Computer Assisted Methods in Engineering and Science
31(1): 113–132, 2024, doi: 10.24423/cames.2024.1296

Stochastic Schemata Exploiter-Based Optimization of Hyper-parameters for XGBoost

Hiroya MAKINO, Eisuke KITA*

Graduate School of Informatics, Nagoya University, Nagoya 464-8601, Japan
** Corresponding Author e-mail: kita@i.nagoya-u.ac.jp*

XGBoost is well-known as an open-source software library that provides a regularizing gradient boosting framework. Although it is widely used in the machine learning field, its performance depends on the determination of hyper-parameters. This study focuses on the optimization algorithm for hyper-parameters of XGBoost by using Stochastic Schemata Exploiter (SSE). SSE, which is one of Evolutionary Algorithms, is successfully applied to combinatorial optimization problems. SSE is applied for optimizing hyper-parameters of XGBoost in this study. The original SSE algorithm is modified for hyper-parameter optimization. When comparing SSE with a simple Genetic Algorithm, there are two interesting features: quick convergence and a small number of control parameters. The proposed algorithm is compared with other hyper-parameter optimization algorithms such as Gradient Boosted Regression Trees (GBRT), Tree-structured Parzen Estimator (TPE), Covariance Matrix Adaptation Evolution Strategy (CMA-ES), and Random Search in order to confirm its validity. The numerical results show that SSE has a good convergence property, even with fewer control parameters than other methods.

Keywords: evolutionary computation, Stochastic Schemata Exploiter, hyper-parameter optimization, XGBoost.

1. Introduction

XGBoost [1] is one of the very useful and popular machine learning algorithms. Generally, its performance depends on the hyper-parameters such as the learning rate, maximum depth, minimum sum of instance weight, fraction of sampled column in constructing each tree, and fraction of row sampling. Since the selection of the hyper-parameters strongly relies on users' experience and knowledge, it is not easy to set hyper-parameters for new problems to be solved.

The selection of the appropriate hyper-parameters is considered a large-scale combinatorial optimization problem and, therefore, many optimization methods have been studied; Grid Search and Random Search are simple and easy to

implement [2–4]. Bayesian optimization is very popular for tuning of deep neural networks [5–7]. Evolutionary algorithms, including Genetic Algorithm (GA) and Covariance Matrix Adaptation Evolution Strategy (CMA-ES) are widely used for such a task [8–10]. These algorithms, however, have some problems in terms of computational cost, accuracy, and large number of control parameters.

To address these issues, the Stochastic Schemata Exploiter (SSE) is applied for the optimization of the hyper-parameters of XGBoost in this study. SSE, which was first proposed by Aizawa [11, 12], is designed to solve combinational optimization problem. SSE is classified as a evolutionary algorithm like GA. The candidate solutions are defined using binary numbers. SSE extracts common schemata from individuals with high fitness and then, generates new individuals from the extracted schemata, through the schemata exploiter process. After that, the mutation operator is applied to randomly changed bits in the candidate solutions. The parameters of SSE are only the population size (number of candidate solutions) and the mutation rate. Besides, the schemata exploiter process distributes better schemata across the whole population. The characteristics of SSE can be summarized as rapid convergence and a small number of control parameters.

In the original SSE algorithm, individuals are defined as binary sequences, like as in the simple GA. For the problem with the design variables defined as the real values a real-coded SSE has been presented by Maruyama and Kita [13, 14]. For hyper-parameter optimization, the original SSE algorithm is extended so that the variables are defined in integer, real, binary, and categorical formats. The proposed algorithm is compared with the previous algorithms in order to confirm its validity.

The rest of this paper is organized as follows. In Sec. 2, XGBoost and hyper-parameter optimization algorithms are introduced. The proposed algorithm is explained in Sec. 3. In Sec. 4, numerical experiments are executed on OpenML and UCI datasets. The obtained results are summarized in Sec. 5.

2. Background

2.1. XGBoost

XGboost is one of the gradient boosted trees algorithm [1].

When a dataset $\mathcal{D} = \{(x_i, y_i)\}$ with n samples and m features is given, a tree ensemble model employs K additive functions as follows

$$\widehat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i). \tag{1}$$

XGBoost minimizes the following objective function

$$\mathcal{L}(\phi) = \sum_i l(\widehat{y_i}, y_i) + \sum_k \Omega(f_k), \tag{2}$$

where l and Ω denote the loss function and the penalty for the complexity of the model, respectively. Ω is described as

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda ||\omega_i||^2, \tag{3}$$

where T is the number of trees, and ω_i is the score of the i-th leaf. For the tree structure $q(x)$, the optimal ω_i is given as

$$\omega_j^* = \frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}, \tag{4}$$

where first- and second-order derivatives are given, respectively as:

$$\begin{aligned} g_i &= \partial_{\widehat{y}^{(t-1)}} l(y_i, \widehat{y}^{(t-1)}), \\ h_i &= \partial^2_{\widehat{y}^{(t-1)}} l(y_i, \widehat{y}^{(t-1)}). \end{aligned} \tag{5}$$

The optimal value is

$$\widetilde{\mathcal{L}}^t(q) = -\frac{1}{2}\sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T. \tag{6}$$

Since many possible structures q exist, it is almost impossible to calculate the score of the structures q. The search starts from a single leaf, and iteratively adds branches. Let a divided instance be I_L, I_R; then, the loss reduction after the split is given by

$$\mathcal{L}_{\text{split}} = \frac{1}{2}\left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda}\right] - \gamma. \tag{7}$$

At each iteration, the split is carried out for minimizing $\mathcal{L}_{\text{split}}$.

The hyper-parameters of XGBoost to be tuned in this paper are as follows:

- booster to use (booster),
- learning rate of the algorithm (learning_rate),
- maximum depth of a tree (max_depth),
- minimum summation of instance weight for a child (min_child_weight),
- fraction of row sampling (subsample),
- fraction of column sampling in constructing each tree (colsample_bytree),
- learning objective (objective).

2.2. Hyper-parameter optimization algorithms

2.2.1. Gradient Boosting Regression Trees. Gradient Boosting is one of non-parametric statistical learning techniques, known as Gradient Boosted Decision Trees (GBDT) for classification task and Gradient Boosted Regression Trees (GBRT) for regression task. The gradient boosting was first presented by Breiman as an optimization algorithm for a function [15]. Regression gradient boosting algorithms were further developed by H. Friedman [16].

