# NiaAML2: An Improved AutoML Using Nature-Inspired Algorithms

Luka Pečnik(✉), Iztok Fister(✉), and Iztok Fister Jr.(✉)

Faculty of Electrical Engineering and Computer Science,
Koroška ul. 46, 2000 Maribor, Slovenia
luka.pecnik@student.um.si, {iztok.fister,iztok.fister1}@um.si

**Abstract.** Using machine learning methods in the real-world is far from being easy, especially because of the number of methods on the one hand, and setting the optimal values of their parameters on the other. Therefore, a lot of so-called AutoML methods have emerged nowadays that also enable automatic construction of classification pipelines to users, who are not experts in this domain. In this study, the NiaAML2 method is proposed that is capable of constructing the classification pipelines using nature-inspired algorithms in two phases: pipeline construction, and hyper-parameter optimization. This method improves the original NiaAML capable of this construction in one phase. The algorithm was applied to four UCI ML datasets, while the obtained results encouraged us to continue with the research.

**Keywords:** Nature-inspired algorithms · Machine learning · AutoML · Classification pipeline

## 1 Introduction

Finding a Machine Learning (ML) method for solving a certain problem usually becomes a very difficult task due to several factors, like proper preparation and data preprocessing, the suitability of the selected ML method for a given problem and setting the optimal values of hyper-parameters for the selected ML methods [7,9]. The domain of Automated Machine Learning (AutoML) has emerged for the sake of simplification and automation of some of the mentioned steps. This domain deals with the problem of finding the optimal classification pipeline automatically. The classification pipeline is a sequence of methods or algorithms, necessary for the successful implementation of the entire ML process, from data preprocessing to the point, where the results are calculated [8]. By automating the individual pipeline discovery steps, ML has become more accessible to the wider community. Researches show that, using the AutoML methods, the classification pipelines can also be constructed by non-expert users [5,12].

For this purpose, many AutoML methods have been developed such as Auto-WEKA [11] and Hyperopt-sklearn [1] that both base on Bayesian optimization.


Y. Tan and Y. Shi (Eds.): ICSI 2021, LNCS 12690, pp. 243–252, 2021.
https://doi.org/10.1007/978-3-030-78811-7_23

Auto-sklearn [5] upgraded the idea of Bayesian optimization by adding a meta-learning step. In the field of AutoML, several methods have been developed that search for classification pipelines using stochastic population-based nature-inspired algorithms. The frameworks that use such methods are TPOT [12] and RECIPE [3], which use genetic algorithms to construct the classification pipeline. One of the more interesting methods is also the NiaAML method [7], which is used to find optimal classification pipelines using the stochastic population-based nature-inspired algorithms.

In this study, an extension of the NiaAML method is presented, i.e., the improved NiaAMLv2. Indeed, the proposed NiaAMLv2 eliminates the main weakness of the original method, where the hyper-parameters' optimization is performed simultaneously with construction of the classification pipelines in a single phase. This means that only one instance of nature-inspired algorithm is needed, where the ML methods and their hyper-parameters are searched for simultaneously in each generation. The improved version divides the construction of the pipeline and hyper-parameter optimization into separate phases, where two nature-inspired instances of algorithms are applied sequentially one after another, i.e., the former by covering the construction of pipeline and the later by optimizing the hyper-parameters of the proposed ML-methods. Results of experiments for both methods are also discussed and compared using the four datasets from the UCI machine learning repository [4].

The paper is organized as follows. In Sect. 2 stochastic population-based nature-inspired algorithms are described along with using the NiaAML. The improved NiaAMLv2 is proposed in Sect. 3, while Sect. 4 describes the experiments and the obtained results. The conclusion of the paper is given in Sect. 5, where we also outline directions for the future work.

## 2 Population-Based Nature-Inspired Algorithms for AutoML

Nowadays, the nature-inspired algorithms present one of the more popular groups for solving the optimization problems [14]. According to the metaphor taken from nature, they are divided into four groups [6], where the SI-based algorithms are a subset of algorithms in the field of biology. This subset also includes Evolutionary Algorithms (EA) working on the principle of Darwin's evolution [2], which proves that a generation of organisms can survive in the environment only if they are well adapted to the environmental conditions [2]. Together with SI-based algorithms, they form a class of nature-inspired algorithms.

