Efficient tuning of automated machine learning pipelines

Nguyen, D.A.

CHAPTER 7

Efficient AutoML via Combinational Sampling

In the previous chapter, the CASH approach converted the ML pipeline optimization problem into a hyperparameter optimization (HPO) problem, where the choice of algorithms was modelled as an additional categorical hyperparameter. In this manner, algorithms and their local hyperparameters are referred to at the same level. Consequently, this approach renders the resulting initial sampling less robust. Unlike the CASH approach, in this study, we used a new hyperparameter class to model the choice of the algorithm under the operator. Additionally, 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 experimented with both experimental scenarios of AutoML with two operators and six operators over 117 benchmark datasets, as introduced in Section 4. The results of our experiments demonstrate that the performance of BO is significantly improved using our sampling approach.

The remainder of this chapter is organized as follows. First, the motivation and introduction are provided in Section 7.1. Next, our contributions are highlighted in Section 7.2, Section 7.3 lays out the experimental setup. The experimental results are discussed in Section 7.4. Finally, the chapter is concluded, and further work is outlined in Section 7.5.

7.1 Introduction

Recall that existing AutoML approaches (e.g., [39], [40]) can be considered as optimization processes for which the best ML pipeline is searched. Each pipeline includes an architecture and a set of hyperparameter settings.

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 [39],

Auto-Weka [40], and Hyperopt-sklearn (HPsklearn) [42]. BO is an efficient global optimization approach (in terms of the number of function evaluations) in which the trade-off between local exploitation and global exploration is well handled. Therefore, in this work, we focus on improving the BO by 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. As HPO was originally developed to find the best hyperparameter setting from a single algorithm, it naturally does not consider the choice of algorithm. The choice of algorithm is then modelled as an extra categorical hyperparameter. Consequently, this HPO-based approach in handling the choice of algorithm mismatches the nature of the AutoML optimization problem.

The search space in the AutoML approach is largely owing to the many possible algorithm choices for pipeline operators. However, including many algorithms in the search space naturally leads BO to slow convergence or to get stuck in a local optimum [30], [31], [35]. One reason is that the initial sampling step in AutoML is typically restricted to a small 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 that 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, for example, the discrepancy-based quasi-random (quasi-random) sampling [30], the Latin Hypercube (LHD) sampling [239], have been employed for the initial sampling in this optimization context. However, they have shown themselves to be insufficiently robust [31], [240], [241] because they have been used in conjunction with the traditional approach of solving an AutoML optimization problem, which, as explained above, consists of converting it to a HPO, thus rendering obscure the differences between the choice of an algorithm and the choice of the algorithm's parameters.

Additionally, to construct a robust surrogate model, BO requires good coverage of the search space [31], but as the number of algorithms increases, the number of samples required to cover the search space increases exponentially. Previous studies [46], [47] pointed out that some algorithms can be grouped based on their technical behaviors.

To assess this theory, in this chapter, we propose a new two-fold approach to improve BO used in AutoML optimization:

- Group the similar operator algorithms when allocating initial sampling budget, e.g., the grouping of linear classifiers vs. the grouping of rule-based

Table 7.1: 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: $\{Continuous, Ordinal, Nominal, Algorithm\}$

classifiers [46]. Table 7.1 summarises different hyperparameter classes with their semantics in our work.

- 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, with the aim of optimizing the performance of the search space of AutoML.

7.2 The Proposed Approaches for Automated Machine Learning

In this section, we first introduce our proposed combination-based sampling approach for increasing the efficiency and robustness of AutoML. Next, we introduce a new BO Python library for AutoML optimization and an AutoML framework that implements this paradigm.

7.2.1 Novel combination-based initial sampling for Bayesian optimization for AutoML optimization

The central idea of our approach is to provide optimized coverage of the algorithm-hyperparameter search space already during the initial sampling of BO in order to characterize the response surface more accurately.

To properly analyze this discussion, we need to utilize the notations that were introduced in Chapter 1 (Section 1.1.1). These notations are crucial for our ongoing analysis and were discussed in detail in their original context in Chapter 1 to ensure a better understanding. Given a search space denoted by $\mathcal{M}$ includes the sequence of z operators $\mathbb{O} = \mathcal{O}_1 \times \ldots \times \mathcal{O}_z$ and its corresponding hyperparameter spaces $\Lambda = \Lambda_{\mathcal{O}_1} \cup \ldots \cup \Lambda_{\mathcal{O}_z}$, as defined in Section 1.1.1 of Chapter 1.

