BlaBoO: A Lightweight Black Box Optimizer Framework

Péter Kiss*, Dávid Fonyó†, Tomáš Horváth‡
Department of Data Science and Engineering
Faculty of Informatics
ELTE – Eötvös Loránd University
Budapest, Hungary
*peter.kiss@inf.elte.hu, †fonyodav@inf.elte.hu, ‡tomas.horvath@inf.elte.hu

***Abstract*—BlaBoO is an open source software initiative, that aims to help find optimal solution for the widest possible range of optimization problems such as, amongst others, hyper-parameter tuning of machine learning algorithms. The software is able to optimize the parameters of any black-box function runnable in command line. It needs no cumbersome installation, launching the application requires running a Java executable file because of what the application runs under various platforms. BlaBoO provides a GUI as well as a batch mode processing abilities. It is easy to extend it by new optimization algorithms. BlaBoO is intended not only for managing black-box optimization experiments but also for education purposes. Its features make it a valuable tool for (not only beginner) machine learning practitioners, students and teachers as well.**

***Index Terms*—Black-box optimization, Hyper-parameter tuning, Open-source software**

I. Introduction

Black-box optimization (BbO), in general, refers to the problem where the task is to optimize an objective, black-box, function $f : X \to \mathbb{R}$ such that one can evaluate $f(x)$ for any $x \in X$ but does not posses any further information on f. Under such circumstances, e.g. when f might not be convex or differentiable, one must rely on derivative-free optimization methods, since variants of gradient descent based algorithms are not applicable. The goal of using BbO algorithms is to find the parametrization of a given task under which it operates best according to some measurable qualities of the output of the black-box function, formally

$$min_{w \in X} f(w), \tag{1}$$

where $X = X_1 \times X_2 \times \cdots \times X_n$ is the parameter space, that is the union of points that represent allowed configurations of the black-box function f given some constraints related to f.

In other words, given a machine learning (ML) algorithm, i.e. our black-box function, which efficiency on a specified dataset depends on the right settings of its (hyper-)parameters, BbO techniques are utilized to find a setup of these parameters providing (sub-)optimal performance of the given ML algorithm on the specified dataset.

EFOP-3.6.3-VEKOP-16-2017-00001: Talent Management in Autonomous Vehicle Control Technologies – The Project is supported by the Hungarian Government and co-financed by the European Social Fund.

In case of restricted search spaces, where $X \subseteq \mathbb{R}^n$, a wide range of BbO techniques were developed [10]. The majority of these techniques were successfully used, among other optimization tasks, for hyper-parameter (HP) tuning of machine learning (ML) algorithms [8], [12], [19].

Such methods include, but are not limited to *Naive algorithms* such as Grid Search (GS) or Random Search (RS) [12]; *Local (direct) search methods* as Pattern Search (PS) [16]; *Meta-modeling approaches*, like Sequential Model-based Optimization (SMO) [18] using Gaussian Processes (GP) [20], Random Forests (RF) [21], Tree-structured Parzen Estimators (TPE) [22] or other learning techniques; *Swarm intelligence methods* (SI) [23]; *Evolutionary algorithms* (EA) [24]; or *Statistical methods* such as Simulated Annealing (SA) [25] or Monte Carlo (MC) [26], for instance.

II. Related works

Lot of the techniques listed in Section I are implemented in commercial or open-source frameworks (some of those from the latter group are listed, along with their main features, in Table I). These optimizers are usually designed for users with good technical and theoretical background and/or they assume users being familiar with specified programming languages. Users with less refined knowledge on BbO or ML need a longer learning phase to use these systems. However, their usability might still remain limited to the range of implemented optimizers, supported programming languages or they may require special preparations such that installing dependencies or integrating API calls of optimizers in their source code.

