

Efficient Automatic CASH via Rising Bandits

Yang Li,¹ Jiawei Jiang,² Jinyang Gao,³ Yingxia Shao,⁴ Ce Zhang,² Bin Cui¹

¹Key Laboratory of High Confidence Software Technologies (MOE), School of EECS, Peking University, Beijing, China

²Department of Computer Science, Systems Group, ETH Zurich, Switzerland

³Beijing University of Posts and Telecommunications, Beijing, China

⁴Alibaba Group, Hangzhou, China

{liyang.cs, bin.cui}@pku.edu.cn, {jiawei.jiang, ce.zhang}@inf.ethz.ch

jinyang.gjy@alibaba-inc.com, shaoyx@bupt.edu.cn

Abstract

The Combined Algorithm Selection and Hyperparameter optimization (CASH) is one of the most fundamental problems in Automatic Machine Learning (AutoML). The existing Bayesian optimization (BO) based solutions turn the CASH problem into a Hyperparameter Optimization (HPO) problem by combining the hyperparameters of all machine learning (ML) algorithms, and use BO methods to solve it. As a result, these methods suffer from the low-efficiency problem due to the huge hyperparameter space in CASH. To alleviate this issue, we propose the alternating optimization framework, where the HPO problem for each ML algorithm and the algorithm selection problem are optimized alternately. In this framework, the BO methods are used to solve the HPO problem for each ML algorithm separately, incorporating a much smaller hyperparameter space for BO methods. Furthermore, we introduce *Rising Bandits*, a CASH-oriented Multi-Armed Bandits (MAB) variant, to model the algorithm selection in CASH. This framework can take the advantages of both BO in solving the HPO problem with a relatively small hyperparameter space and the MABs in accelerating the algorithm selection. Moreover, we further develop an efficient online algorithm to solve the Rising Bandits with provably theoretical guarantees. The extensive experiments on 30 OpenML datasets demonstrate the superiority of the proposed approach over the competitive baselines.

Introduction

Machine learning (ML) has made great strides in many application areas, e.g., recommendation, computer vision, financial market analysis, etc (Goodfellow, Bengio, and Courville 2016; He et al. 2017; Ma et al. 2019; Zhao, Shen, and Huang 2019). However, given a practical application, it is usually knowledge-intensive and labor-intensive to develop customized solutions with satisfied learning performance, where the exploration may include but is not limited to selecting ML algorithms, configuring hyperparameters and network architecture searching. To facilitate the deployment of ML applications and democratize the usage of machine learning, it is of vital importance to reduce human efforts during such exploration. Naturally, automatic

machine learning (Quanming et al. 2018; Zöllner and Huber 2019) has attracted lots of interest from both industry and academia.

Given a learning problem, the first thing is to decide which ML algorithm should be applied – from SVM, Adaboost, GBDT (Jiang et al. 2018; 2017) to deep neural networks. According to the No Free Lunch theorem (Ho and Pepyne 2001), no single ML algorithm can achieve the best performance for all the learning problems; and there is often no golden standard to predict which ML algorithm performs the best. As a result, we typically spend computational resources across all reasonable ML algorithms, and choose the one with the best performance after the optimization of their hyperparameters and network architectures. However, solving the algorithm selection problem after sufficiently optimizing the hyperparameters of each ML algorithm leads to inefficient usage of computational resources. Resources consumed by the poor-performing algorithms are greatly wasted. To this end, the Combined Algorithm Selection and Hyperparameter optimization (CASH) problem (Feurer et al. 2015; Kotthoff et al. 2017) is proposed to jointly optimize the selection of algorithm and its hyperparameters, which is the core focus of this paper.

To solve the CASH problem, a class of methods (Komer, Bergstra, and Eliasmith 2014; Feuerer et al. 2015; Kotthoff et al. 2017) transform the CASH problem into a unified hyperparameter optimization (HPO) problem by merging the hyperparameter space for all ML algorithms and treating the selection of algorithm as a new hyperparameter. Then classical Bayesian optimization (BO) methods (Shahriari et al. 2015) are utilized to solve this HPO problem. Consequently, these methods incorporate a huge optimization space with high-dimensional hyperparameters for BO methods. Past works (Eggenesperger et al. 2013) show that BO methods perform well for relatively low-dimensional hyperparameters. However, for high-dimensional problems, standard BO methods perform even worse than random search (Wang et al. 2013). Thus, such a huge hyperparameter space greatly hampers the efficiency of Bayesian optimization.