A classification pipeline construction can be modeled as a continuous optimization problem that can be solved with a number of optimization algorithms. Similar as the NiaAML [7], this paper focuses on the interesting approach for constructing classification pipelines using stochastic population-based nature-inspired algorithms for solving this problem.

### 2.1 NiaAML

NiaAML is distinguished from the other AutoML methods by mapping of real-valued vectors to classification pipelines. The vectors are evolved using the stochastic population-based nature-inspired algorithms. Indeed, each individual of the population represents one of the potential classification pipelines [7]. According to [9], NiaAML belongs to the process of collecting metadata or meta-learning, which includes: (1) the construction of the classification pipeline, (2) optimization of the hyper-parameters and (3) evaluation of the model. The construction of the classification pipeline with the NiaAML method demands following three mandatory steps, i.e., selection of: (1) the feature selection, (2) the feature transformation, and (3) the classifier algorithms. The aforementioned ML method can be selected from any framework of tools [7].

Individuals representing possible classification pipelines in the NiaAML method are presented in the optimization algorithm as a real-valued vector [7]:

$$\mathbf{x}_i^{(t)} = \{\underbrace{x_{i,1}^{(t)}}_{\text{FSA}}, \underbrace{x_{i,2}^{(t)}}_{\text{FTA}}, \underbrace{x_{i,3}^{(t)}}_{\text{CLA}}, \underbrace{x_{i,4}^{(t)}, \ldots, x_{i,k}^{(t)}, x_{i,k+1}^{(t)}, \ldots, x_{i,D+3}^{(t)}}_{\text{hyperparameters}}\}, \tag{1}$$

where the first three elements are intended for the selection of pipeline components, while the variable length of vector $D$ depends on the number of hyper-parameters used by the components included in the pipeline. Let us notice that the length of each solution is $D$+3. Because both observed methods, i.e., NiaAML and NiaAMLv2, operate on the same set of aforementioned selection components, the detailed description of these is illustrated in the next section.

Pseudo-code of the proposed algorithm for constructing the classification pipelines is presented in Algorithm 1, from which it can be see that the algorithm suits the general form of the SI-based algorithms. After initialization

**Algorithm 1.** A pseudo-code of the NiaAML method

```
1: population ← Initialize real valued vectors x_i regarding Eq. (1)
2: best_pipeline ← Evaluate and select the best (population)
3: while Termination condition not met do
4:     for each x_i ∈ population do
5:         x_trial ← Modify population using variation operators (x_i)
6:         pipeline ← Construct pipeline (x_trial)
7:         Evaluate (pipeline)
8:         if pipeline is better than 'Construct pipeline (x_i)' then
9:             x_i ← x_trial                          ▷ Replace the worse individual
10:        end if
11:        if pipeline is better than best_pipeline then
12:            best_pipeline ← pipeline
13:        end if
14:    end for
15: end while
16: return best_pipeline
```

and evaluation of initial population (line 1 and 2), each individual undergoes acting the variation operators (line 5) by generation of trial solution. The pipeline is constructed by appropriate genotype-phenotype mapping based on collection of selection components (line 6) and evaluated according its quality (line 7). If the quality of constructed pipeline from the trial is better than the quality of the target individual, the target is replaced with the trial (lines 8–10).

## 3 NiaAMLv2

The improved version NiaAMLv2 divides the AutoML process into two phases: (1) classification pipeline construction, and (2) hyper-parameter optimization. In line with this, two algorithms are developed for each particular phase. However, both phases are implemented by a different PSO algorithm.

The first algorithm represents the solutions of the construction process as real-valued vectors, i.e.:

$$\mathbf{x}_i^{(t)} = \{\underbrace{x_{i,1}^{(t)}}_{\text{FSA}}, \underbrace{x_{i,2}^{(t)}}_{\text{FTA}}, \underbrace{x_{i,3}^{(t)}}_{\text{CLA}}\}, \tag{2}$$

which consists of three elements denoting: the Feature Selection Algorithm (FSA), Feature Transformation Algorithm (FTA), and selection of the CLassification Algorithm (CLA). Indeed, the meaning of the three parameters is described in Table 1, from which it can be seen that the four stochastic nature-inspired population-based algorithms can be selected for feature selection, there are two feature transformation options beside the selection of the no transformation, and six classification algorithms. The implementations of all the aforementioned components were taken from the Scikit-learn Python library [13].