2.2.2. Tree-structured Parzen Estimator. Tree-structured Parzen Estimator (TPE) was presented in 2011 [17].

In hyper-parameter optimization problem, fitness is maximized over a graph--structured configuration space. The configuration space is tree-structured with some leaf variables and node variables. The leaf variables denote the number of units on a hidden layer and so on, and the node variables denote the number of layers to use and so on.

The TPE algorithm is basically a Sequential Model-Based Global Optimization (SMBO) [7, 17]. However, the modeling of $p(y\|x)$ is different from the Gaussian process (GP) approach.

SMBO approximates the fitness f with a surrogate function that is easier to evaluate. It often uses the Expected Improvement (EI) criterion

$$\mathrm{EI}_{y^*}(x) := \int_{-\infty}^{\infty} \max(y^* - y, 0) p_M(y\|x) dy, \tag{8}$$

where x is chosen to minimize $\mathrm{EI}_{y^*}(x)$ at each iteration. GP approach is often used for evaluating $p(y\|x)$.

The TPE calculates the density $p(y\|y)$ by the following function

$$p(x\|y) = \begin{cases} l(x) & (y < y^*), \\ g(x) & (y \geq y^*), \end{cases} \tag{9}$$

where $l(x)$ and $g(x)$ are density functions. The function $l(x)$ is formed from the observations $\{x^{(i)}\}$ such that the corresponding loss $f(x^{(i)})$ is less than y^* and then, the function $g(x)$ is formed from the remaining observations. The TPE algorithm chooses that y^* is the best observed loss.

2.2.3. Covariance Matrix Adaptation Evolution Strategy. Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [8], which is one of evolutionary algorithms, is derived from the de-randomized evolution strategy with covariance matrix adaptation. One of the features is its reliability in adapting an arbitrarily

oriented scaling of the search space in a small population. Therefore, CMA-ES is useful for hyper-parameter tuning [9, 10].

2.2.4. Original Stochastic Schemata Exploiter. The Stochastic Schemata Exploiter (SSE) [11, 12] is a population-based search algorithm similar to GA. The candidate solutions are defined as individuals in binary numbers. The subsets of the individuals are determined according to their fitness ranking. The common sequences, referred to as "common schemata", are generated from the individuals in each subset. New individuals are generated from the schemata.

Since SSE iteratively samples schemata and creates new individuals from the schemata, the better schemata are rapidly distributed to the whole population. The parameters of SSE are only the population size (number of candidate solutions) and the mutation rate. Therefore, SSE has two advantages: a few control parameters and fast convergence properties.

3. Proposed algorithm

3.1. Process

The process of the proposed algorithm is the same as the original SSE:

1. Initialization of individuals
2. Evaluation of fitness
3. Definition of subsets
4. Extraction of common schema
5. Generation of new individual
6. Mutation
7. Update of generation

In the original SSE, each gene of the individual takes only 0 or 1. However, when the SSE is applied for the optimization of hyper-parameters for XGBoost, each gene of the individual has to take integer number other than 0 and 1. Therefore, each step of the process is improved as follows.

3.2. Initialization of individuals

As shown in Subsec. 2.1, XGBoost has the following hyper-parameters: a booster to use (booster), a learning rate of the algorithm (learning_rate), a maximum depth of a tree (max_depth), a minimum summation of instance weight for a child (min_child_weight), a fraction of row sampling (sub-sample), a fraction of column sampling for constructing each tree (colsample_bytree) and a learning objective (objective).

The candidate values for each hyper-parameter are specified in advance. SSE selects the set of appropriate values for the hyper-parameters. The genotype of an individual is given as the list of a hyper-parameters

$$x_j = \{x_{j1}, x_{j2}, ..., x_{j8}\}. \tag{10}$$

The genes $x_{j1}, x_{j2}, ..., x_{j8}$ are related to the hyper-parameters such as: the booster to use, the learning rate of algorithm, the maximum depth of a tree, the minimum summation of instance weight for a child, the fraction of row sampling, the fraction of column sampling for constructing each tree and the learning objective.

The genes are encoded by discrete/categorical values, not $0, 1$. The genotype of gene Γ_j $(j = 1, 2, ..., M)$ is defined as

$$\Gamma_j = \{\gamma_{j1}, \gamma_{j2}, ..., \gamma_{jL}\}, \tag{11}$$

where the gene length is L, and the population size is M.

The operator $\mathrm{R}(\cdot)$ is defined first. $\mathrm{R}(A)$ denotes an element selected randomly from the set $A = \{a_1, a_2, ..., a_n\}$ as follows:

$$\mathrm{R}(A) = \begin{cases} a_1 & \text{if} \quad 0 \le p < \dfrac{1}{n}, \\ a_2 & \text{if} \quad \dfrac{1}{n} \le p < \dfrac{2}{n}, \\ \vdots & \\ a_n & \text{if} \quad \dfrac{n-1}{n} \le p < 1, \end{cases} \tag{12}$$

where p is a uniform random number within $[0, 1)$. Individuals in the initial population are defined as a vector as follows

$$[\mathrm{R}(\Gamma_1), \mathrm{R}(\Gamma_2), ..., \mathrm{R}(\Gamma_L)]. \tag{13}$$

3.3. Evaluation of fitness

The individuals are given as $x_1, x_2, ..., x_M$, where M denotes the total number of individuals. The fitness of the individual x_j is evaluated by a fitness $f(x_j)$.

3.4. Definition of subsets

After fitness evaluation, the individuals are sorted in descending order of their fitness values, and then, they are labeled by the indices $c_1, c_2, ..., c_M$. For example, the best individual with the best fitness has the index c_1.

$S(\neq \emptyset)$ denotes a subset that consists of arbitrary elements of $c_1, c_2, ..., c_M$, and $c_{L(S)}$ denotes the individual with the largest index in an arbitrary subset S. The mean fitness value of S is calculated as follows

$$f_m(S) = \frac{1}{|S|} \sum_{x \in S} f(x), \tag{14}$$

where $|S|$ is the number of the elements in the subset S.

For $L(S) < M$, the following relationships hold:

$$f_m(S) > f_m\left(S \bigcup c_{(L(S)+1)}\right), \tag{15}$$

$$f_m(S) > f_m\left((S - c_{L(S)}) \bigcup c_{(L(S)+1)}\right), \tag{16}$$

where $S - c_{L(S)}$ denotes the set in which the element $c_{L(S)}$ is eliminated from the set S and, the notation $\bigcup$ denotes the union of two sets. Equations (15) and (16) are named as the partial order relationship [11, 12].