To alleviate the above issue, it is natural to consider another paradigm where the BO methods are used to solve the HPO problem for each ML algorithm separately, and

the algorithm selection is responsible for determining the allocation of resources to each ML algorithm’s HPO process. Based on this idea, we propose the *alternating optimization framework*, where the HPO problem for each ML algorithm and the algorithm selection problem are optimized alternately. Benefiting from solving the HPO problem for each ML algorithm individually, this framework brings a much smaller hyperparameter space for BO methods. Furthermore, within this framework, the resources can be adaptively allocated to the HPO process of each algorithm based on their performance. Intuitively, spending too many resources in tuning the hyperparameters of poor-performing algorithms should be avoided; instead, more resources should be allocated to the more promising ML algorithms that can achieve the best performance. Unfortunately, which algorithm is the best is unknown unless enough resources are allocated to its HPO process. Therefore, solving the CASH problem efficiently requires to trade off the well-celebrated Exploration vs. Exploitation (EvE) dilemma during algorithm selection: *should we explore the HPO of different ML algorithms to find the optimal algorithm (Exploration), or give more credit to the best algorithm observed so far to further conduct HPO (Exploitation)?*

Since the EvE dilemma has been intensively studied in the context of Multi-Armed Bandits (MAB), here we propose to solve the algorithm selection problem in the framework of MAB. In this setting, each arm is associated with the corresponding HPO process of an ML algorithm. Pulling an arm means that a unit of resource is assigned to the HPO process of the corresponding algorithm, and the reward corresponds to the result from the HPO process. However, the existing MABs cannot be directly applied to model the algorithm selection problem for two reasons: 1) the well-studied objectives of MABs (e.g., accumulated rewards) are not consistent with the target of CASH problem that aims to maximize the observed reward; 2) because the HPO results will be improved with the increase of the HPO resource, the reward distribution of each arm is not stationary over time.

The main contributions of this paper are the following:

- We propose the alternating optimization framework to solve the CASH problem efficiently, which optimizes the algorithm selection problem and the HPO problem for each ML algorithm in an alternating manner. It takes the advantages of both BO methods in solving the HPO problem with a relatively small hyperparameter space and MABs in accelerating the algorithm selection.
- We introduce a novel, CASH-oriented MAB formulation, termed *Rising Bandits*, where each arm’s expected reward increases as a function of the number of times it has been pulled. To the best of our knowledge, this is the first work that models the algorithm selection problem in the framework of non-stationary MABs.
- We present an easy-to-follow online algorithm for the Rising Bandits, accompanied with provably theoretical guarantees.
- The empirical studies on 30 OpenML datasets demonstrate the superiority of the proposed method over the state-of-the-art baselines in terms of final accuracy and

efficiency. Our method can achieve an order of magnitude speedups compared with BO based solutions.

Preliminaries and Related Works

We first introduce the basic notations for the CASH problem. There are K candidate algorithms $\mathcal{A} = \{A^1, \dots, A^K\}$. Each algorithm A^i has a corresponding hyperparameter space Λ_i . The algorithm A^i with a hyperparameter λ is denoted by A^i_λ . Given the dataset $D = \{D_{train}, D_{valid}\}$ of a learning problem, the CASH problem is to find the joint algorithm and hyperparameter configuration $A^*_{\lambda^*}$ that minimizes the loss metric (e.g., the validation error on D_{valid}):

$$A^*_{\lambda^*} = \operatorname{argmin}_{A^i \in \mathcal{A}, \lambda \in \Lambda^i} \mathcal{L}(A^i_\lambda, D). \quad (1)$$

Hyperparameter optimization (HPO) is to find the hyperparameter configuration λ^* of a given algorithm A^i , which has the best performance on the validation set,

$$\lambda^* = \operatorname{argmin}_{\lambda \in \Lambda_i} L(A^i_\lambda, D). \quad (2)$$

Bayesian optimization (BO) has been successfully applied to solve the HPO problem. Sequential Model-based Algorithm Configuration (SMAC) (Hutter, Hoos, and Leyton-Brown 2011), Tree-structure Parzen Estimator (TPE) (Bergstra et al. 2011), and Spearmint (Snoek, Larochelle, and Adams 2012) are three well-established methods. It is important to note that these approaches can be executed in a sequential manner. That is, the HPO process is iterative. Recently, many approaches develop some elaborated mechanisms to allocate the HPO resources adaptively (Huang et al. 2019; Falkner, Klein, and Hutter 2018; Sabharwal, Samulowitz, and Tesauro 2016). In addition, multi-fidelity optimization has been deeply studied in the framework of BO to accelerate the HPO problem (Swersky, Snoek, and Adams 2013; Klein et al. 2017; Kandasamy et al. 2017; Poloczec, Wang, and Frazier 2017; Hu et al. 2019).

