fitness function evaluations was set to $MAXFES = 1000$ (i.e., the number of pipeline evaluations), following the parameter ranges used in prior AutoML and NARM studies, and computational feasibility was then balanced with optimization performance. These parameters were selected empirically after preliminary tuning runs, ensuring that the optimization had sufficient search power without introducing prohibitive computational costs. All other

parameters of the NI algorithms (i.e., GA, DE, PSO, jDE, LSHADE, and ILSHADE) were left at their default parameter settings, i.e., as implemented in the NiaPy framework, to maintain fairness across comparisons. In all the experiments, the lower-level optimization algorithms for ARM were similarly selected as in the example illustrated in Table 2.

Each experimental run produced the best pipeline for a combination of the specific dataset and algorithm. Considering the stochastic nature of the DE and PSO algorithms, the reported results are the average fitness function values of the best obtained pipelines over 30 independent runs.

The quality of the constructed pipeline was evaluated regarding Equation (2) in the higher-level meta-heuristic algorithm, while the fitness function in the lower-level heuristic algorithm was calculated as a weighted sum of the ARM metrics decoded from the corresponding individual by the NiaAutoARM framework.

4.1. Experimental Evaluation

The following experiments were conducted for analyzing the newly proposed NiaAutoARM thoroughly:

- Baseline ARM pipeline optimization that allowed for just one preprocessing component and a disabled ARM metric weight adaptation.
- Studied the influence of adapting the ARM metric weights on the quality of the ARM pipeline construction.
- Studied the influence of selecting more preprocessing components on the quality of the ARM pipeline construction.
- Conducted a comparison with the VARDE state-of-the-art algorithm.

In the remainder of this section, all of the experimental results are presented in detail, showcasing the usefulness and efficiency of the proposed method.

4.1.1. Baseline ARM Pipeline Construction

The purpose of the first experiment was to establish a foundational comparison for all the subsequent experiments. In this experiment, no ARM metric weight adaptation was applied, ensuring that the generated pipelines operated in their default configurations. Additionally, each generated pipeline was restricted to a single preprocessing method, eliminating the variability introduced by multiple preprocessing components.

All the results for this experiment are reported numerically in Tables 5 and 6, and they are graphically represented in Figure 4 for the different PSO and DE higher-level meta-heuristics, respectively. The mentioned tables are structured as follows: The column 'Preprocessing method' denotes the frequency of the preprocessing algorithms in the best obtained pipelines over all 30 runs. The column 'Hyper-parameters' is used for reporting the average obtained population sizes (NP) and the maximum function evaluations ($MAXFES$) for the best obtained ARM pipelines. Lastly, the column 'Metrics & Weights' are used for reporting the average values of each used ARM evaluation metric. The number in the subscript denotes the number of pipelines in which a specific metric was used. Since, in the baseline experiment, no ARM metric weight adaptation was used, all the values are equal to 1. Each row in the tables refer to one experimental dataset.

Table 5. Results for the PSO algorithm, with $P = 1$, without ARM metric weight adaptation.

Dataset	Preprocessing Method						Hyper-Parameters		Metrics & Weights					
	MM	*ZS*	*DS*	*RHC*	*KM*	*N* [a]	*NP*	*MAXFES*	**Supp**	**Conf**	**Cover**	**Amp**	**Incl**	**Comp**
Abalone	0.27	0.07	-	0.20	-	0.47	11.7 ± 5.2	9656.2 ± 796.4	$1.00 \pm 0.00_{25}$	$1.00 \pm 0.00_{23}$	$1.00 \pm 0.00_{19}$	-	$1.00 \pm 0.00_{22}$	$1.00 \pm 0.00_{15}$
Balance scale	0.30	0.07	-	0.10	-	0.53	17.6 ± 9.0	8370.3 ± 2598.6	$1.00 \pm 0.00_{24}$	$1.00 \pm 0.00_{28}$	$1.00 \pm 0.00_{8}$	-	$1.00 \pm 0.00_{19}$	$1.00 \pm 0.00_{16}$
Basketball	0.47	-	-	-	-	0.53	11.7 ± 4.8	9851.2 ± 543.7	$1.00 \pm 0.00_{25}$	$1.00 \pm 0.00_{26}$	$1.00 \pm 0.00_{18}$	-	$1.00 \pm 0.00_{29}$	$1.00 \pm 0.00_{12}$
Bolts	0.23	0.10	-	0.07	-	0.60	13.2 ± 6.6	8946.9 ± 2189.4	$1.00 \pm 0.00_{23}$	$1.00 \pm 0.00_{25}$	$1.00 \pm 0.00_{16}$	$1.00 \pm 0.00_{2}$	$1.00 \pm 0.00_{26}$	$1.00 \pm 0.00_{8}$
Buying	0.23	0.20	-	-	0.07	0.50	17.5 ± 8.3	9039.1 ± 1742.6	$1.00 \pm 0.00_{25}$	$1.00 \pm 0.00_{18}$	$1.00 \pm 0.00_{11}$	$1.00 \pm 0.00_{3}$	$1.00 \pm 0.00_{2}$	$1.00 \pm 0.00_{8}$
German	-	-	0.97	-	-	0.03	20.1 ± 7.2	5871.8 ± 3046.2	$1.00 \pm 0.00_{16}$	$1.00 \pm 0.00_{20}$	$1.00 \pm 0.00_{12}$	$1.00 \pm 0.00_{12}$	$1.00 \pm 0.00_{15}$	$1.00 \pm 0.00_{17}$
House16	0.30	0.13	-	0.10	-	0.47	15.5 ± 8.3	8642.5 ± 2038.8	$1.00 \pm 0.00_{25}$	$1.00 \pm 0.00_{21}$	$1.00 \pm 0.00_{22}$	$1.00 \pm 0.00_{3}$	$1.00 \pm 0.00_{7}$	$1.00 \pm 0.00_{12}$
Ionosphere	0.17	-	-	0.03	0.10	0.70	18.3 ± 9.2	8600.9 ± 2393.7	$1.00 \pm 0.00_{28}$	$1.00 \pm 0.00_{19}$	$1.00 \pm 0.00_{7}$	$1.00 \pm 0.00_{1}$	$1.00 \pm 0.00_{3}$	$1.00 \pm 0.00_{2}$
Quake	0.30	0.03	-	0.07	-	0.60	11.4 ± 4.3	9622.0 ± 1074.7	$1.00 \pm 0.00_{27}$	$1.00 \pm 0.00_{24}$	$1.00 \pm 0.00_{17}$	$1.00 \pm 0.00_{1}$	$1.00 \pm 0.00_{17}$	$1.00 \pm 0.00_{18}$
Wine	0.23	0.03	-	0.10	-	0.63	12.6 ± 6.3	9471.0 ± 1301.1	$1.00 \pm 0.00_{24}$	$1.00 \pm 0.00_{25}$	$1.00 \pm 0.00_{20}$	$1.00 \pm 0.00_{1}$	$1.00 \pm 0.00_{14}$	$1.00 \pm 0.00_{18}$
							14.94 ± 3.06	8807.19 ± 1089.31	$1.00 \pm 0.00_{24.20 \pm 3.06}$	$1.00 \pm 0.00_{22.90 \pm 3.11}$	$1.00 \pm 0.00_{15.00 \pm 4.92}$	$1.00 \pm 0.00_{3.29 \pm 3.65}$	$1.00 \pm 0.00_{15.40 \pm 8.73}$	$1.00 \pm 0.00_{12.60 \pm 5.00}$

[a] No preprocessing of the dataset.