III. The BlaBoO framework

The open-source application called BlaBoO[1] (Black-Box Optimization), presented in this paper, is intended for users with limited ML/BbO knowledge or those who would like to avoid a long preparation process for an off-the-shelf black-box optimizer. BlaBoO has been designed with focus on:

- *Portability* – running the framework requires only `JVM 8+`, and the framework itself can be installed simply by copying and extracting a simple `.zip` file that includes

[1]The application, the manual and the source code are freely available at https://github.com/kppeterkiss/BlackBoxOptimizer

Name of the software	Language	Optimizers included	GUI
Auto-WEKA [8]	Java	GS, SMO	✓
mlr [1], [11]	R	GS, RS, EA (irace), SA, SMO	-
scikit-learn [2], [7]	Python	GS, RS	-
HyperOpt [3], [6]	Python	RS, SMO-TPE	-
MOE [4]	Python, C++	SMO-GP	-
Spearmint [5]	Python, Matlab	GS, RS, SMO-GP	-
BlaBoO	Java	GS, RS, SMO-GP, SA, PS, SI, EA	✓

TABLE I
OPEN-SOURCE BLACK-BOX OPTIMIZATION SOFTWARES AND THEIR MAIN CHARACTERISTICS.

the self-containing `.jar` file of the application along with necessary directory structure.

- *Usability* – to make the software as easily usable as possible, it is provided with a web browser based GUI that can help setting up and configuring the experiments. The developed GUI, basically, creates a `.json` configuration file for the optimization experiment to be conducted. In case of large volume of experiments, configuration files could be generated by a script written by the user.
- *Generality* – the framework, essentially, runs a specified (by the user) terminal command varying its arguments according to the computation of the chosen optimization algorithm, then reads its standard output to further conduct the search of the specified parameter space. Thus, the framework enables to tune/optimize any program or algorithm that accepts arguments from the command line and writes the value(s) of its objective(s) to the standard output, let it be implemented in any language and running on any platform.
- *Flexibility* – the framework enables users to choose various tuning strategies and set up their parameters, set up dependencies between hyper-parameters of black-box functions as well as open and save the configurations and current state of the experiments.
- *Extensibility* – development and testing of new optimization algorithms can be easily carried out by extending an abstract `.java` class describing the general behavior of tuning methods. Developers can follow existing implementations of tuning algorithms integrated into the framework to easily develop their modified versions or implement new optimization methods.

Figure 1 illustrates the architecture and work-flow of the BlaBoO framework. First, it loads the configuration, optionally created by the browser-based GUI (0), some screenshots of which are presented in figures 2 and 3.

The configuration, stored in a `.json` file, serves as a description of the optimization task required to be executed. First it contains the description of hyper parameters of the black-box function (including their constraints), the terminal command that is used to run the black-box function to be optimized, the objectives of the optimization task and the chosen tuning algorithm (1), along with eligible backup setups.

Then, the controller (2) sends the so-called landscape (i.e. the known objective values for previously executed setups), if any, to the tuning algorithm. The tuning algorithm generates a new candidate setup (3), that will be validated according to the objectives w.r.t. the specified parameter constraints (4). If they are valid, the controller calls the black-box function, passes the new parameters (5) to it, and reads its output (6), i.e. the values of the objective(s), from the standard output of the system. The resulting pair of the new parameters together with the returned objective value(s) will be added to the landscape (7). Afterwards, the controller checks, whether termination condition has met (8). Termination can happen if a given iteration number or a threshold value on objectives has been reached or, in case of convergence, when the objective does not change significantly anymore. If such a condition has been met, the landscape will be written in a resulting `.csv` file (9) and the optimization process will be finished.

Algorithm 1 BlaBoO's Optimizer Module

```
1: procedure OPTIMIZE
2:     landscape ← {}                    ▷ initialize the landscape
3:     term ← False                      ▷ for termination checking
4:     w ← initialize                    ▷ initialize configuration
5:     while not term do
6:         w ← generate_configuration(landscape)
7:         obj ← f(w)
8:         term ← check_termination(landscape,obj)
9:         landscape ← landscape ∪ {(w, objective)}
```

A. Parameter space

For now, BlaBoO supports a range of parameter types. The two natural groups are the types of continuous and countable domain sets (including logical type and enumerations as a generalization of discrete sets). Each parameter is defined through specifying its type along with some boundaries between which the actual values can be varied.

