Efficient AutoML via combinational sampling

Nguyen, D.A.; Kononova, A.V.; Menzel, S.; Sendhoff, B.; Bäck T.H.W.

Efficient AutoML via Combinational Sampling

Duc Anh Nguyen*, Anna V. Kononova*, Stefan Menzel§, Bernhard Sendhoff§, and Thomas Bäck*
*Leiden Institute of Advanced Computer Science (LIACS), Leiden University, The Netherlands
Email: {d.a.nguyen,a.kononova,t.h.w.baeck}@liacs.leidenuniv.nl
§Honda Research Institute Europe (HRI-EU), Offenbach/Main, Germany
Email: {stefan.menzel,bernhard.sendhoff}@honda-ri.de

***Abstract*—Automated machine learning (AutoML) aims to automatically produce the best machine learning pipeline, i.e., a sequence of operators and their optimized hyperparameter settings, to maximize the performance of an arbitrary machine learning problem. Typically, AutoML based Bayesian optimization (BO) approaches convert the AutoML optimization problem into a Hyperparameter Optimization (HPO) problem, where the choice of algorithms is modeled as an additional categorical hyperparameter. In this way, algorithms and their local hyperparameters are referred to as the same level. Consequently, this approach makes the resulting initial sampling less robust. In this study, we describe a first attempt to formulate the AutoML optimization problem as its nature instead of transfer it into a HPO problem. To take advantage of this paradigm, we propose a novel initial sampling approach to maximize the coverage of the AutoML search space to help BO construct a robust surrogate model. We experiment with 2 independent scenarios of AutoML with 2 operators and 6 operators over 117 benchmark datasets. Results of our experiments demonstrate that the performance of BO significantly improved by using our sampling approach.**

***Keywords*—Robust AutoML, Optimization, initial sampling**

I. Introduction

Automated machine learning (AutoML) aims at producing the best machine learning pipeline in order to minimize considerably human efforts in the machine learning development cycle to real-world problems. Recent studies have proved the power of AutoML in many real-world problems with minimal human effort while improved performance (e.g., [1], [2]). The existing AutoML approaches (e.g., [3], [4]) can be seen as optimization processes, where the best ML pipeline is searched for. Every such pipeline includes an architecture and a set of hyperparameter settings. A typical AutoML approach has three fundamental parts – a search space, an optimizer, and a target program. The search space defines the feasible search domain. The target program is a particular ML pipeline, resulting in a real-valued performance when evaluated on the target dataset (given by user). The optimizer is used to find the best setting that maximizes the performance of the target program. In other words, the goal of AutoML is to explore an efficient target program out of a large set of possibilities defined by the search space. Bayesian Optimization (BO) is a commonly used approach in AutoML as it has been successfully used in Hyperparameter Optimization (HPO) problems and plays a role of an optimizer in many AutoML frameworks, e.g., Auto-sklearn [3], Auto-Weka [4], and Hyperopt-sklearn (HPsklearn) [5]. BO is an efficient global optimization approach (in terms of the number of function evaluations), where the trade-off between local exploitation and global exploration is well-handled. Therefore, in this work, we focus on improved BO in using it to solve the AutoML optimization problem.

Traditionally, the AutoML optimization problem is treated as a HPO process, where the optimizer is inherited from the HPO domain. Since HPO was originally developed to find the best hyperparameter setting from a single algorithm. Thus, it naturally does not consider the choice of algorithm. The choice of algorithms is then modeled as an extra categorical hyperparameter. Consequently, this HPO-based approach in handling the choice of algorithm is a mismatch with the nature of the AutoML optimization problem.

The search space in AutoML approach is large due to the many possible choices of algorithms for the operators in the pipeline. The main reason for this is the desired automated nature of the AutoML solution which aims to solve ML problems without (or with minimal) human effort and be able to work well for a wide range of ML problems. Unfortunately, the best algorithm for all problems does not exist. Thus, to maximize ML performance on an arbitrary problem, AutoML has to integrate the choice of as many algorithms as possible.

Finally, we note that each choice of an algorithm has a set of local hyperparameters to be tuned (see the discussion regarding CASH in Section III-A). However, including many algorithms in the search space naturally leads to BO being slow convergence or might be stuck in a local optimum [6]–[8]. One reason for this is that the initial sampling step in AutoML is typically restricted to a tiny budget, which is much smaller than the number of possible pipelines that can be constructed in the search space. The reason for this setting is because the effectiveness of BO becomes evident mainly in the later stages of optimization when it learns to produce better configurations. Many well-known sampling approaches, e.g., discrepancy-based quasi-random (quasi-random) [8], Latin Hypercube (LHD) [9], have been employed for initial sampling in this optimization context. However, they have shown themselves not robust enough [6], [10], [11], since, they have been used in conjunction with the traditional approach of solving an AutoML optimization problem which, as explained above, consists in converting it to an HPO, thus rendering obscure the differences between the choice of an algorithm and the choice of algorithm's parameters.

To alleviate the above issues, in this paper, we propose a new two-fold approach to improve BO used in AutoML optimization: 1. use a new hyperparameter class to model the choice of algorithms instead of categorical hyperparameter, namely "Algorithm choice"; 2. group the similar operator algo-

rithms when allocating initial sampling budget. Additionally, building on top of other sampling approaches, we propose a novel sampling method that aims to allocate reasonable budgets for each set of algorithms to maximize the coverage of sampling areas in terms of the grouping of algorithms to provide a robust surrogate model. In other words, our proposed approach is complementary to other sampling approaches rather than competitive, aiming to optimize performance for the search space of AutoML.

To summarize, our main contributions are as follows:

1) We formulate AutoML as an optimization process of the ML pipeline. Under this formulation, the search space is hierarchical with three major levels: Operator, Algorithm, and Hyperparameter.
2) We introduce a new hyperparameter class to model the choice of algorithms in an operator (instead of using categorical hyperparameter as done traditionally), together with an approach to grouping the algorithms that might have similarities. To the best of our knowledge, this is the first work modeling the choice of algorithms and the relationship between algorithms.
3) We propose a robust sampling technique based on the combination of algorithms over operators, which aims to improve the quality and robustness of the surrogate model. We present results of the empirical studies on 117 benchmark datasets in 2 independent scenarios of AutoML with 2 operators and 6 operators that demonstrate the improved performance of BO the help of our proposed sampling approach.

The remainder of this paper is organized as follows. The AutoML optimization problem is defined in Section II. In Section III, the relevant background knowledge on CASH problem and Bayesian optimization are provided. Next, our contributions are highlighted in Section IV, and Section V lays out the experimental setup. Experimental results are discussed in Section VI. Finally, the paper is concluded, and further work is outlined in Section VII.

II. Framework Formulation

A generic ML pipeline $p : \mathbb{X} \rightarrow \mathbb{Y}$ designed for solving problem $\mathcal{P}$ is a sequence of operators that transforms a set of features $x \in \mathbb{X}$ into a target value $y \in \mathbb{Y}$ which can be e.g., a predicted value for a regression problem or a label for a classification problem. Examples of possible pipeline operators depend on problem $\mathcal{P}$ and can include data pre-processing, encoding, feature selection, resampling, etc.

Let $\mathcal{M} = (\mathcal{O}_1, \ldots, \mathcal{O}_z)$ denote the sequence of operators in the pipeline p, where each subsequent operator is applied to the output of the previous operator: $p(x) = \mathcal{O}_z(\mathcal{O}_{z-1}(\ldots(\mathcal{O}_1(x))\ldots))$. Functionality of each such operator can typically be delivered by one of the multiple available ML algorithms: here we assume $\mathcal{O}_{i\in\{1,\ldots,z-1\}} = \{\varnothing, \mathcal{A}_i^1, \ldots, \mathcal{A}_i^{n_i}\}$ for all operators except the last and $\mathcal{O}_z = \{\mathcal{A}_z^1, \ldots, \mathcal{A}_z^{n_z}\}$ for the last operator which defines the learning algorithm – i.e. unlike the first $z-1$ operators, the last operator $\mathcal{O}_z$ has to be selected and cannot be $\varnothing$.