A grouping of algorithms of operator $\mathcal{O}_i$ assumes that the set of all algorithms $\{\varnothing, \mathcal{A}_i^1, \ldots, \mathcal{A}_i^{n_i}\}$ available to be employed for operator[1] $\mathcal{O}_i$ can be partitioned into g_i non-empty and non-overlapping subsets, according to their inner workings[2]: $\{G_i^1, \ldots, G_i^{g_i}\}$, $g_i \leq n_i + 1$[3]. Such partitioning is called a *grouping of algorithms*.

[1] if $i < z$ or $\{\mathcal{A}_z^1, \ldots, \mathcal{A}_z^{n_z}\}$ if $i = z$.
[2] or any other user-defined logic.
[3] if $i < z$ and $g_z < n_z$ otherwise.

The operator can then be represented as $\mathcal{O}_i = \{G_i^1, \ldots, G_i^{g_i}\}$. According to our proposed combination-based initial sampling method (see Algorithm 9), the sequence of pipeline operators $\mathbb{O} = \mathcal{O}_1 \times \ldots \times \mathcal{O}_z$ should be sampled in BO from the domain space of sets $\{G_1^1, \ldots, G_1^{n_1}\} \times \ldots \times \{G_z^1, \ldots G_z^{n_z}\}$ and the total initial sampling budget should be split equally per group. The main idea behind such sampling budget reallocation is the potential exploitation of similarities between algorithms within the group: sampling fewer of the same algorithms frees up the budget to be distributed to other (different) algorithms, thus improving the coverage of algorithm-hyperparameter search space at an 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 exceeds the maximum number of possible combinations computed from the input operation steps $k = \prod_{i=1}^{z} |\mathcal{O}_i|$, then we use Algorithm 10 to randomly regroup algorithms in operators to ensure $k \leq K$. The proposed sampling algorithm, presented in Algorithm 9, consists of the three following steps:

1. Generate the list of combinations: List all k possible combinations of groups for all z operators; apply RandomRegrouping until k is small enough ($k \leq K$) (lines $2-4$).

2. Allocate budget to combinations: first allocate budget to all combinations based on the number of algorithms and hyperparameters behind (lines 5 - 10). 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 the size of the combinations into account to give larger combinations a higher chance of getting a larger budget.

3. Sampling configurations: for each combination s, an existing sampling approach (e.g., LHD, quasi-random, here we use quasi-random) is used to generate a trial sequence $s_j = (G_1, \ldots, G_z)$ (lines $12-16$); Lastly, the generated configurations must be verified by CheckForbidden[4].

Lastly, the generated configurations are shuffled to remove a potential impact of grouped configurations based on combinations. This is highly recommended

[4] An external function that verifies a combination of algorithms/a configuration with the forbidden rules defined by the user.

Algorithm 9: Combination-based sampling

Input: $\mathbb{O}$: sequence of operators, Λ: hyperparameter spaces, B_{init}: number of initial samples, K: maximum 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`

$k = \prod_{i=1}^{z} |\mathcal{O}_i|$ `// maximum number of possible combinations`

if $k > K$ **then**

  $(\mathbb{O}, k) =$ RandomRegrouping$(\mathbb{O}, K)$ `// Algorithm 10`

`// Create a list` s `of all possible combinations from` $\mathbb{O}$

$s = \{s_1, \ldots, s_k\} = \{G_1^1, \ldots, G_1^{g_1}\} \times \ldots \times \{G_z^1, \ldots, G_z^{g_z}\}$

`// 2-Allocate budgets to` k `combinations`

$l_c = \frac{B_{init}}{k}$ `// number of inital samples per combination`

$m = \frac{1}{k} \sum_{i=1}^{k} (|\Lambda_{s_i}| + |s_i|)$ `//` $|s_i|$ `is the number of all unique algorithms and` $|\Lambda_{s_i}|$ `is the number of hyperparameters`

$\Theta = \varnothing$ `// set of initial configurations`

foreach $j \in \{1, \ldots, k\}$ **do**

  $l_j = \lfloor l_c \times \frac{|\Lambda_{s(j)}| + |s_j|}{m} \rfloor$ `//` l_j `is the number of samples for the combination` s_j