In the algorithm selection problem, the objective is to choose a parameterized algorithm $A^*_{\lambda^*}$, which is the most effective with respect to a specified quality metric $Q(\cdot)$. This sub-problem can be stated as a minimization problem:

$$A^*_{\lambda^*} = \operatorname{argmin}_{i \in [1, \dots, K]} Q(A^i_{\lambda^*}, D). \quad (3)$$

In practice, all candidate algorithms with some fixed hyperparameters are evaluated on the validation dataset, and the algorithm with the best performance is chosen. However, this method suffers from the “low accuracy” issue due to the lack of the HPO: the fixed hyperparameters cannot accurately reflect the performance of the algorithm across different problems. Moreover, many methods select algorithms according to some theoretical decision rules, meta-learning methods (Abdulrahman et al. 2015) and supervised learning techniques (Sun and Pfahringer 2013).

To solve the CASH problem effectively in the ML applications, it is necessary to select the algorithm and its hyperparameters simultaneously. Auto-Weka is the first work devoted to the CASH problem, which takes the BO based solutions. Then Auto-Sklearn and Hyperopt-Sklearn also adopt the same BO based framework. In addition, tree-based pipeline optimization tool (TPOT) (Olson and Moore

2019) uses genetic programming to address the CASH problem. Recently, Reinforcement learning method (Efimova, Filchenkov, and Shalamov 2017) and MAB based methods (Liu et al. 2019) have been studied to solve the CASH problem. They model the rewards in the stationary environment and ignore the objective’s difference between MABs and CASH. In the community of MAB, several works (Besbes, Gur, and Zeevi 2014; Jamieson and Talwalkar 2016; Heidari, Kearns, and Roth 2016; Levine, Crammer, and Mannor 2017) focus on the non-stationary bandits, but none of them match the settings in CASH.

The Proposed Method

In this section, we introduce the alternating optimization framework, give the formulation of *Rising Bandits*, and describe the online algorithm to solve this bandit problem.

The Alternating Optimization Framework

We reformulate the CASH problem into the following bilevel optimization problem:

$$\begin{aligned} \min_{i \in [1, \dots, K]} \quad & Q(A_{\lambda^*}^i, D) \\ \text{s.t.} \quad & \lambda^* = \operatorname{argmin}_{\lambda \in \Lambda_i} L(A_{\lambda}^i, D). \end{aligned} \quad (4)$$

Here the CASH problem is decomposed into two kinds of sub-problems: algorithm selection problem (the upper-level sub-problem) and the HPO problem for each ML algorithm (the lower-level sub-problem). We propose to solve this bilevel optimization problem by optimizing the upper-level and lower-level sub-problems alternately. We name it the *alternating optimization framework*. In this framework, Bayesian Optimization (BO) methods are used to conduct HPO for each ML algorithm individually; MAB based method is utilized to solve the algorithm selection problem. This framework brings two benefits:

- Since the hyperparameter space for each ML algorithm is relatively small, BO methods can solve the corresponding HPO problem efficiently.
- The resources can be adaptively allocated to the HPO of each ML algorithm according to its HPO performance in the MAB framework.

As a result, the poor-performing ML algorithms will be equipped with few HPO resources (e.g., the number of trials), and more resources are allocated to the promising algorithms that can achieve better learning performance.

