Thefittest: evolutionary machine learning in Python

Pavel Sherstnev*

Artificial Intelligence Laboratory, Siberian Federal University, Krasnoyarsk, Russia

Abstract. Thefittest is a new Python library specializing in evolutionary optimization methods and machine learning methods that use evolutionary optimization methods. Thefittest provides both classical evolutionary algorithms and efficient modifications of these algorithms that do not have implementations in the open access. Among the advantages of the library are the performance of the implemented methods, accessibility and ease of use. The paper discusses the motivation for developing and leading the project, describes the structure of the library with examples of use, and provides a comparison with other projects with a similar development goal. Thefittest is an open-source project published on GitHub and PyPi, developed using modern methods of code analysis and testing. At the moment of writing the paper, the latest version of the library is 0.2.3.

1 Introduction

Evolutionary algorithms (EA) have at their basis a simplified scheme of evolution developed by Charles Darwin. The basic principles of such a theory of evolution are: variability. Individuals differ in some characteristics; heredity. Individuals who are offspring obtain some characteristics of their parents; natural selection. Individuals with characteristics that make them more adaptable to the environment leave more offspring in the next generation. The fact that EA require only a value of the function to be optimized (a value of the fitness function) and the reliability of EA make them an effective tool for solving complex problems in real-world applications. In particular, EAs are actively used in machine learning methods to improve the quality of the resulting models. For example, EAs are used to design neural networks, which allows optimizing not only the weight coefficients of the network, but also its structure, including individual connections and activation functions of neurons. In addition, EA are successfully used for building a rule base in fuzzy systems and building decision trees. Due to its universality EA can be used for tuning any machine learning model, selecting both numerical and categorical hyperparameters, forming ensembles of models and much more. Because of the universality and efficiency of such optimization methods, both various new EAs and ways to improve old ones are appearing increasingly frequently. Also, in addition to the development of optimization algorithms themselves, the variability of their application for forming machine learning models is growing and expanding. As a rule, a researcher reports the results of experiments with his new approaches in a scientific article,

* Corresponding author: sherstpasha99@gmail.com

adding at the best a description of the developed method or pseudocode. As a rule, a researcher reports the results of experiments with his new approaches in a scientific paper, adding at best a description of the developed method or pseudocode. And then, when another researcher or developer solving his own problem needs to use these modified algorithms, he has to program them himself, spending a lot of time and resources on this process. Of course, this does not concern classical optimization algorithms or machine learning methods, which have long been available in widely used libraries such as scikit-learn and SciPy, but rather effective, but for various reasons not so widely known algorithms and EA modifications. Thus, at the time of writing, there is no available implementation of such an effective method of EA self-tuning as SelfCEA [1,2], and there is no available implementation of the method of automated design of neural networks using genetic programming [3].

Considering the above facts and the variety of different approaches of EA application, and considering the researcher's need to have simple and effective tools for EA application and approaches based on them, the development of a tool that can make it accessible and easy to use EA both for solving optimization problems and for building machine learning models is an actual problem. Among the tools for implementing software, the Python programming language [4] is a favorite, since it is not only easy to use, but has good community support and is continuously evolving. Thus, the goal was formulated to create a Python library that integrates a wide variety of effective EAs and machine learning methods that apply these EAs, and combines accessibility, ease of use, and efficiency. This paper describes Thefittest library created to accomplish the above goal.

The rest of the paper is organized as follows: Section 2 reviews and discusses related work, specifically libraries with similar goals. Section 3 describes the approach to project maintenance and development, as well as the underlying technologies and dependencies. Section 4 discusses the library structure and its application using several examples. Section 5 presents results comparing the performance of the implemented methods against similar methods from other packages, as well as the variety of EA methods provided. Sections 6 and 7 form conclusions on the whole work and discuss future work on the project.

2 Related work

At the moment, there are many Python libraries with similar goals. Let's consider their capabilities and limitations.