  $l_j = \begin{cases} 1, & \text{if } l_j = 0. \\ l_j, & \text{otherwise.} \end{cases}$

if $B_{remain} = B_{init} - \sum_{j=1}^{k} l_j > 0$ **then**

  `// Randomly allocate` B_{remain} `to the top` $\frac{k}{\eta}$ `combinations based on the number of algorithms and hyperparameters`

`// 3-Sampling Configurations`

foreach $j \in \{1, \ldots, k\}$ **do**

  $\Theta_j = \varnothing$ `// feasible configurations in the` j^{th} `combination`

  while $|\Theta_j| \leq l_j$ **do**

    $\Theta_j = \Theta_j \cup$ Sampling$(s_j, \Lambda_j, l_j - |\Theta_j|)$

    `//` Sampling `is done via an existing approach, here we choose quasi-random sampling with minor adjustments`

    foreach $\lambda \in \Theta_j$ **do**

      if CheckForbidden(λ) **then**

        $\Theta_j = \Theta_j \setminus \lambda$

  $\Theta = \Theta \cup \Theta_j$

Algorithm 10: Random Regrouping

Input: $\mathbb{O} = \left(\{G_1^1, \ldots, G_1^{g_1}\} \times \ldots \times \{G_z^1 \ldots G_z^{g_z}\}\right)$: sequence of operators, K: number of combinations

Output: $\mathbb{O}_{new}$: new sequence of operators, k: new number of combinations

$k = K$ `// number of all possible combinations`

$S = \varnothing$ `// split solutions`

$C_1 = \{1, \ldots, g_1\}, \ldots, C_z = \{1, \ldots, g_z\}$ `// set of possible groupings of` $\mathcal{O}_{i \in \{1,\ldots,z\}}$

`// List out all split solutions`

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,` $k \leq K$

while $S = \varnothing$ **do**

  $S = \{h = (c_1, \ldots, c_z) \in H : \left(\prod_j c_j\right) = k\}$

  if $S = \varnothing$ **then**

    $k = k - 1$

$s_{chosen} \sim \mathcal{U}(S)$`// randomly choose one solution`

$\mathbb{O}_{new} = (\varnothing_1, \ldots, \varnothing_z)$, $i = 1$

foreach $c_i \in s_{chosen}$ **do**

  `//` c_i `is the number of groups to be created`

  $n_i = |\mathcal{O}_i|$ `// number of groups in the` i^{th} `operator`

  if $c_i = n_i$ **then**

    $\mathcal{O}_i = \{\{G_i^1\}, \ldots, \{G_i^{n_i}\}\}$ `// when` $c_i = n_i$

  else if $c_i = 1$ **then**

    $\mathcal{O}_i = \{G_i^1, \ldots, G_i^{n_i}\}$ `// merging all predefined groups`

  else

    $G = \varnothing$, $\mathcal{O}_0 = \mathcal{O}_i$

    while $n_i > 0$ **do**

      $G_0 = \varnothing$, $n_{size} = \lceil \frac{n_i}{c_i} \rceil$

      if $n_i > n_{size}$ **then**

        $G_0 =$ {Random pick n_{size} items in $\mathcal{O}_0$}

      else

        $G_0 = \{\mathcal{O}_0\}$

      $G = G \cup G_0$, $\mathcal{O}_0 = \mathcal{O}_0 \setminus G_0$, $n_i = |\mathcal{O}_0|$

    $\mathcal{O}_i = \{G\}$

  $\mathbb{O}_{new}^{(i)} = \mathcal{O}_i$, $i = i + 1$

return $\mathbb{O}_{new}, k$

since, in some cases, the computational optimization budget, i.e., the run time limit, can run out before finishing this initialization step.

The RandomRegrouping method used in Algorithm 9 is presented in Algorithm 10. For a sequence of operators that consists of multiple groupings 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 the number of groupings to be created for the operator $\mathcal{O}_i$.

3. Step 3 (lines $11-27$): For each operator $\mathcal{O}_i$, we randomly group algorithms into c_i groups.

7.2.2 A New Optimization Library for AutoML Optimization

To take advantage of the new sampling approach introduced in Section 7.2.1, we introduce a BO library for AutoML optimization, named BO4ML[5], where the new sampling approach is implemented. In this work, we use the Tree-structured Parzen Estimator (TPE) implemented in Hyperopt [153] for the surrogate model and Expected improvement (EI) [156] for the acquisition function.