In Eqs. (15) and (16), $c_{(L(S))}$ and $c_{(L(S)+1)}$ denote the worst individual in the subset S and the individual worse than it in the ranking, respectively. Therefore, Eqs. (15) and (16) indicate that the mean fitness value of the subset decreases by adding the worse individual to the subset and replacing the worst individual in the subset with the individual worse than it in the ranking, respectively.

According to Eqs. (15) and (16), the list consisting of the M best subsets is defined as follows.

The first subset S_1 consists of the best individual c_1 alone

$$S_1 = \{c_1\}. \tag{17}$$

The second and the third subsets are generated according to Eqs. (15) and (16), respectively, as follows:

$$\begin{aligned} S_2 &= S_1 \bigcup c_{(L(S_1)+1)} = \{c_1\} \bigcup \{c_2\} \\ &= \{c_1, c_2\}, \end{aligned} \tag{18}$$

$$\begin{aligned} S_3 &= (S_1 - c_{L(S_1)}) \bigcup c_{(L(S_1)+1)} = \{\emptyset\} \bigcup \{c_2\} \\ &= \{c_2\}, \end{aligned} \tag{19}$$

where $\{\phi\}$ denotes the empty set. The fourth and fifth subsets are generated according to Eqs. (15) and (16), respectively, as follows:

$$\begin{aligned} S_4 &= S_2 \bigcup c_{(L(S_2)+1)} = \{c_1, c_2\} \bigcup \{c_3\} \\ &= \{c_1, c_2, c_3\}, \end{aligned} \tag{20}$$

$$\begin{aligned} S_5 &= (S_2 - c_{L(S_2)}) \bigcup c_{(L(S_1)+1)} = \{c_1\} \bigcup \{c_3\} \\ &= \{c_1, c_3\}. \end{aligned} \tag{21}$$

Starting from c_1 as the root in the case of $M = 4$, the generated subsets are shown in Fig. 1.

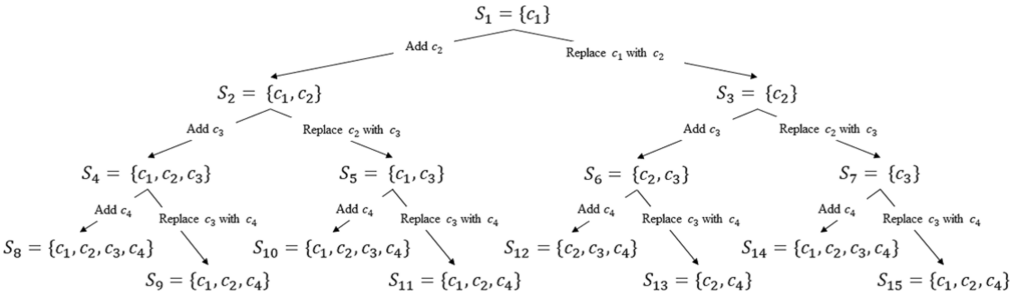

FIG. 1. Generation of subsets with c_1 as the root for $M = 4$.

3.5. Extraction of common schema

Since, in the original SSE, each individual is defined as only 0 and 1, a common schema, which is extracted from the individuals in the subset, can be represented as a string with the symbols 0, 1 and $*$. The symbol $*$ is called the wild card and assumes either 0 or 1.

When SSE is extended to the hyper-parameter optimization, each gene can take more candidates than 0 and 1. So, the symbol $\langle\rangle$ is used instead of $*$, and $\langle\rangle$ represents a set such as $\{a, b, c\}$. $\langle a, b, c\rangle$ means that the gene can assume a, b or c.

The extraction of the common schema

$$H = (H_1, H_2, ..., H_N) \tag{22}$$

from the subset

$$\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_P\} \tag{23}$$

is defined as

$$H_j = \langle x_{1j}, x_{2j}, ..., x_{Pj}\rangle, \tag{24}$$

where $1 \leq j \leq N$ and N is the gene length.

For example, when the subset is composed of three individuals:

$$\mathbf{x}_1 = (1, 4, 8), \quad \mathbf{x}_2 = (2, 4, 7), \quad \mathbf{x}_3 = (1, 4, 9), \tag{25}$$

the following common schema is extracted from the above three individuals,

$$(\langle 1, 2\rangle, 4, \langle 7, 8, 9\rangle). \tag{26}$$

3.6. Generation of new individual

The operator R is applied to the extracted common schema to generate new individuals.

For example, using the operator R for the common schema (26) leads to

$$(\mathrm{R}(\langle 1,2\rangle), \mathrm{R}(\langle 4\rangle), \mathrm{R}(\langle 7,8,9\rangle)).$$

Since $\mathrm{R}(\langle 4\rangle)$ is fixed to 4, the above equation leads to

$$(\mathrm{R}(\langle 1,2\rangle), 4, \mathrm{R}(\langle 7,8,9\rangle)).$$

For generating a new individual, $\mathrm{R}(\langle 1,2\rangle)$ randomly selects 1 or 2, and $\mathrm{R}(\langle 7,8,9\rangle)$ randomly selects 7, 8 or 9.

Schema extraction and new individual generation in SSE are illustrated in Fig. 2.

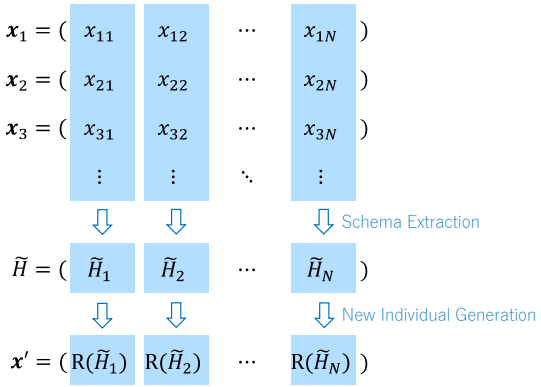

FIG. 2. Schema extraction and new individual generation in SSE.

3.7. Mutation

A mutation changes the value of the gene randomly in GA and the original SSE.

When SSE is extended to the hyper-parameter optimization, genes are encoded by discrete/categorical values. A mutation is expressed by replacing x'_j with the randomly-selected element from the corresponding candidates Γ_j, which is given by R as

$$x'_j \leftarrow \mathrm{R}(\Gamma_j). \tag{27}$$

The effect of the mutation rate should be discussed. For a high mutation rate, there is a high tendency of destroying good individuals. For a low mutation rate, the diversity of the individuals may be lost quickly.