2.1 DEAP

DEAP (Distributed Evolutionary Algorithms on Python) is an evolutionary computing framework for rapid EA prototyping and testing of ideas [5]. Although a number of ready-to-use recipes can be found in the documentation, DEAP does not provide ready-to-use methods, but a set of tools to create them. With this solution, the authors try to motivate researchers to create their own methods by controlling a wide set of algorithm parameters. The main problem with the DEAP framework is that it tries to achieve two conflicting goals: to simplify the construction and application of EAs and to give the user maximum control over the parameters and properties of the constructed EAs. Because of this, users need to put a lot of effort in learning the framework for each optimization problem. In addition, DEAP has only classical genetic operators at its disposal, and the construction of custom operators may require even deeper learning of the framework. Despite its shortcomings, DEAP is one of the most popular Python packages for evolutionary computation, which may indicate that the authors have successfully achieved their goals. Obviously, despite some ready-to-use recipes, the DEAP framework solves a different problem from Thefittest in that it provides a set of tools for constructing EAs rather than ready-to-use methods. However, DEAP is often

used only as a library with which to quickly use genetic algorithm and genetic programming algorithm.

2.2 SciPy

SciPy is a library for performing scientific and engineering calculations built on NumPy [6]. The library has a huge number of methods for gradient optimization, solving differential equations, statistical analysis, and more. SciPy partially relates to EA, as it has its own implementation of the classical differential evolution method.

2.3 PyGAD

PyGAD is an open-source project that aims to provide a high-performance implementation of a genetic algorithm [7]. In addition to the genetic algorithm itself, PyGAD makes it possible to use it to train both a classical neural network and a convolutional neural network. This library aims to simplify the application of the genetic algorithm. Typically, using a genetic algorithm from this library requires only three steps: defining a fitness function by the user, creating an object of the genetic algorithm class, and starting the optimization process by calling the appropriate method of the class. In addition to this advantage, PyGAD is tightly integrated with Keras and PyTorch, allowing the use of a GPU when training a neural network with a genetic algorithm. The obvious disadvantage of PyGAD is only that it does not have other EA optimizations, although the package itself has no such goal. Nevertheless, the classical genetic algorithm is strongly inferior in efficiency to various genetic algorithms with self-tuning of mutation probability and operators with selective pressure and multi-parent recombination [1,2].

2.4 Pymoo

Pymoo is a library that covers many aspects of multi-objective optimization, including decision making and visualization of multidimensional spaces [8]. In addition to multi-criteria optimization, the library has EA implementations for single-objective optimization as well, including differential evolution and genetic algorithm. Pymoo aims not only at EA optimization, but also at classical methods. However, among EAs, the choice of single-criteria optimization methods, as well as genetic operators, is severely limited. Self-tuning EAs would be a good addition for this library.

2.5 GPlearn

Gplearn is a library with a genetic programming algorithm implementation inspired by and compatible with the Scikit-learn API. Similar to PyGAD, Gplearn provides only one algorithm. Thus, a genetic programming algorithm is available to the user to solve regression and classification problems. Among genetic operators, the package implements only standard crossover and classical mutation operators from [9]. The self-configuring or self-tuning methods are not implemented in the package.

3 Project maintenance and underlying technologies

The library is an open-source project published on GitHub [10] and PyPi [11]. It is developed using the static type checker MyPy [12], the testing framework PyTest [13], and in

accordance with pep8 [14]. Thefittest has minimal dependencies, making it easy to use. At the time of writing, only two libraries are required for the package to work:

- NumPy [15]. Provides an implementation of multidimensional arrays and operations on those arrays in the C and Fortran programming languages for Python. Thefittest uses the NumPy toolkit for almost all functions;
- Scipy. Library already discussed earlier. In this project it is necessary only for linear algebra methods;
- Numba [16]. An open-source JIT compiler that translates a subset of Python and NumPy into fast machine code using LLVM via the llvmlite Python package. In the project, Numba is used in the narrowest places, such as executing genetic operators or computing the output of a neural network, allowing high performance methods. A big advantage of using Numba, besides compilation, is the ease of using parallel computing as well as GPUs.

It is likely that in the future the Scikit-learn library [17] will be added to the list of dependencies. It will allow integration of machine learning methods from Thefittest with tools from Scikit-learn, such as cross_validate and Pipeline objects. Optional dependencies currently include the NetworkX visualization package [18], which may need to be imported if the user wants to display the obtained binary tree or neural network.

4 Design and usage of thefittest

Thefittest is an open-source project whose goal is to provide modern and efficient EAs, as well as approaches for design of machine learning models using these EAs. Since one of the goals is to make the package easy and intuitive to use, Thefittest uses a minimum of dependencies and a unified interface. The library allows solving optimization, classification and regression problems, has its own implementations for computing some popular metrics, as well as data transformation methods. In addition, the library contains popular benchmarks for optimization, classification and regression, such as Fisher's Iris [19] or the set of CEC2005 optimization test problems [20]. Figure 1 shows the structure of thefittest library.