7.3 Experimental Setup

This study examines the two experiments introduced in Chapter 4 (Section 4.2 and Section 4.3). In both scenarios, we compare the performance of Bayesian optimization (see Chapter 3. Section 3.1.3) with and without our proposed initial sampling approach.

The first experiment uses similar parameter settings as in Chapter 5; we select two different values of the initial sample size 20 and 50. We use a budget of 500 function evaluations. The 5-fold cross-validation approach and the averaged geometric mean values over 10 repetitions are reported. The selected classification algorithms are not grouped together. The resampling techniques are grouped into

[5] The library is published at `https://github.com/ECOLE-ITN/NguyenSSCI2021`.

four groups "Over-resampling", "Under-resampling", "Combine-resampling", and "No-sampling", as suggested in [47], [48].

In the second experiment, we used a budget of 50 samples for the initial sampling. All the 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 is aborted and returns zero if any folds have an error, for example, due to an infeasible configuration or timeout. Then, the average accuracy values for the test data over 10 runs are reported. Finally, the selected algorithms are grouped, according to the suggestions in [46], [151].

The implementation of the proposed methods is published in a git-repository[5] and PyPi-repository[6]. The experiment scripts for the reproducibility of the reported results are provided in a git-repository[7].

7.4 Results and Discussion

In this section, we report and discuss the results obtained from using the above experimental setups. Our experiments has two objectives. First, we compare the performance of Bayesian optimization with the help of our proposed sampling approach with that without our contributions in terms of AutoML optimization for class-imbalance problems, with a search space of two operators. Second, we compared them against state-of-the-art AutoML frameworks with a search space of six operators.

7.4.1 First experimental results

The results of the first experiment are presented in Table 7.2 to illustrate the performance of BO with and without the help of our proposed approach for two different initial sample sizes, that is, 20 (left, not shaded) and 50 (right, gray shaded). In both scenarios, the best performance for the corresponding dataset is highlighted in bold. A method that performs significantly worse than the best method according to the Wilcoxon signed-rank test with $\alpha = 0.05$ is underlined. A value labeled * indicates the best result obtained for the corresponding dataset. The two extra rows at the end display the additional summaries. The first extra

[6] `https://pypi.org/project/BO4ML`.

[7] `https://github.com/ECOLE-ITN/NguyenSSCI2021/tree/assets/SSCI-Experiments`.

Table 7.2: Average geometric mean (rounded to 4 decimals) based on two different initial sampling settings, i.e., the Hyperopt approach and our approach (BO4ML), over 10 repetitions for the 44 examined datasets, ordered by increasing imbalance ratio (#IR) value. The two extra rows display summaries for each scenario, i.e., 20 and 50 initial samples: (1) *Highest performance* shows the number of times the optimizer achieved the highest value. (2) *Significantly better performance* shows the number of times the optimizer was significantly better than the competitor.

Dataset	**#IR**	**20 initial samples**		**50 initial samples**	
		Hyperopt	**BO4ML**	**Hyperopt**	**BO4ML**
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

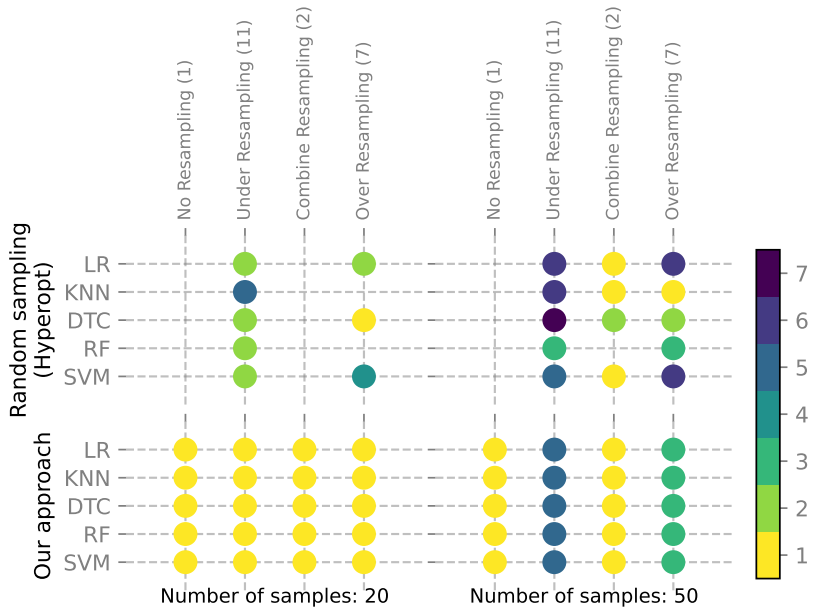

Figure 7.1: Illustration of the number of samples allocated to different combinations of methods in random sampling implemented in Hyperopt (top rows) vs. our proposed approach (bottom rows). The cases with 20 (left) and 50 (right) samples are shown.

