Efficient tuning of automated machine learning pipelines

Nguyen, D.A.

CHAPTER 8

An Efficient Contesting Procedure for AutoML Optimization

Classical AutoML-based Bayesian Optimization approaches often integrate all operator search spaces into a single search space. However, one drawback of this strategy is that it can be less robust when initialized randomly than optimizing each operator-algorithm combination individually. To overcome this issue, a novel contesting procedure, **D**ivide **A**nd **C**onquer **Opt**imization (DACOpt), is proposed in this chapter to make AutoML more robust. The DACOpt partitions the AutoML search space into a reasonable number of sub-spaces based on algorithm similarity and budget constraints. Furthermore, throughout the optimization process, DACOpt allocates resources to each sub-space to ensure that (1) all areas of the search space are covered and (2) more resources are assigned to the most promising sub-space. Two extensive sets of experiments on 117 benchmark datasets demonstrate that DACOpt is significantly better than its competitors. Furthermore, an experiment in surface defect classification in steel manufacturing indicated that the proposed contesting procedure significantly improved the performance of BO in real-world applications. The remainder of this chapter is organized as follows. The motivation and introduction are provided in Section 8.1. Section 8.2 presents the relevant background knowledge Divide and Conquer techniques and early-stop strategies. Our contributions are highlighted in Section 8.3, whereas Section 8.4 outlines the experimental setup. Experimental results are discussed in Section 8.5. Next, an investigation of the use of DACOpt in real-world applications is discussed in Section 8.6. Finally, the chapter is summed up and further work is outlined in Section 8.7.

8.1 Introduction

In this study, we evaluated BO-based approaches for solving the AutoML optimization problem. The AutoML optimization (AO) problem is typically considered as a single optimization problem in the BO-based method by merging the optimization space for all algorithms of all operators – this approach is typically refer to as *integrated approach* [14]. The Combined Algorithm Selection and Hyperparameter Optimization (CASH) approach [40] is a commonly used technique, where the AO problem is treated as a hyperparameter optimization (HPO) problem. However, HPO was initially developed to optimize hyperparameters of a single algorithm, where the considered search space is typically smaller, lower-dimensional, and less (even non)-structured than the AutoML search space. Hence, the HPO-based approach is not ideal for handling the AO problem. In order to alleviate the above limitation, we formulate the AO problem as a ML pipeline optimization problem, which is proposed by [15]. This can be seen as a generalization of the CASH approach, where the parameter classes for operator's algorithms, and hyperparameters in an algorithm were clearly identified.

As an alternative to the *integrated approach*, [241] proposed the so-called CASH-oriented Multi-Armed Bandits (MAB) approach to solve the model selection and hyperparameter optimization problem for the classification problem (i.e., selecting a classification algorithm and tuning the hyperparameters, simultaneously) by applying HPO to each classifier separately. However, this might not be applicable to AutoML scenarios, because the number of combinations of algorithms over operators can be up to thousands. Fortunately, [15], [46], [47], [74] have pointed out that operator algorithms can potentially be grouped and that different groups of algorithms perform better on different types of problems, for example, a group of linear classification algorithms performs best on linear classification tasks.

Hence, this study attempts to further improve BO performance for the AO problems by applying the Divide and Conquer (DAC) strategy: the AutoML search space is divided into multiple sub-spaces based on their similarity[1], and each sub-space is solved independently by a separate BO process (candidate). The budget is then allocated to each candidate using a novel competing mechanism, depending on its performance. Consequently, the most promising candidates have a larger tuning budget than the least promising candidates. Therefore, the worst candidates will be 'terminated' as soon as ample evidence against them has been

[1] see grouping approach proposed in [15]

gathered, saving computation time and resources for future assessments in those search areas.

Notably, as our approach handles BO[2] processes independently, it allows multiple optimization processes to be executed simultaneously without affecting the performance. In other words, our technique achieves the same numerical results in both parallel and sequential settings, with the exception of different execution times.

Our contributions: We summarize our main contributions, which are the following:

- We propose a novel contesting procedure, namely DACOpt, to solve the AutoML optimization problem efficiently, which is complementary to the existing BO approaches.
- DACOpt efficiently allocates resources to each sub-space to ensure that (1) all areas of the search space are covered and (2) more resources are assigned to the most promising sub-space. In addition, we provide a proof that our approach fixes the existing gap between serial and parallel BO execution (see Section 8.3.3).
- Two independent empirical studies on a range of AutoML optimization problems with 2 and 6 operators on a total of 117 benchmark datasets demonstrate the superiority of the proposed approaches.
- An empirical experiment on a real-world application of surface defect classification in steel manufacturing indicated that our proposed approach significantly improved BO's performance.

8.2 Background

In this chapter, we review the relevant techniques to the proposed contesting procedure (Section 8.2.1), and early-stop strategies (Section 8.2.2). Other related research to Bayesian Optimization and AutoML optimization can be found in Section 3.1.3.

[2]BO (a.k.a., Sequential model-based optimization) was originally intended as a sequential approach [25], [36].

8.2.1 Contesting procedure for AutoML optimization

AutoML optimization typically is a high-dimensional mixed-variables (continuous, discrete, nominal) optimization problem. In order to handle such a challenge by a BO approach, three facts are considered: (1) BO performs better for low-dimensional problems [242], (2) AO problems have low effective dimensionality [30], [31], and (3) the complexity of AO problem not only comes from its dimensionality, but also from the number of possible combinations of algorithms within the ML pipeline [15].

Divide and Conquer (DAC) [243] is a well-known strategy for handling large problems via decomposing the target problem into c small-scale and low-dimensional sub-problems. Consider an AO problem $p^* = \text{argmax}_{p \in \mathcal{M}} f(p)$. For applying DAC, the approach has to first decompose the AutoML search space into c sub-spaces, and then solve each sub-space by an optimizer. Assuming that we can split the AutoML search space $\mathcal{M}$ into c smaller spaces $\{\mathcal{M}_1, \ldots, \mathcal{M}_c\}$, the DAC approach can be formulated as:

$$p^* = (\underset{p \in \mathcal{M}_1}{\text{argmax}}\, f(p), \ldots, \underset{p \in \mathcal{M}_c}{\text{argmax}}\, f(p)) = (p_1^*, \ldots, p_c^*) \tag{8.1}$$

where p_i^* is the global optimum of sub-space $\mathcal{M}_i$ and p^* is the global optimum of the original search space $\mathcal{M}$.

The existing DAC studies typically treat elements of the input search space on the same level and decomposed by complementing [244]. That is, variables corresponding to sub-space $\mathcal{M}_i$ can change freely while the remaining $|\mathcal{M} \setminus \mathcal{M}_i|$ dimensions are set to some fixed values. However, such approaches cannot be used for the AutoML search space where dimensions are hierarchical and, thus, dependent. As a result, two challenges are faced when using DAC to solve AutoML problems: (1) how to divide the AutoML space $\mathcal{M}$ onto a set of c sub-spaces efficiently; (2) how to optimize resources during the 'conquer' phase, since some sub-spaces' performance might be significantly worse than others. To answer the above questions, we propose (1) a splitting approach based on the combination of groups of operator algorithms [15], (2) adopting efficient early-stop strategies based on the theoretical guarantees (see our discussion in Section 8.2.2) for optimizing resources for the 'conquer' phase.

In addition, since the number of algorithms (and therefore, the number of sets of their parameters) is smaller in the DAC-formulation of the AutoML problem in Equation 8.1 compared to the original formulation in Equation 1.4, the proposed

contesting procedure also concurs the assumption [30], [31] that the AO problem has low effective dimensionality.

8.2.2 Early-stop strategies

As we discussed in Section 1.1, the k-fold cross-validation is usually applied to the AutoML optimization problem to prevent the over-fitting problem as described in Equation 1.4. For readability, let p denote $p_{(\mathcal{A}_{1,\lambda},\ldots,\mathcal{A}_{z,\lambda})}$. The Equation 1.4 is then formulated as:

$$p^* = \underset{p \in \mathcal{M}}{\operatorname{argmax}} \frac{1}{k} \sum_{j=1}^{k} f\left(p, \mathcal{D}_t^j, \mathcal{D}_v^j\right) \tag{8.2}$$

where $f\left(p, \mathcal{D}_t^j, \mathcal{D}_v^j\right)$ is performance of the pipeline setting p when trained and evaluated on the j^{th} cross-validation data fold $\mathcal{D}_t^j$ and $\mathcal{D}_v^j$, correspondingly. As a consequence of using cross-validation, every function evaluation becomes k times more expensive. An early stop strategy, e.g., [32]–[36], [173] allows limiting this issue, since it avoids wasting time and resources on evaluating worse settings over all k folds.

An important concept is to stop investigating a setting as soon as sufficient information indicates that it is ineffective. A setting will only be examined in a few folds in this manner; an iterative elimination function will analyze its performance on the evaluated folds to compare it to other evaluated settings and determine how many folds should be utilized for the considered setting.

The elimination function in racing procedure approaches (see Section 3.2.1) is based on a statistical test procedure, i.e., Friedman test [176], whereas bandit-based approaches (see Section 3.2.2) compare the setting performance directly to the best-known setting. In a number of case studies, both strategies performed well [14], [33], [35], where the task of proposing new settings was commonly handled by a random search (see Section 3.1.2). Unfortunately, the inconsistencies in how settings are assessed may provide additional noise for BO, making it less reliable in suggesting subsequent settings. This means that such approaches should not be used directly and should be adopted only at the level of search sub-spaces, via the termination of unpromising sub-spaces (the detailed discussion on the termination functions is given in Section 8.3.1).

8.3 Proposed approach

We now discuss our proposed contesting procedure for AutoML optimization problems based on the **D**ivide **A**nd **C**onquer strategy, which we call DACOpt.