Some cases, however, regarding the parameter space as a static structure, can result in unnecessary computation overhead taking into account some previous knowledge on the black-box function f. This stems from the fact that the actual domain of f might be not a convex set, that is, the different variables can influence the ranges of other variables. As a useful example for this phenomenon, it might be worth to mention the scenario in which one wants to find the optimal hyper-parameter setup of a support vector machine, taking into account multiple possible kernel functions. In this case the specific kernels functions may require different sets of hyper-parameters while handling the parameter space as the union

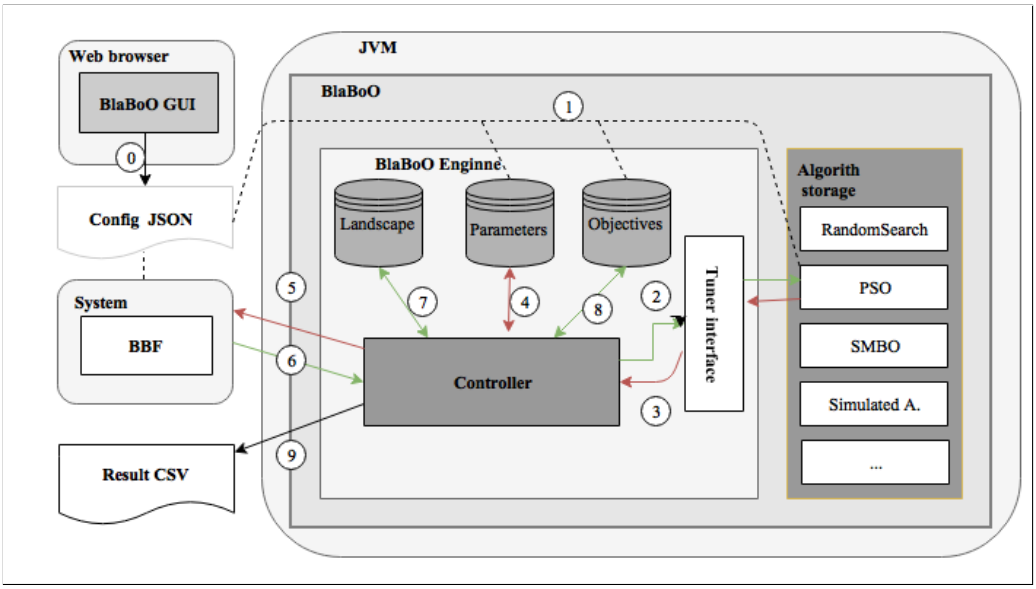

Fig. 1. **Architecture**: The framework runs in the Java Virtual Machine, spawning processes through using native system commands. This process corresponds to an execution of the black-box function, given a specific parameter setup. When a process finished, based on the evolution of objective values (read from the standard output) the optimizer algorithm decides which setup should be executed next, until the termination conditions have been met.

of all the possible domain set end up in a significant number of trials executed in vain.

BlaBoO supports this dynamic nature of search spaces through the concept of bounded parameters. A bounded parameter can have multiple ranges, among which the active one is defined by the recent value of the bounding parameter. The concept also allows specification of empty ranges, that is equivalent with the scenario, where f does not need the bounded parameter for its computations.

B. Optimization Algorithms

At recent state, the framework supports the following black-box optimization methods:

- Naive algorithms:
  - Random search [12]
  - Grid search
- Genetic/Evolutionary algorithms [13]
  - GA with roulette selection and random step mutation
  - GA with tournament selection and random step mutation
  - Differential evolution [14]
- Hit and run approach [27]
- Swarm Intelligence algorithms
  - Particle Swarm Optimization [17]
  - Fish School Search [15]
- Pattern search [16]
- Simulated Annealing [25]
- Sequential Model-Based Optimization with Gaussian processes [18]

The listed algorithms can be classified into two groups based on the characteristics of the search space. The most of the algorithms are designed for searching over continuous search spaces. The naive methods, Random search and Grid search, on the other hand, are capable of working with countable domain sets as well (e.g. $\mathbb{Z}$, $\{True, False\}$, etc.), that can be important in case of some ML techniques as in the above mentioned example with varying a kernel of support vector machines.

One can choose either between these algorithms or can experiment with some own (implementations of) search strategies. BlaBoO framework has been designed with the intention for extreme flexibility and extensibility. In practical point of view it means that if someone would like to implement a new algorithm (or try out a slightly different version of an old one), the only task to do is to implement an `AbstractAlgorithm.java` interface. The abstract methods of this interface make up a tuning algorithm, that is, the developer of the new tuning method should specify essentially what are the rules according to which new parameter configurations should be tested, based on the previous results, and possibly what internal state should be maintained between the subsequent rounds of generation rounds.