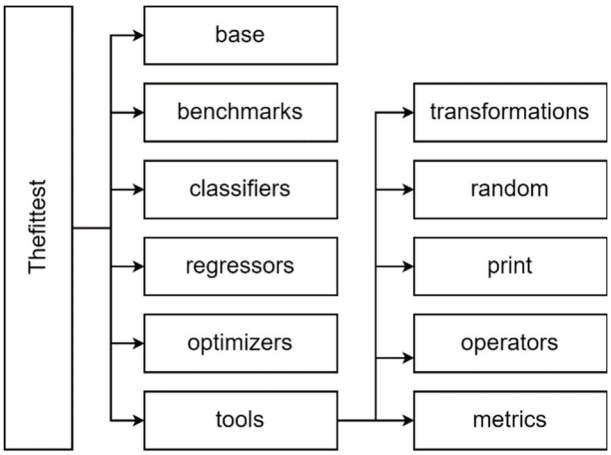

Fig. 1. Thefittest design.

Thefittest has 6 modules. The base module contains machine learning objects and models, as well as some primitives. For example, it contains Tree (n-ary tree) and Net (neural network) objects, as well as functional and terminal set objects for genetic programming algorithm. The benchmarks module contains known problems for testing optimization and

modeling methods, such as CEC2005 [20] or Wine Classification [21]. The classifiers and regressors contain machine learning methods for solving classification and regression problems, respectively. The optimizers module contains EAs that can be used both by machine learning methods and separately for solving other optimization problems. The tools module contains useful instruments used in Thefittest, which can also be used for other purposes. The tools module is divided into a number of other modules. The transformations module contains methods for scaling, ranking, normalization, and encoding numeric variables into binary form. The random module contains special methods for generating data and various random variables. For example, random contains functions for generating trees. In addition, random provides high-performance functions for randomly selecting elements from sets. In the print module one can find functions for visualizing some types of data and models, such as trees or neural networks. The operators contain all the genetic operators used in EA, as well as various activation functions for neural networks. In the last module called metrics, the user can find high-performance functions to calculate known metrics for classification and regression. One of the advantages of Thefittest is its ease of use. Let's consider an example of solving a classical OneMax problem using the genetic algorithm. Figure 2 shows a flowchart with code for solving the OneMax problem using Thefittest. All variables have meaningful names.

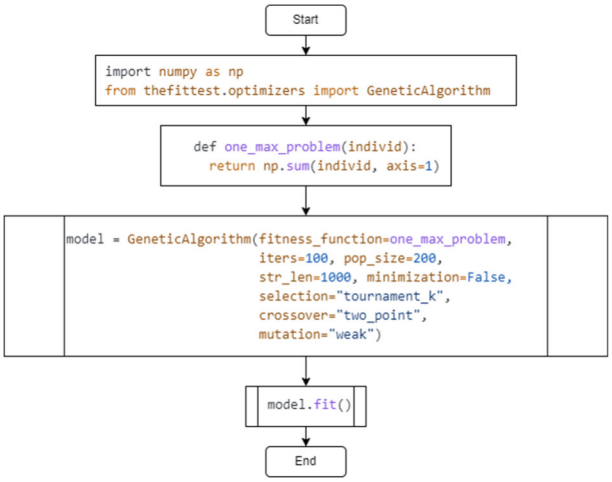

Fig. 2. Solving OneMax problems with Thefittest.

As can be noted, using the genetic algorithm requires only three necessary and one optional step:

- define a fitness function. The fitness function is defined for the whole population at once, not for a single individual;
- initialize the genetic algorithm with the necessary parameters of the optimization process;
- start the optimization process (method fit).

Let's consider another example. Let's solve the classical Iris recognition test problem by using the method of automated neural network formation, proposed in [3]. In this method, the structure of the neural network is optimized using a genetic programming method, and the weight coefficients of the neural network are tuned by some optimizer suitable for real-valued optimization. The fitness value of each structure in the optimization process is the efficiency of the neural network with that structure, so each neural network needs to be trained to evaluate the fitness function. Here, as an example, the weights will be tuned using

a modified SHADE differential evolution method [22], which also has an implementation in Thefittest. Figure 3 shows a flowchart with code to solve this problem.

