Continuous Optimizers for Automatic Design and Evaluation of Classification Pipelines

Iztok Fister Jr.[1](✉), Milan Zorman[1], Dušan Fister[2], and Iztok Fister[1]

[1] Faculty of Electrical Engineering and Computer Science, University of Maribor, Koroška cesta 46, 2000 Maribor, Slovenia
iztok.fister1@um.si

[2] Faculty of Economics and Business, University of Maribor, Razlagova 14, SI-2000 Maribor, Slovenia

1 Introduction

As has been known for a long time, some basic research areas (e.g., medicine, biology) cannot solve some specific problems in their highly specialized laboratories without modern scientific computational methods and algorithms. Bioinformatics stands for a very vibrant interdisciplinary research area that, nowadays, has been solving problems where the aid of digital computers is unavoidable. Therefore, it is no wonder that the scientific discipline encompasses specialists from different research areas, such as, for example, computer scientists, mathematicians, biologists, geneticists, and statisticians.

Indeed, the advent of digital computers has changed the principles of experimental work as well. In past, most of the experiments in science were performed either in vitro or in vivo. The former case refers to controlled experiments that are conducted outside of an organism (e.g., in cellular biology), while the latter to experimentation running on a living organism (e.g., animals). Recently, most experiments have been performed in silico, where they are performed on computers or as computer simulations (e.g., DNA analysis).

For instance, let us imagine the whole human genotype that consists of approximately 25,000 different genes. In line with this, several questions have arisen in bioinformatics, as follows: How can we analyze all complex interactions between them, or even gene expression profiles, without specialized computational algorithms? How to process big data that are produced by the next-generation sequencing (NGS) technology [29] and can easily be measured in gigabytes or even terabytes? How to acquire new information or laws from biodata? Very topical problems have arisen as a consequence of all these questions, and the answers are searched for through the world sponsors of many scientific related projects by many research agencies around the world. For example, in recent years, the famous European Union Project Horizon 2020 supported many projects from these areas. By the same token, many hospitals opened their doors

M. Khosravy et al. (eds.), *Frontier Applications of Nature Inspired Computation*,
Springer Tracts in Nature-Inspired Computing,
https://doi.org/10.1007/978-981-15-2133-1_13

and hired bioinformaticians. Bioinformaticians support clinicians or laboratory researchers with new information that can be obtained from data [19]. Most of the above posted questions coincide with the same as are faced by data science as well [5]. Data science is a multi-disciplinary domain that uses scientific methods, processes, algorithms, and systems for extracting knowledge, and, thus, enabling new insights from structured and unstructured data. Often, data scientists also need to use methods from machine learning (ML) by data analysis. ML is a part of an artificial intelligence (AI) that is capable of building a mathematical model of sample data in order to make predictions or decisions without being programmed explicitly to perform this task [2]. Thus, they were confronted with a question, how to employ these methods as effectively as possible?

As a key solution, automated machine learning (AutoML) became a topic of considerable interest [25]. The purpose of AutoML is to automate some phases of ML tasks, especially those demanding from data scientists to select the proper ML method, or to set the more appropriate hyperparameters. Obviously, these tasks are far from easy for non-experts. Therefore, the AutoML searches for customized ML pipelines which are actually optimized sequences of ML methods, processes, algorithms, and appropriate hyperparameters for controlling their behavior.

AutoML can be modeled as an optimization problem. There is a big pool of different AutoML methods that are based on evolutionary algorithms (EAs) [12]. TPOT [26] is a tree-based pipeline optimization tool for AutoML. It is based on GP. Some improvements of TPOT are offered in the thesis [15]. RECIPE [8] is another excellent example that is based on grammar-based GP that builds customized classification pipelines. Interestingly, paper [34] reports a new EA for the AutoML task of selecting the best ensemble of classifiers and their hyperparameter settings automatically for an input dataset. An example of the swarm intelligence (SI) algorithm ant colony optimization (ACO) was used in the work by Costa and Rodrigues [7].

Classification problems are very common within the bioinformatics area. The classification task of a sample data means to classify each item from the sample, represented by features, into one of a predefined set of classes. Although the task seems trivial from the expert's point of view, a data scientist, for example, needs to perform at least three tasks to accomplish this: The proper feature selection algorithm must be selected first, followed by selection of the appropriate ML classification methods. Finally, the optimal hyperparameter setting must be found for each of the selected algorithms and ML methods. As a result, a classification pipeline is modeled manually that represents a customized sequence of classification methods and algorithms from the data scientist's point of view.

As we can see from the aforementioned example, modeling of a classification pipeline is a really complex task. Additionally, it is worth mentioning that the bioinformatics community also consists of data scientists who are not computer experts. Potentially, they have trouble in conducting classification tasks using their own sample datasets. For that reason, they rely mostly on some existing, usually commercial, software. However, this has many bottlenecks, especially due to a lack of robustness of software solutions on the market.

In this paper, the stochastic nature-inspired population-based algorithms are proposed for evolving classification pipelines automatically in bioinformatics. In line with this, a novel AutoML method, named NiaAML, is presented, which has the following advantages:

- the problem of AutoML is modeled as a continuous optimization problem, where each stochastic nature-inspired population-based algorithm can be used for solving this problem,
- the proposed method is presented in layer style architecture. Hence, adding new components (e.g., classifiers) can be done in a very easy way,
- it is also intended to be used by non-programmers.

The structure of this paper is as follows: Sect. 2 deals with AutoML and its characteristics. Section 3 outlines the proposed NiaAML solution, and Sect. 4 shows performed experiments and results. Section 5 concludes the paper and outlines directions for the future work.

2 AutoML

The advent of big data has brought an explosion of ML researches and applications [20]. Consequently, scientists from various areas have been confronted with issues of how to analyze data as efficiently as possible. Typically, a lot of ML methods are employed in this analysis. Unfortunately, the performance of the analysis depends on the selected ML methods, on the one hand, while all those methods are sensitive to a plethora of parameters controlling their behavior, on the other.

Therefore, finding the optimal sequence of ML methods for application on sample data (also ML pipeline), together with their optimal parameter settings, represents big trouble even for experts, let alone other domain specialists. Obviously, the optimal ML pipeline cannot be found in one step, but demands a "trial-and-error" approach that was confirmed in evolutionary computation (EC) [12]. Unfortunately, this approach cannot be applied here, due to the huge amount of sample data. As a result, searching for the novel approximation methods needs to be performed achieving the results good enough for using in practice.