In the proposed algorithm, a rank-based mutation and a normal mutation are compared. The mutation is applied to the individuals except for the individual generated from the subset S_1 because the individual is the best one at the current population.

Normal mutation: Normal mutation is the mutation with a fixed mutation rate. The mutation rate P_{m} is fixed during the simulation.

Rank-based mutation: Rank-based mutation employs a mutation rate depending on the rank of the subset from which the common schema is created.

For the individual created from the schema of the subset with rank i, the mutation rate is given by

$$p(i) = \frac{i-1}{M} \cdot P_{\mathrm{mmax}}, \tag{28}$$

where M and P_{mmax} denote the population size and the max mutation rate, respectively.

The higher the rank of the common schema, the lower the mutation rate. The aim of the rank-base mutation is to maintain diversity in the population.

3.8. Update of generation

The process of the algorithm is summarized in Algorithm 1.

Algorithm 1. Process of algorithm.

```
Specify the population size and gene definition
Randomly generate individuals to define the initial population
repeat
    Estimate individual fitness
    Define subsets
    Extract common schemata
    Generate a new individual from the extracted schemata
    Perform the mutation on an individual, except for the best individual
    Define a new population
until the number of generation reaches to the value given in advance
```

The population size and the gene definition are given at the initial step. After the estimation of the individual fitness, the subset are generated and the common schemata are extracted from them. New individuals are generated from the extracted schemata and applied to the mutation. A new population is updated with these new individuals.

4. Numerical experiment

4.1. Tasks

Binary classification tasks and regression tasks are executed. Each dataset is randomly separated into 80% for training data ($\mathcal{D}_{\text{train}}$) and 20% for test data ($\mathcal{D}_{\text{test}}$). In classification tasks, we maximize the accuracy on test data after training

$$\boldsymbol{\lambda}^* \in \arg\max_{\boldsymbol{\lambda}\in\Lambda} \text{ACC}(A_{\boldsymbol{\lambda}}, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}}), \tag{29}$$

where $\text{ACC}(A_{\boldsymbol{\lambda}}, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}})$ is the accuracy on $\mathcal{D}_{\text{test}}$ after training on $\mathcal{D}_{\text{train}}$, using $\boldsymbol{\lambda}$. The constraint of $\boldsymbol{\lambda}$ (search range) is described later. The number of gradient boosted trees is 100.

In regression tasks, we maximize the R^2 score

$$\boldsymbol{\lambda}^* \in \arg\max_{\boldsymbol{\lambda}\in\Lambda} \text{R2}(A_{\boldsymbol{\lambda}}, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}}), \tag{30}$$

where $\text{R2}(A_{\boldsymbol{\lambda}}, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}})$ is the R^2 on $\mathcal{D}_{\text{test}}$ after training on $\mathcal{D}_{\text{train}}$, using $\boldsymbol{\lambda}$.

The proposed algorithm based on SSE is called SSEopt. The following algorithms are compared in this study:

- Stochastic Schemata Exploiter-based optimization (SSEopt),
- Gradient Boosted Regression Trees (GBRT),
- Tree-structured Parzen Estimator (TPE),
- Covariance Matrix Adaptation Evolution Strategy (CMA-ES),
- Random Search.

Table 1 shows the hyper-parameters of algorithms. We compare each of the algorithms under the best hyper-parameter. The scikit-optimize GBRT library [18], Optuna TPE library, and Optuna CMA-ES library are used [19]. The mutation methods of SSEopt are normal mutation (SSEopt_nor) and rank-based mutation (SSEopt_rnk).

Table 1. Hyper-parameters of algorithms.

Algorithm	Params	Range
SSEopt_nor	P_{m}	0.05, 0.10, 0.15
SSEopt_rnk	P_{mmax}	0.05, 0.10, 0.15, 0.20
GBRT	initial points	5, 10, 20
TPE	γ	0.01, 0.02, 0.05, 0.10, 0.15

4.2. Dataset

The datasets for the numerical experiments are described in Table 2.

OpenML Mozilla4 (#1046) [20–22] is a binary classification task for recurrent event modeling on software.

Table 2. Dataset.

Dataset	Features	Instances	Task
OpenML Mozilla4	5+1	15 545	Classification
OpenML EEG Eye State	14+1	14 980	Classification
UCI Abalone	10+1	4177	Regression
UCI Wine Quality	11+1	6497	Regression

OpenML EEG Eye State (#1471) [20, 22] is a task of a binary classification to predict the eye-closed or eye-open state from EEG measurement. The features are 14 EEG measurements from the Emotiv EEG Neuroheadset.

UCI Abalone dataset [23] is a regression task to predict the age of abalone from physical measurements. The features are length, height, shell weight and so on.

UCI Wine Quality dataset [24] is a regression task to predict wine quality. The features are acidity, sugar, pH and so on.

4.3. Search range

Tables 3 and 4 show the search ranges. The search range A (Table 3) includes only discrete parameters, while categorical parameters are added in the search range B (Table 4).

Table 3. Search range A (discrete only).

Parameter	Range	Type
learning_rate	$[0.02, 0.04, ..., 0.30]$	discrete
max_depth	$[1, 2, ..., 20]$	discrete
min_child_weight	$[1, 2, ..., 20]$	discrete
subsample	$[0.30, 0.35, ..., 1.00]$	discrete
colsample_bytree	$[0.30, 0.35, ..., 1.00]$	discrete

Table 4. Search range B (discrete and categorical).

Parameter	Range	Type
booster	['gbtree', 'gblinear', 'dart']	categorical
learning_rate	$[0.02, 0.04, ..., 0.30]$	discrete
max_depth	$[1, 2, ..., 20]$	discrete
min_child_weight	$[1, 2, ..., 20]$	discrete
subsample	$[0.30, 0.35, ..., 1.00]$	discrete
colsample_bytree	$[0.30, 0.35, ..., 1.00]$	discrete
learning objective	['squared loss', 'squared log loss']	categorical

4.4. Result

4.4.1. Binary classification tasks. The results of classification tasks are shown in Fig. 3. The horizontal and vertical axes represent trials and the mean best R2 at each trial, respectively. Figures 3a (discrete) and 3b (discrete and categorical) are the results of the OpenML Mozilla4 dataset. Figures 3c (discrete) and 3d (discrete and categorical) are the results of the OpenML EEG Eye State dataset. In all figures, SSEopt, GBRT and TPE are better than Random Search and CMA-ES. Figures 3a and 3c show that TPE and GBRT can find better solutions than SSEopt. Figures 3b and 3d show that SSEopt can find better solutions than TPE and GBRT. Therefore, it can be summarized that SSEopt is more effective for the optimization of hyper-parameters including not only discrete parameters alone but also both discrete and categorical parameters.