**Table 1.** Components of the original NiaAML method.

| Component | Abbreviation | Framework of tools |
|---|---|---|
| Feature selection | FSA | $\{DE, PSO, GWO, BA\}$ |
| Feature transformation | FTA | $\{NONE, SCALING, NORMAL\}$ |
| Classificator | CLA | $\{MLP, LS\_SVM, ADA, RF, ERT, BAG\}$ |

The second algorithm is devoted to the hyper-parameter optimization and, therefore, represents the solution as a variable vector of real-values drawn from domains of feasible values randomly, in other words:

$$\mathbf{y}_i^{(t)} = \{\underbrace{y_{i,1}^{(t)}, \ldots, y_{i,k}^{(t)}, y_{i,k+1}^{(t)}, \ldots, y_{i,D}^{(t)}}_{\text{Hyper-Parameters}}\}. \tag{3}$$

Due to the limitation of the paper, the hyper-parameters for the selected components that enter into the optimization are not illustrated explicitly.

**Algorithm 2.** A pseudo-code of the NiaAMLv2 method

*population*$_1$ ← Initialize real valued vectors $\mathbf{x}_i$ from the Eq. (2)
*population*$_2$ ← Initialize real valued vectors $\mathbf{y}_j$ from the Eq. (3)
*best_pipeline* ← Evaluate and select the best(*population*$_1$,*population*$_2$)
**while** Termination condition$_1$ not met **do**
  **for each** $\mathbf{x}_i \in$ *population*$_1$ **do**
    $\mathbf{x}_{trial}$ ← Modify using variation operators ($\mathbf{x}_i$)
    **while** Termination condition$_2$ not met **do**
      **for each** $\mathbf{y}_j \in$ *population*$_2$ **do**
        $\mathbf{y}_{trial}$ ← Modify using variation operators ($\mathbf{y}_j$)
        *pipeline* ← Construct pipeline ($\mathbf{x}_{trial}, \mathbf{y}_{trial}$)
        Evaluate (*pipeline*)
        *target* ← Construct pipeline ($\mathbf{x}_i, \mathbf{y}_j$)
        **if** *pipeline* is better than *target* **then**
          $\mathbf{x}_i \leftarrow \mathbf{x}_{trial}$
          $\mathbf{y}_j \leftarrow \mathbf{y}_{trial}$
        **end if**
        **if** *pipeline* is better than *best_pipeline* **then**
          *best_pipeline* ← *pipeline*
        **end if**
      **end for**
    **end while**
  **end for**
**end while**
**return** *best_pipeline*

The concept of the NiaAMLv2 is presented in the Algorithm 2. from which it can be seen that both phases, i.e., algorithms' selection and hyper-parameter optimization are launched sequentially one after another by two different nature-inspired algorithms. Interestingly, both algorithms use the same evaluation function (function Evaluate), i.e., pipeline accuracy expressed as follows:

$$f\left(\mathbf{x}_i^{(t)}\right) = \text{Accuracy}\left(M\left(\mathbf{x}_i^{(t)}\right)\right), \tag{4}$$

where Accuracy(.) denotes the accuracy of model $M\left(\mathbf{x}_i^{(t)}\right)$ based on classification pipeline $\mathbf{x}_i$. Thus, the accuracy is a statistical measure reflecting the bias between the true and all number of cases, in other words:

$$\text{Accuracy}\left(M\left(\mathbf{x}_i^{(t)}\right)\right) = \frac{TP + TN}{TP + TN + FP + FN}, \tag{5}$$

where $TP$ = True Positive, $TN$ = True Negative, $FP$ = False Positive and $FN$ = False Negative [7]. The task of the optimization is to find models with the maximum value of accuracy [7].

After we obtained the best pipelines found in both phases, we tested them using the stratified 10-fold cross validation and compared their results using accuracy, precision, F1-score and Cohen's $\kappa$ metrics.