C. Objectives

When the system generates the subsequent configurations for testing with the black-box function, it must be able to decide which configurations performed better and which were less efficient. That requires definition of an ordering over the parameter space. Though every optimization task can be transformed in the form that has been given in Section I, for convenience reason we introduced multiple types of objectives into the framework including minimization, maximization, minimization and maximization until convergence, or reaching a specified threshold.

Additionally, due to the fact that there is no guarantee for a tuning algorithm to terminate on a given task and since there is simply no termination in case of minimization and maximization, a requirement to upper bound for the number of the executed trials is obvious. Default setting of a parameter consist of specifying the type along with the boundaries within it is allowed to move.

D. Usage

To sum up, an optimization task consists of the specification of the black-box function, its parameters as well as their constraints and dependencies, and, the chosen tuning/optimizer method along with the configuration of its parameters. Each such optimization task is described in a `.json` file that will be interpreted and executed by the framework. To enable the widest range of scenarios the software provides a command line and a web based graphical user interface.

a) Command line use: The user can run an optimization task or experiment specifying its configuration in a `.json` file (see figure 5) that contains all the above mentioned characteristics of the task with optional back-up flags. Additional flags can be passed for using secure mode, and specifying the frequency of taking snapshots of the global state of the task. In case of intended or unintended interruption of the process the user can restart the tuning using the snapshot as a basic configuration, thus, continuing the optimization task from the latest backup. When the optimization has terminated, the generated landscape will be saved in a `.csv` output file the lines of which contain given parameter setups (or configurations) and the produced objective value(s) related to those setups.

b) Web GUI: A browser-based interface (introduced in figures 2, 3 and 4) is also provided with the framework in order to help to configure the optimization task. Here, one can specify the way how the parameters can be passed to the black-box function, i.e. the terminal command, the description of its parameters together with their constraints (figure 2), and the chosen optimizer algorithm with its own parameters (figure 3). At the end of the optimization task, there is a possibility to plot the values of the objective(s) w.r.t. the iterations of the optimization process (figure 4). This is especially useful when one wants to compare the performance of various optimizer algorithms.

IV. Conclusions

The presented open-source framework, continuously developed and updated with new black-box optimization algorithms, is intended mainly for non-expert users in the area of black-box optimization, beginner machine learning practitioners as well as educational purposes. Authors, however, believe that BlaBoO can be useful for experiments related to automated machine learning (AutoML [9]), too, allowing to compare various hyper-parameter tuning strategies of machine learning techniques. Integrating more and more algorithms into BlaBoO will also make it a valuable resource of Java implementations of black-box optimization algorithms.

