SoftwareX 29 (2025) 101974

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

NiaAML: AutoML for classification and regression pipelines

Iztok Fister Jr. [b,a], Laurenz A. Farthofer [d], Luka Pečnik [b], Iztok Fister [b], Andreas Holzinger [a,c,*]

[a] Human-Centered AI Lab, Institute of Forest Engineering, Department of Forest and Soil Sciences, University of Natural Resources and Life Sciences Vienna, Austria
[b] Faculty of Electrical Engineering and Computer Science, University of Maribor, Koroška cesta 46, 2000 Maribor, Slovenia
[c] Institute of Interactive Systems and Data Science, Graz University of Technology, Austria
[d] Institute of Computer Graphics and Vision, Graz University of Technology, Austria

ARTICLE INFO

Keywords:
AutoML
Classification
Nature-inspired algorithms
Optimization

ABSTRACT

In this paper we present NiaAML, an AutoML framework that we have developed for creating machine learning pipelines and hyperparameter tuning. The composition of machine learning pipelines is presented as an optimization problem that can be solved using various stochastic, population-based, nature-inspired algorithms. Nature-inspired algorithms are powerful tools for solving real-world optimization problems, especially those that are highly complex, nonlinear, and involve large search spaces where traditional algorithms may struggle. They are applied widely in various fields, including robotics, operations research, and bioinformatics. This paper provides a comprehensive overview of the software architecture, and describes the main tasks of NiaAML, including the automatic composition of classification and regression pipelines. The overview is supported by an practical illustrative example.

Code metadata

Current code version	v2.1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-24-00438
Code Ocean compute capsule	None
Legal Code License	MIT Licence
Code versioning system used	Git
Software code languages, tools, and services used	Python
Compilation requirements, operating environments & dependencies	Python >= v3.9
If available, link to developer documentation/manual	https://niaaml.readthedocs.io/en/latest/
Support email for questions	iztok@iztok-jr-fister.eu

1. Motivation and significance

A machine learning pipeline is a structured sequence of steps or stages that data undergo to move from raw input to a final machine learning model that can make predictions or classifications. It automates the end-to-end workflow of machine learning, ensuring that the process is systematic, efficient, and repeatable [1–3].

Designing robust machine learning pipelines is a challenging task, especially for those who do not have in-depth expertise in this area, because often extensive experience is required to overcome the complexities involved.

Machine learning pipelines usually consist of several stages, including data preprocessing techniques, the selection of classification or regression methods, parameter tuning, and the whole modeling process [4,5].

These steps are also very time-consuming, as they involve many combinations of different parameter settings for each classification or regression method. Furthermore, experiments with a variety of architectures in deep learning networks require significant hardware resources [6]. Evaluating the pipelines by making available numerous metrics is another challenge for the application.

Automated Machine Learning (AutoML) [7] addresses all of these challenges by simplifying critical steps in the machine learning process, aiming to democratize machine learning for a wider audience. There

* Corresponding author at: Human-Centered AI Lab, Institute of Forest Engineering, Department of Forest and Soil Sciences, University of Natural Resources and Life Sciences Vienna, Austria.
E-mail address: andreas.holzinger@human-centered.ai (Andreas Holzinger).

https://doi.org/10.1016/j.softx.2024.101974
Received 15 August 2024; Received in revised form 6 November 2024; Accepted 8 November 2024
Available online 20 November 2024

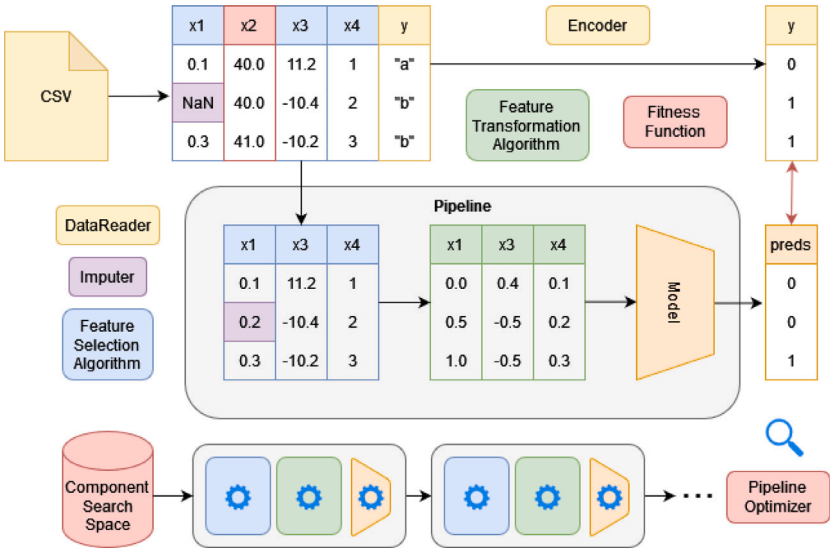

Fig. 1. An overview of the software architecture of NiaAML.