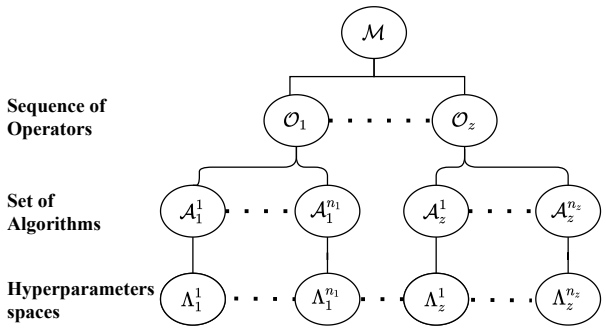

Fig. 1. The structure of AutoML search space Λ

Furthermore, since typically all algorithms have parameters, let $\Lambda = \{\{\Lambda_1^1, \ldots, \Lambda_1^{n_1}\}, \ldots, \{\Lambda_z^1, \ldots \Lambda_z^{n_z}\}\}$ be a set of hyperparameters sets of all considered algorithms for all considered operators. The overall structure of the resulting AutoML search space Λ is illustrated in Fig. 1. For readability, let $\mathcal{A}_{i,\lambda}$ represent algorithm $\mathcal{A}_i$ selected for operator $\mathcal{O}_i$ and configured by a hyperparameter setting $\lambda \in \Lambda_i^j$, $j \leq n_i$. Then, we denote a pipeline with algorithms selected and configured with their hyperparameters for all operators in the pipeline p as $p_{(\mathcal{A}_{1,\lambda},\ldots,\mathcal{A}_{z,\lambda})}$.

In order to eventually solve the AutoML problem (see Equation (4)) and find the best choice of algorithms and their hyperparameters for the operators of the pipeline, every such choice needs to be evaluated. Let $R(\hat{y}, y)$ denote a metric that returns the accuracy of value $\hat{y}$ predicted by the pipeline compared to the real value y. Then, performance f of pipeline configuration $p_{(\mathcal{A}_{1,\lambda},\ldots,\mathcal{A}_{z,\lambda})}$ when trained on a training dataset $D_t = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ and evaluated on a validation dataset $D_v = \{(x_{m+1}, y_{m+1}), \ldots, (x_{m+t}, y_{m+t})\}$ is calculated as:

$$f(p_{(\mathcal{A}_{1,\lambda},\ldots,\mathcal{A}_{z,\lambda})}, D_t, D_v) = \frac{1}{t}\sum_{j=1}^{t} R(\hat{y}_{m+j}, y_{m+j}) \quad (1)$$

To prevent overfitting when solving the AutoML problem, the k-fold cross-validation can be added to Equation (1), which leads to the following formulation of the AutoML problem:

$$(\mathcal{A}_{1,\lambda}, \ldots, \mathcal{A}_{z,\lambda})^* = \arg\max_{l,\lambda} \frac{1}{k}\sum_{j=1}^{k} f\left(p_{(\mathcal{A}_{1,\lambda},\ldots,\mathcal{A}_{z,\lambda})^l}, \mathcal{D}_t^j, \mathcal{D}_v^j\right) \quad (2)$$

where $(\mathcal{A}_1, \ldots, \mathcal{A}_z)^l \in \bigtimes_{i=1}^{z} \mathcal{O}_i$ are all possible choices of algorithms for all pipeline operators, $\lambda = \{(\lambda_1, \ldots, \lambda_z) | \lambda_1 \in \Lambda_1^{j_1}, \ldots, \lambda_z \in \Lambda_z^{j_z}\}$ are algorithms' hyperparameters and $f\left(p_{(\mathcal{A}_{1,\lambda},\ldots,\mathcal{A}_{z,\lambda})^l}, \mathcal{D}_t^{(j)}, \mathcal{D}_v^{(j)}\right)$ is performance of the sequence operators and their corresponding hyperparameter choices when trained and evaluated on the j^{th} data fold $\mathcal{D}_t^j$ and $\mathcal{D}_v^j$, correspondingly.

III. Background

In this section, we first provide a brief introduction of the traditional approach to handle AutoML (Section III-A), and

the Bayesian optimization approaches (Section III-B).

A. AutoML as a CASH problem

In practice, the AutoML optimization problem is commonly referred to as the Combined Algorithm Selection and Hyperparameter optimization (CASH) [3], [4] and Full Model Selection (FMS) [12] problem, in which the choice of algorithm is modeled as an additional categorical hyperparameter. Then, the AutoML optimization (AO) problem is treated as a HPO problem. As such, the choice of algorithms for each operator is modeled as an extra categorical hyperparameter λ^0. The AutoML search space is then defined as:

$$\Lambda = \Big(\underbrace{\{\lambda_1^0, \Lambda_1^1, \ldots, \Lambda_1^{n_1}\}}_{\text{corresponds to operator } \mathcal{O}_1}, \ldots, \underbrace{\{\lambda_z^0, \Lambda_z^1, \ldots, \Lambda_z^{n_z}\}}_{\text{corresponds to operator } \mathcal{O}_z} \Big) \quad (3)$$

Hence, the AO problem becomes the HPO maximizing problem:

$$\lambda^* = \arg\max_{\lambda \in \Lambda} f(\lambda) \ , \quad (4)$$

In this setting, the categorical hyperparameters after the root of this hierarchical search space (see Fig. 1) are known as the choice of algorithm for an operator. Consequently, algorithms and their local hyperparameters are treated at the same level. However, unlike the pure categorical hyperparameter, i.e., choose one in a set of nominal options, the choice of algorithms heavily affects other hyperparameters, i.e., once the algorithm is known, only its hyperparameters are relevant.

Another point worth mentioning is that HPO was originally developed to find the best hyperparameter setting from a single algorithm – a much more straightforward problem compared to the AutoML optimization problem. In addition to HPO, AutoML optimization also searches for an optimized pipeline of algorithms. In AutoML, multiple algorithms have to be considered, and those algorithms can belong to different phases in the ML pipeline, e.g., pre-processing and learning model. This pipeline is restricted by some constraints, such as the learning task, i.e., classification for supervised learning and regression (or clustering) for unsupervised learning, is the last step. For example, Auto-sklearn has up to six sequence operation steps: categorical encoder, numerical transformer, imputation transformer, rescaling, feature preprocessor, and learning operator. In comparison, Auto-Weka and HPsklearn only have two operators: preprocessor and learning operator. Although, generally speaking, AutoML can have different sizes in terms of operators and algorithms under operators, most operators are optional, but the learning operator is mandatory.

Furthermore, the used algorithms/techniques in an ML pipeline are tightly coupled since every operator step is directly affected by the previous step, e.g., the data pre-processing step aims to produce a new dataset (balanced, reduced-dimensions, etc.), which can change the performance of the subsequent operator such as the learning model. Consequently, the traditional approach in handling the choice of algorithm is a mismatch with the nature of the AutoML optimization problem.

B. Bayesian optimization

The Bayesian optimization (BO) is a commonly used approach in many AutoML frameworks, including: Auto-sklearn [3], [13], HPsklearn [5], Auto-Weka [4], [14]. BO is a historical-based approach which uses a probabilistic regression model $P(f|\mathcal{H})$ as a surrogate model of the true objective function f, based on the so-far evaluated configurations $\mathcal{H} = \{(\lambda_i, \Delta_i)_{i=1}^n\}$, where λ denoted for the evaluated configuration and Δ denoted the corresponding reward. This surrogate model is used to predict the performance of a numeric set of configurations with a much cheaper computational budget than evaluating them on the true objective function. The next configuration is chosen by maximizing some acquisition function [15], which handles the trade-off between exploration and exploitation of the search.

Gaussian processes (GPs) [16] are the traditional surrogate models for BO, which model $P(f|\mathcal{H})$ to capture the probability distribution of the reward conditioned on configuration from the historical information. GPs typically work well in low-dimensional optimization problems. However, the search spaces of AutoML problems are typically high-dimensional, structured, and mixed (discrete and continuous), which GPs do not naturally support [3], [17], [18].

An alternative to a GP is a tree-based model called Tree-structured Pazen Estimator (TPE). Instead of modeling the distribution of the true objective function f, TPE models the likelihood $P(\mathcal{H}|f)$ by using a kernel density estimator [19]. In this setting, the so-far evaluated configurations split into two density distributions of a well $l(\lambda)$ and a badly $g(\lambda)$ performing set depending on whether its performance is below or above a predefined threshold α[1]. The next configuration is then chosen by maximizing the ratio $\frac{l(\lambda)}{g(\lambda)}$. Since TPE has been successfully used in several AutoML frameworks [4], [5], here we therefore use TPE as the BO approach.