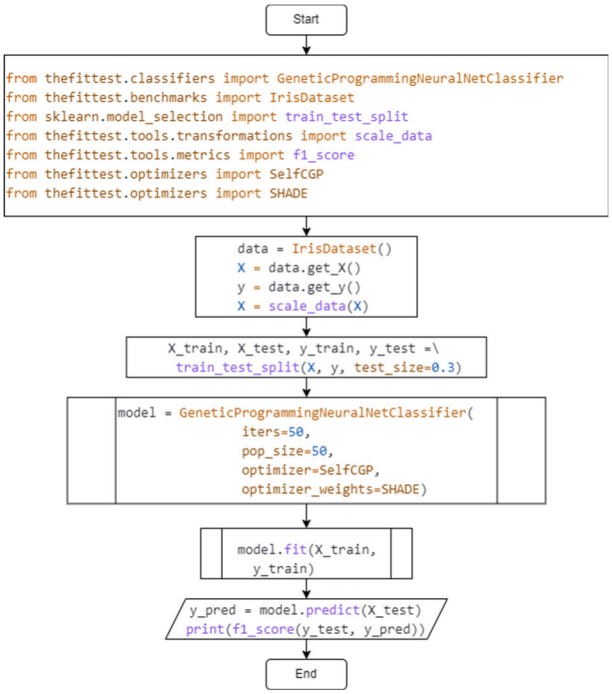

Fig. 3. Building a neural network to recognize Fisher's Iris.

Let's consider the code presented in Figure 3:

- First of all, the import of method [3] is performed, which in Thefittest is named GeneticProgrammingNeuralNetClassifier when solving the classification problem. Import of the IrisDataset test task is also performed. Additionally, a method for splitting the sample into training and test samples, as well as a method for scaling the data and evaluating the resulting prediction must be imported. Next, a self-configuring genetic programming algorithm is imported, which will be used to optimize the network structure. Also imported is the SHADE adaptive differential evolution method, which will be used to tune the weighting coefficients of the neural networks;
- In the next two blocks, the data is initialized, scaled, and split into training and test samples;
- The fifth block initializes the method with the necessary parameters, including information about which optimization methods will be involved in building the machine learning model;
- In the next block, the fit method is executed to start the learning process;
- In the last block, the predict method is called to make predictions from the trained model and outputs the f1 measure between the obtained prediction and the test labels.

One advantage of the method [3] is that it can result in a neural network with a more complex structure than just a fully-connected perceptron, and with different activation functions within the network structure. Such a complex structure would be useful to visualize. The Figure 4 demonstrates the code for visualizing the obtained neural network.

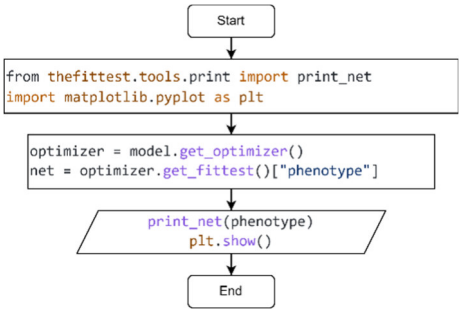

Fig. 4. Code for visualizing the neural network.

The Figure 5 shows the final neural network. Input neurons are shown in green, hidden neurons in blue, and outputs of the neural network in red. The hidden neurons are labeled according to the activation functions used in them. Gs - Gaussian function, rl - linear rectifier.

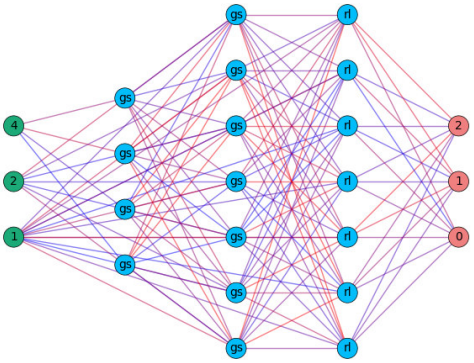

Fig. 5. Neural network obtained by the method [3] when solving the Fisher's Iris problem.

It can be noted that the network has a more complex structure than just a fully-connected perceptron, as well as different activation functions. In addition, the network does not use all variables, but only three of them. This section describes the developed Thefittest library and its internal structure. Examples of the library application are also demonstrated. In order to achieve ease of use, the library has a unified interface for all machine learning and optimization methods.

It is easy to see that model from Thefittest have fit, predict and transform functions similar to models from Scikit-learn. Scikit-learn, besides the machine learning models themselves, provides functions and methods for the full cycle of model training and testing, such as split training and testing, and k-fold validation. In the future, it is planned to add Scikit-learn to the dependencies of the Thefittest library and use special parent classes of machine learning models in order to integrate models from Thefittest into the Scikit-learn environment.