References

[1] https://mlr-org.github.io/mlr-tutorial/devel/html/tune/index.html
[2] http://scikit-learn.org/stable/modules/grid_search.html
[3] https://github.com/hyperopt/hyperopt
[4] https://github.com/Yelp/MOE
[5] https://github.com/HIPS/Spearmint
[6] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins and D. D. Cox, "Hyperopt: a Python library for model selection and hyperparameter optimization", *Computational Science & Discovery* vol. 8 (1), 2015, p. 014008.
[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine Learning in Python", *Journal of Machine Learning Research* vol. 12, 2011, pp. 2825–2830.
[8] Ch. Thornton, F. Hutter, H. H. and Hoos and K. Leyton-Brown, "Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms", *19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2013, pp. 847–855.
[9] M. Feurer, A. Klein, K. Eggensperger, J. T.s Springenberg, M. Blum, F. Hutter, "Efficient and Robust Automated Machine Learning," *28th International Conference on Neural Information Processing Systems* vol. 2, 2015, pp. 2755–2763.
[10] L. M. Rios and N. V. Sahinidis, "Derivative-free optimization: a review of algorithms and comparison of software implementations", *Journal of Global Optimization* vol. 56 (3), 2013, pp. 1247–1293.
[11] B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio and Z. M. Jones, "mlr: Machine Learning in R", *Journal of Machine Learning Research* vol 17, 2016, pp. 1–5.
[12] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization - Semantic Scholar", *Journal of Machine Learning Research* vol. 13, 2012, pp. 281–305.
[13] J. McCall, "Genetic algorithms for modelling and optimisation", *Journal of Computational and Applied Mathematics*, vol. 184 (1), 2005, pp. 205–222.
[14] R. Storn and K. Price, "Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces", *Journal of Global Optimization* vol. 11 (4), 1997, pp. 341–359.
[15] C.J.A. Bastos Filho, F.B. de Lima Neto, A.J.C.C. Lins, A.I.S. Nascimento, M.P. Lima, "A novel search algorithm based on fish school behavior", *IEEE International Conference on Systems, Man and Cybernetics*, 2008, pp. 2646–2651.
[16] R. Hooke and T. A. Jeeves, "Direct Search Solution of Numerical and Statistical Problems", *Journal of the ACM* vol. 8 (2), 1961, pp. 212–229.
[17] J. Kennedy and R. Eberhart, "Particle swarm optimization", *IEEE International Conference on Neural Networks* vol. 4, 1995, pp. 1942–1948.
[18] D. R. Jones, M. Schonlau, W. J. Welch, "Efficient Global Optimization of Expensive Black-Box Functions", *Journal of Global Optimization* vol. 13, 1998, pp. 455–492.
[19] R. G. Mantovani, T. Horváth, R. Cerri, A. C. P. L. F. de Carvalho, J. Vanschoren: "Hyper-parameter Tuning of a Decision Tree Induction Algorithm", *5th IEEE Brazilian Conference on Intelligent Systems*, 2016, pp. 37–42.
[20] C.E. Rasmussen, "Gaussian Processes in Machine Learning", *Lecture Notes in Computer Science* vol. 3176, 2004, pp. 63–71.
[21] L. Breiman, " Random Forests", *Machine Learning* vol. 45 (1), 2001, pp. 5–32.
[22] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization", *24th International Conference on Neural Information Processing Systems*, 2011, pp. 2546–2554.
[23] S-C.Chu , H-C.Huang, J.F., Roddick and J-S. Pan, "Overview of Algorithms for Swarm Intelligence", *Lecture Notes in Computer Science* vol. 6922, pp. 28–41.
[24] D. Simon, *Evolutionary Optimization Algorithms*, Wiley, 2013.
[25] S. Kirkpatrick, C. D. Gelatt Jr and M. P. Vecchi, "Optimization by Simulated Annealing", *Science* vol. 220 (4598), 1983, pp. 671–680.
[26] G. S. Fishman, *Monte Carlo: Concepts, Algorithms, and Applications*, Springer, 1995.
[27] R. L. Smith, "The hit-and-run sampler: a globally reaching markov chain sampler for generating arbitrary multivariate distributions", *Winter Simulation Conference*, 1996, pp. 260–264.

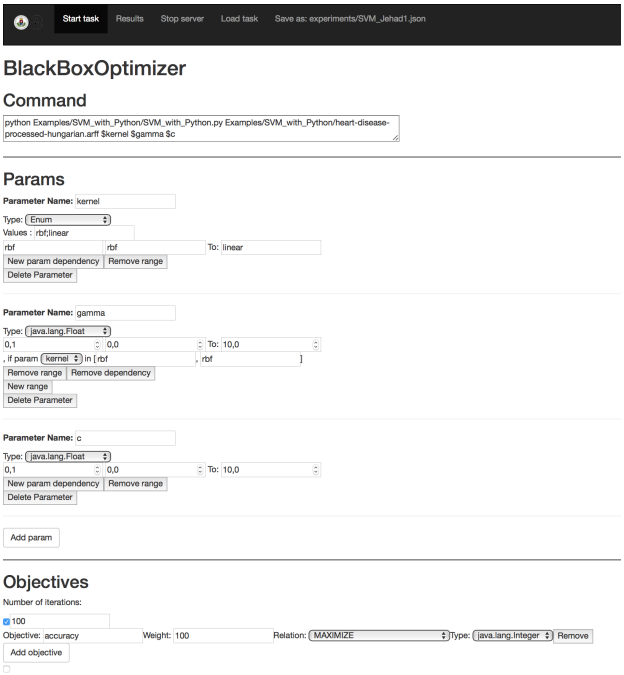

Fig. 2. Black-box setup for an SVM script, with a conditional `gamma` parameter, that is used in case of another parameter `kernel` takes a given value.