## 4 Experiments and Results

The goal of our experimental work was to show that the proposed NiaAMLv2 is suitable for AutoML, on the one hand, and that it constructs classification pipelines of quality equal to, if not better than, the original NiaAML. In line with this, extensive experimental work was conducted, where all experiments were performed on an HP ProDesk 400 G6 MT computer running Microsoft Windows 10, an Intel (R) Core (TM) i7-9700 CPU @ 3.00 GHz processor, and 8 GB of installed physical memory.

When running the NiaAMLv2, we used a population of 15 individuals with 300 fitness function evaluations in both optimization steps, while we used a population of 20 individuals with 400 evaluations with the original NiaAML. Particle Swarm Optimization (PSO) [10] was used in all cases with the following parameters: cognitive component and social component set to 2.0, inertia weight had a value of 0.7 and both minimal and maximal velocities set to 1.5.

During the experimental work, we used four datasets from the UCI Machine Learning Repository [4]. The list and their characteristics are shown in Table 2, from which it can be seen that the first two consist of real-valued attributes, while the other two of mixed ones (i.e., real-valued, integer, and discrete). The Abalone dataset exposed the highest number of instances. Interestingly, the last Cylinder bands dataset also included the missing data. To impute missing data in this dataset, we used an average value for numeric features and a mode for categorical features. To encode categorical features into numeric, we used one-hot encoding.

**Table 2.** Datasets used in the experiment.

| Dataset | Type of attributes | # Instances | # Features | Missing data |
|---|---|---|---|---|
| Ecoli | Real | 336 | 8 | No |
| Yeast | Real | 1484 | 8 | No |
| Abalone | Real, integer, categorical | 4177 | 8 | No |
| Cylinder bands | Real, integer, categorical | 512 | 39 | Yes |

The results of the algorithms were compared according to four classification evaluation metrics: Accuracy, Precision, $F_1$, and Cohen's Kappa. The accuracy is already applied as part of the fitness function evaluation and is defined by using Eq. (5). The other metrics are defined as follows: The precision of the model $M(\mathbf{x}_i)$ is defined as ratio between true positive and all number of cases:

$$\text{Precision}\left(M\left(\mathbf{x}_i\right)\right) = \frac{TP}{TP + FP}. \tag{6}$$

The $F_1$ metric is calculated using Precision and Recall:

$$\mathrm{F}_1\left(M\left(\mathbf{x}_i\right)\right) = 2 \cdot \frac{\text{Precision}\left(M\left(\mathbf{x}_i\right)\right) \cdot \text{Recall}\left(M\left(\mathbf{x}_i\right)\right)}{\text{Precision}\left(M\left(\mathbf{x}_i\right)\right) + \text{Recall}\left(M\left(\mathbf{x}_i\right)\right)}, \tag{7}$$

where Recall $= \frac{TP}{TP+FN}$. Actually, metrics Precision, Recall, and $F_1$ need to be calculated, when we have to do with the classification to two classe.

The Cohen's Kappa is defined as follows:

$$\kappa\left(M\left(\mathbf{x}_i\right)\right) = \frac{n\sum_{i=1}^{k} CM_{i.i} - \sum_{i=1}^{k} CM_{i,*}CM_{*,i}}{n^2 - \sum_{i=1}^{k} CM_{i,*}CM_{*,i}}, \tag{8}$$

where $CM_{i,*}$ is sum of elements in the $i$-th row of confusion matrix $CM$ and $CM_{*,i}$ the sum of the $i$-t columns of the same matrix.

In the remainder of the paper, the results obtained by constructing the optimal classification pipeline are presented in detail.

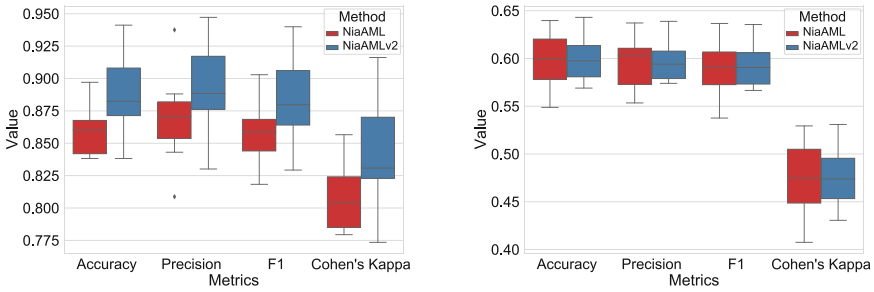
### 4.1 The Results of Constructing the Pipeline on UCI-ML Datasets