are numerous AutoML approaches. In this paper, we focus on stochastic, population-based nature-inspired algorithms for optimization [8].

Stochastic population-based nature-inspired algorithms are powerful tools for solving optimization problems across various domains. Their popularity has increased steadily due to their wide applicability in different problem areas and real-world scenarios.

The observation that nature-inspired optimization methods do not rely on gradients is indeed significant, especially when comparing them to popular deep learning approaches. Traditional deep learning models, such as those using backpropagation, depend heavily on gradient information to update the model's parameters, which can limit their applicability to problems where gradients are difficult or impossible to compute, or where the objective function is non-differentiable, highly irregular, or noisy. Stochastic weight update in backpropagation improves convergence probability and speed, while being simple to implement across various network topologies, making it a viable alternative to classical ordered updates without added complexity or significant loss in convergence quality [9].

In contrast, nature-inspired algorithms like Genetic Algorithms, Particle Swarm Optimization, and Simulated Annealing, operate without requiring gradient information, allowing them to explore complex, multimodal search spaces more effectively. This makes them particularly versatile, and applicable to a broader range of optimization problems, including those where gradient-based methods may struggle or fail entirely, thus offering a distinct advantage in certain scenarios [10,11].

AutoML can also be modeled as an optimization problem, where the task of a stochastic algorithm is to compose the pipeline from a set of data preprocessing methods and classifiers, as well as to perform extensive parameter tuning of classifiers and feature selection algorithms automatically.

The concept of using stochastic methods for optimizing the composition and the parameterization of machine learning pipelines jointly was proposed in [12], while the initial software was introduced in the subsequent papers [13,14]. Since the original NiaAML implementation has undergone some refinements and extensions, in this paper we outline the full potential of the NiaAML framework, highlighting its advantages, presenting the latest architecture in detail, and providing examples of how to use the framework. Additionally, we also discuss future challenges for this framework.

2. Software description

NiaAML is a Python package designed for automated machine learning based on stochastic population-based nature-inspired optimization algorithms. Its primary function is to discover optimal classification or regression pipelines, comprising data selection, preprocessing, and model components automatically (as illustrated in Fig. 1). Notably, NiaAML optimizes both the selection of components and their hyperparameters for a given task. The quality of the pipeline is evaluated using the fitness functions originally proposed in the paper by Pečnik et al. [14]. A Graphical User Interface (GUI) has also been developed for NiaAML, making it much easier for users, especially non-programmers, to work with the framework.[1]

2.1. Software architecture

Developed for Python versions 3.9 and above, NiaAML leverages nature-inspired optimization algorithms from the NiaPy framework [15]. The Application Programming Interface (API) was designed to be compatible with the popular scikit-learn library [16], enabling code reuse from that ecosystem. The package utilizes established data formats, including numpy arrays [17] and pandas data frames [18]. To facilitate code extension and maintainability, NiaAML adheres to object-oriented design principles, separating each component into a class (detailed in the following subsections). The components are parameterized explicitly using the `set_params` method from scikit-learn, avoiding re-initializations during optimization, and reducing the memory footprint of the application [19].

The individual components are composed into a central `Pipeline` class, which gives users access to many convenient methods including inference (application of the pipeline on new samples), serialization and logging.

NiaAML can be integrated seamlessly into Python scripts or Jupyter Notebooks for exploratory data analysis and ad-hoc model development. Additionally, a Command Line Interface (CLI) and a RESTful web API (REST API)[2] are provided to foster reproducibility and model lineage, making it easy to integrate with tools like DVC [20].