IV. The Proposed Approaches for Automated Machine Learning

Now we discuss our contributions in this study. First, we introduce the new class of hyperparameter in terms of using BO for AutoML, namely "Algorithm Choice" and a new attribute of hyperparameter to visualize the relationship of algorithms. Second, we introduce our proposed combination-based sampling approach for increasing the efficiency and robustness of AutoML . Lastly, we introduce a new BO python library for AutoML optimization and an AutoML framework that implements this paradigm.

A. Hyperparameter classes

As mentioned in Section III-A, using a categorical hyperparameter to model the choice of algorithm is a mismatch with the nature of the AutoML optimization problem. Thus, to clarify the categorical hyperparameter vs. algorithm choice in AO problems, we use a new hyperparameter class, named "Algorithm Choice", a sub-class of the categorical class.

[1]By default, $\alpha = 25\%$

TABLE I
HYPERPARAMETER TYPES AND FUNCTIONS USED IN OUR IMPLEMENTATION

Hyperparameter	Annotation	Description
Continuous	FloatParam(min, max)	Choose a float value in range of $[min, max] \cap \mathbb{R}$
Ordinal	IntegerParam(min, max)	Choose a integer value in range of $[min, max] \cap \mathbb{Z}$
Nominal	CategoricalParam($C_1, \ldots, C_n$)	Choose a value in set $\{C_1, \ldots, C_n\}$
*Algorithm	AlgorithmChoice($A_1, \ldots, A_n$)	Choose a value in set $\{A_1, \ldots, A_n\}$
Hierarchical	ConditionalParam($Parent, \{P_{value}\}, \{Child_1, \ldots, Child_n\}$)	when a HyperParam has children
Infeasible	ForbiddenParam($(Param_1, \{P^1_{value(s)}\}), (Param_2, \{P^2_{value(s)}\})$)	when the combination of $Param_1$ and $Param_2$ is forbidden
* Grouping	HyperParam($\underbrace{\{\text{value}^1_1, \ldots, \text{value}^1_{v_1}\}}_{\text{group}_1}, \ldots, \underbrace{\{\text{value}^n_1, \ldots, \text{value}^n_{v_n}\}}_{\text{group}_n}$)	each group$_i$ can be of any type: $\{Continuous, Ordinal, Nominal, Algorithm\}$

Additionally, in order to construct a robust surrogate model, BO requires good coverage of the search space [6], but as the number of algorithms increases, the number of samples needed to cover the search space increases exponentially. Past works [20], [21] have pointed out that some algorithms can be grouped based on their technical behaviors. To take advantage of this property, we introduce a new attribute of hyperparameter for grouping the choices of algorithms for an operator, e.g., the grouping of linear classifiers vs. the grouping of rule-based classifiers [20]. Moreover, this function allows different types of hyperparameters input, which is helpful in the case that some hyperparameters allow different input types. For example, the max_features[2] in the Random Forest classification algorithm implemented in scikit-learn [22], allows three value types: a categorical value, e.g., name of a predefined formula such as "sqrt" (square root of the number of features), "log2" (binary logarithm of the number of features), a real value for the ratio as the percentage of the number of features and an integer value to set the number of features directly. Table I summaries different hyperparameter classes with their semantics in our work.

B. Novel combination-based initial sampling for Bayesian optimization for AutoML optimization

The key idea of our approach is to provide an optimized coverage of the algorithm-hyperparameter search space already at the stage of the initial sampling of BO, to be able to characterize the response surface better earlier.

First, let us introduce a grouping of algorithms of operator $\mathcal{O}_i$. Lets assume that the set of all algorithms $\{\varnothing, \mathcal{A}^1_i, \ldots, \mathcal{A}^{n_i}_i\}$ available to be employed for operator $\mathcal{O}_i$[3] can be partitioned into g_i non-empty and non-overlapping subsets according to their inner workings[4]: $\{G^1_i, \ldots, G^{g_i}_i\}$, $g_i \leq n_i + 1$[5]. We call such partitioning a *grouping of algorithms*.

The operator can then be represented as $\mathcal{O}_i = \{G^1_i, \ldots, G^{g_i}_i\}$. According to our proposed combination-based initial sampling method (see Algorithm 1), the sequence of pipeline operators $\mathcal{M} = (\mathcal{O}_1, \ldots, \mathcal{O}_z)$ should be sampled in BO from the set of sets $\{\{G^1_1, \ldots, G^{n_1}_1\}, \ldots, \{G^1_z, \ldots G^{n_z}_z\}\}$ and the total initial sampling budget should be split equally per group. The main idea behind such reallocation of sampling budget is potential exploitation of similarities between the algorithms within the group: sampling fewer of similar algorithms frees up the budget to be distributed to other (different) algorithms, thus improving the coverage of algorithm-hyperparameter search space at the earlier stage of BO.

As an input parameter for our method, we require a number of data points B_{init} for the initial sampling and a maximum number of combinations K, $K \leq B_{init}$. If K is greater than the maximum number of possible combinations computed based on the input operation steps $k = \prod_{i=1}^{z} |\mathcal{O}_i|$, then we use the Algorithm 2 to randomly regroup algorithms in operators to ensure $k \leq K$. Algorithm 1 consists of the three following steps:

1) Generate the list of combinations: List out k all possible combinations of groups for all z operators; apply RANDOMREGROUPPING until k is small enough ($k \leq K$) (lines $4-6$).
2) Allocate budget to combinations: first allocate budget to all combinations based on the number of algorithms and hyperparameters behind (lines 6 - 12). Then, if there is any remaining budget B_{remain}, randomly allocate B_{remain} to the top $\frac{k}{\eta}$ combinations ordered by their size (i.e. the number of algorithms and hyperparameters in the combination). We take into account the size of the combination to allow larger combination get more chance to get a larger budget.
3) Sampling configurations: each combination s is sampled by an existing sampling approach (e.g., LHD, quasi-random, here we use quasi-random) to get a trial sequence $s_j = (G_1, \ldots, G_z)$ (lines $7-17$); Lastlty, the generated configurations must be verified by CHECKFORBIDDEN[6].

Lastly, the generated configurations are shuffled to remove potential impact of grouped configurations by combinations. This is highly recommended since, in some cases, the computational optimization budget, e.g., run time limit, ran out before finishing this initialization step.

[2] the maximum number of features can be used when splitting a node.
[3] if $i < z$ or $\{\mathcal{A}^1_z, \ldots, \mathcal{A}^{n_z}_z\}$ if $i = z$
[4] or any other user-defined logic
[5] if $i < z$ and $g_z < n_z$ otherwise
[6] an external function uses to verify a combination of algorithms/a configuration with the forbidden rules defined by user.

Algorithm 1: Combination-based sampling

Input:
$\mathcal{M}$: sequence of operators, Λ: hyperparameter spaces, B_{init}: number of initial samples, K: number of combinations of grouping of algorithms over operators, $\eta = 2$: proportion of combinations to be chosen to assign more budget if any remaining budgets are available.
Output: Θ: set of configurations

// **1-GENERATING COMBINATIONS**
1 $k = \prod_{i=1}^{z} |\mathcal{O}_i|$ // maximum number of possible combinations
2 **if** $k > K$ **then**
3 $(\mathcal{M}, k) =$ RANDOMREGROUPPING$(\mathcal{M}, K)$ // Algorithm 2
4 Create a list s of all possible combinations from $\{G_1^1, \ldots, G_1^{g_1}\} \times \ldots \times \{G_z^1, \ldots, G_z^{g_z}\}$
// **2-ALLOCATE BUDGETS TO k COMBINATIONS**
5 $l_c = \frac{B_{init}}{k}$ // number of inital samples per combination
6 $m = \frac{1}{k}\sum_{i=1}^{k}(|\Lambda_{s_i}|+|s_i|)$// average number of algorithms and hyperparameters of all combinations
7 $\Theta = \varnothing$ // set of initial configurations
8 **foreach** $j \in \{1, \ldots, k\}$ **do**
9 $l_j = \lfloor l_c \times \frac{|\Lambda_{s(j)}|+|s_j|}{m} \rfloor$
10 $l_j = \begin{cases} 1, & \text{if } l_j = 0. \\ l_j, & \text{otherwise.} \end{cases}$
11 **if** $B_{remain} = B_{init} - \sum_{i=1}^{k} j > 0$ **then**
// Randomly allocate B_{remain} to the top $\frac{k}{\eta}$ combinations based on the number of algorithms and hyperparameters
// **3-SAMPLING CONFIGURATIONS**
12 **foreach** $j \in \{1, \ldots, k\}$ **do**
13 $\Theta_j = \varnothing$ // feasible configurations in the j^{th} combination
14 **while** $|\Theta_j| \leq l_j$ **do**
15 $\Theta_j = \Theta_j \cup$ SAMPLING$(s_j, \Lambda_j, l_j - |\Theta_j|)$
// SAMPLING is done via an existing approach, here we opted quasi-random sampling with minor adjustments
16 **foreach** $\lambda \in \Theta_j$ **do**
17 **if** CHECKFORBIDDEN(λ) **then**
18 $\Theta_j = \Theta_j \setminus \lambda$
19 $\Theta = \Theta \cup \Theta_j$