row shows the number of times each scenario achieved the highest value over 44 datasets. The last extra row indicates the number of times each approach was significantly better than the others in the group. From the table, we can observe the following.

- In 20 initial samples scenario, Hyperopt achieved the best results 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 behavior of both approaches for the initial sample sizes, we provided two plots: Figure 7.1 shows the distributions of the samples and Figure 7.2 shows the distributions at the level of the individual algorithms. In both plots, the case with 20 samples is shown on the left and 50 on the right of the plot,

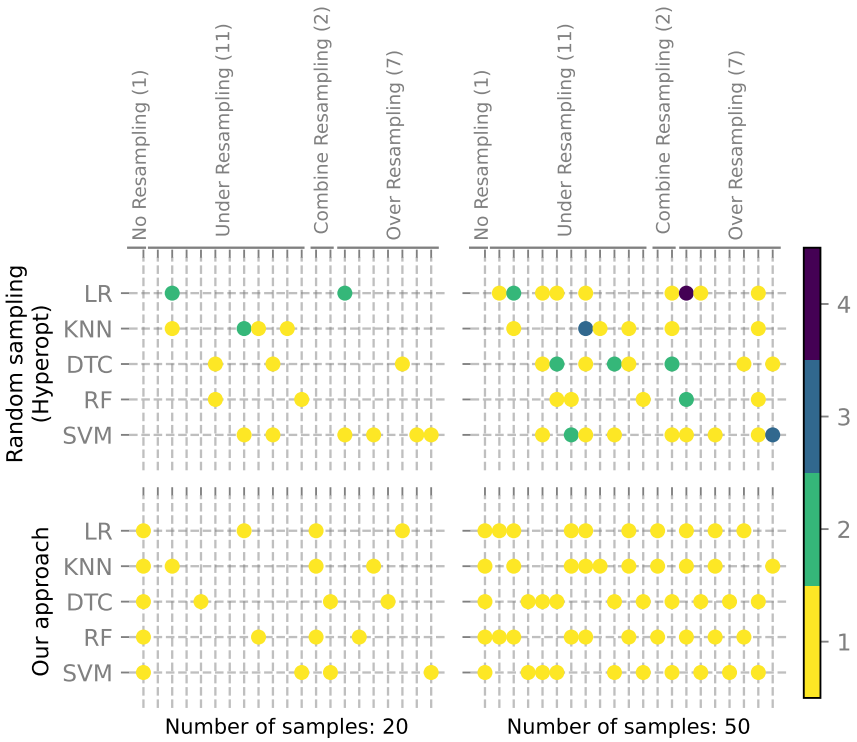

Figure 7.2: Illustration on the distribution of samples obtained via initial sampling methods on the level of individual methods, i.e., under resampling has 11 algorithms, combine resampling has 2 algorithms, over resampling has 7 algorithms and no resampling. The left part shows the case with 20 samples, while the case with 50 samples is shown on the right.

respectively. Looking at these figures, we observe that our approach samples all combinations of groupings over two operators for both sample sizes. By contrast, the sampling strategy used in the Hyperopt samples has less coverage in terms of these combinations. This is because we consider the choice of algorithms in operators to be different from categorical parameters, whereas Hyperopt does not. The plots clearly explain why BO performs better with the help of our approach.

7.4.2 Results of second experiment

The results of the second experiment are presented in Table 7.3. The third and fourth columns show our experimental results, i.e., TPE with and without our sampling approach. The remaining columns contain the results obtained using other AutoML frameworks according to [22]. This table reports the average accuracy over

Table 7.3: Average accuracy (rounded to 5 decimals) over 10 repetitions for the 73 OpenML datasets, ordered by #Task id.