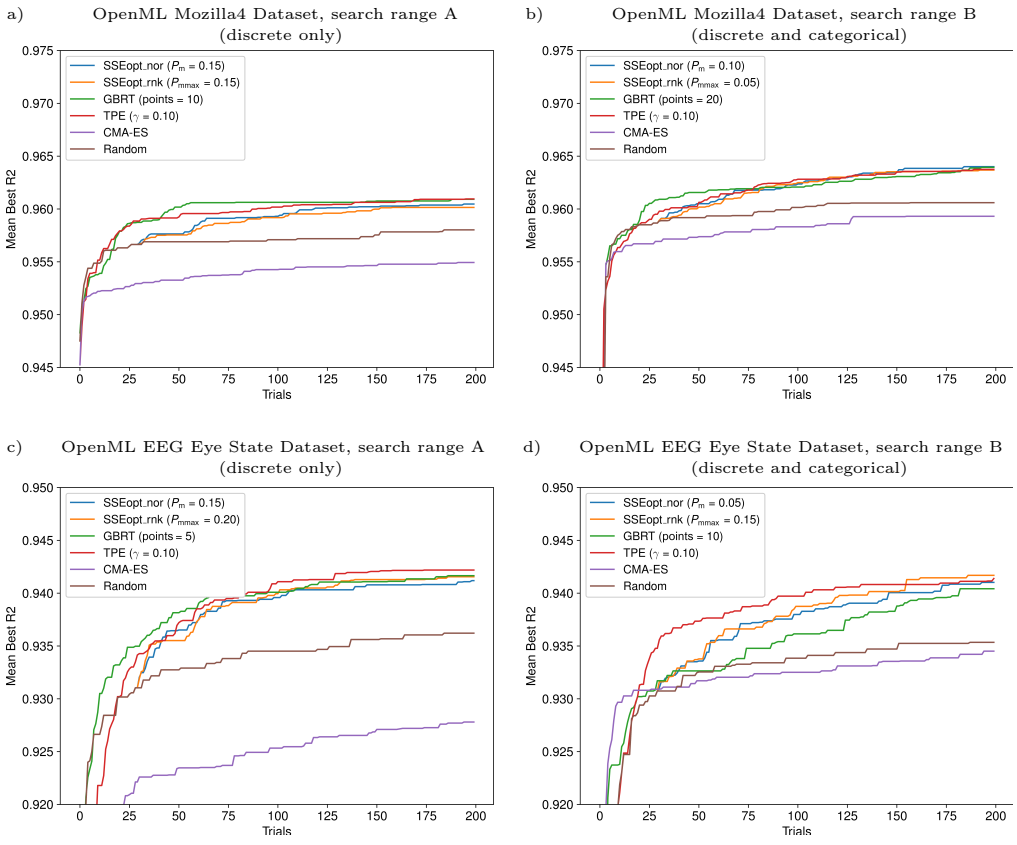

FIG. 3. Mean best R2 at each trial (classification tasks).

Table 5 shows the mean values of the optimal parameters. Refer to Subsec. 4.3 for the search range.

TABLE 5. Mean values of the optimal parameters (classification tasks).

Range	Algorithm	booster (gbtree, gblinear, dart)	lr (SD)	maxdepth (SD)	minchildwt (SD)	subsample (SD)	colsamplebytree (SD)	objectice (sqr, sqrlog)
Dataset "Mozilla4"								
A	SSEopt_nor		0.182 (0.062)	17.2 (2.2)	1.4 (0.5)	0.870 (0.068)	0.465 (0.050)	
	SSEopt_rnk		0.196 (0.069)	17.0 (2.3)	1.6 (0.8)	0.885 (0.092)	0.565 (0.175)	
	GBRT		0.138 (0.060)	18.2 (1.3)	1.0 (0.0)	0.925 (0.087)	0.495 (0.099)	
	TPE		0.192 (0.039)	16.1 (2.0)	1.0 (0.0)	0.910 (0.092)	0.545 (0.140)	
	CMA-ES		0.156 (0.060)	11.0 (3.0)	7.2 (3.7)	0.945 (0.052)	0.580 (0.152)	
	Random		0.174 (0.077)	17.0 (2.8)	1.4 (0.7)	0.870 (0.112)	0.595 (0.181)	
B	SSEopt_nor	50%, 0%, 50%	0.148 (0.062)	17.3 (1.4)	3.1 (1.9)	0.815 (0.092)	0.535 (0.116)	100%, 0%
	SSEopt_rnk	40%, 0%, 60%	0.158 (0.067)	15.2 (2.5)	2.3 (0.8)	0.760 (0.130)	0.540 (0.128)	100%, 0%
	GBRT	50%, 0%, 50%	0.130 (0.055)	16.8 (2.4)	1.5 (0.5)	0.840 (0.137)	0.550 (0.124)	100%, 0%
	TPE	50%, 0%, 50%	0.144 (0.057)	15.7 (3.6)	1.8 (0.7)	0.775 (0.136)	0.545 (0.125)	80%, 20%
	CMA-ES	40%, 0%, 60%	0.262 (0.030)	11.0 (1.3)	9.9 (2.0)	0.780 (0.123)	0.610 (0.143)	100%, 0%
	Random	50%, 0%, 50%	0.122 (0.064)	15.4 (2.5)	2.9 (1.9)	0.735 (0.143)	0.595 (0.123)	90%, 10%
Dataset "EEG Eye State"								
A	SSEopt_nor		0.240 (0.037)	15.9 (3.1)	1.9 (0.7)	0.805 (0.093)	0.885 (0.105)	
	SSEopt_rnk		0.252 (0.036)	16.6 (2.3)	1.6 (0.7)	0.830 (0.112)	0.865 (0.098)	
	GBRT		0.264 (0.027)	15.9 (2.8)	1.0 (0.0)	0.815 (0.114)	0.940 (0.073)	
	TPE		0.238 (0.024)	15.0 (2.6)	1.3 (0.5)	0.830 (0.135)	0.850 (0.081)	
	CMA-ES		0.264 (0.040)	11.5 (2.6)	7.0 (3.0)	0.850 (0.114)	0.910 (0.058)	
	Random		0.226 (0.051)	16.0 (3.3)	1.4 (0.5)	0.865 (0.114)	0.870 (0.117)	
B	SSEopt_nor	60%, 0%, 40%	0.140 (0.041)	17.5 (3.0)	15.0 (4.7)	0.800 (0.077)	0.815 (0.078)	100%, 0%
	SSEopt_rnk	40%, 0%, 60%	0.118 (0.032)	17.6 (1.6)	14.7 (4.9)	0.755 (0.117)	0.825 (0.103)	100%, 0%
	GBRT	70%, 0%, 30%	0.128 (0.038)	18.3 (1.2)	16.1 (3.6)	0.725 (0.138)	0.905 (0.057)	100%, 0%
	TPE	50%, 0%, 50%	0.106 (0.024)	19.0 (0.8)	14.7 (4.0)	0.725 (0.140)	0.860 (0.094)	100%, 0%
	CMA-ES	70%, 0%, 30%	0.142 (0.019)	11.3 (1.3)	10.2 (1.5)	0.720 (0.133)	0.840 (0.073)	100%, 0%
	Random	50%, 0%, 50%	0.100 (0.028)	18.3 (1.2)	14.7 (4.9)	0.685 (0.121)	0.815 (0.116)	100%, 0%