Non-stationary Rewards from Bayesian Optimization

Before introducing the Rising Bandits, we first investigate the rewards (HPO results) from BO methods. Given more HPO resources, the expected rewards (i.e., the best-observed validation accuracy) will increase. Figure 1 provides an intuitive example. Six ML algorithms are equipped with 200 trials to conduct HPO. The rewards $r(\cdot)$ correspond to the best-observed validation accuracy in each trial. As the number of HPO trial increases, this validation accuracy improves gradually, and then gets saturated. Further, we can summarize the following observations about the rewards from BO:

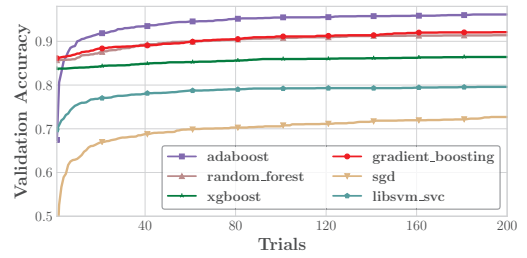


Figure 1: The HPO results of 6 ML algorithms. BO method – SMAC is used to tune the hyperparameters of each algorithm 50 times, and the average validation accuracy across trials is reported.

- For each ML algorithm A^k , the reward sequence $r_k(1), \dots, r_k(n)$ is increasing and bounded, and the limit $\lim_{n \rightarrow \infty} r_k(n)$ exists.
- The reward sequence satisfies the *decreasing marginal returns* approximately. Here we abuse the terminology and refer to this as “concavity”.

Since the rewards increase monotonically across trials, it is evident that the rewards are not identically distributed, but are generated by a non-stationary stochastic process.

The Definition of Rising Bandits

Based on the observations about the HPO results, we give the formulation of Rising Bandits to model the algorithm selection problem with non-stationary rewards. In this bandit variant, the agent is given K arms, and at each time step $t = 1, 2, \dots, T$ one of the arm must be pulled. Each arm k is associated with the HPO process of an ML algorithm A^k . Pulling an arm means that a unit of resource (e.g., an HPO trial) is assigned to the HPO process of an algorithm, and the reward corresponds to the non-stationary HPO results.

In Rising Bandits, we model the non-stationary reward sequences of the arms as follows: each arm k has a fixed underlying reward function denoted by $r_k(\cdot)$. All the reward functions are bounded within $[0, 1]$. When the agent pulls arm k for the n^{th} time, he receives an instantaneous reward $r_k(n)$. We denote the arm that is pulled at time step t as $i(t) \in [K] = [1, \dots, K]$. Let $N_k(t)$ be the number of pulls of arm k at time step t , not including this round’s choice, that’s, $N_k(1) = 0$, and Π the set of all sequence $i(1), i(2), \dots$, where $i(t) \in [K], \forall t \in \mathbb{N}$. i.e., $\pi \in \Pi$ is a sequence of actions (arms), also referred to as a policy. We denote the arm that is chosen by policy π at time step t as $\pi(t)$.

Instead of maximizing the accumulated rewards $\sum_{t=1}^T r_{\pi(t)}(N_{\pi(t)}(t) + 1)$, the objective of the agent in CASH is to maximize the observed reward within T , defined for policy $\pi \in \Pi$ by,

$$J(T; \pi) = \max_{t=1:T} r_{\pi(t)}(N_{\pi(t)}(t) + 1). \quad (5)$$

We consider the equivalent objective of minimizing the regret within T defined by,

$$R(T; \pi) = \max_{\tilde{\pi} \in \Pi} \{J(T; \tilde{\pi})\} - J(T; \pi). \quad (6)$$

Based on the observations about the non-stationary rewards, we introduce the following assumption:

Assumption 1. (Rising) $\forall k \in [K]$, $r_k(n)$ is bounded, increasing, and concave in n .

According to this assumption, the original objective in (5) is equivalent to,

$$J(T; \pi) = \max_k r_k(N_k(T+1)). \quad (7)$$

In the CASH problem, it is clear that the reward function $r(n)$ is bounded and increasing; but the concavity assumption may not always hold. We will discuss the two situations in the following sections. Then we investigate an offline solution for the Rising Bandits. The offline setting means that the optimal arm is known to the agent before the game. Let π^{max} be a policy defined by,

$$\pi^{max}(t) \in \operatorname{argmax}_{k \in [K]} r_k(T). \quad (8)$$

Lemma 1. π^{max} is the optimal policy for the Rising Bandits problem in the offline setting.

Proof: See Appendix A.1 of the supplementary material.

If the best arm is known to the agent, the optimal policy must pull the best arm repeatedly.