The presentation of the results obtained by optimization of both AutoML methods is divided into two parts: In the first part, the optimal classification pipelines are exposed according to each dataset per specific version of the AutoML method. The second part is devoted for comparing the results between the original NiaAML and the improved NiaAMLv2 according to four classification evaluation metrics (i.e., Accuracy, Precision, F1, and Cohen's Kappa), and depicted in the corresponding boxplots. The boxplots show the results of the aforementioned values of metrics obtained by stratified 10-fold cross validation over pipelines optimized by both methods. Each of them presents the average value calculated in each of ten folds.

The boxplots, comparing the results between NiaAml and NiaAMLv2, are presented in Figs. 1 and 2, where each figure is divided into diagrams illustrating one of the observed datasets. In the diagrams, the results of the NiaAML method are presented in the red boxes, while the results of the NiaAMLv2 method in the blue boxes.

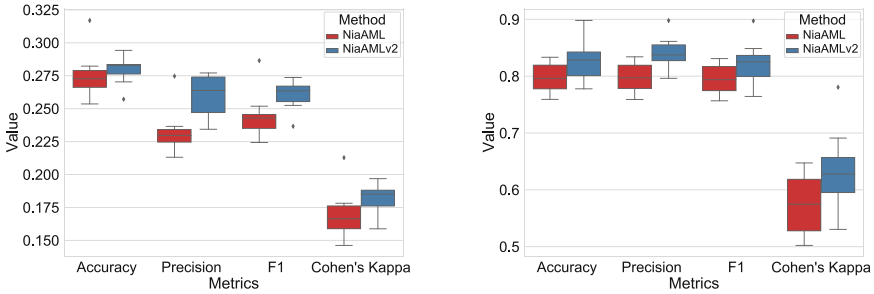
In the case of the Ecoli dataset (Fig. 1(a)), both classification pipelines selected all features. Despite the fact that a few outliers are visible in both cases, the classification pipeline obtained by the NiaAMLv2 shows the better results on average. All features were also selected in the case of the Yeast dataset by both classification pipelines (Fig. 1(b)). On average, the results were very similar, but the pipeline obtained by the NiaAMLv2 method had a smaller interquartile range.

The results of constructing the optimal classification pipelines for the Abalone dataset by the NiaAMLv2 method removed one feature, while the NiaAML used only four features. The classification pipeline optimized with the NiaAMLv2 method showed better results overall (Fig. 2(a)). The pipeline optimized with the NiaAMLv2 method showed better results on average by also optimizing the Cylinder bands dataset, where there are also some visible outliers (Fig. 2(b)).




a: Ecoli dataset. b: Yeast dataset.

**Fig. 1.** Results of the NiaAML and NiaAMLv2 pipelines for the Ecoli and Yeast datasets.




a: Abalone dataset. b: Cylinder bands dataset.

**Fig. 2.** Results of the NiaAML and NiaAMLv2 pipelines for the Abalone and Cylinder bands datasets.

### 4.2 Time Complexity

The time complexity of the NiaAMLv2 was much higher than the time complexity of the original NiaAML as can be seen in Table 3, where the time complexities for each observed database in seconds are presented for both AutoML methods. Moreover, the ratios between time complexities of NiaAML and NiaAMLv2 in percents have been added to the table. As can be seen from the table, the time complexity of the proposed AutoML method was more than 100 times higher than the original one. However, the reason for this must be searched for in separating of the hyper-parameter optimization into an independent phase.

### 4.3 Discussion

Based on the results of the experiment, we found out that the classification pipelines generated by the NiaAMLv2 were slightly better in treating the Ecoli,

**Table 3.** Time complexity of optimization.

| Nr. | Dataset | Time [sec] | | Ratio |
|---|---|---|---|---|
| | | NiaAML | NiaAMLv2 | [%] |
| 1 | Ecoli | 27.48 | 29,031.30 | 0.0945 |
| 2 | Yeast | 93.49 | 33,127.02 | 0.2822 |
| 3 | Cylinder bands | 189.23 | 122,262.46 | 0.1548 |
| 4 | Abalone | 2,868.90 | 295,909.65 | 0.9695 |
| $\sum$ | | 3,538.37 | 55,4863.14 | 0.6377 |