In order to simplify complex ML tasks, the AutoML has been evolved, with the aim to automate creation of the optimal ML pipelines and to enable other domain specialists to use the complex set of ML methods and their optimal hyperparameter settings easily. This means that the AutoML is a way of democratization of ML, wherein ML experts wish to draw the state-of-the-art ML methods nearer to the other domain specialists. Although we are witnesses of an early stage of AutoML development, the AutoML can even outperform human ML experts and show its potential for the future.

This fast-moving area of ML is focused on three main issues:

- Hyperparameter optimization (HPO),
- Meta-learning,

- Neural architecture search (NAS).

Various settings of hyperparameter values are crucial for good performance of ML methods. Typically, these methods have a lot of hyperparameters, with ranges usually unknown in advance. Therefore, it is not easy to determine their optimal values manually. This issue has led researchers in AutoML to HPO, where two optimization types are proposed in summary:

- blackbox optimization,
- multi-fidelity optimization.

The former captures algorithms such as: model-free blackbox HPO and Bayesian optimization. The model-free blackbox HPO is referred to traditional search algorithms starting from a grid search, going through random search, and ending with stochastic nature-inspired population-based algorithms [13,16]. On the other hand, the Bayesian optimization is applied successfully for tuning deep neural networks. Commonly, both mentioned types of blackbox optimization algorithms are too time-consuming to be used in deep learning (DL). The latter type speeds up the manual tuning by probing hyperparameter setting on small subsets of data, casting this into a formal multi-fidelity algorithm, and applying it on sample data. This HPO type presents the best trade-off between optimization performance and runtime.

Meta-learning is the science of observing systematically how different ML methods perform on a wide range of learning tasks, and then learning from this experience [20]. Actually, the challenge in meta-learning is to learn from prior experience in a systematic, data-driven way, with the goal of searching for optimal models for new tasks. This means, the learning does not start from scratch, but bases on the collected meta-data.

Meta-learning consists of two steps [20]:

- Collecting a meta-data, which describes prior learning tools and previously learned models. The meta-data consists of algorithm configurations, including hyperparameter settings, pipeline composition and/or network architecture, and model evaluation expressed as accuracy and training time.
- Exploring the meta-data to learn, extract, and transfer knowledge for guiding the search for the optimal models for new tasks.

Unfortunately, learning from prior experience is effective only until a new task represents completely unrelated phenomena or random noise. However, the real-world tasks are usually not sensitive to the mentioned disturbances.

Deep learning brought a need for using complex network architectures. The more efficient architectures cannot be searched manually. Therefore, growing interest in an automated NAS has been increasing recently. Although NAS has a significant overlap with hyperparameter optimization and meta-learning, it can be seen as a subfield of AutoML. According to Hutter et al. [20], NAS methods are classified with regard to:

- search space,

- search strategy,
- performance estimation strategy.

Incorporating prior knowledge about the search space can reduce its dimensionality, and, thus, simplify the search. A search strategy determines how to explore the search space. A performance estimation strategy searches for those performance measures that allow reduction of the cost of their estimations, on the one hand, and achieve highly predictive performance on unseen data on the other [20].

3 Proposed NiaAML Method

Although classification pipelines have already been composed using various stochastic nature-inspired population-based algorithms (e.g., TPOT, RECIPE. etc.), their origins were usually found in genetic programming (GP) [22], where individuals are represented as trees. This study is focused on the real-valued stochastic nature-inspired population-based algorithms for evolving classification pipelines. A NiaAML method was developed according to the directions of AutoML development, as proposed in [20]. In this sense, the NiaAML is referred to the collecting meta-data step. As already seen, the collecting meta-data is a meta-learning step devoted for pipeline composition including hyperparameter optimization and model evaluation.

An evolving classification pipeline is illustrated in Fig. 1, from which it can be seen that the classification pipeline composition consists of three tools:

- feature selection,
- classifier selection,
- hyperparameter optimization.

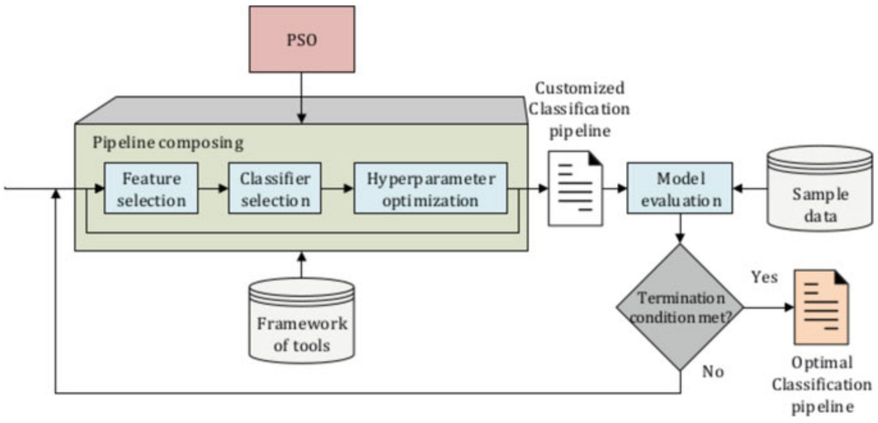
PSO
Customized Classification pipeline
Pipeline composing
Feature selection
Classifier selection
Hyperparameter optimization
Model evaluation
Sample data
Framework of tools
Termination condition met?
Yes
No
Optimal Classification pipeline

Fig. 1. A classification pipeline evolving

The task of the feature selection tool is to determine the suitable algorithm for feature selection, while the classifier selection tool is to select the proper ML method. However, the algorithms, as well as classification methods, are incorporated into a framework of tools. The hyperparameter optimization serves as a searching mechanism for specifying the proper values of the parameters arisen in the feature selection algorithms and classification methods.

The result of the pipeline composing is a customized classification pipeline that needs to be evaluated in the model evaluation step. The task of evaluation is to assess the quality of the composed model on a sample data. The evolving process, consisting of pipeline composing and model evaluation, is launched until the termination condition is satisfied.

An architecture of the NiaAML method is depicted in Fig. 2. The architecture is represented schematically as a feature diagram (FD) [21] and describes tasks needed for composing customized classification pipelines. Actually, the FD is a tree consisting of vertices and arcs. Vertices represent task models, and arcs determine relationships between these. The vertices are either mandatory or optional, where the former are denoted by closed dots, and the latter by open dots.