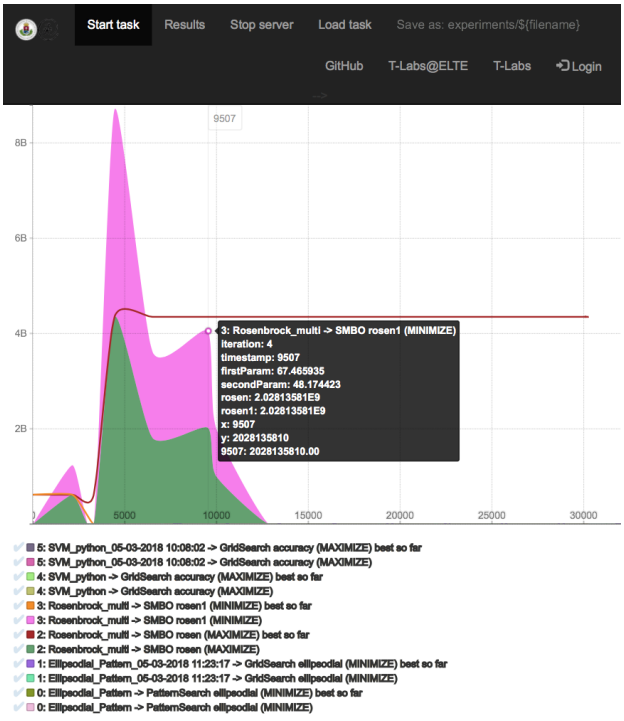

Fig. 4. Landscape, i.e. graph depicting the evolution of 2 objective values (magenta and green areas) in function of the number of executed trials. (The legend contains the results of multiple optimization tasks, that could be displayed as well for purpose of comparison.)

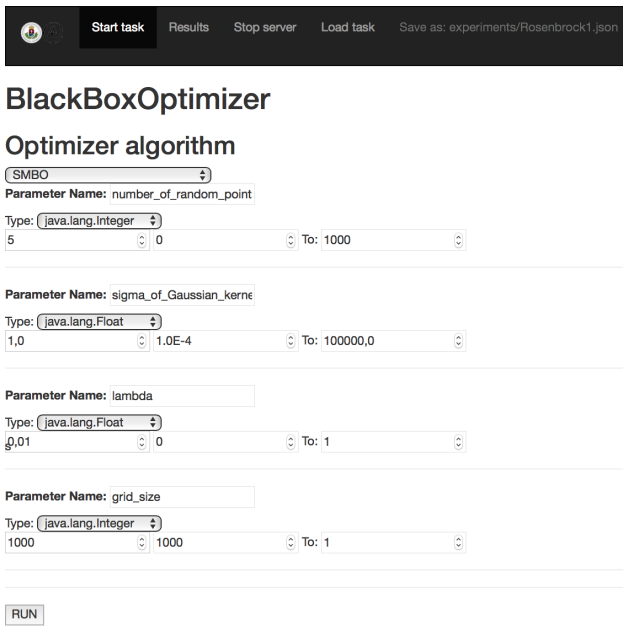

Fig. 3. Optimizer setup - for modifying the parameters of the chosen black-box optimizer algorithm.

```
{
  "baseCommand": "python3 Lasso_poli_reg_on_sinus_curve.py $alpha",
  "algorithmName": "GridSearch",
  "optimizerParameters": [
    {
      "name": "alpha_step_size",
      "typeName": "java.lang.Float",
      "initValue": 0.1,
      "dependencies": [
        {
          "rangeOfThis": {
            "upperBound": 10.0,
            "lowerBound": 1.0E-4
          }
        }
      ]
    }
  ],
  "iterationCount": {
    "value": 100
  },
  "objectiveContainer": {
    "objectives": [
      {
        "relation": "MINIMIZE",
        "terminator": false,
        "weight": 100.0,
        "name": "rss",
        "typeName": "java.lang.Float"
      }
    ]
  },
  "scriptParameters": [
    {
      "name": "alpha",
      "typeName": "java.lang.Float",
      "initValue": 0.0,
      "dependencies": [
        {
          "rangeOfThis": {
            "upperBound": 10.0,
            "lowerBound": 0.0
          }
        }
      ]
    }
  ]
}
```

Fig. 5. Example `.json` configuration file for finding a best parametrization of a python script (Lasso regression), that takes an `alpha` parameter, which will be varied with a simple `GridSearch`, given the specified `stepsize`.