Table 6 shows a comparison of computation time. The computation times of SSEopt, GBRT, and TPE are longer than CMA-ES and Random Search.

TABLE 6. Mean computation time of classification tasks.

Experiment	SSEopt_nor (sec)	SSEopt_rnk	GBRT	TPE	CMA-ES	Random
a) Mozilla4, range A	137.2	132.1	150.0	137.0	95.9	80.7
b) Mozilla4, range B	440.3	384.9	408.7	422.1	184.0	179.0
c) EEG Eye State, range A	199.4	202.9	203.6	205.7	138.0	121.2
d) EEG Eye State, range B	434.7	503.9	451.6	526.8	269.3	249.4

4.4.2. Regression tasks. The results of regression tasks are shown in Fig. 4. The horizontal and the vertical axes represent trials and the mean best accuracy at each trial, respectively. Figures 4a (discrete) and 4b (discrete and categorical)

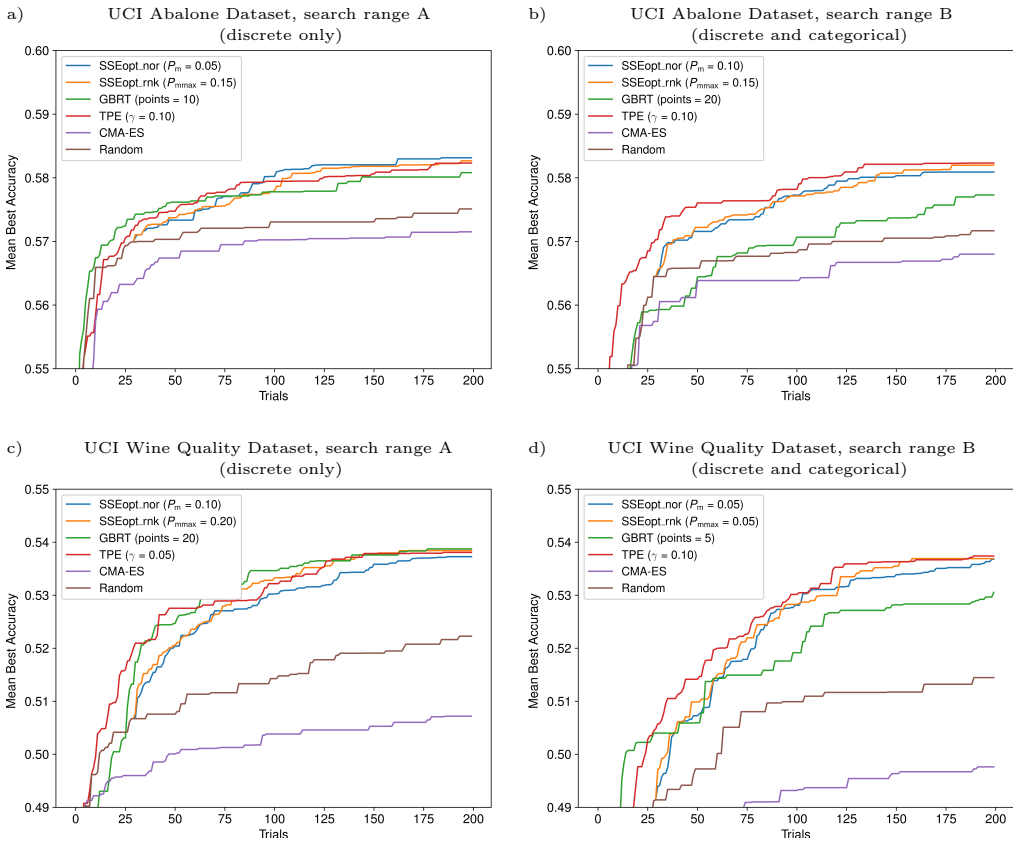

FIG. 4. Mean best accuracy at each trial (regression tasks).

TABLE 7. Mean values of the optimal parameters (regression tasks).