As can be seen from Fig. 2, the task of composing the customized classification pipelines ('NiaAML' vertex) has three conceptual levels, as follows:

- pipeline composing models (features in FD),
- model families (sub-features in FD),
- tools and hyperparameter settings (attributes in FD).

Interestingly, each feature/sub-feature is determined by its own set of sub-features/attributes that are connected between each other with various relationships. There are three different relationships in the FD: 'and', 'one of', and 'more of'. The 'one of' relationship is denoted by an opened semicircle joining

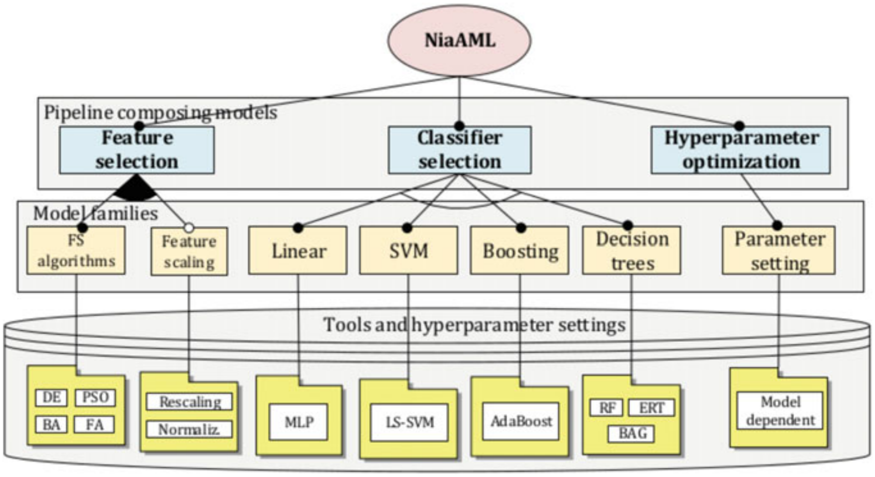

Fig. 2. Feature diagram of NiaAML method

the arcs from a feature to its sub-features, the ‘more of’ by a closed semicircle, while the ‘and’ relationship is without any semicircle. The meaning of these relationships is explained simultaneously with a detailed description of the FD that follows.

Composing the customized classification pipeline with NiaAML consists of three mandatory tasks that include modeling the feature selection, the classifier selection, and the hyperparameter optimization. These tasks are connected with relation ‘and’, which means that all three tasks must be included in the process of composing the pipeline. Modeling the feature selection is composed of modeling the feature selection algorithm and feature scaling. Both sub-features in FD are connected with the relation ‘more of’. Because modeling the feature scaling is optional, the relation means that modeling feature selection can be performed with or without the feature scaling. However, the feature selection algorithm must be modeled anyway. A classification method can even be modeled from four classification families as follows: linear, SVM, boosting, and decision tree. These sub-features in FD are mutually connected with the relation ‘one of’, meaning that one of the classification families cooperates in composing the classification pipelines. The hyperparameter optimization is specific, because it has only one task, i.e., modeling the hyperparameter setting that is dependent on the selected feature selection algorithm, as well as the selected classification method.

The lowest level in an FD tree represents a framework of tools and hyperparameter settings. This consists of feature selection/scaling algorithms, classification methods, and optimized hyperparameter settings. All framework elements are selected with regard to the corresponding model families. For instance, a feature selection algorithm can be modeled using four different feature selection algorithms. Moreover, the rescaling and normalization procedures are available for the feature scaling. At the moment, one classifier method is modeled for each classification family, except a decision tree supporting even three classification methods. Obviously, in the future, we plan to increase the number of classification methods. As already explained, the hyperparameters’ setting is model dependent.

In the remainder of the paper, the algorithm for composing the classification pipeline is presented in detail. The section is concluded with illustrating the model evaluation.

3.1 Composing the Classification Pipeline

The problem can be defined informally as follows: The task is to compose the classification pipeline from tools incorporated into a framework, where the pipeline consists of feature selection and feature scaling algorithms, classification methods, and optimal setting of hyperparameters, controlling the algorithms and/or methods, such that the maximum classification accuracy is achieved. The problem is defined as an optimization and solved by the particle swarm optimization (PSO) [11]. Although it can be solved with the other stochastic population-based nature-inspired algorithms as well, the PSO was preferred due to its simplicity.

3.1.1 Particle Swarm Optimization PSO is a member of the SI-based algorithm family that was developed by Eberhart and Kennedy in 1995 [11]. It is inspired by the social behavior of bird flocking and fish schooling [9,17]. This algorithm works with a swarm (i.e., population) of particles representing candidate solutions $\mathbf{x}_i^{(t)}$ of the problem to be solved. The particles fly virtually through the problem space and are attracted by more promising regions. When the particles are located in the vicinity of these regions, they are rewarded with the better values of fitness function by the algorithm.

The PSO algorithm exploits usage of additional memory, where the particle's personal best $\mathbf{p}_i^{(t)}$, as well as the swarm's global best $\mathbf{g}^{(t)}$ locations in the search space are saved. In each time step t (i.e., generation), all particles change their velocities $\mathbf{v}_i^{(t)}$ toward its personal and global best locations according to the following mathematical formula:

$$\begin{aligned} \mathbf{v}_i^{(t+1)} &= \mathbf{v}_i^{(t)} + C_1 \cdot \text{rand}(0,1) \cdot \left(\mathbf{g}^{(t)} - \mathbf{x}_i^{(t)}\right) + C_2 \cdot \text{rand}(0,1) \cdot \left(\mathbf{p}_i^{(t)} - \mathbf{x}_i^{(t)}\right), \\ \mathbf{x}_i^{(t+1)} &= \mathbf{x}_i^{(t)} + \mathbf{v}_i^{(t)}, \end{aligned} \tag{1}$$

where C_1 and C_2 present learning rates typically initialized to 2, and rand$(0,1)$ is a random value drawn from uniform distribution in the interval $[0,1]$.