8.3.1 Algorithm description

We reformulate the AutoML optimization problem in Equations 1.4, 8.1 and 8.2 into the following:

$$p^* = \underset{p \in \mathcal{M}}{\operatorname{argmax}}(p_1^*, \ldots, p_c^*) \tag{8.3}$$

$$\text{s.t.} \quad p_i^* = \underset{p_i \in \mathcal{M}_i}{\operatorname{argmax}} \frac{1}{k} \sum_{j=1}^{k} f\left(p_i, \mathcal{D}_t^j, \mathcal{D}_v^j\right) \tag{8.4}$$

where $\mathcal{M}_i = \mathbb{O}_i \cup \Lambda_i$ denotes the i^{th} sub-space. $\mathbb{O}_i = \mathcal{O}_1^{(i)} \times \ldots \times \mathcal{O}_z^{(i)}$ denotes the possible sequence of operators in $\mathcal{M}_i$, herein $\forall i \in \{1, \ldots, c\}$, $\mathcal{O}_{l \in \{1,\ldots,z\}}^{(i)} = \{\mathcal{A}_l^1, \ldots, \mathcal{A}_l^{n_l}\}$ denotes a set of algorithms of the l^{th} operator $\forall |\mathcal{O}_l^{(i)}| \leq |\mathcal{O}_l|$, and a set of the corresponding hyperparameters of $\mathcal{O}_l^{(i)}$: $\Lambda_{l \in \{1,\ldots,z\}}^{(i)} = \Lambda_l^1 \cup \ldots \cup \Lambda_l^{n_l}$ and $f\left(p_i, \mathcal{D}_t^{(j)}, \mathcal{D}_v^{(j)}\right)$ denotes performance of the setting, similar to Equation 8.2.

The overall proposed structure of the contesting procedure is summarized in Figure 8.1. The process begins with a *Splitter* function to be applied on the input AutoML search space $\mathcal{M}$ to produce c possible sub-spaces. Here, we extend our work from the previous chapter with improvements (a detailed discussion on this function is given in Section 8.3.2). Then, c BO processes are initialized (in the following discussion, the BO processes shall be called candidates). The whole contest is controlled by the *Controller* function, which allocates budgets to each candidate per contest round based on the feedback from the *Elimination* function that decides which candidates will survive into the next round based on their performances so far. As mentioned in Section 8.2.2, we adopt two possible settings for the early-stop functionality. Therefore, two versions of DACOpt are provided, which differ mainly w.r.t. the elimination criteria: (1) based on highest performance (Section 8.3.1.1) and (2) based on a statistical procedure (Section 8.3.1.2).

8.3.1.1 Elimination criteria based on the highest performances

In BO (Section 3.1.3), the acquisition function maximizes the *best-found value* $\Delta_{(t)}^*$ up to time step t. Due to the fact that the goal of AutoML optimization

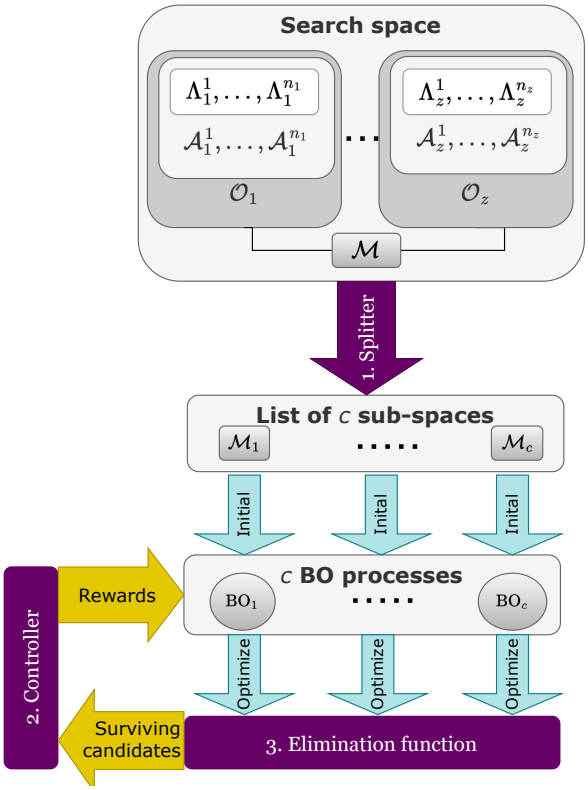


Figure 8.1: The workflow of the contesting procedure.

is to find the setting that achieves the highest performance on the target ML problem, we consider the highest performance as a suitable comparison criterion. Furthermore, the way of computing the budget, step size, and the number of rounds follow the Successive Halving (Chapter 3. Section 3.2.2.1) and Hyperband (Chapter 3. Section 3.2.2.2) approaches with minor adjustments: input parameters of our procedure include the maximum number of sub-spaces to be split c, total optimizing budget B, and the ratio of candidates discarded in each round[3] η. The number of rounds in the contest is then calculated as: $R_{max} = \lfloor \log_\eta(c) \rfloor$. Each round has the same budget $B_r = \frac{B}{R_{max}}$. That is, each of m surviving candidates at the round can have a budget of $b = \lfloor \frac{B_r}{m} \rfloor$. At the end of the round, the *Elimination* function keeps $\lceil \frac{m}{\eta} \rceil$ candidates for the following round. Therefore, the surviving candidate has η times the budget from the previous round.

Our approach is elaborated in Algorithm 11 which requires the maximum

[3] $\eta = 3$, can be changed by user.

number K of sub-spaces to be split and the ratio of candidates discarded in each round, η, as input parameters. This approach consists of the following steps:

- Initialize: Split the original search space into c ($c \leq K$) sub-spaces (line 1). Next, initialize c corresponding Bayesian optimization candidates (lines 2-3). Next, the number of contest rounds is calculated, $R_{max} = \lfloor \log_{\eta}(c) \rfloor$ (line 4).

- Parameter for each round: Based on the number of surviving candidates from the previous round, the number of candidates c_r for the current round r is computed in line 8 for the first round and in line 11 for subsequent rounds. The elimination function discards candidates labeled as badly performing and returns a set of c_r good candidates (line 11). Herein, we simply select the top c_r candidates based on their best-found values. A reasonable budget for each candidate is computed based on the remaining budget, remaining rounds, and the number of surviving candidates (lines 13-14). All the above steps (lines 10-14) are repeated every round, except the first round. In the *first round*, all candidates survive and are given a budget of $b = b_{init}$ (lines 7-9).

- Finally, using value b obtained in the previous step, all surviving candidates continue their optimization processes (lines 15-19).

8.3.1.2 Elimination criteria based on a statistical procedure

As mentioned in Section 8.2.2, our second option was the approach of racing procedures to determine well and badly-performing candidates. This approach also requires a maximum number of sub-spaces K and a level of significance α. Since the effectiveness of BO is mostly seen in the later phases of optimization when it learns to produce better settings, we only consider the best-found value of the initial sampling step for further statistical tests. Unlike the first elimination criteria method, this method does not compute the number of rounds or budget for each round since it completely depends on the statistical results; instead, we use a step size[4] γ to limit budget per round. At the end of the round, a Friedman test [176] is performed to verify whether there is a significant difference between the pair of candidates. If it is the case, a Holm post-hoc test [245][5] is applied to compare the highest-ranked candidate to others. Any candidate that fails the test

[4] $\gamma = 1$, can be changed by user.
[5] Following the recommendations by [246], [247].

Algorithm 11: DACopt based on the highest performance

Input: $\mathcal{M}$: Search space, K: number of sub-spaces to be split, f: objective function, B: maximal number of evaluations, b_{init}: number of evaluations for the initial step in each of c BO processes, η: ratio controlling the proportion of candidates discarded in each round

Output: p^*: the best pipeline setting, Δ^*: the best value

1 $(\{\mathcal{M}_1, \ldots, \mathcal{M}_c\}, c) \leftarrow$ `Splitter`$(\mathcal{M}, K)$ `// divide the input search space into` c `sub-spaces`

Initialization : INITIALIZATION PHASE

2 **for** $\mathcal{M}_i \in \{\mathcal{M}_1, \ldots, \mathcal{M}_c\}$ **do**

3 $\quad (\text{BO}_i, \mathcal{H}_i) \leftarrow$ `BayesianOptimizer`$(\mathcal{M}_i, f, b_{init})$ `// Initialize` BO_i `and its historical data` $\mathcal{H}_i = \{(p_n, \Delta_n)_{n=1}^{\texttt{evaluated}}\}$. (p, Δ) `represent configuration and performance.`

Initialization : CONTESTING PHASE

4 $R_{max} \leftarrow \lfloor \log_\eta(c) \rfloor$ `//` R_{max}`: number of rounds`

5 $r \leftarrow 0$ `//` r`: round number`

6 **while** $r \leq R_{max}$ **do**

7 $\quad$ **if** $r = 0$ **then**

8 $\quad\quad c_r \leftarrow c$; $(\text{BO}_1, \ldots, \text{BO}_{c_r}) \leftarrow$ `RandomPermute`$(\text{BO}_1, \ldots, \text{BO}_c)$ `// Note: at the first round` $c_r = c$`, but the order of candidates are shuffled.`

9 $\quad\quad b \leftarrow b_{\text{init}}$ `//all candidates have an equal budget` $b_{\texttt{init}}$

10 $\quad$ **else**

11 $\quad\quad c_r \leftarrow \lceil \frac{c_{\text{previous}}}{\eta} \rceil$ `// number candidates for` $r^{\texttt{th}}$ `round`

12 $\quad\quad (\text{BO}_1, \ldots, \text{BO}_{c_r}) \leftarrow$ `Eliminate`$((\text{BO}, \mathcal{H})_{i \in \{1,\ldots,c\}}, c_r)$ `// Select good` c_r `candidates, ordered by performance/rank`

13 $\quad\quad B_r \leftarrow \lfloor \frac{B}{R_{max} - r} \rfloor$ `//` B_r`:total budget for` $r^{\texttt{th}}$ `round`

14 $\quad\quad b \leftarrow \lfloor \frac{B_r}{c_r} \rfloor$ `//` B_r`:budget per candidate`

15 $\quad$ **for** $\text{BO}_i \in \{\text{BO}_1, \ldots, \text{BO}_{c_r}\}$ **do**

16 $\quad\quad \text{BO}_i$ .ADDBUDGET(b) `//Add budget` b `to` BO_i

17 $\quad\quad (\text{BO}_i, \mathcal{H}_i) \leftarrow \text{BO}_i$ .OPTIMIZE() `//Continues` BO_i `process`

18 $\quad\quad B \leftarrow B - b$`// Update the remaining budgets`

19 $\quad c_{\text{previous}} \leftarrow c_r$; $r \leftarrow r + 1$

20 **Return** $p^*, \Delta^* = \text{argmax}_{p,\Delta} \{\mathcal{H}\}_{i \in \{1,\ldots,c\}}$;

is removed from the list of surviving candidates. This loop is repeated until the best candidate is found. This process, summarized in Algorithm 12, consists of the following steps:

- Initialize: Using the same split function as Algorithm 11, to produce c ($c \leq K$) sub-spaces (line 1). All candidates are initialized with the minimum

required budget b_{init} (line 2-3).