RANDOMREGROUPING method used in Algorithm 1 is presented in Algorithm 2. For a sequence of operators which consists of multiple grouping of algorithms, it produces, via a regrouping, k combinations of operators ($k \leq K$) using the following steps:

1) Step 1 (lines $3-9$): Based on the number of the grouping in operators, we list out all possible solutions of regrouping to have k combinations.
2) Step 2 (line 10): Randomly choose one solution $s_{chosen} = (c_1, \ldots, c_z)$ where c_i is number of grouping to be created for the operator $\mathcal{O}_i$.
3) Step 3 (lines $12-28$): For each operator $\mathcal{O}_i$, we randomly group into c_i groups.

Algorithm 2: Random Regrouping

Input:
$\mathcal{M} = \Big(\{G_1^1, \ldots, G_1^{g_1}\}, \ldots, \{G_z^1 \ldots G_z^{g_z}\}\Big)$: sequence of operators, K: number of combinations
Output: $\mathcal{M}_{new}$: new sequence of operators, k: new number of combinations

1 $k = K$ // number of all possible combinations
2 $S = \varnothing$ // split solutions
3 $C_1 = \{1, \ldots, g_1\}, \ldots, C_z = \{1, \ldots, g_z\}$ // set of possible groupings of $\mathcal{O}_{i \in \{1,\ldots,z\}}$
// List out all split solutions
4 Create a list of all possible splits $H = \{h_i\}$ where $h_i = (c_1, \ldots, c_z) : c_j \in C_j \forall j$ // Select split solutions which can produce k combinations, $k \leq K$
5 **while** $S = \varnothing$ **do**
6 $S \leftarrow \{h = (c_1, \ldots, c_z) \in H : \Big(\prod_j c_j\Big) = k\}$
7 **if** $S = \varnothing$ **then**
8 $k = k - 1$
9 $s_{chosen} \sim \mathcal{U}(S)$// randomly choose one solution
10 $\mathcal{M}_{new} = (\varnothing_1, \ldots, \varnothing_z)$, $i = 1$
11 **foreach** $c \in s_{chosen}$ **do**
12 $n_i = |\mathcal{O}_i|$
13 **if** $c = n_i$ **then**
14 $\mathcal{O}_i = \{\{G_i^1\}, \ldots, \{G_i^{n_i}\}\}$
15 **else if** $c = 1$ **then**
16 $\mathcal{O}_i = \{G_i^1, \ldots, G_i^{n_i}\}$
17 **else**
18 $G = \varnothing$, $\mathcal{O}_0 = \mathcal{O}_i$
19 **while** $n_i > 0$ **do**
20 $G_0 = \varnothing$, $n_{size} = \lceil \frac{n_i}{c} \rceil$
21 **if** $n_i > n_{size}$ **then**
22 $G_0 =$ {Random pick n_{size} items in $\mathcal{O}_0$}
23 **else**
24 $G_0 = \{\mathcal{O}_0\}$
25 $G = G \cup G_0$, $\mathcal{O}_0 = \mathcal{O}_0 \setminus G_0$, $n_i = |\mathcal{O}_0|$
26 $\mathcal{O}_i = \{G\}$
27 $\mathcal{M}_{new}^{(i)} = \mathcal{O}_i$, $i = i + 1$
28 **return** $\mathcal{M}_{new}, k$

C. The proposed BO and AutoML approaches

To take advantage of the new hyperparameter class and the regrouping of algorithms proposed in Section IV-A and the new sampling approach introduced in Section IV-B, we introduce a new BO library for AutoML and an AutoML framework:

- We introduce a BO library for AutoML optimization, named BO4AutoML[7], where the new hyperparameter classes and our sampling approaches are implemented. In this work, we use Tree-structured Parzen Estimator (TPE) implemented in Hyperopt [23] for the surrogate model and Expected improvement (EI) [15] for the acquisition function. The schematic overview of BO4AutoML is shown in the grey rounded rectangle in Fig. 2.

[7] Available at https://github.com/ECOLE-ITN/BO4ML

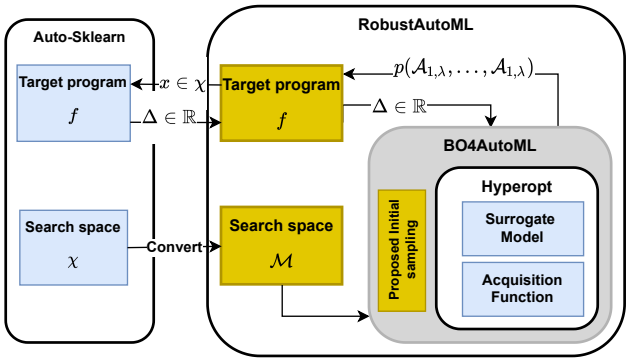

Fig. 2. Schematic overview of the proposed RobustAutoML framework

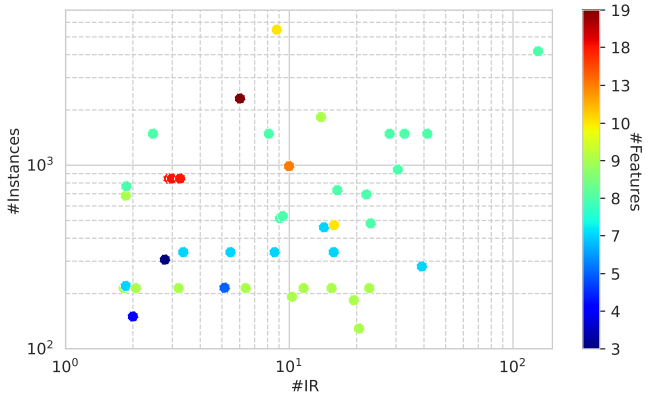

Fig. 3. Overview of the characteristics of 44 imbalanced benchmark datasets. The scatter plot shows the Imbalance Ratio (#IR) and the number of instances (#Instances) for all considered imbalanced datasets on a logarithmic scale. The color indicates the number of attributes (#Features).

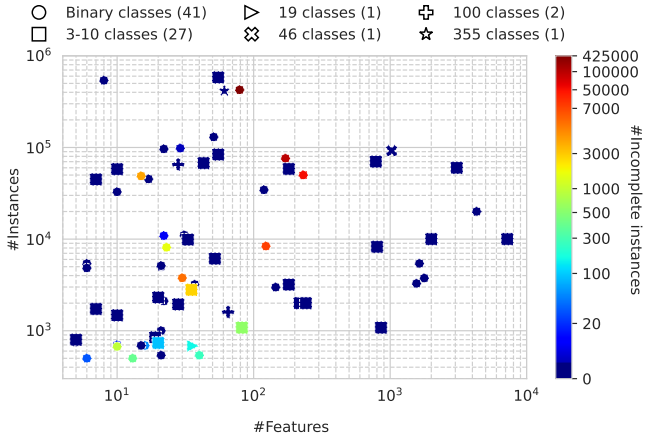

Fig. 4. Overview of the characteristics of 73 AutoML benchmark datasets. The scatter plot shows the number of features (#Features) and the number of instances (#Instances) for all examined datasets on a logarithmic scale. The color indicates the number of samples that contain missing value (#Incomplete instances). The symbols indicate the number of classes, which ranges from 2 to 355 classes and includes 41 binary-class datasets, 32 multi-class datasets.

- Built on top of BO4AutoML and the well-known AutoML framework -Auto-sklearn, we introduce a new AutoML framework[7], here we dub RobustAutoML[8], where BO4AutoML opts as the underlying optimizer and search space is converted from Auto-sklearn. The overall structure of Robust4AutoML is summarized in Fig. 2.