[1] https://github.com/firefly-cpp/NiaAML-GUI
[2] https://github.com/alenrajsp/NiaAML-API

2.2. Data loading and preprocessing

NiaAML operates on tabular input data and provides two implementations of the `DataReader` class: a `CSVDataReader` for loading CSV files and a `BasicDataReader` for in-memory objects. Certain preprocessing steps, such as data imputation and target encoding, are always performed and are therefore excluded from the component selection optimization.

2.3. Pipelines

A `Pipeline` represents a sequential composition of data processing components and a model. In the background, NiaAML translates these pipelines into optimization problems for NiaPy.

2.3.1. Feature selection

It is often beneficial to consider only a subset of features as input for the model. How this subset is chosen is determined by the concrete implementation of a `FeatureSelectionAlgorithm` component. The following algorithms are currently included: VarianceThreshold [16], Self-Adaptive Differential Evolution (jDEFSTH) [15], SelectPercentile [16], Particle Swarm Optimization (PSO) [21], Bat Algorithm (BA) [22], Differential Evolution (DE) [23], and SelectKBest [16]. Since NiaAML is based on the NiaPy framework [15], users can utilize any of the population-based nature-inspired algorithms that are already included in NiaPy.[3]

2.3.2. Feature transformation

Transforming features, e.g. scaling, can be very beneficial for the performance of the model. NiaAML's `FeatureTransformAlgorithm` [15] method provides several implementations of these methods as follows: Normalizer, StandardScaler, MaxAbsScaler, RobustScaler and QuantileTransformer.

2.3.3. Classification and regression models

Because NiaAML was originally designed exclusively for classification tasks, each model is still referred to as a `Classifier`. This was a deliberate choice to avoid breaking changes when introducing regression models. All models follow the API from scikit-learn [19], which allows re-using many implementations. The package currently includes: random forest, multi-layer perceptron, linear SVC, AdaBoost, bagging, extremely randomized trees, decision tree, decision tree regression, K-Neighbors, Gaussian process, Gaussian process regression, Gaussian NB, quadratic discriminant analysis, linear regression, ridge regression and Lasso regression.

2.4. Optimization and inference

The main entry point for users is the `PipelineOptimizer` class, which defines the components for the pipeline optimization (see Fig. 3). This class handles the dual optimization of the component selection and the parameterization of each component.

Once such a pipeline optimizer is defined, an optimization using one of the provided fitness functions and optimization algorithms can be triggered with the `optimize` method. The returned pipeline can be serialized (saved and loaded) with `export` and `load`.

To use the pipeline for inference, new samples are passed to `run`.

[3] For the full list of included algorithms check the following repository link.[4]

[4] https://github.com/NiaOrg/NiaPy/blob/master/Algorithms.md

2.5. Time complexity of proposed framework

Stochastic optimization algorithms (like Evolutionary Algorithms and Swarm Intelligence algorithms) incorporate randomness and probability in the variation operators (like crossover and mutation) to efficiently find the optimum or near optimum solutions. Therefore, these algorithms are compared with each other according to the characteristics as follows:

- convergence speed,
- ability to avoid local optima,
- computational efficiency,
- scalability.

On the other hand, the ML complexity depends on the number of training examples necessary or sufficient to learn hypotheses of a given accuracy. As a result, the ML complexity depends on:

- size or expressiveness of the hypothesis space,
- accuracy to which target concept must be approximated,
- probability with which the learner must produce a successful hypothesis,
- manner in which training examples are presented, e.g. randomly or by query to an oracle.

In general, the time complexity of the proposed framework depends on the problem to be solved.

3. Illustrative examples

The NiaAML framework now offers multiple ways for users to interact with it. These include the NiaAML GUI and the NiaAML API, which provide convenient options for users. Another method involves using the well-documented command-line interface (CLI), as shown in Fig. 2. However, the most flexible approach is to write short programs using the NiaAML framework directly in Python.

An example of such a program is presented in Fig. 3, where we see a basic script designed to find the optimal classification combination. In the first part of the code, the NiaAML and numpy libraries are loaded. The second part involves generating a random synthetic dataset for testing purposes. In the third part, the problem is defined by passing the data to the appropriate class, selecting the classifiers to test, and choosing the feature selection and transformation algorithms. Finally, we configure the remaining parameters, such as the metric for fitness function calculation (in the current case, we use Accuracy), the control parameters for the search algorithm (including population size and the number of function evaluations), and the stochastic population-based nature-inspired algorithm that will search for the best combination. In this example, Particle Swarm Optimization is selected for both stages: pipeline composition and hyperparameter tuning. Users are encouraged to visit the following URL[5] for more detailed and well-documented examples.