- The main operates in the contesting phase: maintain a set of surviving candidates[6]. A statistical test is performed at each round to determine if there are any pairs of candidates that are significantly different (lines 7-10). If the null hypothesis is false, we first perform a rank test, e.g., the Wilcoxon signed rank test, to detect the highest-ranked candidate (line 12). Next, a post-hoc test is applied to the pair of every candidate and the highest-ranked candidate (line 13). Any candidate that fails the test is removed from the set of surviving candidates (line 14). Next, a budget γ is added to each candidate in the survived set (line 18) and continues the tuning process with the added budget (line 19). This procedure is repeated until the total budget runs out.

Lastly, both options naturally support parallel implementation. We require the number of maximum available threads $\tau, (\tau \geq 1)$, as an extra input parameter for the parallel mode. The parallel mode will be discontinued when the best sub-space is found. In both algorithms, parallel mode is applied to execute the BO processes.

[6]Note for the contesting phase: Since the effectiveness of BO is mainly determined during the initial sampling step as it learns to produce better settings. Therefore, we consider only the best-found value from the initial sampling step for further statistical tests, and we perform statistical tests only when the sample size exceeds 2.

Algorithm 12: DACOpt based on the statistical test

Input: $\mathcal{M}$: Search space, K: number of sub-search spaces, f: objective function, B: maximal number of evaluations, b_{init}: minimum evaluations per sub-search space, $\gamma = 1$: step size, $\alpha = 0.05$: level of significance

Output: p^*: the best pipeline setting, Δ^*: the best value

$\{\mathcal{M}_1, \ldots, \mathcal{M}_c\}, c \leftarrow$ SPLITTER$(\mathcal{M}, K)$ // divide the input search space into c sub-spaces, $c \leq K$

// BEGINNING OF INITIAL PHASE

for $\mathcal{M}_i \in \{\mathcal{M}_1, \ldots, \mathcal{M}_c\}$ **do**

  $(\text{BO}_i, \mathcal{H}_i) \leftarrow$ BayesianOptimizer$(\mathcal{M}_i, f, b_{init})$ // Initialize BO_i and its historical data $\mathcal{H}_i = \{(p_n, \Delta_n)_{n=1}^{\text{evaluated}}\}$. (p, Δ) represent configuration and performance.

// BEGINNING OF CONTESTING PHASE

$\{(\text{BO}_i, \mathcal{H}_i)\}_{i=1}^{survive} \leftarrow \{(\text{BO}_i, \mathcal{H}_i)\}_{i=1}^{c}$ // All candidates survive

$c_r \leftarrow c$ // c_r number of surviving candidates

while $B \geq 0$ **do**

  if $c_r < 3$ **then**

    STAC $\leftarrow$ WILCOXONTEST() // Init WILCOXONTEST if $c_r < 3$

  else

    STAC $\leftarrow$ FRIEDMANTEST() // Init FRIEDMANTEST if $c_r \geq 3$

  // Performs the chosen statistical test with α to detect if there is at least one pair of candidates that are significantly different

  if $(\neg STAC(\{\mathcal{H}_i\}_{i=1}^{survive}, \alpha))$ & $c_r > 1$ **then**

    $\mathcal{H}_{i^*} = \text{argmax}\, \text{RANKING}(\{\mathcal{H}_i\}_{i=1}^{survive})$ // detect the highest ranked $\mathcal{H}_{i^*}$ among the surviving candidates based on a ranking test, e.g., Wilcoxon signed rank test

    $\{(\text{BO}_i, \mathcal{H}_i)\}_{i=1}^{survive} \leftarrow$ HOLM_POST_HOC_TEST// detects candidates not significantly worse than $\mathcal{H}_{i^*}$

    $c_r \leftarrow$ number of surviving candidates

  else if $c_r = 1$ **then**

    $\gamma \leftarrow B$// If $c_r = 1$, allocate the remaining budget

  for $\text{BO}_i \in \{\text{BO}_i\}_{i=1}^{survive}$ **do**

    BO_i .ADDBUDGET(γ) // Add budget γ to the selected BO_i

    $(\text{BO}_i, \mathcal{H}_i) \leftarrow \text{BO}_i$ .OPTIMIZE() // Continues BO_i process.

    $B \leftarrow B - \gamma$ // Update the remaining budgets

Return $p^*, \Delta^* = \text{argmax}_{p,\Delta} \{\mathcal{H}\}_{i \in \{1,\ldots,c\}}$

8.3.2 The Splitting approach

In this section, we briefly describe of the splitting function (line 1 of the Algorithm 11 and Algorithm 12). The AutoML search space is complex owing to the number of operators and their choice of algorithms. In practice, the search space can lead to thousands of algorithm combinations over operators. Because the tuning budget is relatively small compared to a large number of possible pipelines over operators, we propose grouping them based on their similarities with the assumption that a good choice for one algorithm in the group can also serve as a good choice for other algorithms in the group. Consequently, the sampler can maximize the coverage of the search space by sampling at the group level instead of at the algorithm level.

The concept of grouping is similar to that done in Chapter 7. However, it mainly focused on initial sampling, where the budget was typically much smaller than the number of combinations of algorithm's groups. As a result, the group's level is limited to only 1, i.e., the group's item is a specific choice for the algorithm. In this study, we consider a scenario in which a set of algorithms under a group might be slightly different. For example, while the RandomOverSampler and SMOTE algorithms are both oversampling techniques (see the bottom plot in Figure 8.2), they differ significantly: RandomOverSampler randomly generates more data for minority classes, whereas SMOTE is based on interpolation. To account for the possible hierarchical groupings of the algorithms, we extended Algorithm 10 to allow any group at any level to contain child groups. Therefore, the required groups are produced by downing (or upping) the levels to minimize randomness. Consequently, the resulting subspaces are purer, that is, the difference between items in a group is minimized, representing their actual relationship.

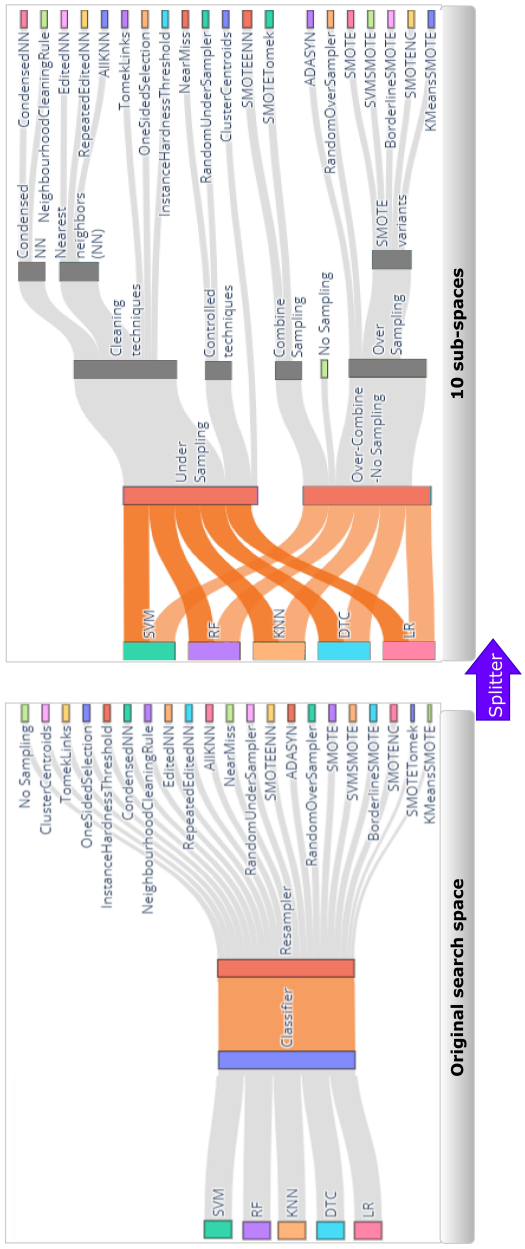


Figure 8.2: Illustration of the Splitting approach on a search space of two operators, i.e., Classifier and Resampler, used in our first experiment. The connection in orange indicates a search space/ sub-space.

8.3.3 Fixing the gap between serial and parallel BO

Bayesian optimization, called otherwise the Sequential model-based optimization (SMBO), is naturally sequential. However, most modern optimizer-based BO approaches include a parallelized version in addition to the original BO method. AutoML-based BO is typically parallelized by either assessing in parallel (1) cross-validation folds or (2) multiple settings, e.g., [26], [38], [248]. While the first approach focuses on parallelizing evaluations inside the objective function, it does not affect BO, however, it is efficient when k is less than the available resources. The second approach might lead to inefficient solutions proposed by BO, in terms of the number of function evaluations. Since the objective function is expensive, we have to choose a configuration that might perform best. In the following, we discuss how parallelized BO can lead to poorer results compared to serial approaches.

Let us consider a noiseless function $f : \mathcal{M} \subset \mathbb{R}^d \rightarrow \mathbb{R}$ and its real-valued surrogate model $\hat{f} = \{\mathcal{P}(p_i, \Delta_i)_{i=1}^t\}$ for time step t. At a new step $t+1$, a sampling approach (randomly) generates a set of solutions $\{\hat{p}_1, \ldots, \hat{p}_n\}$. Those later will be estimated by the surrogate model $\hat{f}$ and used to propose **one** setting $p_{t+1} \in \{\hat{p}_1, \ldots, \hat{p}_n\}$ by maximizing the acquisition function in Eq. 3.5. The set of m next settings from the time step t of the sequential approach is $\{p_{t+1}^{\curlywedge} = \text{argmax}_{p \in \mathcal{M}} \mathbb{E}[I(p_t)], \ldots, p_{t+m}^{\curlywedge} = \text{argmax}_{p \in \mathcal{M}} \mathbb{E}[I(p_{t+m})]\}$. In contrast, the parallel approach proposes a set of solutions $\{p_{t+1}^{\shortparallel_1}, \ldots, p_{t+1}^{\shortparallel_m}\} \in \text{argmax}_{p \in \mathcal{M}} \mathbb{E}[I(p_t)]$. Let $\overline{p} = |f(p) - \hat{f}(p)|$ denote the difference between the performance of the setting p on the true objective function f and its surrogate $\hat{f}$. Clearly, the quality of BO in suggesting new solution(s) is highly dependent on $\hat{f}$ and the statistical property of $\hat{f}$ (i.e., uncertainty) at time t, which significantly increases as more historical data is collected. Thus, $\sum_{j=1}^{m} \overline{p_{t+j}^{\curlywedge}} \geq \sum_{j=1}^{m} \overline{p_{t+j}^{\shortparallel_j}}$. Hence, the quality of m additional time steps in the sequential method may be more robust than those in the parallel technique. Thus, there is a discrepancy between the current serial and parallel BOs.