The pseudo-code of the original PSO is illustrated in Algorithm 1, from which it can be seen that the PSO is distinguished from the classical EAs by four specialties:

- does not have survivor selection,
- does not have parent selection,
- does not have crossover operator, and

Algorithm 1 The original PSO algorithm

```
1: procedure ParticleSwarmOptimization
2:     $t \leftarrow 0$;
3:     $P^{(t)} \leftarrow$ Initialize;                       ▷ initialization of population
4:     while not TerminationConditionMeet do
5:         for all $\mathbf{x}_i^{(t)} \in P^{(t)}$ do
6:             $f_i^{(t)} = $ Evaluate($\mathbf{x}_i^{(t)}$);          ▷ evaluation of candidate solution
7:             if $f_i^{(t)} \le f_{best_i}^{(t)}$ then
8:                 $\mathbf{p}_i^{(t)} = \mathbf{x}_i^{(t)}$; $f_{best_i}^{(t)} = f_i^{(t)}$;
9:             end if                                    ▷ preserve the local best solution
10:            if $f_i^{(t)} \le f_{best}^{(t)}$ then
11:                $\mathbf{g}^{(t)} = \mathbf{x}_i^{(t)}$; $f_{best}^{(t)} = f_i^{(t)}$;
12:            end if                                    ▷ preserve the global best solution
13:            $\mathbf{x}_i^{(t)} = $ Move($\mathbf{x}_i^{(t)}$);            ▷ move the candidate w.r.t. Eq. (1)
14:        end for
15:        $t = t + 1$;
16:    end while
17: end procedure
```

- the mutation operator is replaced by the move operator changing each element of particle $\mathbf{x}_i^{(t)}$ with probability of mutation $p_m = 1.0$.

Let us mention that the selection is implemented in the PSO implicitly, i.e., by improving the personal best solution permanently. However, when this improving is not possible anymore, the algorithm gets stuck in the local optima.

In what follows, the necessary modifications to the PSO algorithm are described in order to prepare it for composing the classification pipelines. The section is concluded with a discussion about the classification pipeline evaluation.

3.1.2 Representation of Individuals In the modified PSO algorithm, individuals are represented by floating-point vectors as follows:

$$\mathbf{x}_i^{(t)} = \{ \underbrace{x_{i,1}^{(t)}}_{FeatureSelection}, \underbrace{x_{i,2}^{(t)}}_{FeatureScaling}, \underbrace{x_{i,3}^{(t)}}_{Classification}, \underbrace{x_{i,4}^{(t)}, \ldots, x_{i,k}^{(t)}, x_{i,k+1}^{(t)}, \ldots, x_{i,D}^{(t)}}_{HyperParameters} \}, \tag{2}$$

where each element of the individual $x_{i,j}^{(t)}$ is mapped to the attribute from the corresponding feature sets according to genotype–phenotype mapping illustrated in Table 1. The table illustrates a mapping of the elements of a vector to the features, and then into the attributes of the corresponding feature sets. Indeed, the first three elements of the floating-point vector are mapped to attributes, as follows:

Table 1. Genotype–phenotype mapping

Vector elements	Feature name	Feature set
$x_{i,1}^{(t)}$	*FeatureSelection*	$\{DE, PSO, GWO, BA\}$
$x_{i,2}^{(t)}$	*FeatureScaling*	$\{No, Rescaling, Normalization\}$
$x_{i,3}^{(t)}$	*Classification*	$\{MLP, LS\text{-}SVM, ADA, RF, ERT, BAG\}$
$x_{i,4}^{(t)}, \ldots, x_{i,k}^{(t)}, x_{i,k+1}^{(t)}, \ldots, x_{i,D}^{(t)}$	*HyperParameters*	Model dependent

$$attr_{feat}^{(t)} = \left\lfloor \frac{x_{i,j}^{(t)}}{|attr_{feat}|} \right\rfloor, \quad \text{for } j = 1, \ldots, 3, \tag{3}$$

where $attr_{feat}^{(t)}$ denotes the specific attribute of the feature and the $|attr_{feat}|$ is the size of feature set $feat \in \{FeatureSelection, FeatureScaling, Classification\}$. Thus, the result of the division is truncated. The remaining elements of the vector represent the absolute values of the corresponding hyperparameter laying in the corresponding hyperparameter domain. Although the maximum number of elements is fixed according to the algorithm and method with the maximum number of control hyperparameters, the number of effective elements is variable and depends on the selected tools. However, the hyperparameter domain depends on the particular hyperparameter and must be determined experimentally.

3.1.3 Hyperparameter Domains This subsection focuses on defining the hyperparameter domains. Actually, searching for the optimal setting of hyperparameters is a part of the NiaAML optimization process, and not by the separate process as proposed by the AutoML community. Consequently, the NiaAML performs HPO simultaneously with composing the classification pipelines, and, therefore, can be faster than the traditional AutoML.

In our study, we dealt with feature rescaling or normalization, four feature selection algorithms, and six classification methods. The feature scaling can either be selected (i.e., rescaling or normalization) or not selected. In both cases, however, there are no special parameters for controlling these algorithms. Among the feature selection algorithms, the following stochastic nature-inspired population-based algorithms were proposed: Differential evolution (DE) [32], PSO, gray wolf optimization (GWO) [23], and bat algorithm (BA) [35]. Obviously, even six methods, like multilayer perceptron (MLP) [27], least squares support vector machine (LS-SVM) [33], AdaBoost (ADA) [28], random forest (RF) [4], extremely andomized trees (ERT) [14], and BAGging (BAG) [3], can be selected for classification.

The hyperparameters of the proposed algorithms and methods, that were the subject of HPO, are illustrated in Table 2.

Table 2. Hyperparameter domains

Alg.	Hyperparameter	Domain of values
DE	F, CR	$F \in [0.5, 0.9]$, $CR \in [0.0, 1.0]$
PSO	C_1, C_2	$C_1 \in [1.5, 2.5], C_2 \in [1.5, 2.5]$
GWO	a	$a \in [0.0, 2.0]$
BA	$A, r, Q_{\min}, Q_{\max}$	$A \in [0.5, 1.0], r \in [0.0, 0.5], Q_{\min} \in [0.0, 1.0], Q_{\max} \in [1.0, 2.0],$
MLP	act, sol, lr	$act \in \{identity, logistic, tanh, relu\}$, $sol \in \{lbfgs, sgd, adam\}$
		$lr \in \{constant, invscaling, adaptive\}$
LS-SVM	$gamma$, c	$gamma \in [0.1, 100], c \in [0.1, 100]$
ADA	n_estim, alg	$n\_estim = [10, 110]$, $alg \in \{samme, samme.r\}$
RF	n_estim	$n\_estim \in [10, 110]$
ERT	n_estim	$n\_estim \in [10, 110]$
BAG	n_estim	$n\_estim \in [10, 110]$

Let us mention that the maximum number of hyperparameters occurring in the selected algorithms and classification methods is seven. As a result, the size of the real-valued vector for composing the classification pipeline is 10.