Online Solution for Rising Bandits

The CASH problem falls into the online setting, where the best arm is unknown to the agent. In this circumstance, the above Lemma 1 fails. However, it guides us to derive an efficient policy in the online setting: 1) first identify the best arm by using as few time steps as possible, and then 2) pull the best arm until the time step T meets. That is, solving the best arm identification problem first and then fully exploiting the best arm can efficiently optimize the objective in (7).

Based on the Assumption 1, we can obtain an interval that bounds the final reward of an arm. The reward function is concave, that's, for any $n > 2$, we have $r(n) - r(n-1) \geq r(n+1) - r(n)$. Suppose the arm k has been pulled n times, and n rewards $r_k(1), \dots, r_k(n)$ are observed. Given that $r_k(\cdot)$ is increasing, bounded and concave, we have for any $t > n$,

$$r_k(t) \leq \min(r_k(n) + (t-n)\omega_k(n), 1), \quad (9)$$

where $\omega_k(n)$ equals $r_k(n) - r_k(n-1)$, and we name $\omega(n)$ as the growth rate at the n^{th} step. We refer to the right-hand side of Inequality 9 as the upper bound $u_k(t)$ of $r_k(t)$. Naturally, the lower bound $l_k(t)$ of $r_k(t)$ is $r_k(n)$. If the arm i has the lower bound $l_i(t)$ that is no less than the upper bound $u_k(s)$ of the arm k , the arm k cannot be the optimal arm. By using this elimination criterion, we can gradually dismiss the arm that cannot be the optimal arm. After finding the best arm, this arm will be pulled repeatedly until the game ends.

Algorithm 1 illustrates both the pseudo-code of the proposed online algorithm and its usage in the alternating optimization framework. It operates as follows: it maintains a set of candidate arms (ML algorithms) in which the best arm is guaranteed to lie (Line 1). At each round, it pulls all the arms in the candidate set once, and it means that each corresponding algorithm in the candidate set is given one unit of

Algorithm 1 Online algorithm for Rising Bandit

Input: ML algorithm candidates $\mathcal{A} = \{A_1, \dots, A_K\}$, the total time steps T , and one unit of HPO resource \hat{b} .

```

1: Initialize  $S_{cand} = \{1, 2, \dots, K\}$ ,  $t = 0$ .
2: while  $t < T$  do
3:   for each  $k \in S_{cand}$  do
4:      $t = t + 1$ .
5:     Pull arm  $k$  once:  $H_k \leftarrow \text{Iterate\_HPO}(A_k, \hat{b})$ .
6:     Calculate  $\omega_k(t)$  according to  $H_k$ .
7:     Update  $u_k^t(T) = \min(y_k(t) + \omega_k(t)(T-t), 1)$ .
8:     Update  $l_k^t(T) = y_k(t)$ .
9:   end for
10:  for  $i \neq j \in S_{cand}$  do
11:    if  $l_i^t(T) \geq u_j^t(T)$  then
12:       $S_{cand} = S_{cand} \setminus \{j\}$ 
13:    end if
14:  end for
15: end while
16: return the corresponding ML algorithm  $A^*$  and its best hyperparameter configuration.
```

resource to tune its hyperparameters with BO methods. Then both the upper bound and lower bound of the final reward at time step T are updated (Line 5-10). If the upper bound of the final reward of an arm k (algorithm A_k) is less than or equal to the lower bound of another arm's in the candidate set, then the arm k will be eliminated from the candidate set (Line 11-15). The above process iterates until the time step T meets. Finally, the best algorithm along with the corresponding hyperparameter configuration is returned.

Rising Bandits with ‘‘Loose’’ Concavity

As stated previously, the concavity in Assumption 1 may not always hold in the CASH problem. In this case, the problematic growth rate $\omega_k(t) = r_k(t) - r_k(t-1)$ may lead to a wrong upper bound. To alleviate this issue, we propose to use the following growth rate calculated by,

$$\omega_k(t) = \frac{y_k(t) - y_k(t-C)}{C}, \quad (10)$$

where C is a constant. Intuitively, by averaging the latest C growth rates, this smooth growth rate is more robust to the case with ‘‘loose’’ concavity. In the next section, we provide theoretical guarantees for the proposed methods.