For the reasons above, we use sequential BO to solve each search sub-space. Fortunately, BO processes in our proposed procedure are independent (see Figure 8.1). Therefore, we introduce a partly-parallel approach instead of fully parallel. Instead of proposing a set of future solutions from a single search area like the fully parallel technique does, in order to ensure the best performance of BO at every iteration, DACOpt proposes a set of next setting solutions as sequential approach from multiple independent search areas: $p_{t+1}^{\curlywedge_i} = \text{argmax}_{p \in \mathcal{M}_i} \mathbb{E}[I(p_t)], \forall i \in \{1, \ldots, c\}$.

Table 8.1: Proposed DACOpt approaches compared in this study

Name	Contesting procedure		BO variant	
	Highest	Statistical	BO4ML	Hyperopt
DAC-HB	✓		✓	
DAC-HH	✓			✓
DAC-SB		✓	✓	
DAC-SH		✓		✓

Thus, $p_{t+1}^{r_i}$ in either serial or parallel situations are exactly the same. For parallel computing, a parallel pool of m available workers will be repeated $\lceil \frac{c}{m} \rceil$ times to finish c processes. The last iteration of that parallel pool is partly parallel if (c mod m) > 0 and fully parallel otherwise. As a result, our approach holds the same effectiveness in both cases.

The key benefits of our DACOpt approach are as follows:

- Based on the performance of the related BO process, the budget adaptively redistributes to the search area[7]. As a result, the budget is distributed effectively.
- As a partly-parallel BO variant, the proposed approach has parallel efficiency without harming BO performance.
- BO performance and robustness can be increased since each BO process optimizes a relatively small low-dimensional search space independently.

8.4 Experimental Setup

In order to evaluate the robustness and general applicability of our proposed approach, we compare it to other state-of-the-art AutoML optimization approaches. We reproduce the experimental setup with a total of 117 benchmark datasets on two scenarios with optimization of 2 operators (Section 4.2) and 6 operators (Section 4.3). In both scenarios, we compare the performance of BO-based variants with the TPE surrogate model BO4ML (Chapter 7) and Hyperopt [153], with the two proposed contesting procedures against those without such procedure (see Table 8.1). Our local parameter settings are summarized in Table 8.2.

Both experiments used similar parameter settings as Chapter 7. All approaches use an initial sample size of 50 function evaluations.

[7] In this thesis, we use the term *search area* to refer to an area (subset) of the search space.

Table 8.2: Parameter settings

	1st experiment	**2nd experiment**
Total budgets (B_{max})	500 (func. eval.)	1 (hour)
DACOpt parameters		
- Number of candidates (K)	10	10
- Initial sample size per candidate (B_{init})	5	5
- DAC variants used		
DAC-HB	✓	✓
DAC-HH	✓	✓
DAC-SB	✓	✓
DAC-SH	✓	✓
HyperOpt parameters		
- Initial sample size per	50	50
BO4ML parameters		
- Number of candidates (K)	10	10
- Initial sample size per candidate (B_{init})	5	5

The first experiment used a budget of 500 function evaluations. The 5-fold cross-validation approach and the averaged geometric mean values over 10 repetitions were reported. The selected classification algorithms were not grouped together. The resampling techniques were grouped by a hierarchical graph as shown on the right-hand side of Figure 8.2, following the suggestion in [48].

In the second experiment, all experiments are based on 10 runs with different random seeds, and a time limit of 1 hour. The performance evaluation 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 to zero if any of the folds have an error, for example, infeasible configuration or timeout. The average accuracy values for the test data over 10 runs were reported. Finally, the selected algorithms used a hierarchical tree of similarity of algorithms[8].

Reproducibility and Open Science: The implementation of the proposed methods is published in a git-repository[9] and PyPi-repository[10]. The experiment scripts

[8] based on the hierarchy used in [151] and discussed in [46].

[9] `https://github.com/ECOLE-ITN/NguyenIEEEAccess2022`

[10] `https://pypi.org/project/DACOpt`

for the reproducibility of the reported results are provided in a git-repository[11].

8.5 Results and Discussion

In this section, we report and discuss the results obtained from the two experimental setups introduced above. Generally speaking, we target three goals: (1) to compare the performance of our two contesting procedures in terms of *number of function evaluations* and *wall-time limit*; (2) to compare the performance of BO with and without the proposed contesting procedures; (3) to compare those against the current state-of-the-art AutoML frameworks.

The first experiment's results are provided in Table 8.3, and the second in Table 8.4. Both tables highlight the highest performance for the corresponding dataset/task in **bold**. According to the Wilcoxon signed-rank test, the method that performs significantly worse than the best with $\alpha = 0.05$ is underlined. Two extra rows at the end of the corresponding table display additional summaries. The first extra row shows the number of times each scenario got the highest value over tested datasets/tasks. The last extra row indicates the number of times each approach was significantly better than the other in a group.

For each tested case, the method that achieved the highest performance was counted as winning, provided that its performance was significantly better than that of all other methods. The method that performed significantly worse than the best was counted as a loss. They are considered equal if there is no significant difference in performance between the two methods. The method is counted as performing well if it either achieves the best performance or is not significantly worse than the best-found method in the corresponding case.

[11] `https://github.com/ECOLE-ITN/NguyenIEEEAccess2022/tree/main/Experiments`

Table 8.3: Average geometric mean (rounded to 4 decimals) based on six approaches, i.e., DAC-HB, DAC-HH, DAC-SB, DAC-SH, BO4ML and Hyperopt, over 10 repetitions for the 44 examined datasets, ordered by increasing imbalance ratio (#IR) value.

Dataset	#IR	DAC-HB	DAC-HH	DAC-SB	DAC-SH	BO4ML	Hyperopt
glass1	1.82	0.8015	0.8004	**0.804**	0.7945	0.7922	0.7968
ecoli-0_vs_1	1.86	0.9864	0.9864	0.9864	0.9864	**0.9868**	0.9864
wisconsin	1.86	0.9813	0.9816	0.9813	0.9814	0.9814	**0.9819**
pima	1.87	0.769	**0.7725**	0.768	0.7719	0.7694	0.7705
iris0	2.0	**1**	**1**	**1**	**1**	**1**	**1**
glass0	2.06	0.876	**0.8777**	0.8736	0.8757	0.8736	0.8745
yeast1	2.46	0.7332	0.7333	0.7322	0.7321	**0.7335**	0.7325
haberman	2.78	**0.7057**	0.701	0.7023	0.6974	0.6968	0.7012
vehicle2	2.88	0.9903	0.991	0.9902	0.9898	**0.9912**	0.991
vehicle1	2.9	**0.8709**	0.8707	0.8512	0.8445	0.862	0.8701
vehicle3	2.99	0.84	0.8476	0.8166	0.8202	**0.848**	0.8461
glass-0-1-2-3_vs_4-5-6	3.2	0.9571	0.9568	0.9545	**0.9572**	0.9562	0.9514
vehicle0	3.25	0.9865	**0.9868**	0.9837	0.9837	0.9864	0.9855
ecoli1	3.36	0.9034	0.9047	0.9036	0.9029	0.9031	**0.9047**
new-thyroid1	5.14	0.9975	**0.9986**	0.9966	0.9972	0.9972	0.9978
new-thyroid2	5.14	0.9975	**0.9978**	0.9972	0.9972	0.9975	0.9978
ecoli2	5.46	0.9375	**0.9375**	0.9365	0.9362	0.9361	0.9358
segment0	6.02	**0.9993**	0.9993	0.9992	0.9992	0.9991	0.9993
glass6	6.38	0.952	**0.9547**	0.9516	0.9503	0.9489	0.9524
yeast3	8.1	0.9436	**0.9437**	0.9422	0.942	0.9428	0.9425
ecoli3	8.6	0.9075	0.9079	0.907	0.9072	0.9054	**0.9091**
page-blocks0	8.79	0.9471	0.9467	0.9468	0.9463	**0.948**	0.9472
yeast-2_vs_4	9.08	0.9535	0.952	0.9533	0.951	**0.9538**	0.9533
yeast-0-5-6-7-9_vs_4	9.35	0.8145	**0.8258**	0.8146	0.8169	0.8238	0.8195
vowel0	9.98	**0.9628**	0.9569	0.9598	0.9554	0.9564	0.9555
glass-0-1-6_vs_2	10.29	**0.8515**	0.8424	0.845	0.8359	0.8436	0.8342
glass2	11.59	0.8593	0.8546	**0.8601**	0.8534	0.8578	0.856
shuttle-c0-vs-c4	13.87	**1**	**1**	**1**	**1**	**1**	**1**
yeast-1_vs_7	14.3	0.8017	**0.8043**	0.8003	0.8026	0.8017	0.8001
glass4	15.46	0.9355	**0.9372**	0.9291	0.935	0.9192	0.9334
ecoli4	15.8	**0.9743**	0.9709	0.9701	0.9637	0.9661	0.9698
page-blocks-1-3_vs_4	15.86	**0.9944**	0.9889	0.9929	0.9882	0.9901	0.99
abalone9-18	16.4	**0.8951**	0.8864	0.8834	0.8846	0.8873	0.8838
glass-0-1-6_vs_5	19.44	0.9655	0.9535	0.9588	0.9597	**0.9681**	0.9644
shuttle-c2-vs-c4	20.5	**1**	**1**	**1**	**1**	**1**	**1**
yeast-1-4-5-8_vs_7	22.1	**0.7166**	0.7037	0.7155	0.7011	0.6979	0.7011
glass5	22.78	**0.9716**	0.9699	0.9659	0.96	0.9625	0.96
yeast-2_vs_8	23.1	0.8242	0.8259	0.8135	0.8117	**0.828**	0.8254
yeast4	28.1	0.8794	**0.8812**	0.8694	0.8726	0.8708	0.8773
yeast-1-2-8-9_vs_7	30.57	0.7515	**0.7546**	0.7416	0.7459	0.7391	0.7429
yeast5	32.73	0.9801	**0.9806**	0.9795	0.9796	0.9801	0.9798
ecoli-0-1-3-7_vs_2-6	39.14	0.866	0.8799	0.8892	0.9035	0.9034	**0.9057**
yeast6	41.4	0.9007	**0.9018**	0.8994	0.9003	0.897	0.9004
abalone19	129.44	**0.8059**	0.8031	0.8022	0.8003	0.8049	0.8021
Cases achieved the highest values		14	18	5	4	11	7
Significant wins over other approaches		10	8	1	1	4	0