3.1.4 Fitness Function Evaluation In order to calculate the value of fitness function, tenfold cross-validation is used [1]. Thus, data are split into $k = 10$ equal parts using stratified sampling, and each of the k parts are classified by the classification pipeline. The main advantage of this approach is that all these so-called training sets have 80% of data in common when $k = 10$. Consequently, the trade-off between the bias and variance parts of the prediction error is minimized due to reducing both as much as possible.

The performance of the classification is estimated by accuracy ($Accuracy$), a statistical measure that estimates the proportion of true predictions (i.e., true positives and true negatives) among the total number of samples. Mathematically, this proportion can be expressed as:

$$Accuracy(M(\mathbf{x}_i)) = \frac{TP + TN}{TP + TN + FP + FN}, \tag{4}$$

where $M(\mathbf{x}_i)$ denotes a model built on the basis of the customized classification pipeline $\mathbf{x}_i$, TP = True Positive, TN = True Negative, FP = False Positive, and FN = False Negative.

The fitness function is then defined as follows:

$$f(\mathbf{x}_i) = 1 - \frac{1}{k}\sum_{i=1}^{k} Accuracy(M(\mathbf{x}_i)), \tag{5}$$

where $Accuracy(M(\mathbf{x}_i))$ is calculated according to Eq. (4). Let us emphasize that the fitness function evaluates the average performance of the classification methods obtained in 10-folds of training data. The task of the optimization is to minimize the value of the fitness function.

3.2 Model Evaluation

In this step, the quality of the composed classification pipeline is evaluated, found in the last step that consists of selected features, classifier, and hyperparameters. Typically, performance of the classification method in data science is evaluated by applying the evolving model to unseen test data. A standard 80–20% holdout validation is used, where 80% of the data is used for training and the other 20% for testing. The classification performance is then assessed according to the accuracy as expressed in Eq. (4).

4 Experiments and Results

The purpose of our experimental work was to show that the algorithm for composing the classification pipeline finds comparable, if not better, results than those tuned by the real data science experts. In line with this, three experiments were performed using well-known datasets from the UCI machine learning repository [10]. Actually, three datasets from the life sciences domain were taken into account. The characteristics of these datasets are depicted in Table 3.

Table 3. Datasets used

Dataset name	Characteristics	Attribute characteristics	#instances	#features	Missing data
Yeast	Multivariate	Real	1,484	8	No
Ecoli	Multivariate	Real	336	8	No
Abalone	Multivariate	Categorical, integer, real	4,177	8	No

As can be seen from the table, all datasets are multivariate with 8 features. As the number of features (denoted as #features in the table) is low, we can expect that the feature selection algorithm cannot reduce this number a lot. On the other hand, the number of instances (denoted as #instances in the table) increases from 336 toward 4,177.

Although the NiaAML is prepared to tune the algorithm's parameters as well, the parameter values were fixed in this preliminary study due to simplicity. The parameter settings of the used algorithms are illustrated in Table 4.

Let us mention that the population size $NP = 20$ and the number of fitness function evaluations $nFES = 400$ are fixed for all algorithms due to fairness by mutual comparison. However, the parameter settings of the NiaAML algorithm are the same as presented in Table 4 for the PSO, except the population size of $Np = 15$ and the number of fitness function evaluations $nFES = 500$. Interestingly, the GWO algorithm is parameterless, and, therefore, does not demand any parameter setting. The optimal setting of hyperparameters is the subject of the HPO as presented in Table 2.

Table 4. Parameter setting

Algorithm	Acronym	Parameter 1	Parameter 2	Parameter 3	Parameter 4
Differential evolution	DE	$F = 0.5$	$CR = 0.9$		
Gray wolf optimizer	GWO				
Particle swarm algorithm	PSO	$C_1 = 2.0$	$C_2 = 2.0$	$w = 0.7$	$v \in [-4, 4]$
Bat algorithm	BA	$A = 0.5$	$r = 0.5$	$Q \in [0.0, 2.0]$	

The results of the optimization were measured according to three statistical measures: *precision*, Cohen's kappa κ, and F_1-score. The precision is defined as follows [1]:

$$precision(M(\mathbf{x}_i)) = \frac{TP}{TP + FP}, \tag{6}$$

where TP = True Positives, and FP = False Positives.

Metric for handling multivariate class problems is Cohen's kappa defined as [6]:

$$\kappa(M(\mathbf{x}_i)) = \frac{n \sum_{i=1}^{k} CM_{ii} - \sum_{i=1}^{k} CM_{i.} CM_{.i}}{n^2 - \sum_{i=1}^{k} CM_{i.} CM_{.i}}, \tag{7}$$

where $CM_{i.}$ is the sum of the elements in the ith row of confusion matrix CM and $CM_{.i}$ the sum of the elements in the ith columns of the CM [31]. The metric measures a level of agreement between two annotators on a multivariate classification problem and can occupy values in the interval $[-1, 1]$. Values below 0.8 mean that there is no agreement between annotators.

Finally, the F_1-score based on *precision* and *recall* is expressed as follows:

$$F_1(M(\mathbf{x})) = 2 \cdot \frac{precision \cdot recall}{precision + recall}, \tag{8}$$

where $recall = \frac{TP}{TP+FN}$, FN= False Negatives, and TP= True Positives.

It is worth mentioning that *precision*, *recall*, and F_1 can be determined for the binary classification only, i.e., two prediction classes. However, using the *sklearn*'s metrics library, one can compute a weighted average among many binary classification tasks and thus can use those measures for multiclass classification.

Let us emphasize that we focus on the results of the composing of the classification pipeline in the preliminary study. This means that the results of the model evaluation phase were left for the future. In the remainder of the paper, the aforementioned experiments are described in detail.

4.1 The Results on the Yeast Dataset

The three top results of composing the classification pipeline according to measure *Accuracy* on the Yeast dataset are illustrated in Table 7, from which it can be seen that these were all obtained by applying the RF classification method.