Range	Algorithm	booster (gbtree, gblinear, dart)	lr (SD)	maxdepth (SD)	minchildwt (SD)	subsample (SD)	colsamplebytree (SD)	objectice (sqr, sqrlog)
Dataset "Abalone"								
A	SSEopt_nor		0.088 (0.039)	4.8 (1.2)	9.7 (6.4)	0.560 (0.176)	0.885 (0.158)	
	SSEopt_rnk		0.096 (0.069)	5.1 (1.8)	11.4 (4.9)	0.505 (0.181)	0.900 (0.107)	
	GBRT		0.128 (0.092)	7.2 (6.7)	9.0 (5.8)	0.435 (0.132)	0.920 (0.084)	
	TPE		0.064 (0.008)	5.5 (0.5)	13.7 (4.7)	0.475 (0.119)	0.890 (0.146)	
	CMA-ES		0.050 (0.013)	10.2 (2.6)	11.8 (1.6)	0.520 (0.229)	0.850 (0.107)	
	Random		0.086 (0.035)	6.2 (3.2)	15.2 (5.3)	0.560 (0.158)	0.825 (0.186)	
B	SSEopt_nor	60%, 0%, 40%	0.110 (0.059)	4.6 (1.5)	8.6 (4.5)	0.635 (0.112)	0.855 (0.137)	100%, 0%
	SSEopt_rnk	70%, 0%, 30%	0.090 (0.062)	5.0 (1.5)	11.1 (4.6)	0.525 (0.127)	0.815 (0.155)	100%, 0%
	GBRT	40%, 0%, 60%	0.068 (0.016)	8.1 (4.9)	19.4 (1.5)	0.325 (0.046)	0.870 (0.110)	100%, 0%
	TPE	50%, 0%, 50%	0.068 (0.016)	5.9 (1.5)	11.6 (4.3)	0.515 (0.105)	0.725 (0.212)	100%, 0%
	CMA-ES	70%, 0%, 30%	0.054 (0.030)	9.7 (3.2)	10.9 (1.7)	0.545 (0.104)	0.810 (0.173)	100%, 0%
	Random	50%, 0%, 50%	0.086 (0.040)	5.3 (3.1)	9.8 (5.4)	0.570 (0.138)	0.745 (0.149)	100%, 0%
Dataset "Wine"								
A	SSEopt_nor		0.074 (0.016)	16.5 (1.6)	2.4 (1.2)	0.905 (0.076)	0.585 (0.123)	
	SSEopt_rnk		0.070 (0.013)	17.6 (1.6)	2.1 (0.8)	0.885 (0.112)	0.585 (0.143)	
	GBRT		0.070 (0.013)	18.7 (1.4)	1.6 (1.2)	0.955 (0.047)	0.540 (0.073)	
	TPE		0.068 (0.010)	18.0 (2.3)	1.7 (0.9)	0.910 (0.094)	0.585 (0.074)	
	CMA-ES		0.136 (0.033)	12.0 (1.4)	9.6 (2.2)	0.920 (0.093)	0.640 (0.097)	
	Random		0.086 (0.027)	18.2 (1.7)	3.4 (2.8)	0.890 (0.089)	0.740 (0.184)	
B	SSEopt_nor	40%, 0%, 60%	0.088 (0.035)	17.4 (2.1)	2.8 (1.3)	0.910 (0.086)	0.590 (0.086)	100%, 0%
	SSEopt_rnk	50%, 0%, 50%	0.070 (0.013)	17.4 (2.3)	1.9 (0.5)	0.860 (0.092)	0.700 (0.124)	100%, 0%
	GBRT	60%, 0%, 40%	0.074 (0.016)	18.1 (2.0)	2.0 (1.3)	0.920 (0.095)	0.585 (0.095)	100%, 0%
	TPE	60%, 0%, 40%	0.074 (0.016)	16.0 (2.4)	1.9 (0.8)	0.840 (0.109)	0.675 (0.135)	100%, 0%
	CMA-ES	50%, 0%, 50%	0.132 (0.032)	10.6 (1.7)	9.3 (3.3)	0.875 (0.117)	0.730 (0.140)	100%, 0%
	Random	40%, 0%, 60%	0.100 (0.024)	15.8 (2.5)	4.6 (2.3)	0.820 (0.169)	0.635 (0.116)	100%, 0%

present the results of the UCI Abalone dataset. Figures 4c (discrete) and 4d (discrete and categorical) depict the results of the UCI Wine Quality dataset. In all figures, SSEopt, GBRT and TPE are better than Random Search and CMA-ES. In the search range A (discrete only), as shown in Figs. 4a and 4c, SSEopt and TPE perform as good as GBRT and TPE. On the other hand, in search range B (discrete and categorical), as shown in Figs. 4b and 4d, SSEopt is as good as TPE and they are both better than GBRT.

Table 7 shows the mean values of the optimal parameters. Refer to Subsec. 4.3 for the search range.

Table 8 shows a comparison of computation time. In search range A (discrete only), the computation time of SSEopt is as short as TPE. On the other hand, in search range B (discrete and categorical), the computation time of SSEopt is shorter than TPE but longer than GBRT.

Table 8. Mean computation time of regression tasks.

Experiment	SSEopt_nor (sec)	SSEopt_rnk	GBRT	TPE	CMA-ES	Random
a) Abalone, range A	33.5	35.4	71.5	34.3	48.7	49.9
b) Abalone, range B	41.0	40.5	77.8	58.0	45.3	41.9
c) Wine, range A	167.3	167.3	173.7	146.0	77.3	74.7
d) Wine, range B	228.3	183.6	131.7	250.5	67.0	65.5

4.5. Discussion

As shown in Figs. 3 and 4, the final R2 scores of SSEopt, TPE, and GBRT are almost the same in some experiments, but SSEopt and TPE are better than GBRT in the regression tasks (discrete and categorical). CMA-ES and Random Search exhibit worse performance than other algorithms. These results show that the type of parameters (discrete, categorical, continuous) can strongly affect the results. CMA-ES for real-valued spaces is recommended in [7].

Figures 3d and 4c show that SSEopt with rank-based mutation is slightly better than normal mutation. Since the rank-base mutation tends to keep the diversity of individuals in the population, it can prevent good solutions from being destroyed. This mutation will work effectively in a complex space.

Tables 6 and 8 compare the computation time of SSEopt, GBRT, and TPE is longer than CMA-ES and Random Search. The results of the classification tasks are shown in Table 6. Figure 5 shows that SSEopt, GBRT, and TPE are better than CMA-ES and Random Search, so the good parameters, which are searched by SSEopt, GBRT, and TPE intensively, require a long computation time. The computation time on the Wine dataset (regression tasks) is also similar to this situation. The computation time mainly consists of computing the objective

function values and estimating optimal parameters for the next iteration. Each XGBoost task takes a long time, so the former time is usually longer than the latter time.

5. Conclusions

The optimization algorithm of the hyper-parameters XGBoost was presented in this study. The proposed algorithm is the extension of SSE to hyper-parameter optimization in XGBoost. The proposed algorithm was compared with the traditional algorithms such as GBRT, TPE, CMA-ES, and Random Search in both binary classification and regression tasks. Mozilla4 and EEG Eye datasets were used for binary classification task UCI Abalone and UCI Wine Quality datasets were employed for regression task. The sets of hyper-parameters to be optimized included discrete parameters only (range A) and discrete and categorical parameters (range B).