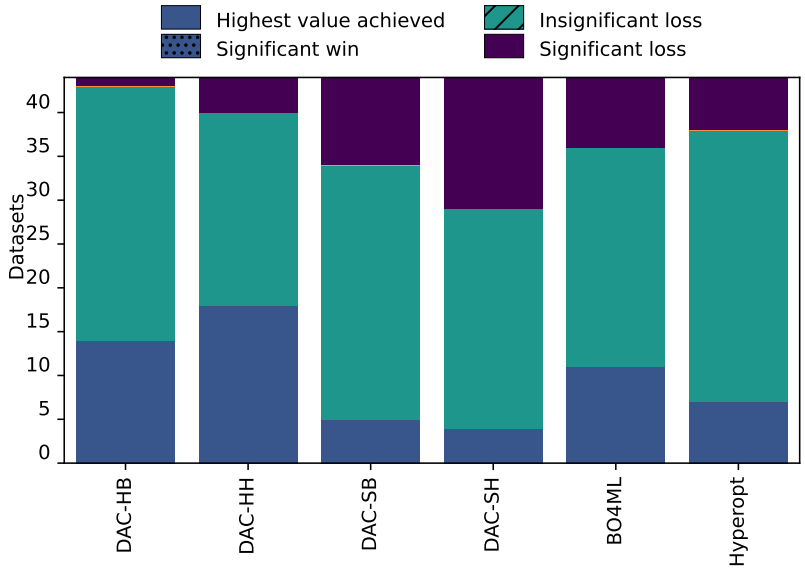


Figure 8.3: Overview of the results over 10 repetitions for the 44 binary imbalanced benchmark datasets.

8.5.1 First experiment results

The results of the first experiment are presented in Table 8.3 to illustrate the performance between 2 BO variants based on TPE surrogate model with and without proposed contesting procedures using 2 elimination criteria – highest performance and statistical test procedure, i.e., DAC-HB, DAC-SB, DAC-HH, DAC-SH compared to BO4ML and Hyperopt. Additionally, these results are summarized in Figure 8.3. This figure is based on the average geometric mean over a 5-fold cross-validation over 44 imbalanced binary benchmark datasets. We make the following observations:

- Comparing two methods that use the highest performance as the elimination criteria (highest value-based contest), DAC-HH achieved the highest performance more times than DAC-HB (18 vs. 14). However, DAC-HB significantly won on more tested cases than DAC-HH. Additionally, DAC-HB loses on fewer cases than DAC-HB (1 vs. 5).
- Compared to the contesting procedures that used statistical tests as the elimination criteria (statistical-based contest), two methods, i.e., DAC-SH and DAC-SB, achieved similar performance.

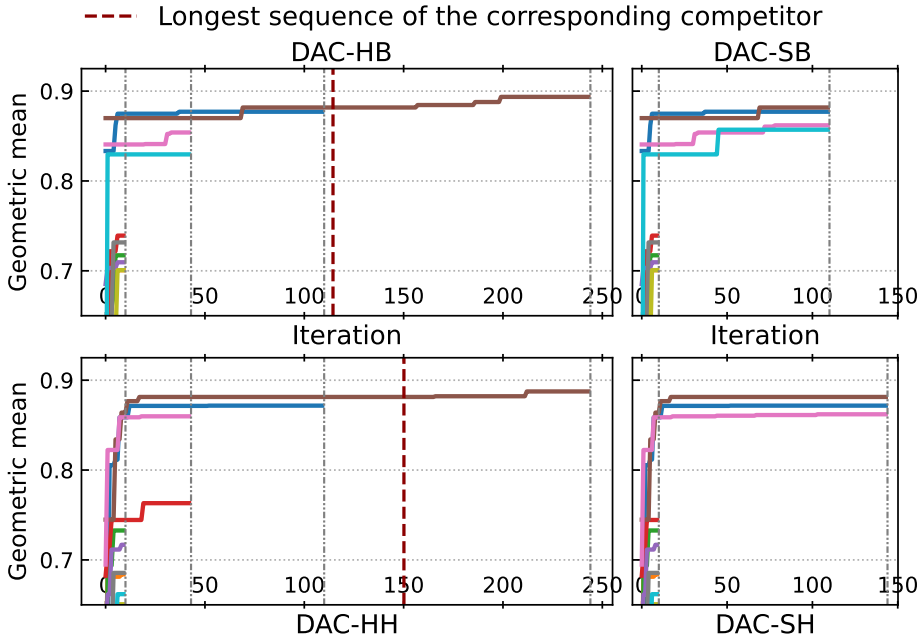


Figure 8.4: Illustration of the contesting process on dataset *abalone9-18*. This figure shows the optimization convergence plots of DAC-HB (top-left), DAC-SB (top-right), DAC-HH (bottom-left) and DAC-SH (bottom-right) approaches. All approaches are initialized with the same random seed. The colors represent BO processes on sub-spaces.

- Overall, DAC-HB performs well in most of the tested cases. More precisely, over 44 tested dataset, DAC-HB loses only on dataset *pima*, where DAC-HH is the winner.

Lastly, another point worth mentioning is that we expected the statistical-based approaches, i.e., DAC-SB and DAC-SH, to perform better than the highest-based approaches, i.e., DAC-HB and DAC-HH. However, the experimental results contradict our assumptions. To investigate their optimizing behavior, we plot a single run of these approaches on the dataset *abalone9-18* in Figure 8.4. The two plots on the left show the convergence behavior of the highest-based contests and the statistical-based contests are shown on the right. All approaches used a total budget of 500 function evaluations, and the search space was split into 10 sub-spaces. The colors represent BO processes on sub-spaces. The dashed-grey vertical lines indicate a contest round cutoff point, i.e., the end of the round where the elimination function is called. The extra dashed-red vertical line on the two left

plots shows the most extended sequence of the corresponding underlying optimizer. The statistical-based approach maintained more candidates throughout the contest than the highest-based approach. Consequently, the best candidate was found late with less budget than the best candidate in the competitor approach.

Table 8.4: Average accuracy (rounded to 5 decimals) over 10 repetitions for the 73 OpenML datasets, ordered by #Task ID. The first fourth columns after "Dataset" shows our experimental results, i.e., 4 variants of the contesting procedure. The remaining columns contain results obtained by other AutoML frameworks according to results from Chapter 7 and [22].

OpenML IDs	DACOpt contesting procedure				Chapter 7		[22]					
#TaskID	DAC-HB	DAC-HH	DAC-SB	DAC-SH	BO4ML	Hyperopt	Auto sklearn	Random search	HP sklearn	TPOT	ATM	H2O
3	**0.99802**	0.99666	0.99802	0.99666	0.99656	0.9951	0.98986	0.99062	0.99051	0.99431	0.99326	0.99426
12	**0.98617**	0.98383	0.98617	0.98383	0.98417	0.98117	0.97767	0.97633	0.94758	0.97333	0.98178	0.97433
15	0.98095	0.97524	0.98095	0.97524	0.97952	0.97048	0.96875	0.95873	0.96	0.96571	**0.98474**	0.96286
23	0.57376	0.57805	0.57376	0.57805	0.57285	0.55158	0.54638	0.53262	0.53047	0.55882	**0.581**	0.53733
24	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0.99993**	**1**	**1**	**1**	0.99848
29	0.88647	0.87778	0.88647	0.87778	0.88744	0.86522	0.87289	0.85507	0.85956	0.86377	**0.89133**	0.86184
31	**0.767**	0.74533	0.767	0.74533	0.766	0.72733	0.73433	0.724	0.70121	0.744	0.76578	0.74867
41	0.94927	0.93659	0.94927	0.93659	**0.95171**	0.92878	0.91954	0.91911	0.92585	0.92732	0.94504	0.93122
53	0.86299	0.8374	0.86299	0.8374	**0.86457**	0.83858	0.82008	0.81969	0.75787	0.81811	0.81522	0.82717
2079	**0.69864**	0.69548	0.69864	0.69548	0.69502	0.66018	0.63886	0.6267	0.64072	0.65566	0.6419	0.6557
3021	0.99134	0.9909	0.99134	0.9909	**0.99152**	0.98737	0.98288	0.9855	0.97438	0.98746		0.98419
3543	**1**	**1**	**1**	**1**	**1**	**1**	0.99019	0.99081	0.99404	0.99091	**1**	0.97967
3560	0.2375	0.22417	0.2375	0.22417	0.23042	0.21125	0.20365	0.20382	0.19139	0.20833	**0.27028**	0.19542
3561	0.69901	0.67228	0.69901	0.67228	0.63119	0.64752	0.65687	0.64563	0.63762	0.66832	**0.71221**	0.71089
3904	**0.82535**	0.82171	0.82535	0.82171	0.82404	0.81393	0.81344	0.81126	0.80998	0.8181	0.821	0.74819
3917	0.87172	0.86572	0.87172	0.86572	**0.87393**	0.85972	0.85118	0.8534	0.84044	0.86019	0.86856	0.80869
3945	**0.98325**	0.98285	0.98325	0.98285	0.98323	0.98197	0.98244	0.98228	0.98189	0.98182		0.96555
3946	0.92863	0.92809	0.92863	0.92809	**0.92901**	0.92624	0.92725	0.92586	0.92599	0.92624		0.78802
3948	0.9506	**0.95137**	0.9506	0.95137	0.94345	0.94116	0.95094	0.9503	0.95068	0.95085		0.93415
7592	0.86906	0.86527	0.86906	0.86527	0.86251	0.85769	0.86938	0.87013	0.86727	**0.87089**	0.85448	0.86656
7593	0.87738	0.93199	0.87738	0.93199	0.70278	0.80902	**0.96395**	0.89143	0.95227	0.94542	0.6639	0.92908
9910	**0.80595**	0.80062	0.80595	0.80062	0.80107	0.78073	0.7889	0.77762	0.77798	0.80249	0.77087	0.80044
9952	**0.91726**	0.91196	0.91726	0.91196	0.91319	0.90826	0.89716	0.89205	0.89273	0.9045	0.89963	0.89205
9955	**0.69562**	0.6775	0.69562	0.6775	0.67167	0.65146	0.65172	0.62795	0.54667	0.61146	0.61097	0.56435
9977	0.97132	0.97098	0.97132	0.97098	0.96525	0.95924	0.96903	0.96656	0.96891	0.97026	0.96055	**0.97146**
9981	**0.96235**	0.95432	0.96235	0.95432	0.95093	0.94228	0.94167	0.93117	0.94012	0.94784	0.96049	0.95216
9985	**0.61983**	0.61133	0.61983	0.61133	0.61029	0.59853	0.59695	0.58601	0.58293	0.61291	0.60272	0.61656
10101	0.80622	0.79644	0.80622	0.79644	0.80667	0.76578	0.76667	0.77778	0.78044	0.78711	**0.81956**	0.73378
14952	**0.97422**	0.97272	0.97422	0.97272	0.97094	0.96623	0.9659	0.96244	0.96964	0.96913	0.96464	0.9716
14954	0.83395	0.82037	0.83395	0.82037	**0.83642**	0.81111	0.79012	0.76173	0.76667	0.81009	0.81701	0.78333
14965	0.90695	0.90625	0.90695	0.90625	0.90307	0.90007	0.90447	0.90398	0.90451	**0.90705**	0.89957	0.9006
14967	**1**	**1**	**1**	**1**	**1**	**1**	0.98265	0.99841	0.97131	**1**		**1**
14968	**0.84259**	0.84074	0.84259	0.84074	0.8358	0.80432	0.77353	0.77058	0.75823	0.81173	0.79155	0.8
14969	0.66722	0.6812	0.66722	0.6812	0.64001	0.61864	0.67733	0.65004	0.67272	0.67586	0.66217	**0.70165**
34538	**1**	**1**	**1**	**1**	**1**	0.99907	**1**	0.99907	0.99983	**1**	**1**	**1**
34539	0.94915	0.94702	0.94915	0.94702	0.94825	0.94557	0.94761	0.94444	0.9475	0.94891	0.94606	**0.95114**
125920	0.61533	0.6	0.61533	0.6	0.63133	0.562	0.56667	0.55556	0.56844	0.56867	**0.66978**	0.584
146195	0.82042	0.82628	0.82042	0.82628	0.77358	0.77321	0.82109	0.79628	0.82886	0.84123	0.77698	**0.865**