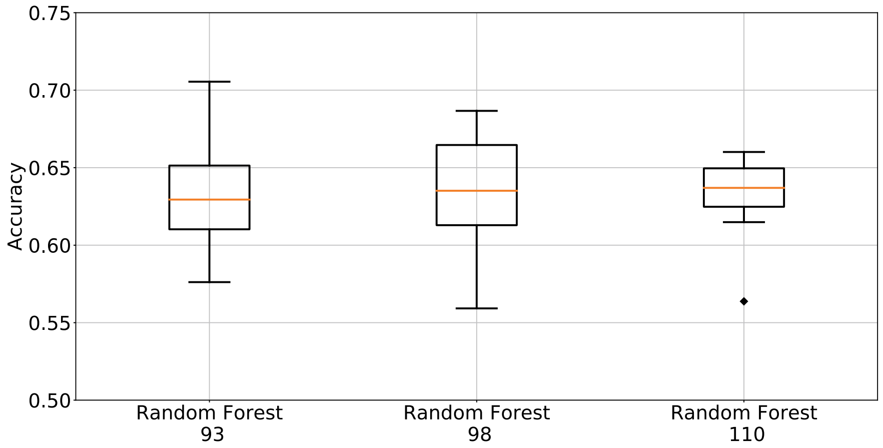

Fig. 3. Top three configurations found on the Yeast dataset

The numerical results of data, depicted in Fig. 3, are presented in Table 5, where each of the best results is denoted by places from 1 to 3. In the table, the values in square brackets denote the number of reduced features according to the total set in the column 'Feature selection' and the optimized number of estimators used by the corresponding classification method in the column 'Classification method'. The sign 'n/a' (i.e., not applicable) in the column 'Feature scaling' indicates that no feature scaling was applied. The column '*Accuracy*' denotes the accuracy calculated according to the Eq. (4).

Table 5. Numerical results obtained on the Yeast dataset

Place	Feature selection	Feature scaling	Classification method	*Accuracy*
1	[8/8]	n/a	Random forest [93]	0.6337
2	[8/8]	n/a	Random forest [98]	0.6329
3	[8/8]	n/a	Random forest [110]	0.6321

Interestingly, the best results were obtained using the PSO FS algorithm, no feature scaling, and the RF classification method. Although the FS procedure was performed, it was found out on the basis of the top three configurations that the best results are obtained by incorporating all the features. This indicates that each feature plays a relevant role for interpretation of results and that low redundancy is present among the features. RF, with 93 estimators, is the best among all built models. We can see that accuracy performance lowers by increasing model complexity.

The results according to the statistical measures, i.e., *Accuracy*, *Precision*, Cohen's kappa, and F_1-score, are depicted in Table 6.

Table 6. Statistical measures obtained on the Yeast dataset

Statistical measure	Statistical score
Accuracy	0.6465
Precision	0.6382
Cohen's kappa, κ	0.5356
F_1-score	0.6387

As can be seen from the Cohen's kappa measure $\kappa = 0.5356$ in the table, the value is higher than 0.5. This indicates moderate agreement between true and prediction labels. Obtained $Accuracy = 0.6465$ is higher than in cross-validation $Accuracy = 0.6337$.

4.2 The Results on the Ecoli Dataset

Three of the best results of the NiaAML algorithm for composing the classifier pipeline according to measure *Accuracy* obtained on the Ecoli dataset are illustrated in Table 7. The meaning of the variables in the table is the same as discussed in the last experiment. Also, in this case, the RF classification method was the most preferable by the NiaAML, and the FS procedure has not shown to be beneficial. The feature scaling was not applied in any case, while the number of estimators by classification method was more than >100 for each instance. Although the number of estimators are somehow similar to the Yeast dataset, classification accuracy is much higher for the Ecoli dataset.

Table 7. Numerical results obtained on the Ecoli dataset

Place	Feature selection	Feature scaling	Classification method	*Accuracy*
1	[7/7]	n/a	Random forest [104]	0.8899
2	[7/7]	n/a	Random forest [101]	0.8882
3	[7/7]	n/a	Random forest [109]	0.8880

The graphical representation of the same results are presented in Fig. 4, from which it can be seen that the second configuration with the least number of estimators provides the shortest standard deviation, and the third configuration with the highest number of estimators, the largest standard deviation. All three configurations seize an approximately equal highest classification accuracy score, which is set at 0.9697. We can conclude that using an arbitrary number of estimators (unless too low), high classification accuracy can be scored, but standard deviation increases, due to a possible over-fitting problem. It is desired to lower the classification method complexity maximally to avoid such problems.

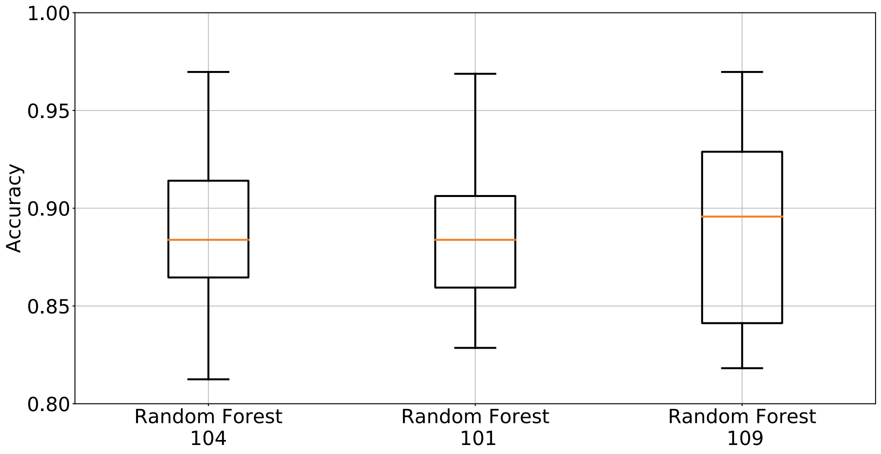

Fig. 4. Top three configurations found on the Ecoli dataset

The results according to the four statistical measures are illustrated in Table 8, from which it can be seen that Cohen's kappa κ increases significantly

Table 8. Statistical measures obtained on the Ecoli dataset

Statistical measure	Statistical score
Accuracy	0.9412
Precision	0.9373
Cohen's kappa, κ	0.9017
F_1-score	0.9318

compared to the Yeast dataset. This indicates very high agreement between true and predictive classes, and might highlight the more predictive (correlative) nature of the Ecoli dataset. Using the optimal classification method configuration, $Accuracy = 0.9412$ is improved drastically compared to the cross-validation $Accuracy = 0.8899$.

4.3 The Results on the Abalone Dataset

The last experiment was conducted on the Abalone dataset. In line with this, the results of the NiaAML for composing the classification pipeline obtained on this dataset are presented in Table 9, from which it can be seen that ERT and bagging classification methods proved to be the most beneficial results. Overall *Accuracy* is the lowest among the three datasets, reaching barely over 0.55, which might indicate the increased complexity (i.e., number of instances) of the Abalone dataset. In our opinion, this is also the main reason why the RF performed worse.