5 Comparison of libraries with evolutionary optimization methods

Although the Thefittest project is only six months old, it can already provide many efficient optimization methods and some machine learning methods that do not have implementations in the free access. It would be useful to compare the packages described in Section 2 with Thefittest with regard to the methods provided as well as their performance. As stated earlier,

the packages in Section 3 have similar but not identical goals to Thefittest. Sometimes the researcher wants variability in algorithm parameter settings and full control over the entire optimization process, in which case a library such as Deap may be a suitable solution. In other cases, the researcher may want to quickly try as many different methods as possible. In this case, Thefittest will be the right choice. In addition, the choice may also depend on the type of problem to be solved. For example, if the multimodal optimization problem needs to be solved, the pymoo package would be a better choice. Therefore, the advantage of Thefittest in this comparison can only be fair if the researcher's needs are the same as Thefittest's goals. If there are other needs, another library or framework may be more effective. Let's start with a list of algorithms and methods that the packages provide. The Table 1 provides information about the provided methods in Thefittest and other packages. We consider mainly evolutionary algorithms for single-objective optimization.

Table 1. EAs available in the packages under consideration.

Algorithm\package	**Thefittest**	**Deap**	**Scipy**	**PyGAD**	**Pymoo**	**GPlearn**
Genetic algorithm	+	+	-	+	+	-
SelfCGA	+	-	-	-	-	-
SHAGA	+	-	-	-	-	-
Evolution Strategy	-	+	-	-	+	-
Differential evolution	+	-	+	-	+	-
jDE	+	-	-	-	-	-
SHADE	+	-	-	-	-	-
Particle Swarm Optimization	-	+	-	-	+	-
Genetic programming	+	+	-	-	-	+
SelfCGP	+	-	-	-	-	-
Genetic programming of neural networks (GPNN)	+	-	-	-	-	-
Multilayer perceptron trained by evolutionary algorithms	+	-	-	+	-	-
Convolutional neural networks trained by evolutionary algorithms	-	-	-	+	-	-

Table 1 provides information about the presence of EA and machine learning methods in Thefittest and the packages in Section 2. It is not really correct to compare the number of methods in the packages, as they have slightly different goals from Thefittest. Nevertheless, we can note the fact that none of them provides adaptive EAs such as SelfCGA, SHAGA or SHADE. In addition, the packages under consideration provide only a minimal choice of genetic operators for EAs. Thefittest, on the other hand, provides both classical operators and modified multi-parent crossover operators with selective pressure [1, 2], which have shown high efficiency on optimization problems for both genetic algorithm and genetic programming algorithm. Classical EAs from different libraries were run to solve the same problem (for each type of EA the problem was different) under the same settings. The only difference was in the EA implementations. The Table 2 shows the running time (in seconds) of each implementation averaged over the 30 runs. If the method was not in the package, a dash was placed.

Table 2. Comparison of EA runtime on different packages.

Algorithm\package	**Thefittest**	**Deap**	**Scipy**	**PyGAD**	**Pymoo**	**GPlearn**
Genetic algorithm	8.209	145.1	-	68.97	341.4	-
Differential evolution	0.191	-	91.86	-	1.339	-
Genetic programming	30.72	37.16	-	-	-	39.15

As can be noted, Thefittest has the lowest execution time compared to the other packages (in bold). The differential evolution algorithm implemented in Thefittest ran more than 480 times faster than the same algorithm implemented in Scipy. This was achieved through the use of the Numba library discussed earlier. Due to the complexity of the data type, it works with, the genetic programming algorithm has a minimum of functions implemented in Numba. Because of this, the performance difference is not as significant as for other EAs. This situation may change in the future when Numba finalizes class support. It may seem that the performance of the EA implementation can be ignored since the main consumer of resources will be the fitness function of this EA, but in some algorithms, such as in [3] there are two layers of EAs, one of which works inside the fitness function. In such case, the performance of this EA is critical. Another point concerns the use of parallel computing. In such case, again there may be a view that optimizing the code does not make sense, since one can simply split the problem into a sufficient number of processes, thereby reducing the execution time. But even in this case the optimized code will win because separate processes with optimized code will also execute faster than with non-optimized code. At the same time, the Numba library allows you to execute code in parallel.