Theoretical Analysis and Dissussions

For the coming theorem, we define a quantity that captures the time steps required to distinguish the optimal arm from the others. More precisely, we define $\gamma(T) = \max_k \gamma_k(T)$, where

$$\gamma_k(T) = \arg \min_t \{u_k^t(T) \leq l_{k_*}^t(T)\} \quad (11)$$

and k_*^T is the optimal arm in the given T . Thus $\gamma_k(T)$ specifies the smallest number of time steps needed to pull both arm k and the optimal arm so that the sub-optimal arms can be distinguished from the optimal arm.

We prove that the upper bound of the policy regret of the proposed algorithm exists.

Theorem 1. *Suppose Assumption 1 holds. The proposed algorithm achieves regret bounded by,*

$$R(T; \bar{\pi}) \leq r_{k_*^T}(T) - r_{k_*^T}(T - (n - 1)\gamma(T)). \quad (12)$$

Proof: See Appendix A.2 of the supplementary material. This bound contains a problem-dependent term $\gamma(T)$. If identifying the optimal arm is easier, $\gamma(T)$ will be smaller.

Compare with Average Policy

An intuitive policy π_{uni} is to pull each arm $\frac{T}{K}$ times. The regret of this policy is,

$$R(T; \pi_{uni}) = r_{k_*^T}(T) - \max_k r_k\left(\frac{T}{K}\right). \quad (13)$$

We now establish the regret connection between the proposed algorithm and the average policy.

Corollary 1. *When the problem-dependent term $\gamma(T)$ satisfies: $\gamma(T) \leq \frac{KT-T}{K(K-1)}$, the regret of the proposed algorithm will not be worse than the average strategy's.*

$$R(T; \bar{\pi}) \leq R(T; \pi_{uni}). \quad (14)$$

Proof: See Appendix A.3 of the supplementary material.

Theoretical Guarantee for ‘‘Loose’’ Concavity

Here we provide a theoretical guarantee for the smooth growth rate. For any reward sequence $y_k(1), \dots$, we can find a reward function $r_k(\cdot)$ that satisfies the Assumption 1. At each time step t , $r_k(t) \geq y_k(t)$, and they have the same limit. We denote the bias between $r_k(\cdot)$ and $y_k(\cdot)$ by $\Delta_k(t) = r_k(t) - y_k(t)$.

Theorem 2. *If the following condition holds, the proposed algorithm with smooth growth rate can be used to identify the optimal arm without any loss,*

$$\frac{\Delta_k(t)}{\Delta_k(t - C)} \leq \frac{T - t}{T - t + C}. \quad (15)$$

Proof: See Appendix A.4 of the supplementary material.

Towards Cost-Aware CASH

In the previous sections, the limited resource is the number of HPO trials, and here we consider the time B as the limited resource. Both the algorithm's performance and its HPO evaluation cost in runtime should be taken into consideration. In CASH, conducting an HPO trial for different ML algorithms usually takes a different time cost. For example, for large datasets, training linear models is much faster than the tree-based model such as gradient boosting. To solve the cost-aware CASH, we develop a variant of Algorithm 1. For each ML algorithm, we first compute its average time overhead c_k in each HPO trial; then we predict the upper bound of the final reward within the given time B by,

$$u_k^t(B) = r_k(t) + \omega_k \frac{B'}{c_k}, \quad (16)$$

where B' is the time left, and ω_k is the growth rate at time t .

Relationship and Comparison with Previous Works

Our approach takes an adaptive resource allocation scheme. From a theoretical perspective, our method is similar, in spirit, to some previous works (Huang et al. 2019; Falkner, Klein, and Hutter 2018; Sabharwal, Samulowitz, and Tesauro 2016). In addition, one work (Heidari, Kearns, and Roth 2016) also supports concave reward functions. Compared with these works, our main contribution is to apply a similar methodology to a new application, i.e., CASH. In the CASH problem, we find some additional structures that we can use, e.g., CASH has the concave structure in the reward function. Furthermore, instead of optimizing the accumulated regrets in Heidari, Kearns, and Roth, CASH focuses more on identifying the best arm. These additional structures allow us to perform significantly better over simply applying these previous approaches to CASH.

Compared with BO based solutions, our method explicitly reduces the hyperparameter space in the CASH problem by dismissing the poor algorithms successively. Without any modification, this method can also be used to solve the regression tasks by mapping the loss into $[0, 1]$. In addition, the proposed approach can handle the cost-aware CASH; however, the existing solutions for the CASH problem do not take the evaluation cost into consideration.