First, finally obtained solutions were discussed. In all examples, SSEopt, GBRT and TPE are better than Random Search and CMA-ES. In the binary classification task, TPE and GBRT found better solutions than SSEopt when the hyper-parameters were discrete alone. SSEopt can find better solutions than TPE and GBRT when the hyper-parameters are discrete and categorical. In the regression task, SSEopt and TPE are as good as GBRT and TPE when the hyper--parameters were discrete alone. SSEopt is as good as TPE and they are better than GBRT when the hyper-parameters are discrete and categorical.

Secondly, the algorithms were compared from the view-point of computational time. The computation time of SSEopt, GBRT, and TPE was longer than CMA-ES and Random Search in all examples. In the binary classification task, the computational task of SSEopt was as long as GBRT and TPE and longer than CMA-ES and Random Search. The computation time on the Wine dataset (regression tasks) was also similar to this situation. Computation time mainly consisted of computing the objective function values and estimating optimal parameters for the next iteration.

The simulation results showed that the convergence property of the proposed algorithm was better than the random search and CMA-ES and then, almost equal to or better than GBRT and TPE. Moreover, the proposed algorithm needs only three control parameters: population size, generation size, and mutation rate.

References

1. T. Chen, C. Guestrin, XGBoost: A scalable tree boosting system, [in:] *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016, doi: 10.1145/2939672.2939785.

2. J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, *Journal of Machine Learning Research*, **13**(10): 281–305, 2012.

3. R.G. Mantovani, A.L. Rossi, J. Vanschoren, B. Bischl, A. C. De Carvalho, Effectiveness of random search in SVM hyper-parameter tuning, [in:] *Proceedings of 2015 International Joint Conference on Neural Networks*, Killarney, Ireland, pp. 1–8, 2015, doi: 10.1109/IJCNN.2015.7280664.

4. A.C. Florea, R. Andonie, Weighted random search for hyperparameter optimization, *International Journal of Computers, Communications and Control*, **14**(2): 154–169, 2019.

5. Y. Xia, C. Liu, Y.Y. Li, N. Liu, A boosted decision tree approach using Bayesian hyper-parameter optimization for credit scoring, *Expert Systems with Applications*, **78**: 225–241, 2017, doi: 10.1016/j.eswa.2017.02.017.

6. J. Snoek, H. Larochelle, R.P. Adams, Practical Bayesian optimization of machine learning algorithms, *Advances in Neural Information Processing Systems*, **25**: 2960–2968, 2012.

7. M. Feurer, F. Hutter, Hyperparameter optimization, [in:] F. Hutter, L. Kotthoff, J. Vanschoren [Eds.], *Automated Machine Learning: Methods, Systems, Challenges*, pp. 3–33, Springer, Cham, 2019, doi: 10.1007/978-3-030-05318-5_1.

8. N. Hansen, S.D. Müller, P. Koumoutsakos, Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES), *Evolutionary Computation*, **11**(1): 1–18, 2003, doi: 10.1162/106365603321828970.

9. F. Friedrichs, C. Igel, Evolutionary tuning of multiple SVM parameters, *Neurocomputing*, **64**: 107–117, 2005, doi: 10.1016/j.neucom.2004.11.022.

10. I. Loshchilov, F. Hutter, CMA-ES for hyperparameter optimization of deep neural networks, *arXiv*, 2016, arXiv:1604.07269v1.

11. A.N. Aizawa, Evolving SSE: A stochastic schemata exploiter, [in:] *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, Orlando, FL, USA, Vol. 1, pp. 525–529, 1994, doi: 10.1109/ICEC.1994.349895.

12. A.N. Aizawa, Evolving SSE: A new population-oriented search scheme based on schemata processing, *Systems and Computers in Japan*, **27**(2): 41–52, 1996, doi: 10.1002/scj.4690270204.

13. T. Maruyama, E. Kita, Extension of stochastic schemata exploiter to real-valued problem, *The Special Interest Group MPS Technical Reports of Information Processing Society of Japan*, **61**: 17–20, 2006.

14. T. Maruyama, E. Kita, Investigation of real-valued stochastic schemata exploiter, *Information Processing Society of Japan Transactions on Mathematical Modeling and its Applications*, **48**: 10–22, 2007.

15. L. Breiman, *Arcing the Edge*, Technical Report 486, Statistics Department, University of California, Berkeley, CA, 1997.

16. J.H. Friedman, Greedy function approximation: A gradient boosting machine, *Annals of Statistics*, **29**(5): 1189–1232, 2001.

17. J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-parameter optimization, [in:] *Proceedings of the 24th International Conference on Neural Information Processing Systems*, Granada, Spain, pp. 2546–2554, 2011.

18. T. Head, M. Kumar, H. Nahrstaedt, G. Louppe, I. Shcherbatyi, scikitoptimize/scikit-optimize (v0.8.1), 2020, https://zenodo.org/records/4014775.

19. T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, Optuna: A next-generation hyperparameter optimization framework, [in:] *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2623–2631, 2019, doi: 10.1145/3292500.3330701.

20. M. Feurer*et al.*, OpenML-Python: An extensible Python API for OpenML, *The Journal of Machine Learning Research*, **22**(1): 4573–4577, 2019.

21. A.G. Koru, D. Zhang, H. Liu, Modeling the effect of size on defect proneness for open-source software, [in:] *29th International Conference on Software Engineering (ICSE'07 Companion)*, Minneapolis, MN, USA, pp. 115–124, 2007, doi: 10.1109/ICSECOMPANION.2007.54.

22. J. Vanschoren, J.N. van Rijn, B. Bischl, L. Torgo, OpenML: Networked science in machine learning, *SIGKDD Explorations*, **15**(2): 49–60, 2013, doi: 10.1145/2641190.2641198.

23. W.J. Nash, T.L. Sellers, S.R. Talbot, A.J. Cawthorn, W.B. Ford, *The population biology of abalone (Haliotis species) in Tasmania. I. Blacklip abalone (H. rubra) from the North Coast and Islands of Bass Strait*, Technical Report, No. 48, Sea Fisheries Division, Department of Primary Industry and Fisheries, Tasmania, 1994.

24. P. Cortez, A. Cerdeira, F. Almeida, T. Matos, J. Reis, Modeling wine preferences by data mining from physicochemical properties, *Decision Support Systems*, **47**(4): 547–553, 2009.

Received May 20, 2023; revised version November 13, 2023;
accepted December 11, 2023; published online February 29, 2024.