Table 6. Results for DE algorithm, with $P = 1$, without ARM metrics weight adaptation.

Dataset	Preprocessing Method						Hyper-Parameters		Metrics & Weights					
	MM	*ZS*	*DS*	*RHC*	*KM*	*N* [a]	*NP*	*MAXFES*	**Supp**	**Conf**	**Cover**	**Amp**	**Incl**	**Comp**
Abalone	0.43	0.10	-	0.20	-	0.27	13.2 ± 5.4	9360.9 ± 1150.8	$1.00 \pm 0.00_{27}$	$1.00 \pm 0.00_{22}$	$1.00 \pm 0.00_{20}$	-	$1.00 \pm 0.00_{23}$	$1.00 \pm 0.00_{8}$
Balance scale	0.33	0.07	-	0.20	-	0.40	14.8 ± 7.4	8216.3 ± 2234.2	$1.00 \pm 0.00_{23}$	$1.00 \pm 0.00_{28}$	$1.00 \pm 0.00_{7}$	-	$1.00 \pm 0.00_{23}$	$1.00 \pm 0.00_{20}$
Basketball	0.47	0.17	-	0.13	-	0.23	12.9 ± 3.7	9160.8 ± 1468.8	$1.00 \pm 0.00_{22}$	$1.00 \pm 0.00_{25}$	$1.00 \pm 0.00_{17}$	-	$1.00 \pm 0.00_{28}$	$1.00 \pm 0.00_{9}$
Bolts	0.27	0.13	-	0.10	-	0.50	15.4 ± 6.3	9107.0 ± 1343.4	$1.00 \pm 0.00_{25}$	$1.00 \pm 0.00_{21}$	$1.00 \pm 0.00_{15}$	-	$1.00 \pm 0.00_{23}$	$1.00 \pm 0.00_{10}$
Buying	0.33	0.17	-	0.10	0.10	0.30	13.8 ± 6.6	8793.3 ± 1813.5	$1.00 \pm 0.00_{28}$	$1.00 \pm 0.00_{13}$	$1.00 \pm 0.00_{6}$	-	$1.00 \pm 0.00_{1}$	-
German	-	-	1.00	-	-	-	18.7 ± 7.4	7992.1 ± 2403.1	$1.00 \pm 0.00_{16}$	$1.00 \pm 0.00_{13}$	$1.00 \pm 0.00_{15}$	$1.00 \pm 0.00_{19}$	$1.00 \pm 0.00_{15}$	$1.00 \pm 0.00_{13}$
House16	0.50	0.20	-	0.03	-	0.27	14.2 ± 6.4	8751.8 ± 1865.9	$1.00 \pm 0.00_{23}$	$1.00 \pm 0.00_{25}$	$1.00 \pm 0.00_{23}$	-	$1.00 \pm 0.00_{8}$	$1.00 \pm 0.00_{17}$
Ionosphere	0.30	-	-	0.10	0.10	0.50	15.1 ± 6.6	8769.9 ± 2080.5	$1.00 \pm 0.00_{30}$	$1.00 \pm 0.00_{19}$	$1.00 \pm 0.00_{4}$	-	-	$1.00 \pm 0.00_{2}$
Quake	0.13	0.23	-	0.17	-	0.47	11.1 ± 2.9	9406.5 ± 899.3	$1.00 \pm 0.00_{24}$	$1.00 \pm 0.00_{18}$	$1.00 \pm 0.00_{18}$	-	$1.00 \pm 0.00_{21}$	$1.00 \pm 0.00_{15}$
Wine	0.27	0.07	-	0.13	-	0.53	11.8 ± 2.8	9506.5 ± 827.2	$1.00 \pm 0.00_{27}$	$1.00 \pm 0.00_{28}$	$1.00 \pm 0.00_{15}$	-	$1.00 \pm 0.00_{21}$	$1.00 \pm 0.00_{10}$
							14.09 ± 2.03	8906 ± 478.47	$1.00 \pm 0.00_{24.50 \pm 3.72}$	$1.00 \pm 0.00_{21.20 \pm 5.21}$	$1.00 \pm 0.00_{14.00 \pm 5.98}$	$1.00 \pm 0.00_{19.00 \pm 0.00}$	$1.00 \pm 0.00_{18.11 \pm 8.10}$	$1.00 \pm 0.00_{11.56 \pm 5.06}$

[a] No preprocessing of the dataset.

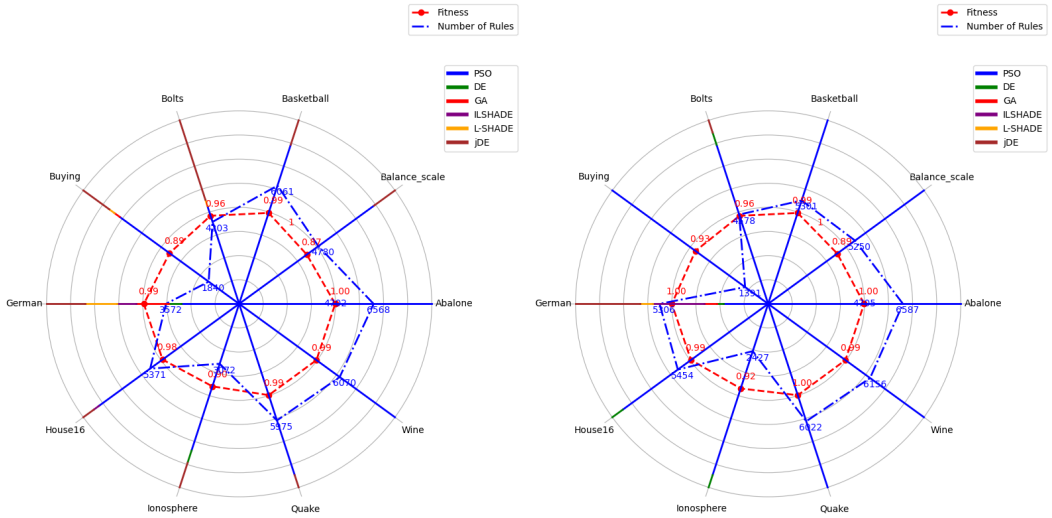


Figure 4. Results for the baseline ARM pipeline optimization, where the averages of the best pipelines in terms of fitness values, number of generated rules, and the used lower-level heuristic algorithms are reported. (**a**) Results for the PSO higher-level meta-heuristic algorithm without ARM metric weight adaptation and just one preprocessing method. (**b**) Results for the DE higher-level meta-heuristic algorithm without ARM metric weight adaptation and just one preprocessing method.

Figure 4 presents the obtained average fitness values along with the average number of rules generated by the best obtained pipelines. Additionally, the frequencies of the lower-level heuristic algorithms are depicted. The fitness values are marked with blue dash/dotted lines, whereas the number of rules is marked with a red dotted line. The frequencies of the lower-level heuristic algorithms are presented as different colored lines from the center of the graph, and they are outward to each dataset.

The results in Table 5, developed by the PSO higher-level meta-heuristic algorithm, justified that the preprocessing methods, like MM, ZS, and RHC, were selected more frequently. Meanwhile, in general, 'No preprocessing' was selected in most of the pipelines, regardless of the dataset. The ARM metrics support, confidence, and coverage appeared consistently across most datasets. Notably, the support and confidence were present in nearly all the pipelines for datasets like Abalone, Balance scale, and Basketball, indicating that these metrics are essential for the underlying optimization process. Metrics like amplification, which are used less frequently, are absent in many datasets, suggesting that the current algorithm configuration does not prioritize such metrics. The hyper-parameters *NP* and *MAXFES* varied depending on the dataset, influencing the ARM pipeline optimization process.