Table 9. Numerical results obtained on the Abalone dataset

Place	Feature selection	Feature scaling	Classification method	*Accuracy*
1	[8/8]	n/a	Extremely randomized trees [110]	0.5650
2	[8/8]	n/a	Bagging [81]	0.5545
3	[8/8]	n/a	Bagging [83]	0.5542

The same results are presented graphically in Fig. 5. As can be seen from the figure, the first configuration reaches the highest overall *Accuracy*. ERTs are comprehensive classification methods which perform well for complex datasets. Second configuration, i.e., Bagging [81], scores the lowest standard deviation among them but produces two outliers, i.e., black spots significantly higher and lower than the boxplot.

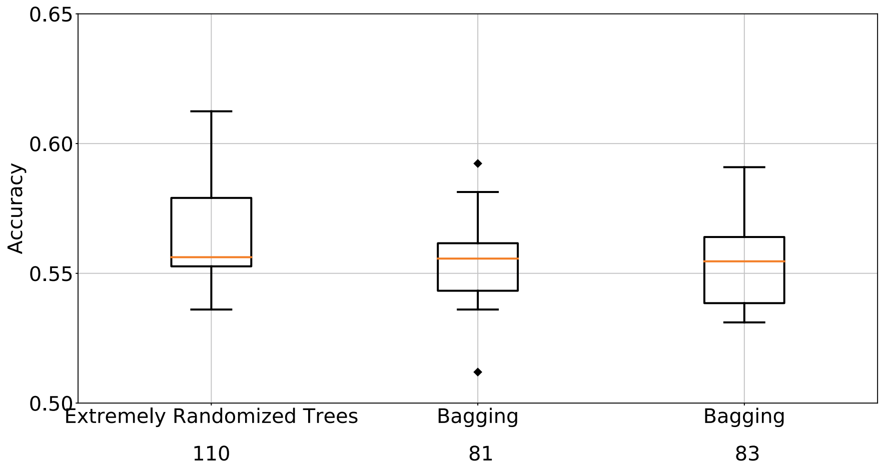

Fig. 5. Top three configurations found on the Abalone dataset

Finally, the results according to four statistical measures are depicted in Table 10, from which it can be seen that Cohen's kappa κ indicates fair agreement only. Although the $Accuracy = 0.5754$ is improved, compared to cross-validation $Accuracy = 0.5650$, one should consider the obtained predictive performance. The *Precision* also scores the lowest value among all three datasets, which indicates an increase of FP predictions.

Table 10. Statistical measures obtained on the Abalone dataset

Statistical measure	Statistical score
Accuracy	0.5754
Precision	0.5701
Cohen's kappa, κ	0.3589
F_1-score	0.5725

4.4 Summary

In order to compare the quality of classification pipelines, the best results obtained on the various datasets are presented in Fig. 6. The following conclusions can be summarized after analyzing the results: The Yeast and Ecoli datasets are shorter (i.e., 1,484 and 336 instances), compared to the Abalone dataset consisting of 4,177 instances. Therefore, the RF overcomes the results of the other classification methods by classifying the shorter datasets. However, by increasing the number of instances, e.g., the Abalone dataset, more comprehensive methods such as ERT and Bagging emerge.

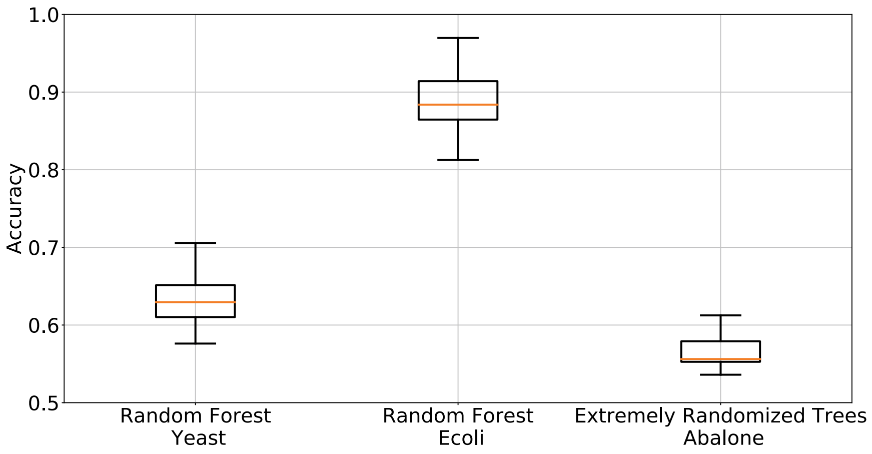

Fig. 6. The best classification pipelines obtained on the three datasets

The results obtained by NiaAML are comparable or better than the other domain experts, e.g., for the Yeast dataset authors [36] report $Accuracy = 0.5792$ on a so-called infinite latent SVM classifier. This is significantly less than in our case, where we reached $Accuracy = 0.6465$. For the Ecoli dataset, the following accuracies were reported by [24]: $Accuracy = 0.8214$ for the J48 decision tree classifier, $Accuracy = 0.8095$ for the Ridor classifier and $Accuracy = 0.8125$ for the JRip classifier. The authors in [30] for the best case report the mean $Accuracy = 0.8880$ by cross-validation. Using the NiaAML, we scored $Accuracy = 0.9412$ for evaluation, and $Accuracy = 0.8899$ by cross-validation. While the evaluation is significantly improved, cross-validation is comparable. Review of [18] revealed following results for the Abalone dataset: $Accuracy = 0.5422$ for decision trees, $Accuracy = 0.5478$ for Naïve Bayes, $Accuracy = 0.5393$ for k-nearest neighbors, and $Accuracy = 0.5456$ for SVM. The proposed approach gives $Accuracy = 0.5754$, which is at least comparable to the domain experts.

5 Conclusions and Future Work

Recently, ML methods like classification became useful tools in various sciences. Unfortunately, using these methods is far from being easy. Typically, these methods are controlled by many hyperparameters, where the optimal setting depends on the problem to be solved. Fortunately, the different methods solving a particular step in the ML phase, like feature selection/scaling, classification, and HPO, can be composed into pipelines and can be executed one after another. Therefore, a new discipline of the ML has emerged, i.e., AutoML, with the goal to help the ordinary users by applying the ML methods. The AutoML is capable of selecting the most appropriate ML methods, as well as finding their optimal parameters. Normally, the algorithms for composing the ML methods into ML pipelines are of deterministic nature.