continued on the next page

Table 8.4: Average accuracy (rounded to 5 decimals) over 10 repetitions for the 73 OpenML datasets, ordered by #Task ID. The first fourth columns after "Dataset" show the experimental results, i.e., 4 variants of the contesting procedure. The remaining columns contain results obtained by other AutoML frameworks according to results presented in Chapter 7 and [22]. – continued from previous page

OpenML IDs	DACOpt contesting procedure				Chapter 7		[22]					
#TaskID	DAC-HB	DAC-HH	DAC-SB	DAC-SH	BO4ML	Hyperopt	Auto sklearn	Random search	HP sklearn	TPOT	ATM	H2O
146212	**0.9999**	0.99989	0.9999	0.99989	0.99965	0.99945	0.99978	0.99968	0.99253	0.99974	0.99955	0.99987
146606	0.7115	0.71645	0.7115	0.71645	0.70605	0.69761	**0.72296**	0.7193	0.70743	0.72031	0.67135	0.71281
146607	**0.87383**	0.8708	0.87383	0.8708	0.86611	0.85871	0.86291	0.86225	0.86661	0.86392	0.86128	0.84968
146800	**1**	**1**	**1**	**1**	0.99969	0.99321	0.99043	0.99506	0.9638	0.99506	**1**	0.99551
146817	**0.80892**	0.79966	0.80892	0.79966	0.80497	0.78216	0.78268	0.76364	0.75955	0.79091	0.76415	0.78062
146818	0.88889	0.88261	0.88889	0.88261	0.88647	0.85845	0.87053	0.85556	0.86913	0.86184	**0.8905**	0.87633
146819	0.96728	0.96605	0.96728	0.96605	0.95802	0.93951	0.94074	0.92407	0.92593	0.94547	**0.96975**	0.93642
146820	**0.98864**	0.98747	0.98864	0.98747	0.97355	0.97906	0.98612	0.98581	0.95289	0.9854	0.98657	0.98574
146821	**0.99981**	0.99884	**0.99981**	0.99884	0.99441	0.98748	0.97264	0.97958	0.98786	0.99422	0.96763	0.99191
146822	**0.94603**	0.94055	0.94603	0.94055	0.94473	0.93189	0.93088	0.93333	0.90664	0.94055	0.92564	0.94185
146824	0.9835	0.98233	0.9835	0.98233	**0.98433**	0.98117	0.97783	0.97367	0.98121	0.96883	0.9775	0.976
146825	0.85193	0.86744	0.85193	0.86744	0.843	0.83891	**0.87844**	0.8445	0.8506	0.78089	0.82114	0.87341
167119	0.84892	0.86674	0.84892	0.86674	0.84647	0.83956	0.86775	0.85378	0.88691	0.88735	0.8754	**0.90047**
167120	**0.52457**	0.52385	0.52457	0.52385	0.52257	0.52134	0.51926	0.51939	0.52033	0.52082	0.51941	0.50635
167121	0.7812	0.89001	0.7812	0.89001	0.86652	0.7491	0.74009	0.02169	0.86438		**0.8947**	0.5822
167124	0.33284	**0.40956**	0.33284	0.40956	0.39675	0.37813			0.32093	0.29429	0.32001	0.36389
167125	0.97856	**0.97876**	0.97856	0.97876	0.97713	0.97033	0.97774	0.97114	0.97358	0.97398	0.969	
167140	0.96475	0.96287	0.96475	0.96287	0.96485	0.95397	0.95962	0.95889	0.96109	0.95931	0.95282	**0.96904**
167141	**0.9636**	0.96133	0.9636	0.96133	0.96273	0.95367	0.9562	0.95313	0.94533	0.96	0.95007	0.9537
168329	**0.33594**	0.32871	0.33594	0.32871	0.3169	0.29294	0.30692	0.29566	0.28741	0.33576	0.32108	
168330	0.68921	0.70161	0.68921	0.70161	0.68479	0.6667	**0.71814**	0.69273	0.68494	0.69642	0.63788	0.71786
168331	0.6319	0.65658	0.6319	0.65658	0.60445	0.5952	0.66933	0.63762	0.65451	0.65075	**0.6794**	0.67841
168332	0.39963	0.44613	0.39963	0.44613	0.42507	0.38497	**0.44843**	0.39922	0.34203		0.35252	
168335	0.93435	0.93889	0.93435	0.93889	0.92248	0.91035	0.94334	0.92891	0.87477	0.9385	0.90234	**0.94604**
168337	0.7521	0.81453	0.7521	0.81453	0.78205	0.72707	0.64227		0.74347	0.72548	0.66063	**0.81928**
168338	0.9701	**0.99552**	0.9701	0.99552	0.98303	0.98035	0.74757	0.75042	0.82518	0.98495	0.90729	0.95625
168868	0.99183	0.99318	0.99183	0.99318	0.98985	0.989	0.99287	0.99137	0.9936	0.99339	0.97097	**0.99369**
168908	**0.75609**	0.75166	0.75609	0.75166	0.73659	0.72565	0.74754	0.73081	0.7163	0.72645	0.72169	0.72811
168909	0.96513	0.983	0.96513	0.983	0.9504	0.94437	**0.98357**	0.94793	0.97243	0.96254	0.95391	0.96988
168910	0.69563	0.69308	0.69563	0.69308	0.67565	0.66177	0.70255	0.67395	0.69104	0.68336	0.67357	**0.71752**
168911	0.82578	0.81942	0.82578	0.81942	**0.83259**	0.80748	0.82009	0.80603	0.80078	0.82366	0.79911	0.80906
168912	0.95033	0.95273	0.95033	0.95273	**0.95709**	0.94655	0.93921	0.94753	0.94675	0.95533	0.93476	0.9251
189354	0.65866	0.6634	0.65866	0.6634	0.64626	0.63957	0.66665	0.59845	0.6508	**0.66895**	0.63671	0.61266
189355	0.81678	**0.82433**	0.81678	0.82433	0.73916	0.68112			0.77971		0.38666	
189356	0.65874	0.66576	0.65874	0.66576	0.6481	0.64737	0.68314	0.66709	0.66694	0.6611	**0.80064**	0.64798
Cases with the highest values achieved	28	10	6	5	13	3	8	1	1	6	16	12
Significant wins over other approaches	26	8	4	3	11	2	6	0	0	4	13	11

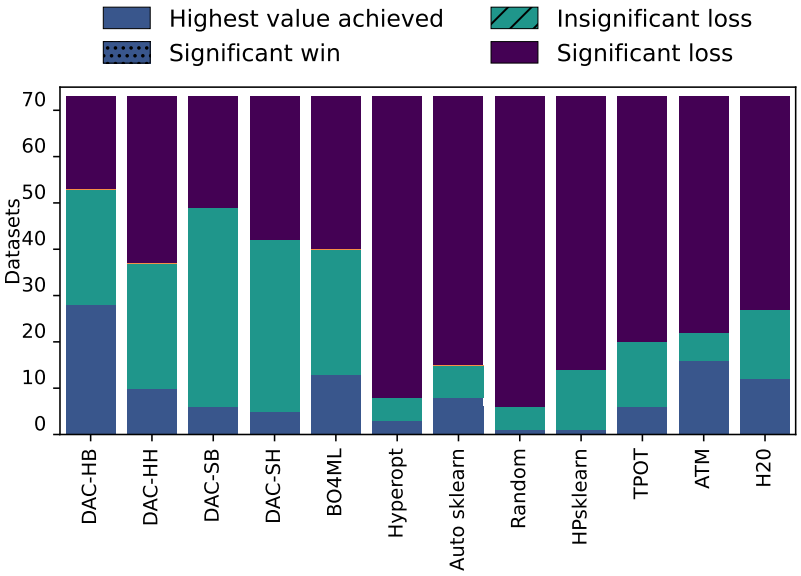


Figure 8.5: Overview of the results over 10 repetitions for the 73 AutoML benchmark datasets.