Table 6 shows the results for the DE higher-level meta-heuristic algorithm. Similar to the results of the PSO, key ARM metrics, like support, confidence, and coverage, are found consistently in many of the generated pipelines. However, there are subtle differences in the distribution of these metrics across the pipelines. For instance, the metric amplitude was selected just for the dataset German. Regarding the preprocessing methods and hyper-parameters, a similar distribution can be found as in the results of the PSO algorithm.

The graphical results showcase that both DE and PSO obtained similar results regarding the fitness value. The number of rules was slightly dispersed, although no big deviations were detected. The key differences were in the selection of the lower-level heuristic algorithm. For the majority of datasets, the PSO and jDE algorithms were selected more often as the lower-level heuristic algorithms. This was also true for both the higher-level meta-heuristic algorithms. Other used algorithms, such as GA, DE, ILSHADE and LSHADE, were selected rarely as the lower-level heuristic, probably due to their complexity or their lack of it.

To summarize the results of the baseline experiment, we can conclude that the best results were obtained when either no preprocessing was applied or when MM was used on the dataset. The *NP* parameter seemed to be higher for more complex datasets (i.e., more attributes), such as Buying, German, House16 and Ionosphere, while it remained lower for the others, which were less demanding. Regarding the selection of specific ARM evaluation metrics, it seems that both algorithms focused on the more common ones, i.e., those usually used in Evolutionary ARM [30]. Overall, these results indicate the DE and PSO algorithms' robustness as a higher-level meta-heuristic while reinforcing the potential benefits of further exploration into ARM metric weight adaptation and diversified preprocessing strategies.

Please note that all the subsequent results are reported in the same manner.

4.1.2. Influence of the ARM Metric Weights Adaption on the Quality of ARM Pipeline Construction

The purpose of this experiment was to analyze the impact of selecting ARM metric weight adaptation on the performance of the ARM pipeline construction. The ARM metric weights play a crucial role in guiding the optimization process as they influence the evaluation and selection of the candidate association rules. By incorporating the ARM weight adaptation mechanism, the pipeline can adjust the importance of ARM metrics dynamically, such as support, confidence, coverage, and others, and it is tailored to the characteristics of the dataset. This experiment aimed to determine whether adapting these weights improved the quality of the discovered rules; therefore, they are reflected in the pipeline's metrics. The results were compared to the baseline configuration, where no weight adaptation was applied.

Tables 7 and 8 present the results obtained by the PSO and DE higher-level meta-heuristic algorithms, respectively. A similar selection of the preprocessing methods as in the last experiment was also employed in this experiment, where the preprocessing methods MM, ZS, and None were applied the most frequently. The hyper-parameters yielded higher values for the harder datasets. Considering the ARM metrics, the support and confidence still arose with high weight values in the majority of the pipelines, whereas the ARM metrics, like amplification or comprehensibility, were utilized less with lower weights.

From the results in Figure 5, we can deduce similar conclusions as from those in the baseline experiment, but the ARM metric weight adaptation provided slightly higher fitness values than those achieved in the last experiment. Although these differences were not significantly different to those according to the Wilcox test (p-value = 0.41), they still offered overall better ARM pipelines for the majority of datasets.

Table 7. Results for the outer algorithm PSO with ARM metric weight adaptation and just one preprocessing method.

Dataset	Preprocessing Method						Hyper-Parameters		Metrics & Weights					
	MM	*ZS*	*DS*	*RHC*	*KM*	*N* [a]	*NP*	*MAXFES*	Supp	Conf	Cover	Amp	Incl	Comp
Abalone	0.40	0.07	-	0.10	-	0.43	11.6 ± 5.0	9448.7 ± 1608.1	$0.89 \pm 0.23_{23}$	$0.81 \pm 0.29_{25}$	$0.67 \pm 0.33_{17}$	-	$0.63 \pm 0.35_{23}$	$0.41 \pm 0.29_{11}$
Balance scale	0.40	0.10	-	0.10	-	0.40	16.6 ± 8.8	6563.6 ± 3507.9	$0.56 \pm 0.39_{23}$	$0.77 \pm 0.30_{23}$	$0.62 \pm 0.25_{8}$	-	$0.66 \pm 0.31_{14}$	$0.74 \pm 0.27_{15}$
Basketball	0.63	-	-	0.07	-	0.30	14.8 ± 7.9	9285.8 ± 1723.5	$0.83 \pm 0.28_{29}$	$0.84 \pm 0.22_{24}$	$0.63 \pm 0.36_{10}$	-	$0.76 \pm 0.34_{22}$	$0.88 \pm 0.24_{9}$
Bolts	0.23	0.07	-	0.03	-	0.67	10.9 ± 3.6	8642.9 ± 2285.0	$0.86 \pm 0.21_{19}$	$0.68 \pm 0.32_{19}$	$0.75 \pm 0.28_{15}$	-	$0.84 \pm 0.27_{25}$	$0.98 \pm 0.04_{5}$
Buying	0.43	0.03	-	0.13	0.03	0.37	17.6 ± 8.4	8695.0 ± 2184.4	$0.75 \pm 0.31_{27}$	$0.83 \pm 0.33_{13}$	$0.61 \pm 0.40_{6}$	$1.00 \pm 0.00_{1}$	$0.98 \pm 0.00_{1}$	$0.99 \pm 0.01_{2}$
German	-	-	1.00	-	-	-	20.4 ± 7.0	5921.3 ± 2437.6	$0.53 \pm 0.28_{13}$	$0.60 \pm 0.35_{14}$	$0.47 \pm 0.36_{15}$	$0.62 \pm 0.36_{11}$	$0.66 \pm 0.35_{15}$	$0.61 \pm 0.29_{19}$
House16	0.30	0.03	-	0.03	-	0.63	13.7 ± 6.5	9141.0 ± 1947.1	$0.79 \pm 0.28_{24}$	$0.88 \pm 0.20_{18}$	$0.62 \pm 0.36_{14}$	$0.03 \pm 0.03_{4}$	$0.67 \pm 0.46_{6}$	$0.41 \pm 0.29_{10}$
Ionosphere	0.23	-	-	0.13	0.03	0.60	13.3 ± 6.1	8799.0 ± 2451.1	$0.77 \pm 0.35_{29}$	$0.68 \pm 0.34_{20}$	$0.73 \pm 0.22_{3}$	$0.03 \pm 0.00_{1}$	-	$0.34 \pm 0.20_{2}$
Quake	0.40	-	-	0.13	-	0.47	12.1 ± 5.9	9941.2 ± 239.3	$0.80 \pm 0.29_{25}$	$0.74 \pm 0.34_{16}$	$0.83 \pm 0.21_{15}$	-	$0.72 \pm 0.32_{17}$	$0.87 \pm 0.29_{13}$
Wine	0.37	0.07	-	0.03	-	0.53	10.8 ± 2.0	9454.8 ± 1539.8	$0.85 \pm 0.24_{23}$	$0.88 \pm 0.26_{25}$	$0.74 \pm 0.30_{10}$	-	$0.73 \pm 0.33_{24}$	$0.70 \pm 0.23_{6}$
							14.16 ± 3.00	8589.34 ± 1240.34	$0.76 \pm 0.12_{23.50 \pm 4.54}$	$0.77 \pm 0.09_{19.70 \pm 4.24}$	$0.67 \pm 0.10_{11.30 \pm 4.38}$	$0.42 \pm 0.41_{4.25 \pm 4.09}$	$0.74 \pm 0.10_{16.33 \pm 7.89}$	$0.69 \pm 0.23_{9.20 \pm 5.29}$

[a] No preprocessing of the dataset.