6 Conclusion

This paper demonstrated a library that aims to provide productive and state-of-the-art EA optimizations, as well as efficient methods for building machine learning models using EA. Among the advantages of Thefittest compared to other libraries specializing in evolutionary computation are the availability of both classical and efficient modified methods, the ease of working with the library, the availability of benchmarks, and the optimized code of the algorithms. The results of comparison with other libraries have shown that, despite the short

period of the project's existence, Thefittest is a competitive library, and in some cases has an advantage over other libraries with similar development goals.

7 Future Work

The project development work is planned in several directions at once:

- Expanding the set of EAs as well as the class of problems to be solved, including multi-criteria optimization, clustering, and control;
- Expanding the set of machine learning methods that utilize EAs, including fuzzy logic systems, decision trees, and ensembles;
- Providing compatibility with the Scikit-learn library for easy handling of models;
- Implementation of tools for automatic parallelization of algorithms;
- Implementing tools for using GPUs to train neural networks using the Numba library.

In addition, at the time of writing the paper, Thefittest has no documentation. Accordingly, the task of documenting the code is also relevant. Additionally, there is a need for further analysis of existing libraries on the topic of EA and their applications.

This work was supported by the Ministry of Science and Higher Education of the Russian Federation (Grant № 075-15-2022-1121).

References

1. E. Semenkin, M. Semenkina, LNCS. Self-configuring Genetic Algorithm with Modified Uniform Crossover Operator **7331**, 414-421 (2012)
2. E. Semenkin, M. Semenkina, IEEE Congress on Evolutionary Computation, CEC 2012. Self-configuring genetic programming algorithm with modified uniform crossover (2012)
3. L. Lipinskiy, E. Semenkin, Bulletin of the Siberian State Aerospace University **3(10)**, 22-26 (2006)
4. The official home of the Python Programming Language [Internet]. [cited 2023 Aug 20]. Available from: https://www.python.org/
5. F.A. Fortin, F.M. De Rainville, M.A. Gardner, M GC Parizeau, Journal of Machine Learning Research. DEAP: Evolutionary algorithms made easy, 2171-2175 (2012)
6. P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau et al., Nat Methods. SciPy 1.0: fundamental algorithms for scientific computing in Python **17(3)**, 261-272 (2020)
7. A.F. Gad, PyGAD: An Intuitive Genetic Algorithm Python Library (2021). Available from: http://arxiv.org/abs/2106.06158
8. J. Blank, K. Deb, IEEE Access. Pymoo: Multi-Objective Optimization in Python **8**, 89497-8509 (2020)
9. Welcome to gplearn's documentation! [Internet] (2023). Available from: https://gplearn.readthedocs.io/en/stable/
10. Thefittest: Implementation of data mining methods that use evolutionary algorithms [Internet] (2023). Available from: https://github.com/sherstpasha/thefittest;
11. thefittest · PyPI [Internet] (2023). Available from: https://pypi.org/project/thefittest/

12. Optional Static Typing for Python [Internet] (2023). Available from: https://mypy-lang.org/
13. pytest: helps you write better programs [Internet] (2023). Available from: https://docs.pytest.org/en/7.4.x/;
14. PEP 8: The Style Guide for Python Code [Internet] (2023). Available from: https://pep8.org/;
15. C.R. Harris, K.J. Millman, S.J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau et al., Nature. Nature Research. Array programming with NumPy **585**, 357-362 (2020)
16. Numba documentation [Internet] (2023). Available from: https://numba.readthedocs.io/en/stable/index.html;
17. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel et al., Scikit-learn: Machine Learning in Python (2012). Available from: http://arxiv.org/abs/1201.0490
18. A.A. Hagberg, Los lanlgov, D.A. Schult, P.J. Swart, Exploring Network Structure, Dynamics, and Function using NetworkX [Internet] (2008). Available from: http://conference.scipy.org/proceedings/SciPy2008/paper_2
19. Iris. UCI Machine Learning Repository [Internet] (2023). Available from: https://archive.ics.uci.edu/dataset/53/iris;
20. J. Liang, K. Deb, Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization [Internet] (2005). Available from: https://www.researchgate.net/publication/235710019
21. Wine. UCI Machine Learning Repository [Internet] (2023). Available from: https://archive.ics.uci.edu/dataset/109/wine
22. R. Tanabe, A. Fukunaga, Evolutionary Computation (CEC). Success-history based parameter adaptation for Differential Evolution, 71-78 (2013)