In this work, we use a budget of 50 samples for the initial sampling step, and the selected algorithms are grouped follows the suggestions in [20] and [22].

V. Experimental Setup

The performance of our proposed approach is empirically evaluated on two scenarios with 2 and 6 operators in the pipelines. In both scenarios, we will compare the results against the TPE approach without our proposed initial sampling approach. The first scenario (section V-A) involves the class imbalanced problems with a search space of two operators, i.e., imbalanced resampling techniques and classification algorithms, which are used in [21]. The second scenario (section V-B) will be implemented with the full search space used in the Auto-Sklearn framework [3].

A. First experiment

In this experiment, we compared the performance of our proposed BO approach, i.e., BO4AutoML, to against the BO implemented in Hyperopt [23] (Hyperopt) on the scenario of optimizing the ML pipeline of two operators, i.e., imbalanced-resampling operator and classification operator for the class-imbalanced problems. We reproduced the experimental setup of [21]:

- **Dataset:** we used 44 binary class imbalanced datasets, which are presented in Fig. 3. For each dataset, the *Imbalance Ratio* (IR)[9] on the x-axis and the number of instances (#Instances) on the y-axis. The number of attributes marks with colors. Their number of instances ranges from 129 to 5472, and their number of features ranges from 3 to 19. The full list of datasets is given in the Section II-A of the supplementary material.
- **Operators:** The first operator is the resampling operator, aiming to resamples the imbalanced input dataset to have a balanced dataset. The resampling operator includes 21 resampling approaches; they fall into 4 major groups, such as No resampling, Over-resampling (7 algorithms), Under-resampling (11 algorithms), Combine-resampling (2 algorithms). The final operator is the classification operator, with 5 classification algorithms (i.e., Support Vector Machines (SVM), Random Forest (RF), K-Nearest Neighbors (KNN), Decision Trees (DTC), and Logistic Regression (LR)).
- **Parameter setting:** We used a budget of 500 function evaluations, with the original experimental setup, data pre-processed and source code provided by [21]. Our results for Hyperopt resemble those of [21], with minor differences because of our higher versions of python environment (Python-3.7.2). We select two different values

[8]Due to the page limitation, the detailed discussion on RobustAutoML can be found in Section I of the supplementary material.

[9]The ratio of the number of majority class instances to that of minority class instances, ranges here from 1.82 to 129.44.

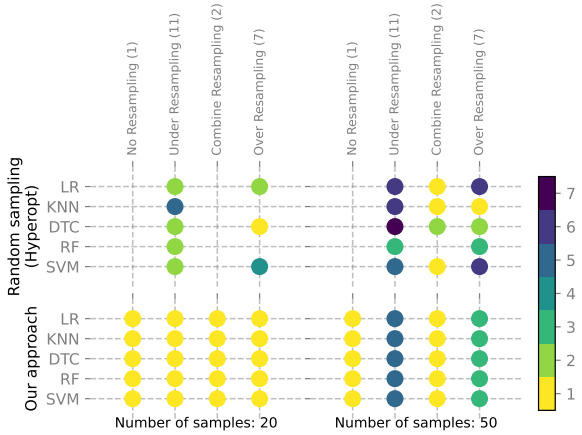

Fig. 5. Illustration on the number of samples allocated to different combination of methods in random sampling implemented in Hyperopt (top) vs. our proposed approach (bottom). Cases with 20 (left) and 50 (right) samples are shown here. Figure best viewed in color.

of the initial sample size as 20 and 50. The 5-fold cross-validation approach is used, and the averaged geometric mean values over 10 repetitions are reported.

B. Second experiment

In order to evaluate the proposed approach in a higher number of operators, we compared the performance of our AutoML framework, includes two BO approaches, to six well-known AutoML frameworks, i.e., Auto-sklearn (BO and Random search), HPsklearn, TPOT [24], ATM [25], H20 [26]. We experiment with our approaches as the same experimental setup with 73 AutoML benchmark datasets reported in [18], with their original data train/test split technique, i.e., 30% for testing and the remaining for training. The full list of datasets is given in Section II-B of the supplementary materials file.

For a fair comparison, we use a similar computational resource with that used in [18]. For clarification, all experiments are conducted using our available computation clusters, namely DAS-5 [27], each computation node (32 cores) parallelly runs 4 experiments, i.e., a fixed 8 cores for one experiment. All experiments performed 10 runs with different random seeds, with a time limit of 1 hour. The performance of a single configuration is limited to 10 minutes with 4-folds cross-validation on training data, i.e., the evaluation of a fold is allowed to take 150 seconds. The evaluation of a configuration will be aborted and returned a zero if any folds got an error, e.g., infeasible configuration, timeout. Lastly, the average accuracy values on test data over 10 runs are reported.

VI. Results and Discussion

In this section, we report and discuss the results obtained from the above experimental setups. We have two goals in performing our experiments. Firstly, to compare the performance of the Bayesian optimization with the help of our proposed sampling approach against that without our contributions in terms of AutoML optimization for class-imbalance problems, with a search space of two operators. Secondly, we compare those against the state-of-the-art AutoML frameworks with a search space of six operators.

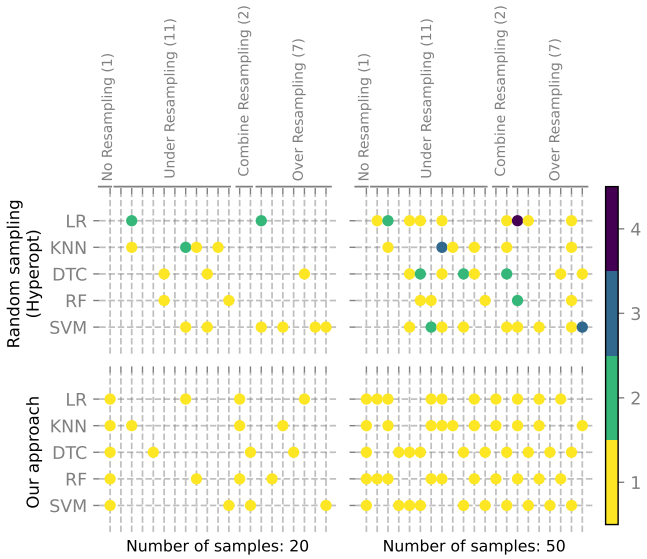

Fig. 6. Illustration on the distribution of samples obtained via initial sampling methods on the level of individual methods. The left part shows the case with 20 samples, while the case with 50 samples is shown on the right.

A. First experiment results

The results of the first experiment are presented in Table II to illustrate the performance between BO with and without the help of our proposed approach in two different initial sample sizes, i.e., 20 (left, not shaded) and 50 (right, grey shaded). In both scenarios, the highest performance for the corresponding dataset is highlighted in bold. The method performs significantly worse than the best according to the Wilcoxon sign-rank test with $\alpha = 0.05$ is underlined. A value labeled with * indicates the highest result obtained for the corresponding dataset. Two extra rows at the end display additional summaries. The first extra row shows the number of times each scenario got the highest value over 44 datasets. The last extra row indicates the number of times each approach was significantly better than the other in group. Looking at the table, we can observe that:

- In the scenario of 20 initial samples, Hyperopt achieves the highest result on $28/44$ cases, and our approach - on $20/44$ cases. However, our approach significantly wins on 2 tested cases, i.e., "ecoli3" and "yeast-2_vs_8" and is not significantly worse than Hyperopt in any tested cases.
- In the second scenario, our approach achieves the highest value on $31/44$ cases and Hyperopt- on $16/44$ cases. Similarly, our approach is not significantly worse than Hyperopt in any tested cases but significantly better on 4 examined datasets, i.e., "glass0", "yeast1", "ecoli4", "yeast-1-2-8-9_vs_7".

To investigate the sampling behaviour of both approaches in those initial sample sizes, we provide two plots: Fig. 5 shows the distributions of samples, and Fig. 6 displays those distributions on the level of individual algorithms. In both plots, the case with 20 samples on the left and 50 on the right of the plot. Looking at that figures, we can observe that our approach sampled over all combinations of grouping over

TABLE II
AVERAGE GEOMETRIC MEAN (ROUNDED TO 4 DECIMALS) BASED ON TWO DIFFERENT INITIAL SAMPLING SETTINGS, I.E., HYPEROPT APPROACH AND OUR APPROACH (OUR), OVER 10 REPETITIONS FOR THE 44 EXAMINED DATASETS, ORDERED BY INCREASING IMBALANCE RATIO (#IR) VALUE.