Table 8. Results for the outer algorithm DE with ARM metric weight adaptation and just one preprocessing method.

Dataset	Preprocessing Method						Hyper-Parameters		Metrics & Weights					
	MM	*ZS*	*DS*	*RHC*	*KM*	*N* [a]	*NP*	*MAXFES*	Supp	Conf	Cover	Amp	Incl	Comp
Abalone	0.60	0.03	-	0.10	-	0.27	12.1 ± 3.9	8808.1 ± 1628.4	$0.78 \pm 0.29_{24}$	$0.84 \pm 0.18_{20}$	$0.67 \pm 0.32_{19}$	-	$0.63 \pm 0.33_{26}$	$0.65 \pm 0.30_{11}$
Balance scale	0.37	0.10	-	0.03	-	0.50	19.3 ± 8.0	8727.0 ± 1780.1	$0.66 \pm 0.30_{25}$	$0.80 \pm 0.19_{20}$	$0.66 \pm 0.29_{9}$	-	$0.85 \pm 0.26_{15}$	$0.66 \pm 0.36_{15}$
Basketball	0.37	0.17	-	0.27	-	0.20	13.0 ± 5.1	8858.1 ± 1383.7	$0.70 \pm 0.31_{22}$	$0.85 \pm 0.21_{22}$	$0.63 \pm 0.26_{11}$	-	$0.69 \pm 0.34_{27}$	$0.56 \pm 0.33_{9}$
Bolts	0.17	0.20	-	0.07	-	0.57	16.1 ± 7.7	8495.1 ± 2678.4	$0.67 \pm 0.28_{20}$	$0.59 \pm 0.34_{23}$	$0.69 \pm 0.36_{16}$	$0.25 \pm 0.00_{1}$	$0.59 \pm 0.27_{23}$	$0.78 \pm 0.30_{12}$
Buying	0.27	0.13	-	0.10	0.07	0.43	16.8 ± 6.9	9124.0 ± 1631.4	$0.73 \pm 0.33_{30}$	$0.68 \pm 0.33_{14}$	$0.69 \pm 0.34_{3}$	-	$0.10 \pm 0.00_{1}$	$0.49 \pm 0.44_{2}$
German	-	-	1.00	-	-	-	19.4 ± 7.6	5848.0 ± 3014.7	$0.87 \pm 0.17_{12}$	$0.88 \pm 0.17_{10}$	$0.57 \pm 0.30_{10}$	$0.61 \pm 0.37_{11}$	$0.63 \pm 0.27_{12}$	$0.59 \pm 0.28_{10}$
House16	0.33	0.07	-	0.23	0.03	0.33	15.4 ± 6.7	8682.6 ± 1810.6	$0.64 \pm 0.30_{23}$	$0.76 \pm 0.31_{16}$	$0.67 \pm 0.32_{20}$	-	$0.41 \pm 0.37_{9}$	$0.60 \pm 0.33_{18}$
Ionosphere	0.40	-	-	0.07	0.13	0.40	14.7 ± 6.4	8727.3 ± 1754.6	$0.72 \pm 0.27_{28}$	$0.73 \pm 0.32_{14}$	$0.79 \pm 0.23_{6}$	-	$0.46 \pm 0.40_{3}$	$0.48 \pm 0.17_{3}$
Quake	0.33	0.10	-	0.17	-	0.40	11.1 ± 2.6	9471.8 ± 1115.7	$0.66 \pm 0.32_{26}$	$0.68 \pm 0.25_{18}$	$0.69 \pm 0.30_{18}$	-	$0.74 \pm 0.30_{13}$	$0.59 \pm 0.33_{17}$
Wine	0.40	0.10	-	0.10	-	0.40	11.8 ± 4.0	9293.9 ± 1261.7	$0.75 \pm 0.26_{24}$	$0.77 \pm 0.24_{19}$	$0.54 \pm 0.35_{9}$	-	$0.61 \pm 0.34_{20}$	$0.55 \pm 0.34_{11}$
							14.95 ± 2.84	8603 ± 961.74	$0.72 \pm 0.07_{23.40 \pm 4.67}$	$0.76 \pm 0.09_{17.60 \pm 3.85}$	$0.66 \pm 0.07_{12.10 \pm 5.52}$	$0.43 \pm 0.18_{6.00 \pm 5.00}$	$0.57 \pm 0.20_{14.90 \pm 8.62}$	$0.60 \pm 0.08_{10.80 \pm 5.02}$

[a] No preprocessing of the dataset.

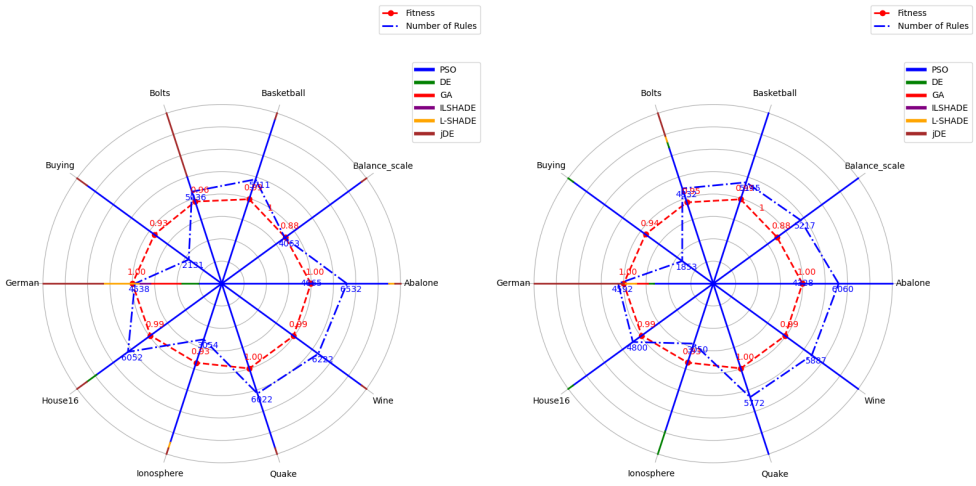


Figure 5. Results for the ARM pipeline construction using ARM metric weight adaptation, where the averages of the best pipelines in terms of fitness values, number of generated rules, and the used inner optimization algorithms are presented. (**a**) Results for the PSO higher-level meta-heuristic algorithm with ARM metric weight adaptation and just one preprocessing method. (**b**) Results for the DE higher-level meta-heuristic algorithm DE with ARM metric weight adaptation and just one preprocessing method.