4. Impact

Democratizing machine learning for a broad audience is crucial in today's data-driven world. This aligns well with the principles of human-centered AI, making these technologies accessible, understandable, and usable by a diverse range of people, not just experts in the field. By democratizing machine learning, the goal is to empower more individuals and organizations to leverage such technologies, ensuring that the benefits of AI are widely distributed and that its development is inclusive. This approach fosters transparency, collaboration, and equity, which are key tenets of human-centered AI [24].

[5] https://github.com/firefly-cpp/NiaAML/tree/master/examples

```
NiaAML on  master [$?] is  v2.1.0 via  v3.11.2 (niaaml-py3.11) took 15s
❯ niaaml optimize .\tests\tests_files\dataset_header_classes.csv --number-of-pipeline-evaluations 2 --number-of-inner-ev
aluations 2
2024-11-05 10:13:54.908 | INFO     | niaaml.cli:optimize:42 - 📄 reading `tests\tests_files\dataset_header_classes.csv`
2024-11-05 10:13:54.913 | INFO     | niaaml.cli:optimize:51 - 🦾 start the optimization process ...
INFO:niaaml:Currently optimizing 1: classifier - Multi Layer Perceptron, feature selection algorithm - Select K Best, fe
ature transform algorithm - Normalizer
INFO:niaaml:Evaluation 1
INFO:niaaml:Evaluation 2
INFO:niaaml:Currently optimizing 2: classifier - AdaBoost, feature selection algorithm - Variance Threshold, feature tra
nsform algorithm - Normalizer
INFO:niaaml:Evaluation 1
INFO:niaaml:Evaluation 2
2024-11-05 10:13:55.226 | SUCCESS  | niaaml.cli:optimize:65 - 💾 saving optimized pipeline to `pipeline.ppln`
```

Fig. 2. An example of using the command line interface of NiaAML to optimize a classification pipeline.

```python
from niaaml import PipelineOptimizer, Pipeline
from niaaml.data import BasicDataReader
import numpy

data_reader = BasicDataReader(
    x=numpy.random.uniform(low=0.0, high=15.0, size=(50, 3)),
    y=numpy.random.choice(['Class 1', 'Class 2'], size=50)
)

pipeline_optimizer = PipelineOptimizer(
    data=data_reader,
    classifiers=['AdaBoost', 'Bagging', 'MultiLayerPerceptron', 'RandomForest', 'ExtremelyRandomizedTrees'],
    feature_selection_algorithms=['SelectKBest', 'SelectPercentile', 'ParticleSwarmOptimization', 'VarianceThreshold'],
    feature_transform_algorithms=['Normalizer', 'StandardScaler']
)

pipeline = pipeline_optimizer.run('Accuracy', 15, 15, 300, 300, 'ParticleSwarmAlgorithm', 'ParticleSwarmAlgorithm')
```

Fig. 3. A classification pipeline definition and optimization.

As the data revolution pushes many individuals beyond fundamental data analysis to discover new insights across various fields, modern AutoML solutions are essential. They speed up experimental work and eliminate the need for a manual trial-and-error approach, thus reducing the human effort required. NiaAML addresses both these aspects, helping users manage machine learning pipelines efficiently and accelerate the knowledge discovery process from datasets [25].

One of the strengths of NiaAML is its highly extensible architecture, which makes it easy to add new components, whether for data preprocessing or incorporating new classifiers. Additionally, being written in Python and well-documented, NiaAML is ready for adoption by users in academic and industrial settings where Python is widely used.

5. Conclusion and future outlook

In conclusion, this paper introduced the NiaAML AutoML framework, developed for the automatic composition of machine learning pipelines and hyperparameter tuning. By framing pipeline composition as an optimization challenge, the framework leverages stochastic, population-based nature-inspired algorithms to find optimal solutions. The software architecture was outlined meticulously, with a focus on NiaAML's key functions of generating classification and regression pipelines automatically. The framework's capabilities were demonstrated further through practical examples.