Dataset	#IR	20 initial samples		50 initial samples	
		Hyperopt	Our	Hyperopt	Our
glass1	1.82	0.7935	**0.7944**	*__0.7970__	0.7944
ecoli-0_vs_1	1.86	**0.9864**	**0.9864**	0.9864	*__0.9868__
wisconsin	1.86	0.9814	**0.9817**	0.9818	*__0.9819__
pima	1.87	*__0.7712__	0.7696	0.7703	**0.7707**
iris0	2	*__1__	*__1__	*__1__	*__1__
glass0	2.06	**0.8777**	0.8748	0.8740	*__0.8853__
yeast1	2.46	0.7319	**0.7332**	0.7321	*__0.7345__
haberman	2.78	*__0.7049__	0.7012	0.6991	**0.7040**
vehicle2	2.88	0.9908	*__0.9927__	0.9912	**0.9918**
vehicle1	2.9	**0.8690**	0.8684	0.8713	*__0.8735__
vehicle3	2.99	0.8463	**0.8486**	0.8416	*__0.8506__
glass-0-1-2-3_vs_4-5-6	3.2	*__0.9567__	0.9539	0.9534	**0.9553**
vehicle0	3.25	*__0.9876__	0.9867	**0.9867**	0.9867
ecoli1	3.36	0.9038	*__0.9053__	**0.9050**	0.9043
new-thyroid1	5.14	**0.9980**	0.9972	*__0.9983__	0.9966
new-thyroid2	5.14	*__0.9972__	0.9964	0.9952	**0.9966**
ecoli2	5.46	**0.9363**	0.9353	*__0.9365__	0.9360
segment0	6.02	*__0.9993__	0.9992	0.9992	**0.9992**
glass6	6.38	0.9488	**0.9514**	*__0.9518__	0.9511
yeast3	8.1	**0.9423**	0.9421	0.9427	*__0.9441__
ecoli3	8.6	0.9038	**0.9059**	0.9064	*__0.9072__
page-blocks0	8.79	*__0.9475__	0.9472	**0.9464**	0.9457
yeast-2_vs_4	9.08	**0.9549**	0.9542	*__0.9554__	0.9531
yeast-0-5-6-7-9_vs_4	9.35	**0.8245**	0.8177	*__0.8261__	0.8193
vowel0	9.98	0.9567	*__0.9593__	0.9525	**0.9561**
glass-0-1-6_vs_2	10.29	0.8404	**0.8421**	0.8334	*__0.8460__
glass2	11.59	*__0.8504__	0.8461	0.8462	**0.8471**
shuttle-c0-vs-c4	13.87	*__1__	*__1__	*__1__	*__1__
yeast-1_vs_7	14.3	0.7991	**0.8013**	*__0.8033__	0.8010
glass4	15.46	*__0.9390__	0.9230	0.9299	**0.9324**
ecoli4	15.8	**0.9712**	0.9694	0.9632	*__0.9737__
page-blocks-1-3_vs_4	15.86	**0.9931**	0.9874	0.9917	*__0.9944__
abalone9-18	16.4	*__0.8899__	0.8829	0.8856	**0.8859**
glass-0-1-6_vs_5	19.44	0.9494	*__0.9571__	0.9564	**0.9565**
shuttle-c2-vs-c4	20.5	*__1__	*__1__	*__1__	*__1__
yeast-1-4-5-8_vs_7	22.1	0.6989	**0.7024**	*__0.7052__	0.7045
glass5	22.78	**0.9589**	0.9558	0.9591	*__0.9595__
yeast-2_vs_8	23.1	0.8136	*__0.8348__	0.8136	**0.8150**
yeast4	28.1	0.8764	**0.8788**	0.8782	*__0.8788__
yeast-1-2-8-9_vs_7	30.57	**0.7500**	0.7489	0.7397	*__0.7538__
yeast5	32.73	*__0.9802__	0.9798	*__0.9802__	0.9800
ecoli-0-1-3-7_vs_2-6	39.14	*__0.9265__	0.9076	**0.9113**	0.8982
yeast6	41.4	**0.8953**	0.8918	0.8939	*__0.8955__
abalone19	129.44	0.7958	**0.7974**	0.7992	*__0.7998__
Highest performance		15	8	12	19
Significantly better performance		0	2	0	4

two operators in both sample sizes. In contrast, the sampling strategy used in Hyperopt samples has much less coverage in terms of those combinations. This is because we consider the choice of algorithms in operators is different with categorical parameters, while Hyperopt does not. The plots clearly explain the reason BO performs better with the help of our approach.

B. Second experiment results

The experimental results of the second experiment are presented in Table III. This table reports the average accuracy over 10 repetitions to illustrate the performance differences between the two implemented approaches in our AutoML framework[10], i.e., TPE with (RobustAutoML) and without (Auto-Hyperopt)

[10]For readability, RobustAutoML stands for TPE with our sampling approach, and Auto-Hyperopt stands for the original version of TPE implemented by Hyperopt without our improvement

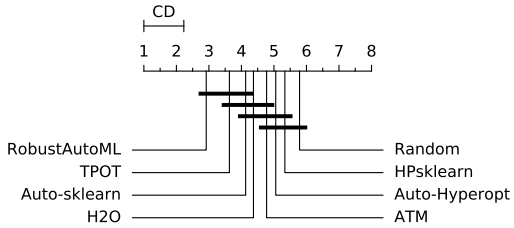

Fig. 7. Comparison of all approaches against each other with the Nemenyi test with 5% significance level.

our sampling approach, to compare them against other well-known AutoML frameworks, i.e., Auto-sklearn-SMAC (Auto-sklearn) and Auto-sklearn-Random search (Random), HP-sklearn, TPOT, ATM, H20. Values in bold indicate the highest value in the corresponding dataset. Underline indicates significantly different results from the best method according to a Wilcoxon signed-rank test with $p < 0.05$. Two extra rows at the end show additional summaries. The first extra row shows the number of times each approach achieved the highest performance over 73 examined datasets. The last row presents the number of cases that those methods significantly win other compared methods. The results allow the following insights:

- Comparing the results of approaches using the search space of Auto-sklearn includes our two approaches, Auto-sklearn and Random. Firstly, it should not be surprising that all Bayesian optimization approaches perform better than Random search in most tested cases. This has been proved in other works [4], [21]. Secondly, Auto-sklearn won on more tested cases than Auto-Hyperopt with the same search space. A possible explanation for this might be that Hyperopt lacks support for K-fold cross-validation yet, while SMAC, BO variant used in Auto-sklearn, uses racing algorithms to skip performing on unnecessary folds. Consequently, within the same budget of time, Auto-Hyperopt evaluated much less number of configurations than Auto-sklearn. Lastly, the experimental results clearly indicate that the performance of TPE with the help of our sampling approach significantly improved.
- From the results of three approaches using TPE, we can observe that: Firstly, comparing the two approaches not use our sampling, i.e., HPsklearn vs. Auto-Hyperopt, we can conclude that the search space of Auto-sklearn does not improve the final performance of TPE. Secondly, results clearly demonstrate that significant improvement was achieved with the help of our sampling approach. More precisely, in our 23 times significantly outperforms others, our approach significantly won Auto-Hyperopt 16 cases, won HPsklearn -20 cases. Furthermore, in all 3 cases where Auto-Hyperopt achieves the highest results, e.g., tasks 24, 3543, and 14967, both our approach and Auto-Hyperopt get maximum accuracy in those cases. On the other hand, HPsklearn got the highest results in 3 cases, e.g., tasks 24, 146607, 189355, but never significantly better than our approach in any of those.