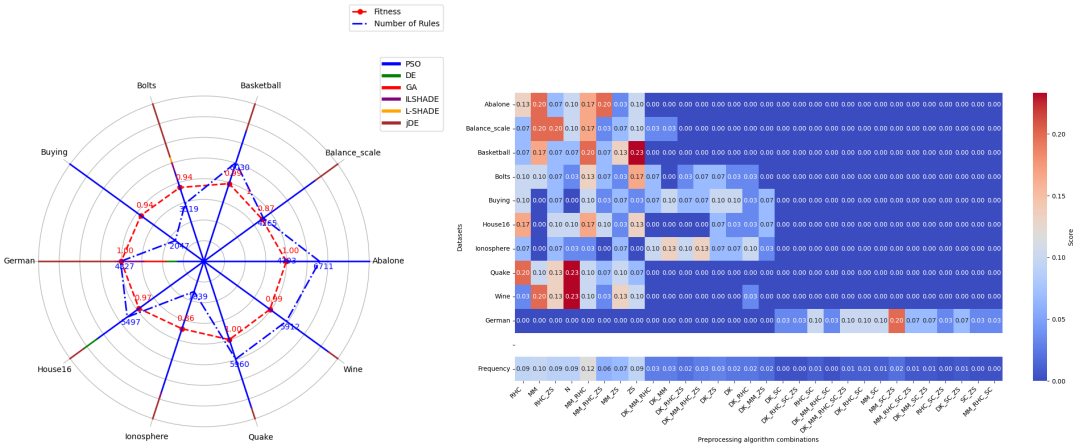
4.1.3. Influence of Selecting More Preprocessing Methods on the Quality of ARM Pipeline Construction

The parameter P controls the number of preprocessing components allowed in an ARM pipeline. By increasing P beyond 1, we introduce the possibility of combining multiple preprocessing dataset methods, which can, potentially, enhance the quality of the generated rules. This increased flexibility enables the pipeline to address complex data characteristics (e.g., variability in feature scaling, noise reduction, or dimensionality reduction) more effectively. However, this increased complexity also poses challenges, including higher computational costs and a broader search space to be discovered by the inner optimization algorithms. In this section, we analyze the impact of setting the parameter as $P > 1$ on the quality of the ARM pipelines, focusing on the resulting ARM metrics and their corresponding weights, as well as onthe computational trade offs for the experimental datasets. The results of the selected preprocessing algorithms are depicted as heatmaps of all the possible combinations. The results in Tables 9 and 10 suggest that the support and confidence ARM metrics were again included heavily in the calculation of the fitness function, achieving high values in the majority of the pipelines for both the higher-level meta-heuristic algorithms. The coverage and inclusion ARM metrics were also involved in many pipelines, although their average weights were smaller. There was no notable difference in the selected hyper-parameters when compared to the previous two experiments.

Since this experiment included selecting more preprocessing methods, their selection frequency is reported in terms of heatmaps in Figure 6b for the PSO meta-heuristic algorithm and Figure 7b for the DE meta-heuristic algorithm. The selection of the preprocessing method varied, of course, if we observed a particular dataset, as the data were distributed differently. However, if we look at the overall selection process, specific combinations stand out. For the PSO algorithm, the most frequent combinations were $\{\text{MM}, \text{RHC}\}$ and MM, while, for the DE meta-heuristic algorithm, it was $\{\text{RHC}, \text{ZS}\}$, $\{\text{MM}, \text{RHC}, \text{ZS}\}$, and RHC. The MM preprocessing method was frequently selected across all datasets in both algorithms, likely due to its ability to normalize feature values to a standard range (which enhances the ability of the inner optimization algorithm to explore the search space

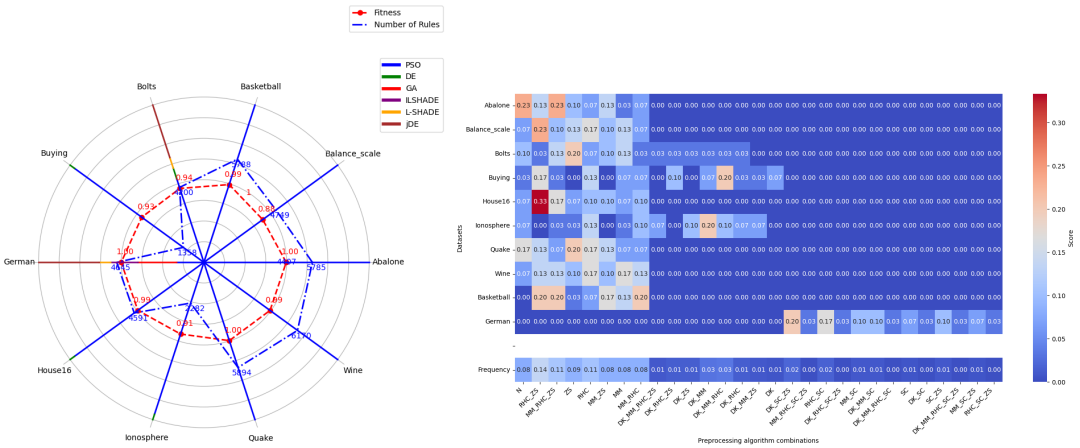
more efficiently). This preprocessing method ensures that all features equally contribute during the optimization process, mitigating the influence of features with larger numeric ranges and facilitating better rule generation.

Figures 6a and 7a illustrate the fitness values and the number of generated rules for the PSO and DE meta-heuristic algorithms. The DE meta-heuristic algorithm produced ARM pipelines with slightly higher fitness values, while the PSO meta-heuristic algorithm generated a greater number of rules. It is also evident that the PSO algorithm was selected the most as the lower-level heuristic algorithm in both scenarios.



(**a**) (**b**)

Figure 6. Results of the PSO ARM pipeline optimization using ARM metric weight adaptation and selecting more preprocessing components, where the averages of the best pipelines in terms of fitness values, number of generated rules, and the used lower-level heuristic algorithms and preprocessing methods are reported. (**a**) Results of the preprocessing components for the PSO higher-level meta-heuristic algorithm with ARM metric weight adaptation and more preprocessing methods. (**b**) Heatmap of the preprocessing components for the PSO higher-level meta-heuristic algorithm with ARM metric weight adaptation and more preprocessing methods.



(**a**) (**b**)

Figure 7. Results for the DE ARM pipeline optimization using ARM metric weight adaptation and selecting more preprocessing methods, where the averages of the best pipelines in terms of fitness values, number of generated rules, and the used inner optimization algorithms and preprocessing methods are reported. (**a**) Results of the preprocessing components for the DE higher-level meta-heuristic algorithm with ARM metric weight adaptation and more preprocessing methods. (**b**) Heatmap of the preprocessing components for the higher-level meta-heuristic algorithm with ARM metric weight adaptation and more preprocessing methods.

Table 9. Results for the PSO higher-level meta-heuristic algorithm with ARM metric weight adaptation and selecting more preprocessing methods.