In our study, we go a step further, and propose the stochastic NiaAML for composing the classification pipelines. Actually, the stochastic NiaAML is capable of: (1) performing automated feature selection and feature scaling to reduce the complexity of a dataset, (2) classifier selection, and (3) HPO to find the optimal configuration of the classifier. Classifier configurations are tested using cross-validation.

The proposed NiaAML was tested on three different ML datasets: Yeast, Ecoli, and Abalone. Although feature selection was applied to each dataset, it was not found to be beneficial due to the too low number of features. The solution thus might help non-technical users obtain good classification performance. It was found that NiaAML searches successfully for the optimal classification method and its configuration. As a result, we can conclude that the obtained results are comparable to those proposed by domain experts.

In future, we would like to implement the NiaAML framework for detection and accounting the imbalanced datasets. Furthermore, constrained feature selection might be proposed, where a user could specify the maximal number of features to be incorporated into the classifier. Nevertheless, a Web graphical user interface and automated visualization framework might be desired as well.

References

1. Aggarwal Charu C (2014) Data classification: algorithms and applications. Chapman and Hall/CRC, Chapman & Hall/CRC data mining and knowledge discovery series
2. Bishop Christopher M (2007) Pattern recognition and machine learning, 5th edn. Springer, Information Science and Statistics
3. Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140
4. Breiman L (2001) Random forests. Mach Learn 45(1):5–32
5. Cleveland William S (2014) Data science: an action plan for expanding the technical areas of the field of statistics. Stat Anal Data Mining 7:414–417
6. Cohen J (1960) A coefficient of agreement for nominal scales. Educ Psychol Measure 20(1):37–46

7. Costa VO, Rodrigues CR (2018) Hierarchical ant colony for simultaneous classifier selection and hyperparameter optimization. In: 2018 IEEE congress on evolutionary computation (CEC). IEEE, pp 1–8
8. de Sá AGC, Pinto WJGS, Oliveira LOVB, Pappa GL (2017) RECIPE: a grammar-based framework for automatically evolving classification pipelines. In: European conference on genetic programming. Springer, pp 246–261
9. Dey N (2017) Advancements in applied metaheuristic computing. IGI Global
10. Dua D, Graff C (2017) UCI machine learning repository
11. Eberhart R, Kennedy J (1995) Particle swarm optimization. In: Proceedings of ICNN '95—international conference on neural networks, vol 4, pp 1942–1948
12. Eiben AE, James E (2015) Introduction to evolutionary computing, 2nd edn. Springer Publishing Company, Incorporated, Smith
13. Fister I Jr, Yang X-S, Fister I, Brest J, Fister D (2013) A brief review of nature-inspired algorithms for optimization. Elektrotehniški vestnik 80(3):116–122
14. Geurts P, Ernst D, Wehenkel L (2006) Extremely randomized trees. Mach Learn 63(1):3–42
15. Gijsbers P (2018) Automatic construction of machine learning pipelines. Master's thesis, Eindhoven University of Technology
16. Gupta N, Khosravy M, Patel N, Senjyu T (2018) A bi-level evolutionary optimization for coordinated transmission expansion planning. IEEE Access 6:48455–48477
17. Gupta N, Khosravy M, Patel N, Sethi I (2018) Evolutionary optimization based on biological evolution in plants. Proc Comput Sci 126:146–155
18. Herranz J, Matwin S, Nin J, Torra V (2010) Classifying data from protected statistical datasets. Comput Sec 29(8):875–890
19. Holzinger A, Dehmer M, Jurisica I (2014) Knowledge discovery and interactive data mining in bioinformatics—state-of-the-art, future challenges and research directions. BMC Bioinf 15(6):I1
20. Hutter F, Kotthoff L, Vanschoren J (eds) (2019) Automatic machine learning: methods, systems, challenges. Series on challenges in machine learning. Springer
21. Kang KC, Cohen SG, Hess JA, Novak WE, Peterson AS (1990) Feature-oriented domain analysis (FODA) feasibility study. Technical report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA
22. Koza John R (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge, MA, USA
23. Mirjalili S, Mirjalili SM, Lewis A (2014) Grey wolf optimizer. Adv Eng Softw 69:46–61
24. Mohamed WNHW, Salleh MNM, Omar AH (2012) A comparative study of reduced error pruning method in decision tree algorithms. In: 2012 IEEE international conference on control system, computing and engineering. IEEE, pp 392–397
25. Olson RS, Bartley N, Urbanowicz RJ, Moore JH (2016) Evaluation of a tree-based pipeline optimization tool for automating data science. In: Proceedings of the genetic and evolutionary computation conference 2016, GECCO 2016. ACM, New York, NY, pp 485–492
26. Olson RS, Moore JH (2016) TPOT: a tree-based pipeline optimization tool for automating machine learning. In: Workshop on automatic machine learning, pp 66–74
27. Rosenblatt F (1961) Principles of neurodynamics. Perceptrons and the theory of brain mechanisms. Cornell Aeronautical Lab Inc, Buffalo, NY
28. Schapire RE (1999) A brief introduction to boosting. In: Proceedings of the 16th international joint conference on artificial intelligence, IJCAI '99, vol 2. Morgan Kaufmann Publishers Inc, San Francisco, CA, pp 1401–1406

29. Schuster Stephan C (2007) Next-generation sequencing transforms today's biology. Nat Methods 5(1):16
30. Soda P, Iannello G (2010) Decomposition methods and learning approaches for imbalanced dataset: an experimental integration. In: 2010 20th international conference on pattern recognition. IEEE, pp 3117–3120
31. Stehman Stephen V (1997) Selecting and interpreting measures of thematic classification accuracy. Remote Sens Environ 62(1):77–89
32. Storn R, Price K (1997) Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J Glob Opt 11(4):341–359
33. Suykens JAK, Vandewalle J (1999) Least squares support vector machine classifiers. Neural Process Lett 9(3):293–300
34. Xavier-Júnior JC, Freitas AA, Feitosa-Neto A, Ludermir TB (2018) A novel evolutionary algorithm for automated machine learning focusing on classifier ensembles. In: 2018 7th Brazilian conference on intelligent systems (BRACIS). IEEE, pp 462–467
35. Yang X-S (2010) A new metaheuristic bat-inspired algorithm. Springer Berlin Heidelberg, Berlin, Heidelberg, pp 65–74
36. Zhu J, Chen N, Xing EP (2011) Infinite latent SVM for classification and multi-task learning. In: Advances in neural information processing systems, pp 1620–1628