Abalone and Cylinder bands datasets than the pipelines optimized by the original NiaAML. On the other hand, we were surprised due to observation that there were no noticeable differences in comparing the results obtained by the Yeast dataset. Indeed, we expected that the NiaAMLv2 would give much better results than NiaAML in all datasets, but this assumption was not justified in all cases. The reason for such results was most likely in the size of the search spaces introduced by continuous domains of hyper-parameters, and in the sensitivity of both methods to occasional discovery of a very good fitness value, presumably due to a random favorable distribution of data for training and testing pipelines.

## 5 Conclusion

In this article, we described the NiaAMLv2 method for automatic construction of classification pipelines that presents an extension of the original NiaAML. The main weakness of the NiaAML was that this method constructs a classification pipeline and optimizes the corresponding hyper-parameters simultaneously in one step. The proposed method separates both mentioned steps in two phases conducted sequentially one after another. The results of our experiments showed that the NiaAMLv2 outperformed the results of the original NiaAML by constructing the classification pipelines for three of the four UCI ML datasets in tests, while the achieved results on the other two datasets were similar.

In the further research, a stratified 10-fold cross validation could be used to provide less sensitivity to occasional discovery of good fitness values due to a random favorable distribution of data for training and testing pipelines. The better results would also likely be brought by discretizing the huge continuous domains of hyper-parameters into more discrete classes, and, thus, getting the search algorithm more chance to explore this search space more effectively.

**Acknowledgements.** The authors acknowledge the financial support from the Slovenian Research Agency (research core funding No. P2-0041 - Digital twin).

## References


1. Bergstra, J., Komer, B., Eliasmith, C., Yamins, D., Cox, D.D.: HyperOpt: a python library for model selection and hyperparameter optimization. Comput. Sci. Discov. **8**(1), 014008 (2015)
2. Dasgupta, D., Michalewicz, Z.: Evolutionary algorithms in engineering applications. Springer Science and Business Media (2013). https://doi.org/10.1007/978-3-662-03423-1
3. de Sá, A.G.C., Pinto, W.J.G.S., Oliveira, L.O.V.B., Pappa, G.L.: RECIPE: a grammar-based framework for automatically evolving classification pipelines. In: McDermott, J., Castelli, M., Sekanina, L., Haasdijk, E., García-Sánchez, P. (eds.) EuroGP 2017. LNCS, vol. 10196, pp. 246–261. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55696-3_16
4. Dua, D., Graff, C.: UCI machine learning repository. http://archive.ics.uci.edu/ml (2017)
5. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 28, pp. 2962–2970. Curran Associates Inc, (2015)
6. Fister Jr, I., Yang, X.-S., Fister, I., Brest, J., Fister, D.: A brief review of nature-inspired algorithms for optimization. Elektrotehniški vestnik **80**(3), 116–122 (2013)
7. Fister Jr, I., Zorman, M., Fister, D., Fister, I.: Continuous optimizers for automatic design and evaluation of classification pipelines. In: Frontier Applications of Nature Inspired Computation, pp. 281–301 (2020)
8. Guyon, I., et al.: Design of the 2015 chalearn automl challenge. In: 2015 International Joint Conference on Neural Networks (IJCNN), pp. 1–8 (2015)
9. He, X., Zhao, K., Chu, X.: AutoML: a survey of the state-of-the-art. Knowl.-Based Syst. **212** 106622 (2020)
10. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of ICNN 1995 - International Conference on Neural Networks, vol. 4, pp. 1942–1948 (1995)
11. Kotthoff, L., Thornton, C., Hoos, H.H., Hutter, F., Leyton-Brown, K.: Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. J. Mach. Learn. Res. **18**(1), 826–830 (2017)
12. Olson, R.S., Bartley, N., Urbanowicz, R.J., Moore, J.H.: 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, pp. 485–492, New York, NY, USA (2016)
13. Pedregosa, F., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
14. Vrbančič, G., Brezočnik, L., Mlakar, U., Fister, D., Fister Jr, I.: NiaPy: Python microframework for building nature-inspired algorithms. J. Open Source Softw. **3**, **613** (2018)