OpenML IDs		Our Experiments		Existing AutoML frameworks [22]					
#TaskID	Dataset Name (#ID)	**TPE** *with our sampling* (**RobustAutoML**)	**TPE** *without our sampling* (**Auto-Hyperopt**)	**Auto-sklearn**		**HPsklearn**	**TPOT**	**ATM**	**H2O**
				SMAC	**Random**				
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

continued on the next page

Table 7.3: Average accuracy (rounded to 5 decimals) over 10 repetitions for the 73 OpenML datasets, ordered by #Task id. – continued from previous page

OpenML IDs		Our Experiments		Existing AutoML frameworks [22]					
#TaskID	Dataset Name (#ID)	TPE *with our sampling* (**RobustAutoML**)	TPE *without our sampling* (**Auto-Hyperopt**)	Auto-sklearn		HPsklearn	TPOT	ATM	H2O
				SMAC	Random				
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

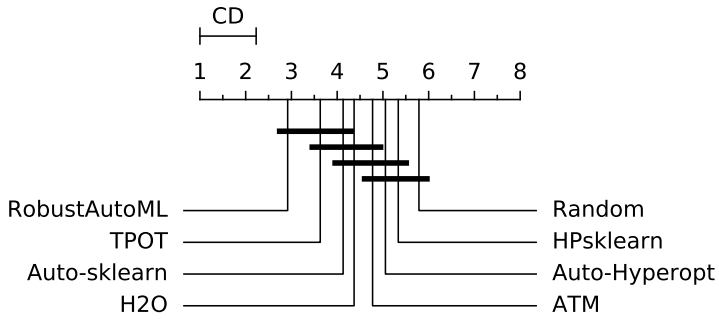

Figure 7.3: Comparison of all approaches against each other with the Nemenyi test with 5% significance level.

10 repetitions to illustrate the performance differences between the two implemented approaches in our AutoML framework[8], i.e., TPE with (RobustAutoML) and without (Auto-Hyperopt) 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), HPsklearn, TPOT, ATM, and H2O. Values in bold indicate the highest values in the corresponding dataset. Underline values indicate significantly different results from the best method according to a Wilcoxon signed-rank test with $p < 0.05$. The two extra rows at the end show the 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 in which these methods significantly outperformed the 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 Search. First, it is not surprising that all Bayesian optimization approaches perform better than random search in most tested cases. This has been demonstrated in other studies [40], [47]. Second, Auto-Sklearn won 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, the BO variant used in Auto-Sklearn, uses racing algorithms to skip performing on unnecessary folds. Consequently, within the

[8]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.

same budget of time, Auto-Hyperopt evaluated a much smaller 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 improves.

- From the results of three approaches using TPE, we can observe that: Firstly, comparing the two approaches that do 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, the results clearly demonstrate that significant improvement was achieved with the help of our sampling approach. Our approach outperforms others 23 times, significantly winning Auto-Hyperopt in 16 cases and HPsklearn in 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 performed significantly better than our approach in any of those.

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

When all approaches are compared, Friedman's test reveals a significant difference in average accuracy with $p = 6.35 \cdot 10^{-11}$. Thus, we performed a post-hoc multiple comparison test with the Nemenyi test ($\alpha = 0.05$), shown in Figure 7.3. Approaches that have a distance higher than CD[9] are considered significantly different. According to this figure, we conclude that RobustAutoML is better than both TPE-based approaches and better than five other AutoML frameworks, such as, H20, ATM, Auto-Hyperopt, HPSklearn, and Random Search.

7.5 Conclusions and Future Work

In this chapter, we formulated AutoML as an optimization process for the machine learning pipeline. Then, we built on this paradigm, we proposed a new class for modeling the choice of algorithms and the concept of grouping algorithms. Second,

[9]Critical Difference, here CD=1.2288.

a robust sampling approach for Bayesian optimization for AutoML optimization problems was introduced; Third, a BO approach for AutoML optimization was presented, where our proposed sampling approach and new hyperparameter classes were implemented. Lastly, a robust AutoML framework was presented which takes advantage of the proposed BO approach mentioned above.

The experimental results demonstrate 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 study. First, we intend to apply the proposed sampling approach to other AutoML frameworks. Additionally, we plan to apply some pruning approaches such as Hyperband [35] and racing algorithm to reduce the time for evaluating configurations that are not promising by evaluating fewer folds.