Dataset	Preprocessing Method						Hyper-Parameters		Metrics & Weights					
	MM	*ZS*	*DS*	*RHC*	*KM*	*N* [a]	*NP*	*MAXFES*	Supp	Conf	Cover	Amp	Incl	Comp
Abalone	-	-	-	-	-	-	15.6 ± 8.4	9570.9 ± 1477.9	$0.83 \pm 0.30_{17}$	$0.82 \pm 0.24_{21}$	$0.83 \pm 0.28_{19}$	-	$0.65 \pm 0.40_{17}$	$0.76 \pm 0.36_{16}$
Balance scale	-	-	-	-	-	-	14.8 ± 7.6	7869.1 ± 2986.2	$0.69 \pm 0.37_{23}$	$0.74 \pm 0.29_{24}$	$0.48 \pm 0.35_{10}$	-	$0.82 \pm 0.27_{16}$	$0.70 \pm 0.28_{14}$
Basketball	-	-	-	-	-	-	13.4 ± 6.5	9700.8 ± 907.4	$0.73 \pm 0.34_{24}$	$0.83 \pm 0.30_{19}$	$0.88 \pm 0.25_{11}$	-	$0.66 \pm 0.38_{21}$	$0.76 \pm 0.33_{10}$
Bolts	-	-	-	-	-	-	15.7 ± 7.9	8379.6 ± 2697.1	$0.79 \pm 0.29_{25}$	$0.86 \pm 0.24_{18}$	$0.82 \pm 0.24_{14}$	-	$0.76 \pm 0.28_{21}$	$0.79 \pm 0.27_{8}$
Buying	-	-	-	-	-	-	19.3 ± 8.7	9364.9 ± 1770.2	$0.80 \pm 0.29_{26}$	$0.88 \pm 0.21_{13}$	$0.79 \pm 0.32_{7}$	-	-	$0.66 \pm 0.23_{2}$
German	-	-	-	-	-	-	19.4 ± 6.5	6091.4 ± 3015.5	$0.61 \pm 0.29_{13}$	$0.67 \pm 0.30_{14}$	$0.51 \pm 0.38_{13}$	$0.76 \pm 0.28_{14}$	$0.66 \pm 0.31_{18}$	$0.54 \pm 0.33_{14}$
House16	-	-	-	-	-	-	16.0 ± 8.3	8451.8 ± 2975.0	$0.71 \pm 0.33_{24}$	$0.80 \pm 0.29_{22}$	$0.65 \pm 0.32_{17}$	$0.01 \pm 0.00_{2}$	$0.48 \pm 0.37_{10}$	$0.52 \pm 0.42_{10}$
Ionosphere	-	-	-	-	-	-	21.3 ± 8.2	6776.0 ± 3324.5	$0.64 \pm 0.41_{23}$	$0.82 \pm 0.32_{14}$	$0.25 \pm 0.20_{5}$	$0.76 \pm 0.23_{5}$	$0.81 \pm 0.16_{3}$	$0.59 \pm 0.41_{2}$
Quake	-	-	-	-	-	-	11.6 ± 4.8	9585.9 ± 899.3	$0.91 \pm 0.20_{19}$	$0.87 \pm 0.24_{18}$	$0.64 \pm 0.40_{15}$	-	$0.68 \pm 0.36_{13}$	$0.71 \pm 0.33_{16}$
Wine	-	-	-	-	-	-	14.4 ± 7.3	8685.9 ± 2585.8	$0.82 \pm 0.31_{24}$	$0.86 \pm 0.24_{21}$	$0.69 \pm 0.30_{18}$	$0.33 \pm 0.29_{2}$	$0.53 \pm 0.38_{17}$	$0.74 \pm 0.31_{13}$
							16.15 2.86	8447.63 ± 1170.97	$0.75 \pm 0.09_{21.80 \pm 3.92}$	$0.82 \pm 0.06_{18.40 \pm 3.56}$	$0.65 \pm 0.19_{12.90 \pm 4.41}$	$0.47 \pm 0.32_{5.75 \pm 4.92}$	$0.67 \pm 0.11_{15.11 \pm 5.40}$	$0.68 \pm 0.09_{10.50 \pm 4.92}$

[a] No preprocessing of the dataset.

Table 10. Results for the DE higher-level meta-heuristic algorithm with ARM metric weight adaptation and selecting more preprocessing methods.

Dataset	Preprocessing Method						Hyper-Parameters		Metrics & Weights					
	MM	*ZS*	*DS*	*RHC*	*KM*	*N* [a]	*NP*	*MAXFES*	Supp	Conf	Cover	Amp	Incl	Comp
Abalone	-	-	-	-	-	-	11.6 ± 4.2	8989.0 ± 1818.6	$0.73 \pm 0.23_{25}$	$0.74 \pm 0.29_{17}$	$0.85 \pm 0.23_{15}$	-	$0.72 \pm 0.30_{21}$	$0.67 \pm 0.40_{9}$
Balance scale	-	-	-	-	-	-	15.0 ± 6.7	7358.0 ± 3076.0	$0.61 \pm 0.31_{24}$	$0.74 \pm 0.32_{24}$	$0.44 \pm 0.32_{12}$	-	$0.82 \pm 0.27_{16}$	$0.60 \pm 0.29_{10}$
Basketball	-	-	-	-	-	-	13.6 ± 5.5	8971.7 ± 1704.6	$0.69 \pm 0.27_{25}$	$0.72 \pm 0.33_{19}$	$0.57 \pm 0.33_{13}$	-	$0.74 \pm 0.31_{20}$	$0.61 \pm 0.37_{15}$
Bolts	-	-	-	-	-	-	15.6 ± 6.5	8468.6 ± 2388.3	$0.73 \pm 0.27_{21}$	$0.76 \pm 0.28_{20}$	$0.71 \pm 0.35_{17}$	$0.34 \pm 0.17_{3}$	$0.81 \pm 0.23_{26}$	$0.63 \pm 0.36_{10}$
Buying	-	-	-	-	-	-	15.1 ± 6.3	9024.1 ± 1431.3	$0.72 \pm 0.32_{30}$	$0.61 \pm 0.32_{12}$	$0.67 \pm 0.33_{2}$	-	-	-
German	-	-	-	-	-	-	22.2 ± 7.6	6033.7 ± 2926.3	$0.55 \pm 0.33_{11}$	$0.62 \pm 0.33_{22}$	$0.40 \pm 0.35_{14}$	$0.57 \pm 0.32_{15}$	$0.59 \pm 0.31_{13}$	$0.72 \pm 0.32_{11}$
House16	-	-	-	-	-	-	15.8 ± 7.3	7880.9 ± 2238.0	$0.77 \pm 0.29_{25}$	$0.74 \pm 0.28_{23}$	$0.68 \pm 0.29_{21}$	-	$0.55 \pm 0.36_{13}$	$0.54 \pm 0.38_{14}$
Ionosphere	-	-	-	-	-	-	16.8 ± 7.3	8059.6 ± 2564.3	$0.71 \pm 0.34_{28}$	$0.82 \pm 0.28_{21}$	$0.52 \pm 0.36_{5}$	-	$0.53 \pm 0.39_{5}$	-
Quake	-	-	-	-	-	-	11.8 ± 2.8	8982.1 ± 1247.7	$0.78 \pm 0.29_{27}$	$0.66 \pm 0.30_{15}$	$0.73 \pm 0.28_{13}$	$0.21 \pm 0.00_{1}$	$0.64 \pm 0.36_{18}$	$0.71 \pm 0.30_{18}$
Wine	-	-	-	-	-	-	14.6 ± 5.9	9342.5 ± 1265.8	$0.65 \pm 0.34_{24}$	$0.83 \pm 0.24_{29}$	$0.66 \pm 0.33_{17}$	$0.08 \pm 0.00_{1}$	$0.63 \pm 0.33_{22}$	$0.67 \pm 0.32_{11}$
							15.22 ± 2.82	8311.03 ± 963.66	$0.69 \pm 0.07_{24.00 \pm 4.92}$	$0.72 \pm 0.07_{20.20 \pm 4.58}$	$0.62 \pm 0.13_{12.90 \pm 5.36}$	$0.30 \pm 0.18_{5.00 \pm 5.83}$	$0.67 \pm 0.10_{17.11 \pm 5.86}$	$0.64 \pm 0.05_{12.25 \pm 2.90}$

[a] No preprocessing of the dataset.