The potential for extending NiaAML to incorporate deep learning techniques, neural architecture search (NAS), and support for explainable artificial intelligence (XAI) opens up several intriguing questions to stimulate future research:

(1) How can nature-inspired algorithms be effectively combined with deep learning techniques to optimize neural network training beyond traditional gradient-based methods? Particularly, how can hybrid approaches, including a human-in-the-loop bringing in conceptual understanding and domain knowledge [26], lead to faster convergence, better generalization, and more robust training in complex, high-dimensional spaces.

(3) How can the NiaAML framework be extended to support the development of explainable AI models, and what nature-inspired techniques are most effective in enhancing model interpretability while maintaining high performance? This question seeks to explore the intersection of nature-inspired algorithms and explainable AI, aiming to develop models that not only perform well but also provide transparent, interpretable decisions, which is critical in sensitive application domains including medicine [27] and bioinformatics.

4. In what ways can NiaAML be adapted to optimize hyperparameters and architectures of deep neural networks in real-time or under dynamic conditions, such as changing data distributions or computational constraints? This research would investigate the potential for dynamic optimization in deep learning, using nature-inspired algorithms to adapt models on-the-fly to changing conditions, which could be particularly valuable in real-time systems or environments with non-stationary data.

5. How can nature-inspired approaches within the NiaAML framework be utilized to enhance the robustness of deep learning models against adversarial attacks, data noise, and other forms of uncertainty? This question explores the potential for nature-inspired algorithms to contribute to the development of more resilient models, potentially offering new ways to mitigate the risks posed by adversarial examples and other vulnerabilities.

CRediT authorship contribution statement

Iztok Fister Jr.: Writing – review & editing, Writing – original draft, Validation, Software, Project administration, Methodology, Investigation, Conceptualization. **Laurenz A. Farthofer:** Writing – original draft, Visualization, Validation, Software, Formal analysis, Data curation. **Luka Pečnik:** Writing – original draft, Visualization, Validation, Software, Formal analysis. **Iztok Fister:** Writing – original draft, Validation, Supervision, Formal analysis. **Andreas Holzinger:** Writing – review & editing, Writing – original draft, Supervision, Resources, Investigation, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was funded in part by the Austrian Science Fund (FWF) https://doi.org/10.55776/P32554. For open access purposes, the authors have applied a CC BY public copyright license to any author accepted manuscript version arising from this submission. This publication reflects only the authors' view and the founder is not responsible for any use that may be made of the information it contains. The authors are very grateful to the international research community, having already reported some bugs in the NiaAML software and to all who made improvements to the framework (see all the contributors[6]). Last but not least the authors are grateful for the valuable comments of the three reviewers.

Data availability

We have shared the link to the repo in the paper.