8.5.2 Second experimental results

In this experiment, we compare all approaches used in the first experiment to the current state-of-the-art AutoML frameworks, i.e., Auto-sklearn-SMAC (Auto-sklearn) and Auto-sklearn-Random search (Random), HPsklearn ([42], TPOT [43], ATM [90], H2O [89]), based on the results obtained by [22]. The detailed results of the second tested scenarios are presented in Table 8.4. We note that entries with missing values in the last 6 columns indicate arbitrary fails reported by [22]. Additionally, the results of the second experiment are summarized in Figure 8.5. This figure is based on the accuracy of the test dataset over 10 repetitions to show the performance differences between the two BO variant-based TPE surrogate models, namely BO4ML and Hyperopt, to compare both with and without the proposed contesting procedure, as well as with two elimination criteria, namely the highest value and a statistical procedure (see Table 8.1).

- First, when comparing the three approaches that use Hyperopt as the underlying optimizer, i.e., DAC-HH, DAC-SH, and Hyperopt, we observed that both proposed contesting procedures won on more tested cases than Hyperopt. More precisely, DAC-HH, DAC-SH, and Hyperopt significantly

outperformed others in $8, 3, and$ 2 cases, respectively. However, in these tested cases of Hyperopt, it is never significantly better than both DAC-SH and DAC-HH; DAC-SH and DAC-HH are not significantly different. In contrast, correspondingly, DAC-HH and DAC-SH significantly outperform Hyperopt in 5 and 1 cases. Therefore, we can conclude that (1) both contesting procedures significantly improve the performance of BO, (2) DAC-HH won against Hyperopt in more cases compared to DAC-SH.

- Secondly, we analyze the results of three approaches that use BO4ML as the underlying optimizer, i.e., DAC-HB, DAC-SB, and BO4ML. We observe that: (1) all three approaches performed well on $73\%, 67\%$, and 55% tested cases, respectively; (2) DAC-HB achieved the highest performances on most of the tested cases, followed by BO4ML and DAC-SB. In 11 cases where BO4ML significantly outperformed others, it was not significantly better than any of the competitors in this comparison. DAC-SB was significantly better than BO4ML on 1 tested case, i.e., task 146821, but it never won DAC-HB. In comparison, DAC-HB outperformed DAC-SH and BO4ML on 3 and 7 cases, correspondingly.

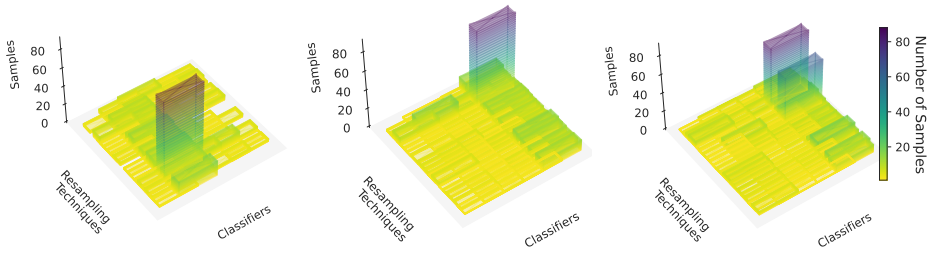
- Comparing the results of 8 approaches using the search space of Auto-sklearn, i.e., our four approaches, BO4ML, Hyperopt, Auto-sklearn, and Random search, we can observe that: First, all BO-based approaches performed better than random searches over all tested cases. Random search achieves the highest result in 1 case (#ID:24), in which all competitors perform equally (no win). Second, it can be seen that DAC-HB won in most tested cases, followed by BO4ML, DAC-HH, Auto-Sklearn, DAC-SH, DAC-SH, Hyperopt, and Random search. We conclude that the proposed approach clearly improves the efficiency of BO in solving AutoML optimization problems. This finding may be explained by the fact that the HPO-based approach does not consider the relationship between algorithms under operators; thus, it requires more resources to cover a large and complex search space in this experiment. In contrast, by grouping similar algorithms together and splitting the original search space into smaller independent subspaces, the proposed approach better utilizes the given budget. Consequently, the search space can be covered within a relatively small budget, and the most promising subspace can be identified early. As a result, resources are efficiently distributed. Additionally, BO is known to perform better for low-dimensional problems [31], [241], [242].

Our approach transfers the original high-dimensional problem of AutoML into multiple low-dimensional problems, thus improving the performance of BO.

- Additionally, when comparing all contesting variants, it can be seen that DAC-HB won on more tested cases than others. The contesting procedure based on the highest performance, i.e., DAC-HH, DAC-HB, won on more cases than those based on the statistical procedure, i.e., DAC-SH, DAC-SB. This finding may be explained by the fact that executing a statistical method adds to the overall computational cost of the procedure. As a result, the contesting technique that used statistical procedures examined fewer configurations in the same amount of time as the others.

- Finally, Figure 8.5 shows that the proposed contesting procedures performed well on up to 73% and at least 53% of all tested cases, when compared to Random Search -8% of all cases, Hyperopt - 11% of all cases, AutoSklearn - 21% of all cases, TPOT - 27% of all cases, ATM - 30% of all cases and H2O - 37% of all cases.

8.6 Application on Surface Defect Classification in Steel Manufacturing



(a) TPE (b) BO4ML (c) DACOpt

Figure 8.6: Illustration on the distribution of 500 samples across a search space of 5 classifiers and 21 resampling techniques of the three optimization algorithms, namely TPE, BO4ML, DACOpt. In this run, different approaches explore specific combinations of algorithms (cells in the figure) to find the combination that can achieve the best performance. BO4ML and DACOpt cover more combinations of algorithms (cells in the figure) than TPE. Specifically, TPE, BO4ML, and DACOpt have 35, 13, and 8 combinations with no samples (white cells color in the figure), respectively.

In this section, we present an application of DACOpt in a real-world application for surface classification in steel manufacturing. This multi-class imbalance problem was introduced in Chapter 6. We have two main objectives in this section:

- Firstly, we aim to enhance the performance of the current classification system used for surface defect detection at our industry partner, TATA, by applying AutoML optimization. This study will use a standard performance metric, i.e., geometric mean (micro), as the objective function. This is different from Chapter 6.
- Secondly, we apply our new method, DACOpt, to the real-world application for surface classification in steel manufacturing. As presented in Section 8.5.1 and Section 8.5.2, the experimental results show that DAC-HB won on more test cases compared to other variants of DACOpt. Thus, we use DAC-HB as the mere variant of DACOpt in this study. Additionally, we aim to investigate the efficiency of DACOpt as compared to BO4ML (Chapter 7) and TPE [158]. The difference between the three optimization approaches is illustrated in Figure 8.6. The illustration shows the sample distribution of 500 samples across the search space of 5 classification and 21 resampling algorithms. The TPE algorithm is shown in Fig. 8.6a, while the our two optimization algorithms are BO4ML (Fig. 8.6b) and DACOpt (Fig. 8.6c). The height and color of each bar represent the number of samples. The white cell shows unexplored algorithm combinations. The figure shows that TPE has more unexplored combinations than the other algorithms, indicating that some ML algorithm combinations were never explored.

In the remainder of this section, we present the experimental setup (Section 8.6.1) followed by the experimental results and discussion (Section 8.6.2).

8.6.1 Experimental setup

As mentioned, this study reuses the experimental setup introduced in Chapter 6[12], which includes the search space (Section 6.3), experimental procedure (Section 6.3.2), and datasets (Section 6.3.1). As a reminder, the search space includes five classification algorithms, namely Support Vector Machines (SVM), Random Forest (RF), k-nearest Neighbors (KNN), Decision Tree (DT), and Logistic Regression (LR), along with 3 commonly used multiple-class classification

[12]The experiment scripts for the reproducibility of the reported results are provided in a git-repository :`https://github.com/anh05/AutoML-Multiclass-Imbalanced`

techniques (Multi-class direct classification (Direct), One-vs-One (OvO), and One-vs-Rest (OvR)). We direct interested readers to Chapter 6 (Section 6.2) for a detailed discussion of the relevant background. Additionally, there are 21 options for resampling techniques, including the option of not using any resampler, leading to a total of 84 hyperparameters in the search space. We have improved the practicality of selected resampling techniques in tackling multi-class imbalanced problems by introducing a hyperparameter called `sampling strategy`. It offers a range of values including {`majority/minority`[13], `not minority`, `not majority`, `all`, `auto`}. We use the geometric mean micro ($\mathrm{GM}_{\mathrm{micro}}$) as the objective function to maximize. For M classes $(\mathrm{A}, \mathrm{B}, \ldots, \mathrm{M})$ in a multi-class classification problem, we calculate the $\mathrm{GM}_{\mathrm{micro}}$ as:

$$\begin{aligned} \mathrm{GM}_{\mathrm{micro}} &= \sqrt{\mathrm{Specificity}_{\mathrm{micro}} \times \mathrm{Sensitivity}_{\mathrm{micro}}} \\ &= \sqrt{\frac{\sum_{i=1}^{\mathrm{M}} \mathrm{TN}_i}{\sum_{i=1}^{\mathrm{M}} \mathrm{TN}_i + \sum_{i=1}^{\mathrm{M}} \mathrm{FP}_i} \times \frac{\sum_{i=1}^{\mathrm{M}} \mathrm{TP}_i}{\sum_{i=1}^{\mathrm{M}} \mathrm{TP}_i + \sum_{i=1}^{\mathrm{M}} \mathrm{FN}_i}} \end{aligned} \tag{8.5}$$

where $\mathrm{TP}_i, \mathrm{TN}_i, \mathrm{FP}_i, \mathrm{FN}_i$ denote the number of true positives, true negatives, false positives and false negatives samples in class $i \in \mathrm{M}$, respectively.

For this particular study, we conducted 9 optimization processes for a given dataset using different classification strategies and optimizers. To be specific, for each optimization approach, we set up three independent experiments, each representing a different approach– One vs. Rest (OvR), One vs. One (OvO), and Direct classification (Direct) strategies. Therefore, we had $3 \times 3 = 9$ optimization processes for a dataset. All 9 optimization processes have a budget of 500 for function evaluations. Additionally, our experiments aim to compare the *current classification system* (current system) used by our industry partner[14]. We use the same training and test datasets as the current system for a fair comparison. The current system executes 10 times on each of the tested datasets. For each execution, the considered dataset is randomly split into training (80%) and test (20%) sets. Those train/test sets are exported to use in our experiments, i.e., we have $2 \times 10 = 20$ different train/test sets in total.

[13] `majority` is an option for under resampling, `minority` is for over/combine resampling techniques

[14] For reasons of confidentiality, since proprietary software of a supplier is used by the industrial partner, no details about the algorithmic approach taken by the currently used system are available.