TABLE III

AVERAGE ACCURACY (ROUNDED TO 5 DECIMALS) OVER 10 REPETITIONS FOR THE 73 OPENML DATASETS, ORDERED BY #TASK ID. THE THIRD AND FOURTH COLUMNS SHOWS OUR EXPERIMENTAL RESULTS, I.E., TPE WITH AND WITHOUT OUR SAMPLING APPROACH. THE REMAINING COLUMNS CONTAIN RESULTS OBTAINED BY OTHER AUTOML FRAMEWORKS ACCORDING TO [18]

OpenML IDs		Our AutoML framework		Existing AutoML frameworks [18]					
#TaskID	Dataset Name (#ID)	TPE *with our sampling* (RobustAutoML)	TPE *without our sampling* (Auto-Hyperopt)	Auto-sklearn SMAC	Auto-sklearn Random	HPsklearn	TPOT	ATM	H2O
3	kr-vs-kp (3)	**0.99656**	0.99510	0.98986	0.99062	0.99051	0.99431	0.99326	0.99426
12	mfeat-factors (12)	**0.98417**	0.98117	0.97767	0.97633	0.94758	0.97333	0.98178	0.97433
15	breast-w (15)	0.97952	0.97048	0.96875	0.95873	0.96000	0.96571	**0.98474**	0.96286
23	cmc (23)	0.57285	0.55158	0.54638	0.53262	0.53047	0.55882	**0.58100**	0.53733
24	mushroom (24)	**1**	**1**	**1**	0.99993	**1**	**1**	**1**	0.99848
29	credit-approval (29)	0.88744	0.86522	0.87289	0.85507	0.85956	0.86377	**0.89133**	0.86184
31	credit-g (31)	**0.76600**	0.72733	0.73433	0.72400	0.70121	0.74400	0.76578	0.74867
41	sick (42)	**0.95171**	0.92878	0.91954	0.91911	0.92585	0.92732	0.94504	0.93122
53	soybean (54)	**0.86457**	0.83858	0.82008	0.81969	0.75787	0.81811	0.81522	0.82717
2079	vehicle (188)	**0.69502**	0.66018	0.63886	0.62670	0.64072	0.65566	0.64190	0.65570
3021	eucalyptus (38)	**0.99152**	0.98737	0.98288	0.98550	0.97438	0.98746	-	0.98419
3543	irish (451)	**1**	**1**	0.99019	0.99081	0.99404	0.99091	**1**	0.97967
3560	analcatdata_dmf (469)	0.23042	0.21125	0.20365	0.20382	0.19139	0.20833	**0.27028**	0.19542
3561	profb (470)	0.63119	0.64752	0.65687	0.64563	0.63762	0.66832	**0.71221**	0.71089
3904	jm1 (1053)	**0.82404**	0.81393	0.81344	0.81126	0.80998	0.81810	0.82100	0.74819
3917	kc1 (1067)	**0.87393**	0.85972	0.85118	0.85340	0.84044	0.86019	0.86856	0.80869
3945	KDDCup09_appete (1111)	**0.98323**	0.98197	0.98244	0.98228	0.98189	0.98182	-	0.96555
3946	KDDCup09_churn (1112)	**0.92901**	0.92624	0.92725	0.92586	0.92599	0.92624	-	0.78802
3948	KDDCup09_upsell (1114)	0.94345	0.94116	**0.95094**	0.95030	0.95068	0.95085	-	0.93415
7592	airlines (1590)	0.86251	0.85769	0.86938	0.87013	0.86727	**0.87089**	0.85448	0.86656
7593	bank-marketing (1596)	0.70278	0.80902	**0.96395**	0.89143	0.95227	0.94542	0.66390	0.92908
9910	blood-transfusi (4134)	0.80107	0.78073	0.78890	0.77762	0.77798	**0.80249**	0.77087	0.80044
9952	cnae-9 (1489)	**0.91319**	0.90826	0.89716	0.89205	0.89273	0.90450	0.89963	0.89205
9955	first-order-the (1492)	**0.67167**	0.65146	0.65172	0.62795	0.54667	0.61146	0.61097	0.56435
9977	nomao (1486)	0.96525	0.95924	0.96903	0.96656	0.96891	0.97026	0.96055	**0.97146**
9981	phoneme (1468)	0.95093	0.94228	0.94167	0.93117	0.94012	0.94784	**0.96049**	0.95216
9985	one-hundred-pla (1475)	0.61029	0.59853	0.59695	0.58601	0.58293	0.61291	0.60272	**0.61656**
10101	adult (1464)	0.80667	0.76578	0.76667	0.77778	0.78044	0.78711	**0.81956**	0.73378
14952	covertype (4534)	0.97094	0.96623	0.96590	0.96244	0.96964	0.96913	0.96464	**0.97160**
14954	Bioresponse (6332)	**0.83642**	0.81111	0.79012	0.76173	0.76667	0.81009	0.81701	0.78333
14965	Amazon_employee (1461)	0.90307	0.90007	0.90447	0.90398	0.90451	**0.90705**	0.89957	0.90060
14967	PhishingWebsite (23380)	**1**	**1**	0.98265	0.99841	0.97131	**1**	-	**1**
14968	GesturePhaseSeg (6332)	**0.83580**	0.80432	0.77353	0.77058	0.75823	0.81173	0.79155	0.80000
14969	MiceProtein (4538)	0.64001	0.61864	0.67733	0.65004	0.67272	0.67586	0.66217	**0.70165**
34538	cylinder-bands (4550)	**1**	0.99907	**1**	0.99907	0.99983	**1**	**1**	**1**
34539	cylinder-bands (4135)	0.94825	0.94557	0.94761	0.94444	0.94750	0.94891	0.94606	**0.95114**
125920	cjs (23381)	0.63133	0.56200	0.56667	0.55556	0.56844	0.56867	**0.66978**	0.58400
146195	dresses-sales (40668)	0.77358	0.77321	0.82109	0.79628	0.82886	0.84123	0.77698	**0.86500**
146212	higgs (40685)	0.99965	0.99945	0.99978	0.99968	0.99253	0.99974	0.99955	**0.99987**
146606	numerai28.6 (23512)	0.70605	0.69761	**0.72296**	0.71930	0.70743	0.72031	0.67135	0.71281
146607	SpeedDating (40536)	0.86611	0.85871	0.86291	0.86225	**0.86661**	0.86392	0.86128	0.84968
146800	connect-4 (40966)	0.99969	0.99321	0.99043	0.99506	0.96380	0.99506	**1**	0.99551
146817	dna (40982)	**0.80497**	0.78216	0.78268	0.76364	0.75955	0.79091	0.76415	0.78062
146818	shuttle (40981)	0.88647	0.85845	0.87053	0.85556	0.86913	0.86184	**0.89050**	0.87633
146819	churn (40994)	0.95802	0.93951	0.94074	0.92407	0.92593	0.94547	**0.96975**	0.93642
146820	Devnagari-Scrip (40983)	0.97355	0.97906	0.98612	0.98581	0.95289	0.98540	**0.98657**	0.98574
146821	CIFAR_10 (40975)	**0.99441**	0.98748	0.97264	0.97958	0.98786	0.99422	0.96763	0.99191
146822	MiceProtein (40984)	**0.94473**	0.93189	0.93088	0.93333	0.90664	0.94055	0.92564	0.94185
146824	car (40979)	**0.98433**	0.98117	0.97783	0.97367	0.98121	0.96883	0.97750	0.97600
146825	Internet-Advert (40996)	0.84300	0.83891	**0.87844**	0.84450	0.85060	0.78089	0.82114	0.87341
167119	mfeat-pixel (41027)	0.84647	0.83956	0.86775	0.85378	0.88691	0.88735	0.87540	**0.90047**
167120	Australian (23517)	**0.52257**	0.52134	0.51926	0.51939	0.52033	0.52082	0.51941	0.50635
167121	steel-plates-fa (40923)	0.86652	0.74910	0.74009	0.02169	0.86438	-	**0.89470**	0.58220
167124	wilt (40927)	**0.39675**	0.37813	-	-	0.32093	0.29429	0.32001	0.36389
167125	segment (40978)	0.97713	0.97033	**0.97774**	0.97114	0.97358	0.97398	0.96900	-
167140	climate-model-s (40670)	0.96485	0.95397	0.95962	0.95889	0.96109	0.95931	0.95282	**0.96904**
167141	Fashion-MNIST (40701)	**0.96273**	0.95367	0.95620	0.95313	0.94533	0.96000	0.95007	0.95370
168329	jungle_chess_2p (41169)	0.31690	0.29294	0.30692	0.29566	0.28741	**0.33576**	0.32108	-
168330	APSFailure (41168)	0.68479	0.66670	**0.71814**	0.69273	0.68494	0.69642	0.63788	0.71786
168331	christine (41166)	0.60445	0.59520	0.66933	0.63762	0.65451	0.65075	**0.67940**	0.67841
168332	jasmine (41165)	0.42507	0.38497	**0.44843**	0.39922	0.34203	-	0.35252	-
168335	sylvine (41150)	0.92248	0.91035	0.94334	0.92891	0.87477	0.93850	0.90234	**0.94604**
168337	albert (41159)	0.78205	0.72707	0.64227	-	0.74347	0.72548	0.66063	**0.81928**
168338	MiniBooNE (41161)	0.98303	0.98035	0.74757	0.75042	0.82518	**0.98495**	0.90729	0.95625
168868	guillermo (41138)	0.98985	0.98900	0.99287	0.99137	0.99360	0.99339	0.97097	**0.99369**
168908	riccardo (41142)	0.73659	0.72565	**0.74754**	0.73081	0.71630	0.72645	0.72169	0.72811
168909	dilbert (41163)	0.95040	0.94437	**0.98357**	0.94793	0.97243	0.96254	0.95391	0.96988
168910	fabert (41164)	0.67565	0.66177	0.70255	0.67395	0.69104	0.68336	0.67357	**0.71752**
168911	robert (41143)	**0.83259**	0.80748	0.82009	0.80603	0.80078	0.82366	0.79911	0.80906
168912	volkert (41146)	**0.95709**	0.94655	0.93921	0.94753	0.94675	0.95533	0.93476	0.92510
189354	dionis (1169)	0.64626	0.63957	0.66665	0.59845	0.65080	**0.66895**	0.63671	0.61266
189355	jannis (41167)	0.73916	0.68112	-	-	**0.77971**	-	0.38666	-
189356	helena (41147)	0.64810	0.64737	0.68314	0.66709	0.66694	0.66110	**0.80064**	0.64798
Number of cases achieved the highest values		28	3	11	-	3	9	18	15
Significant wins over other approaches		23	-	9	-	1	6	13	13