References

[1] Drori I, et al. AlphaD3M: Machine learning pipeline synthesis. 2021, arXiv preprint arXiv:2111.02508.
[2] Olson RS, Moore JH. TPOT: A tree-based pipeline optimization tool for automating machine learning. In: Workshop on automatic machine learning. PMLR; 2016, p. 66–74.
[3] Kotthoff L, Thornton C, Hoos HH, Hutter F, Leyton-Brown K. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. J Mach Learn Res 2017;18(25):1–5.
[4] Jordan MI, Mitchell TM. Machine learning: Trends, perspectives, and prospects. Science 2015;349(6245):255–60. http://dx.doi.org/10.1126/science.aaa8415.
[5] Russell SJ, Norvig P. Artificial intelligence: a modern approach. Pearson; 2016.
[6] Gharibi G, Walunj V, Alanazi R, Rella S, Lee Y. Automated management of deep learning experiments. In: Proceedings of the 3rd international workshop on data management for end-to-end machine learning. 2019, p. 1–4.
[7] Hutter F, Kotthoff L, Vanschoren J. Automated machine learning: methods, systems, challenges. Cham: Springer Nature; 2019, http://dx.doi.org/10.1007/978-3-030-05318-5.
[8] Fister Jr. I, Yang X-S, Fister I, Brest J, Fister D. A brief review of nature-inspired algorithms for optimization. Elektrotehniški Vestnik 2013;80(3):116–22.
[9] Koščak J, Jakša R, Sinčák P. Stochastic weight update in the backpropagation algorithm on feed-forward neural networks. In: The 2010 international joint conference on neural networks. IJCNN, IEEE; 2010, p. 1–4. http://dx.doi.org/10.1109/IJCNN.2010.5596870.
[10] Freitas AA. A survey of evolutionary algorithms for data mining and knowledge discovery. In: Ghosh A, Tsutsui S, editors. Advances in evolutionary computing. Springer; 2003, p. 819–45. http://dx.doi.org/10.1007/978-3-642-18965-4_33.
[11] Freitas AA. Evolutionary algorithms for data mining. In: Maimon O, Rokach L, editors. Data mining and knowledge discovery handbook. New York: Springer; 2010, p. 435–67.
[12] Fister I, Zorman M, Fister D, Fister I. Continuous optimizers for automatic design and evaluation of classification pipelines. In: Khosravy M, Gupta N, Patel N, Senjyu T, editors. Frontier applications of nature inspired computation. springer tracts in nature-inspired computing.. Singapore: Springer; 2020, p. 281–301. http://dx.doi.org/10.1007/978-981-15-2133-1_13.
[13] Pečnik L, Fister I. Niaaml: Automl framework based on stochastic population-based nature-inspired algorithms. J. Open Source Softw. 2021;6(61):2949.
[14] Pečnik L, Fister I, Fister I. NiaAML2: An improved automl using nature-inspired algorithms. In: Advances in swarm intelligence: 12th international conference, ICSI 2021, qingdao, China, July 17–21, 2021, proceedings, part II 12. Springer; 2021, p. 243–52.
[15] Vrbančič G, Brezočnik L, Mlakar U, Fister D, Fister Jr I. NiaPy: Python microframework for building nature-inspired algorithms. J. Open Source Softw. 2018;3(23). http://dx.doi.org/10.21105/joss.00613.
[16] Pedregosa F, et al. Scikit-learn: Machine learning in Python. J Mach Learn Res 2011;12:2825–30.
[17] Harris CR, et al. Array programming with NumPy. Nature 2020;585(7825):357–62. http://dx.doi.org/10.1038/s41586-020-2649-2.
[18] pandas development team T. Pandas-dev/pandas: Pandas. 2020.
[19] Buitinck L, et al. API design for machine learning software: experiences from the scikit-learn project. In: ECML PKDD workshop: languages for data mining and machine learning. 2013, p. 108–22.
[20] Kuprieiev R, et al. DVC: Data Version Control - Git for Data & Models. 2024, http://dx.doi.org/10.5281/ZENODO.13137317, URL https://zenodo.org/doi/10.5281/zenodo.13137317.
[21] Kennedy J, Eberhart R. Particle swarm optimization. In: Neural networks, 1995. proceedings., IEEE international conference on. 4, IEEE; 1995, p. 1942–8.
[22] Yang X-S. A New Metaheuristic Bat-Inspired Algorithm. In: González JR, Pelta DA, Cruz C, Terrazas G, Krasnogor N, editors. Nature inspired cooperative strategies for optimization (NICSO 2010). Berlin, Heidelberg: Springer Berlin Heidelberg; 2010, p. 65–74. http://dx.doi.org/10.1007/978-3-642-12538-6_6.
[23] Storn R, Price K. Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. J Global Optim 1997;11(4):341–59. http://dx.doi.org/10.1023/A:1008202821328.
[24] Shneiderman B. Human-centered artificial intelligence: Reliable, safe and trustworthy. Int. J. Hum.-Comput. Interact. 2020;36(6):495–504. http://dx.doi.org/10.1080/10447318.2020.1741118.
[25] Holzinger A, Zupan M. KNODWAT: A scientific framework application for testing knowledge discovery methods for the biomedical domain. BMC Bioinformatics 2013;14(1):191. http://dx.doi.org/10.1186/1471-2105-14-191.
[26] Hudec M, Minarikova E, Mesiar R, Saranti A, Holzinger A. Classification by ordinal sums of conjunctive and disjunctive functions for explainable AI and interpretable machine learning solutions. Knowl. Based Syst. 2021;220:106916. http://dx.doi.org/10.1016/j.knosys.2021.106916.
[27] Combi C, Amico B, Bellazzi R, Holzinger A, Moore JH, Zitnik M, Holmes JH. A manifesto on explainability for artificial intelligence in medicine. Artif Intell Med 2022;133(11):102423. http://dx.doi.org/10.1016/j.artmed.2022.102423.

[6] https://github.com/firefly-cpp/NiaAML/graphs/contributors