4.1.4. Comparison with the VARDE State-of-the-Art Algorithm

The last experiment was reserved for an indirect comparison with the VARDE state-of-the-art algorithm [30] for ARM, which represents a hybridized version of DE and was designed specifically for the exploration and exploitation of the ARM search space. Thus, the best reported variations of VARDE were used in this comparative study. It was not a direct comparison since the pipelines produced by NiaAutoARM are dataset-specific. Therefore, for each dataset, we observed which components of the pipeline provided the best results (i.e., the lower-level heuristic algorithm, preprocessing component and rule evaluation metrics), and we performed 30 independent runs with these settings. The results of these dataset-specific independent runs were compared to the results of VARDE using the Wilcoxon signed rank test.

The results are depicted in Table 11.

Table 11. Results of the Wilxocon test when comparing the NiaAutoARM-generated pipelines with VARDE.

Method	Baseline		WO, $P = 1$		WO, $P > 1$	
	PSO	**DE**	**PSO**	**DE**	**PSO**	**DE**
VARDE_pos_15_2000 [30]	**0.03**	0.34	**0.01**	0.08	**0.01**	**0.01**
VARDE_neg_15_2000 [30]	0.61	0.17	0.97	0.54	0.75	0.98

As is evident from the table, the pipelines found by the NiaAutoARM provided significantly better results in some instances compared to the VARDE method. Therefore, NiaAutoARM was distinguished as an effective framework for ARM.

4.2. Discussion

The results show notable trends in the optimization of ARM pipelines. The PSO algorithm was selected predominantly over jDE, DE, LSHADE, and ILSHADE as the lower-level heuristic method. This preference can be attributed to the PSO's ability to balance exploration and exploitation effectively, enabling it to navigate the search space efficiently and avoid premature convergence. In contrast, the other algorithms may converge too quickly, potentially limiting their effectiveness in identifying diverse high-quality pipelines, thus making them less suitable for this specific optimization task. Min-max scaling was the most frequently used preprocessing method, likely due to its simplicity and ability to standardize data efficiently. Additionally, support and confidence were the dominant metrics in the generated pipelines, reflecting their fundamental role in ARM.

While the approach exhibits a slightly higher computational complexity due to the iterative optimization and exploration of diverse preprocessing combinations, this is a manageable trade-off (see Table 12). The superior results achieved, particularly in comparison to the VARDE state-of-the-art hybrid DE method, underscore the robustness of the approach. Notably, the method operates without requiring prior knowledge of the algorithms or datasets, making it adaptable and versatile for various applications.

In summary, the NiaAuroARM framework is capable of finding the best association rules automatically, without any intervention from the user. This makes the framework aligned with the goals of democratizing ML. However, the basic problem remains unsolved from the user's perspective, i.e., how to make explanations and predictions on the basis of the mined association rules. Therefore, the primary research direction for the future remains to integrate the NiaAutoARM with emerging technologies, like eXplainable AI (XAI). On the other hand, the hybridization of meta-heuristics presents a promising research issue for the future.

Table 12. The average execution times of both the higher-level meta-heuristic algorithms, which are needed for finding the best pipelines for each experimental dataset in seconds.

Dataset	PSO	DE
Abalone	27584.0 ± 7238.7	23486.5 ± 4702.6
Balance scale	15356.1 ± 6617.0	11598.7 ± 1298.3
Basketball	23442.6 ± 5271.6	15476.7 ± 1893.6
Bolts	22325.9 ± 9694.9	18603.7 ± 4979.5
Buying	33819.2 ± 10046.0	34449.2 ± 4134.3
German	25322.6 ± 10027.3	25958.7 ± 3230.3
House	34444.4 ± 8286.6	34464 ± 7709.4
Ionosphere	32299.7 ± 9396.3	40831.1 ± 7365.6
Quake	17897.9 ± 4523.5	18393.1 ± 4162.1
Wine	28541.7 ± 7341.7	24963.4 ± 3111.6

5. Conclusions

This paper presents NiaAutoARM, an innovative framework designed for the optimization of the ARM pipelines using stochastic population-based NI algorithms. The framework integrates the selection of the following: a lower-level heuristic, its hyper-parameter optimization, dataset preprocessing techniques, and searching for the more suitable fitness function represented as a weighted sum of ARM evaluation metrics (which is where the weights are the subjects of the adaptation). Extensive evaluations on ten widely used datasets from the UC Irvine repository underscore the framework's effectiveness, and it is particularly useful for users with limited domain expertise. Comparative analysis against the VARDE state-of-the-art hybrid DE highlights the superior performance of the proposed framework in generating high-quality ARM pipelines. In general, the obtained results underscore the effectiveness of NiaAutoARM's layered metaheuristic design in optimizing full NARM pipelines, offering clear advantages over conventional or single-layer optimization methods in terms of flexibility, adaptability, and also overall performance.

Our future work aims to address several key areas: First, integrating additional NI algorithms with adaptive parameter tuning could enhance the pipeline optimization process further. Second, incorporating other advanced preprocessing techniques and alternative metrics might improve pipeline diversity and domain-specific applicability. Third, exploring parallel and distributed computing strategies could mitigate computational complexity, making the framework more scalable for larger datasets and more complex mining tasks.

In addition, extending the framework to support multi-objective optimization would allow a deeper exploration of trade-offs between potentially conflicting metrics, advancing its utility for real-world applications that demand interpretable and actionable rule sets. Furthermore, a promising and underexplored direction is to investigate how the heterogeneity of the attribute type. Specifically, how the varying proportions of numerical and categorical attributes influence the performance, quality, and interpretability of the mined association rules. To date, this question has received little systematic attention in the literature, and examining it could lead to tailored strategies that further enhance the effectiveness of NiaAutoARM across mixed-attribute datasets.

Author Contributions: Conceptualization, I.F.J. and I.F.; Methodology, U.M. and I.F.J.; Software, U.M. and I.F.J.; Validation, U.M.; Formal analysis, U.M. and I.F.; Investigation, I.F.J. and I.F.; Writing—original draft, U.M., I.F.J. and I.F.; Writing—review and editing, U.M., I.F.J. and I.F.; Visualization, U.M.; Supervision, I.F.; Project administration, I.F.; Funding acquisition, U.M. and I.F.J. All authors have read and agreed to the published version of the manuscript.

Funding: Iztok Fister, Jr. wishes to thank the Slovenian Research Agency (Program No. P2-0057) for their financial support. Uroš Mlakar also wishes to thank the Slovenian Research and Innovation Agency (Program No. P2-0041) for their financial support.

Data Availability Statement: The data used in this study are available on request from the corresponding authors. The code of the proposed NiaAutoARM is publicly available on https://github.com/firefly-cpp/NiaAutoARM (accessed on 30 April 2025).