- Overall, our proposed approach shows the highest results in more cases than all compared approaches, $28/73$. Moreover, according to the results of the Wilcoxon signed-rank test, our approach significantly outperforms other compared approaches in $23/73$ test cases. In contrast, Auto-Hyperopt, without our improvement, does not significantly win in any datasets.

Comparing all approaches together, they are significantly different according to Friedman's test in average accuracy with $p = 6.35E - 11$. Thus, we perform a post-hoc multiple comparison test with the Nemenyi test ($\alpha = 0.05$), shown in Fig. 7, approaches that have a distance greater than CD [11] are considered significantly different. Looking at this figure, we conclude that the proposed approach is significantly better than both TPE-based approaches. Additionally, the proposed approach is significantly better than five other AutoML frameworks, e.g., H20, ATM, Auto-Hyperopt, HPSklearn, and Random Search.

VII. Conclusions and Future Work

In this study, we formulate AutoML as an optimization process of the machine learning pipeline. Then, built on this paradigm, we proposed: a new class for modeling the choice of algorithms and the concept of grouping algorithms; Second, a robust sampling approach for Bayesian optimization for AutoML optimization problems; Third, a BO approach for AutoML optimization, where our proposed sampling approached and new hyperparameter classes are implemented; Lastly, a robust AutoML framework, which taking advantage of the proposed BO approach above. The experimental results demonstrated the effectiveness of our approaches in two independent experiments over 117 datasets. The results clearly show significant improvement achieved by using our approach.

There are several interesting research directions for extending this work. First, we intend to apply the proposed sampling approach to other AutoML frameworks. Additionally, we plan to apply some pruning approaches such as Hyperband [7] and racing algorithm to reduce the time of evaluating configurations that are not promising by evaluating fewer folds.

Acknowledgment

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 766186 (ECOLE).

References

[1] C. Wang, T. Bäck, H. H. Hoos, M. Baratchi, S. Limmer, and M. Olhofer, "Automated machine learning for short-term electric load forecasting," in *Symposium Series on Computational Intelligence*, pp. 314–321, 2019.

[2] M. Kefalas, M. Baratchi, A. Apostolidis, D. van den Herik, and T. Bäck, "Automated machine learning for remaining useful life estimation of aircraft engines," in *International Conference on Prognostics and Health Management*, pp. 1–9, 2021.

[3] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *The International Conference on Neural Information Processing Systems*, (Cambridge, MA, USA), p. 2755–2763, MIT Press, 2015.

[11] Critical Difference, here CD=1.2288

[4] C. Thornton, F. Hutter, H. Hoos, and K. Leyton-Brown, "Auto-weka: Combined selection and hyperparameter optimization of classification algorithms," *KDD*, 08 2012.

[5] B. Komer, J. Bergstra, and C. Eliasmith, *Hyperopt-Sklearn*, pp. 97–111. 2019.

[6] Z. Wang, M. Zoghi, F. Hutter, D. Matheson, and N. De Freitas, "Bayesian optimization in high dimensions via random embeddings," in *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, p. 1778–1784, AAAI Press, 2013.

[7] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *Journal of Machine Learning Research*, vol. 18, pp. 6765–6816, Jan. 2017.

[8] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[9] M. D. McKay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, pp. 239–245, 1979.

[10] G. Muniraju, B. Kailkhura, J. J. Thiagarajan, P.-T. Bremer, C. Tepedelenlioglu, and A. Spanias, "Coverage-based designs improve sample mining and hyperparameter optimization," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 3, pp. 1241–1253, 2021.

[11] Y. Li, J. Jiang, J. Gao, Y. Shao, C. Zhang, and B. Cui, "Efficient automatic CASH via rising bandits," in *The Conference on Artificial Intelligence*, pp. 4763–4771, AAAI Press, 2020.

[12] H. J. Escalante, M. Montes, and L. E. Sucar, "Particle swarm model selection," *Journal of Machine Learning Research*, vol. 10, no. 15, pp. 405–440, 2009.

[13] M. Feurer, K. Eggensperger, S. Falkner, M. L., and F. Hutter, "Auto-sklearn2.0:the next generation," 2020.

[14] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown, "Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka," *J. of Mach. Lear. Re.*, pp. 1–5, 2017.

[15] J. Močkus, "On bayesian methods for seeking the extremum," in *Optimization Techniques IFIP Technical Conference Novosibirsk* (G. I. Marchuk, ed.), pp. 400–404, 1975.

[16] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[17] S. Falkner, A. Klein, and F. Hutter, "BOHB: Robust and efficient hyperparameter optimization at scale," in *The International Conference on Machine Learning*, vol. 80 of *Proceedings of Machine Learning Research*, pp. 1437–1446, PMLR, 10–15 Jul 2018.

[18] M.-A. Zöller and M. Huber, "Benchmark and survey of automated machine learning frameworks," *J. Artif. Intell. Res.*, vol. 70, pp. 409–472, 2021.

[19] E. Parzen, "On estimation of a probability density function and mode," *Annals of Mathematical Statistics*, vol. 33, pp. 1065–1076, 1962.

[20] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?," *J. Mach. Learn. Res.*, vol. 15, p. 3133–3181, Jan. 2014.

[21] D. A. Nguyen, J. Kong, H. Wang, S. Menzel, B. Sendhoff, A. V. Kononova, and T. Bäck, "Improved automated cash optimization with tree parzen estimators for class imbalance problems," *IEEE International Conference on Data Science and Advanced Analytics*, 2021.

[22] 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 E. Duchesnay, "Scikit-learn: Machine learning in Python," *J. of Mach. Lear. Res.*, vol. 12, pp. 2825–2830, 2011.

[23] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. Cox, "Hyperopt: A python library for model selection and hyperparameter optimization," *Computational Science & Discovery*, vol. 8, p. 014008, 07 2015.

[24] R. S. Olson and J. H. Moore, *TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning*, pp. 151–160. 2019.

[25] T. Swearingen, W. Drevo, B. Cyphers, A. Cuesta-Infante, A. Ross, and K. Veeramachaneni, "Atm: A distributed, collaborative, scalable system for automated machine learning," in *2017 IEEE International Conference on Big Data (Big Data)*, pp. 151–162, 2017.

[26] E. LeDell and S. Poirier, "H2O AutoML: Scalable automatic machine learning," *ICML Workshop on Automated Machine Learning*, 2020.

[27] H. Bal, D. Epema, C. de Laat, R. van Nieuwpoort, J. Romein, F. Seinstra, C. Snoek, and H. Wijshoff, "A medium-scale distributed system for computer science research: Infrastructure for the long term," *Computer*, vol. 49, pp. 54–63, may 2016.