8.6.2 Experimental results and discussion

In this section, we present our findings and insights. We have summarized the experimental results in Table 8.5 to showcase the performance differences between three optimization algorithms, namely TPE, BO4ML, and DACOpt. For each optimizer, we have provided their optimization performance with the use of three classification strategies, namely multi-class direct classification (Direct), One vs. One (OvO), and One vs. Rest (OvR). This results in 9 experimental outcomes for each dataset. We have compared these results against the classification approach used in the current system. The highest performance for each dataset is highlighted in bold. The methods performing significantly worse than the best according to the Wilcoxon signed-rank test with $\alpha = 0.05$ are underlined. Additionally, the

Table 8.5: Average geometric mean (micro), rounded to 5 decimals over 10 repetitions for the 2 datasets. Boldface highlights the best-performing method per dataset and underline indicates results that are significantly different from the best method in that group according to a Wilcoxon signed-rank test ($p < 0.05$).

Dataset	Current	TPE			BO4ML			DACOpt		
	system	Direct	OvO	OvR	Direct	OvO	OvR	Direct	OvO	OvR
Top side	0.87269	0.92068	0.91957	0.92394	0.92076	0.91982	0.92474	0.92115	0.92069	**0.92554**
Bottom side	0.86064	0.94085	0.94175	0.94146	0.94096	0.9198	0.94165	0.94137	**0.94247**	0.94176

distribution of geometric mean micro over 10 repetitions for the two tested datasets, is visualized in Fig. 8.7. Each box plot represents 10 repetitions. The horizontal inner line shows the median. The whiskers show the lowest and the highest observed value[15]. The color dots show the observed values, and the dots outside the whisker represent the outliers. The box covers the first to the third quantiles. The results allow the following insights:

- According to the results of the Wilcoxon signed-rank test, our experimental approaches significantly outperform the current approach used at our industry partner (current system). Additionally, from Fig. 8.7, the median and whiskers of all optimization approaches on three classification scenarios are higher than those of the current system. In other words, our procedure has successfully enhanced the performance of the current classification system used by our industry partner.

[15]The whisker scale is set as 1.5.

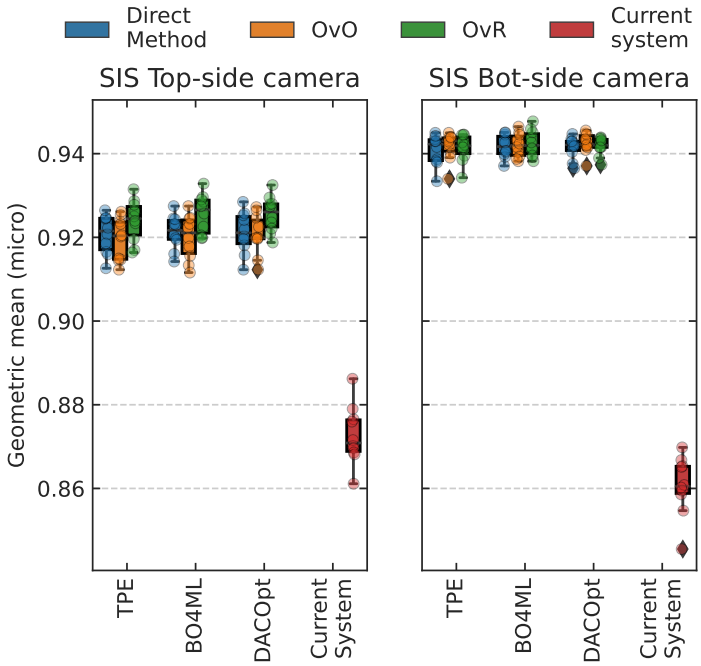


Figure 8.7: Box plots showing the distribution of classification results for two examined datasets.

- Overall, DACOpt achieved the best classification performance on both datasets that were examined. Specifically, for the SIS top-side dataset, the DACOpt approach using the OvR classification strategy outperformed TPE in all experiments. It also significantly outperformed BO4ML in two cases of direct classification and OvO strategies. For the SIS bottom-side dataset, the DACOpt method with the OvR classification strategy also achieved the highest result. It significantly outperformed TPE with the direct classification strategy.

- We conducted a Friedman's Test on all three strategies to determine the most effective classification strategy among direct classification, OvO, and OvR when used with AutoML optimization approaches. Surprisingly, the results showed no significant difference in the average GM (micro) with a p-value of 0.13169. It is surprising because the decomposition approach is the most commonly recommended for dealing with multi-class problems from literature, as it converts the multi-class problem into multiple binary-class

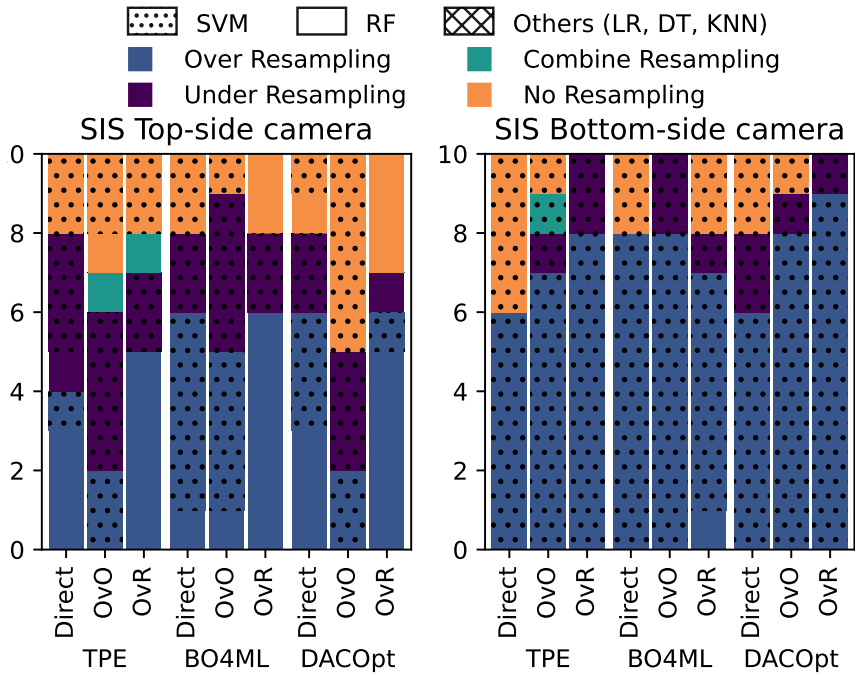


Figure 8.8: Illustration of the different combinations of resampling and classification algorithms generated by the best ML pipeline resulting from the optimization processes conducted by TPE, BO4ML, and DACOpt. Within each of the 9 optimization approaches (3 for each method), 3 distinct classification strategies, namely Direct classification (Direct), One-vs-One (OvO), and One-vs-Rest (OvR). The optimization process is repeated ten times on two examined datasets.

problems, which makes the classifier work more efficiently. However, our finding indicates that applying AutoML optimization performs similarly for three classification strategies.

We have presented Fig. 8.8 to investigate the final combination of choices for resampling and classification algorithms of all optimization methods. Our findings indicate that SVM and RF are the algorithms most frequently selected, while other classification algorithms have not been chosen in any of the test cases. SVM wins in 61% and 99% of cases for the SIS Top-side and SIS Bottom-side datasets, respectively. RF obtains 39% wins on the SIS Top-side dataset, but only one win in the SIS Bottom-side dataset by BO4ML with the OvR classification strategy. Regarding the usefulness of resampling techniques, 81% of the runs yield the highest results using some form of resampling technique. Over resampling, under

Figure 8.9: Comparison of all optimization algorithms compared to each other using Nemenyi test with a 5% significance level.

resampling, and combined resampling obtain 109 (61%), 34 (19%), and 3 (2%) wins over (2 datasets $\times$ 3 classification strategies $\times$ 3 optimizers) $\times$ 10 repetitions = 180 runs. At the same time, 34 (19%) runs yield the highest performance without using any resampling techniques.

Based on the results of Friedman's test in average GM (micro), we found significant differences among all optimization approaches, with a p-value of 9.8e-4. As a follow-up, we performed post hoc multiple comparison tests using the Nemenyi test at a significance level of 0.05, as shown in Fig. 8.9. Approaches with a distance greater than CD[16] are considered significantly different. Upon analyzing the figure, we can conclude that DACOpt outperforms all competitors, while BO4ML and TPE perform similarly.

Based on the results of our experiment, we have arrived at **four main conclusions**:

1. Our experiments demonstrated that AutoML optimization approaches significantly improved classification performance compared to the current system.

2. In addition, we found that our new approach, DACOpt, outperforms the two competitors, i.e., BO4ML and TPE. Therefore, we recommend the use of the DACOpt method for AutoML optimization.

3. Resampling techniques are recommended to deal with multi-class imbalanced problems, as 81% of runs yield the best performance using them.

4. Lastly, our findings indicate that applying AutoML optimization to direct classification yields similar performance compared to using it in OvO and OvR strategies. Therefore, we highly recommend using the direct classification approach to address similar problems, as it is much more cost-effective than OvO and OvR.

[16]Critical Difference, here CD=1.353

It is worth noting that our study focuses solely on enhancing the current classification component of the surface defect detection system used by our industry partner. Moving forward, we plan to expand our research to cover the entire surface defect detection system, including image processing and feature extraction. Additionally, we may look into utilizing deep learning and convolutional neural networks.

8.7 Conclusions and future work

In this study, we proposed a novel contesting procedure for the AutoML optimization problem, namely DACOpt, which is complementary to the existing BO approaches. DACOpt partitions the AutoML search space into multiple relatively small sub-spaces based on algorithm similarity and budget constraints. Next, BO approaches are employed to optimize these sub-spaces independently. The budget is then adaptively distributed to the search area based on the performance of the corresponding BO processes. The proposed contesting procedure has two different variants of elimination criteria – based on the highest performance and a statistical procedure. Additionally, we presented a partly parallel approach to using BO to address AutoML optimization problems with provably theoretical guarantees. Two extensive experiments on a total of 117 benchmark datasets demonstrated the superiority of our novel contesting procedures over the current state-of-the-art AutoML optimization approaches. Additionally, an experiment was conducted on surface defect classification in steel manufacturing. It was concluded that our proposed approach significantly improves BO's performance. In future studies, we intend to incorporate meta-learning approaches to identify search areas that may perform well in the early stages. Finally, the scope of this study was limited to the AutoML optimization problem; we plan to extend our research to Neural Architecture Search (NAS) problems in the future.