Acknowledgments: The authors express their gratitude to Žiga Stupan for his insightful input during the initial discussions of this research.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Yao, Q.; Wang, M.; Chen, Y.; Dai, W.; Li, Y.F.; Tu, W.W.; Yang, Q.; Yu, Y. Taking human out of learning applications: A survey on automated machine learning. *arXiv* **2018**, arXiv:1810.13306.
2. Hutter, F.; Kotthoff, L.; Vanschoren, J. *Automated Machine Learning: Methods, Systems, Challenges*; Springer Nature: Berlin/Heidelberg, Germany, 2019.
3. He, X.; Zhao, K.; Chu, X. AutoML: A survey of the state-of-the-art. *Knowl.-Based Syst.* **2021**, *212*, 106622. [CrossRef]
4. Conrad, F.; Mälzer, M.; Schwarzenberger, M.; Wiemer, H.; Ihlenfeldt, S. Benchmarking AutoML for regression tasks on small tabular data in materials design. *Sci. Rep.* **2022**, *12*, 19350. [CrossRef]
5. Eiben, A.E.; Smith, J.E. *Introduction to Evolutionary Computing*, 2nd ed.; Springer Publishing Company: Berlin/Heidelberg, Germany, 2015.
6. Blum, C.; Merkle, D. *Swarm Intelligence: Introduction and Applications*; Springer: Berlin/Heidelberg, Germany, 2008. [CrossRef]
7. Wolpert, D.H.; Macready, W.G. No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.* **1997**, *1*, 67–82. [CrossRef]
8. Agrawal, R.; Srikant, R. Fast Algorithms for Mining Association Rules in Large Databases. In Proceedings of the 20th International Conference on Very Large Data Bases, VLDB'94, San Francisco, CA, USA, 12–15 September 1994; pp. 487–499.
9. Han, J.; Cheng, H.; Xin, D.; Yan, X. Frequent Pattern Mining: Current Status and Future Directions. *Data Min. Knowl. Discov.* **2007**, *15*, 55–86. [CrossRef]
10. Alatas, B.; Akin, E.; Karci, A. MODENAR: Multi-objective differential evolution algorithm for mining numeric association rules. *Appl. Soft Comput.* **2008**, *8*, 646–656. [CrossRef]
11. Altay, E.V.; Alatas, B. Differential evolution and sine cosine algorithm based novel hybrid multi-objective approaches for numerical association rule mining. *Inf. Sci.* **2021**, *554*, 198–221. [CrossRef]
12. Fister, I.; Iglesias, A.; Galvez, A.; Del Ser, J.; Osaba, E.; Fister, I. Differential evolution for association rule mining using categorical and numerical attributes. In Proceedings of the Intelligent Data Engineering and Automated Learning–IDEAL 2018: 19th International Conference, Madrid, Spain, 21–23 November 2018; Proceedings, Part I 19; Springer: Berlin/Heidelberg, Germany, 2018; pp. 79–88.
13. Minaei-Bidgoli, B.; Barmaki, R.; Nasiri, M. Mining numerical association rules via multi-objective genetic algorithms. *Inf. Sci.* **2013**, *233*, 15–24. [CrossRef]
14. Heraguemi, K.E.; Kamel, N.; Drias, H. Association rule mining based on bat algorithm. *J. Comput. Theor. Nanosci.* **2015**, *12*, 1195–1200. [CrossRef]
15. Kuo, R.J.; Chao, C.M.; Chiu, Y. Application of particle swarm optimization to association rule mining. *Appl. Soft Comput.* **2011**, *11*, 326–336. [CrossRef]
16. Garey, M.R.; Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1st ed.; Series of Books in the Mathematical Sciences; W. H. Freeman: New York, NY, USA, 1979.
17. Glover, F.; Kochenberger, G.A. (Eds.) *Handbook of Metaheuristics*; International Series in Operations Research & Management Science; Springer: Berlin/Heidelberg, Germany, 2003.
18. Grefenstette, J. Optimization of control parameters for genetic algorithms. *IEEE Trans. Syst. Man Cybern.* **1986**, *16*, 122–128. [CrossRef]
19. Cui, H.; Bai, J. A new hyperparameters optimization method for convolutional neural networks. *Pattern Recognit. Lett.* **2019**, *125*, 828–834. [CrossRef]
20. Stang, M.; Meier, C.; Rau, V.; Sax, E. An Evolutionary Approach to Hyper-Parameter Optimization of Neural Networks. In *Human Interaction and Emerging Technologies, Proceedings of the 1st International Conference on Human Interaction and Emerging Technologies (IHIET 2019), Nice, France, 22–24 August 2019*; Ahram, T., Taiar, R., Colson, S., Choplin, A., Eds.; Springer: Cham, Switzerland, 2020; pp. 713–718.

21. Holzinger, A. Interactive machine learning for health informatics: When do we need the human-in-the-loop? *Brain Inform.* **2016**, *3*, 119–131. [CrossRef]
22. Zöller, M.A.; Huber, M.F. Benchmark and survey of automated machine learning frameworks. *J. Artif. Intell. Res.* **2021**, *70*, 409–472. [CrossRef]
23. Escalante, H.J. Automated Machine Learning—A Brief Review at the End of the Early Years. In *Automated Design of Machine Learning and Search Algorithms*; Springer: Cham, Switzerland, 2021; pp. 11–28.
24. Musigmann, M.; Akkurt, B.H.; Krähling, H.; Nacul, N.G.; Remonda, L.; Sartoretti, T.; Henssen, D.; Brokinkel, B.; Stummer, W.; Heindel, W.; et al. Testing the applicability and performance of Auto ML for potential applications in diagnostic neuroradiology. *Sci. Rep.* **2022**, *12*, 13648. [CrossRef]
25. Barreiro, E.; Munteanu, C.R.; Cruz-Monteagudo, M.; Pazos, A.; González-Díaz, H. Net-Net auto machine learning (AutoML) prediction of complex ecosystems. *Sci. Rep.* **2018**, *8*, 12340. [CrossRef]
26. Fister, I.; Zorman, M.; Fister, D.; Fister, I. Continuous optimizers for automatic design and evaluation of classification pipelines. In *Frontier Applications of Nature Inspired Computation*; Springer Tracts in Nature-Inspired Computing; Springer: Singapore, 2020; pp. 281–301.
27. Pečnik, L.; Fister, I.; Fister, I., Jr. NiaAML2: An Improved AutoML Using Nature-Inspired Algorithms. In Proceedings of the Advances in Swarm Intelligence: 12th International Conference, ICSI 2021, Qingdao, China, 17–21 July 2021; Proceedings, Part II 12; Springer: Berlin/Heidelberg, Germany, 2021; pp. 243–252.
28. Stupan, Ž.; Fister, I. NiaARM: A minimalistic framework for Numerical Association Rule Mining. *J. Open Source Softw.* **2022**, *7*, 4448. [CrossRef]
29. Vrbančič, G.; Brezočnik, L.; Mlakar, U.; Fister, D.; Fister, I., Jr. NiaPy: Python microframework for building nature-inspired algorithms. *J. Open Source Softw.* **2018**, *3*, 613. [CrossRef]
30. Mlakar, U.; Fister, I. Variable-Length Differential Evolution for Numerical and Discrete Association Rule Mining. *IEEE Access* **2023**, *12*, 4239–4254. [CrossRef]
31. Dua, D.; Graff, C. UCI Machine Learning Repository. 2017. Available online: https://archive.ics.uci.edu/ (accessed on 30 October 2024)
32. Telikani, A.; Gandomi, A.H.; Shahbahrami, A. A survey of evolutionary computation for association rule mining. *Inf. Sci.* **2020**, *524*, 318–352. [CrossRef]
33. Yan, D.; Zhao, X.; Lin, R.; Bai, D. PPQAR: Parallel PSO for quantitative association rule mining. *Peer-to-Peer Netw. Appl.* **2019**, *12*, 1433–1444. [CrossRef]
34. Su, T.; Xu, H.; Zhou, X. Particle swarm optimization-based association rule mining in big data environment. *IEEE Access* **2019**, *7*, 161008–161016. [CrossRef]