Experiments and Results

In the evaluation of the proposed method, we demonstrate its superiority from the following three perspectives: 1) the efficiency compared with the state-of-the-art BO based solutions, 2) the empirical performance compared with all considered baselines in terms of final accuracy and efficiency, and 3) practicability and effectiveness in the AutoML systems.

We compared our method with the following baselines, including the BO based methods and the traditional bandit based methods in the MAB community:

AVG The average policy that allocates the same HPO resources to each ML algorithm.

SMAC BO based method that uses a modified random forest as the surrogate model to conduct BO.

TPE BO based method that utilizes the tree-structured Parzen density estimators as the surrogate model.

CMAB Bandit based method that models the stationary reward and maximizes the accumulated rewards with Thompson sampling (Russo et al. 2018; Liu et al. 2019).

UCB UCB policy is used to solve the traditional MAB.

Softmax Softmax policy (Sutton and Barto 2018) is leveraged to solve the traditional MAB.

BOHB This method takes an adaptive strategy to conduct HPO (Falkner, Klein, and Hutter 2018).

In addition, to investigate its practicability and the empirical performance in the AutoML systems, we also take the following AutoML systems into account:

Auto-Sklearn (ASK) The state-of-the-art AutoML system. It utilizes the BO based solution – SMAC to solve the CASH problem.

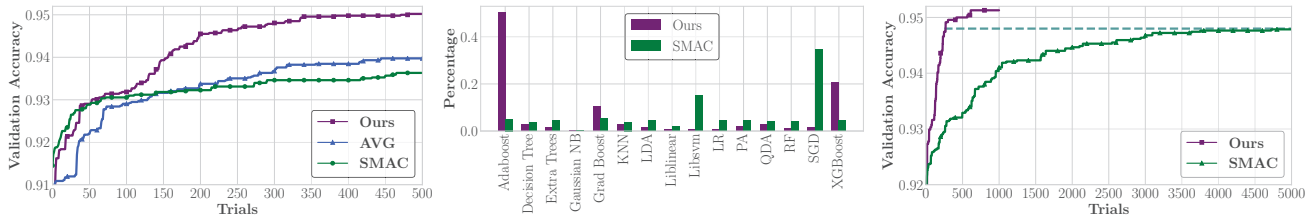


Figure 2: Performance comparison between BO based solutions and the proposed method on PC4 dataset.

Hyperopt-Sklearn (HPSK) Similar to Auto-Sklearn, it also adopts the BO based solution, and it uses TPE to conduct HPO instead.

TPOT It leverages the genetic algorithm to solve CASH.

CASH space, Datasets and Basic Settings In all experiments, the optimization space of the CASH problem is the same as the one in Auto-Sklearn. It comprises 16 ML classification algorithms with 78 hyperparameters. More details about the space can be found in Appendix B of the supplemental materials. We considered 30 classification datasets from the OpenML repositories. These datasets are widely used in the related works (Feurer et al. 2015; Efimova, Filchenkov, and Shalamos 2017; Olson and Moore 2019; Liu et al. 2019), and the details are listed in Appendix C. For each run, the original dataset will be partition into three subsets: training set, validation set and test set, in the proportion of 64%, 16%, 20%. Accuracy is used as the metric of the objective. We repeated each method 10 times on each dataset and reported the average accuracy. For the sake of fairness, we assured that all compared methods use the data with the same preprocessing operations. That is, we processed the raw datasets with the necessary operations only (e.g., label encoder, one-hot encoding); and we disabled the original preprocessing module in Auto-Sklearn and TPOT. Like Auto-Sklearn and Auto-Weka, the proposed method leverages SMAC to solve the HPO problem for each ML algorithm individually. In the following experiments, we used the initial version of our method (in Algorithm 1) by default (except when specified the concrete version). The parameter C for computing the smooth growth rate is set to 7. Our method is not sensitive to the choice of C , and the detailed sensitivity analysis can be found in Appendix D.

More Results about the Concave Rewards We ran experiments on 5 datasets, and analyzed the reward functions for different ML algorithms. Ten figures in the supplementary materials illustrate the rewards functions for each algorithm in details. We found that the concave behavior about the reward function is largely consistent with the result we showed in Figure 1.

Comparison with BO based Methods

The empirical evaluation of BO methods shows that SMAC performs best on the benchmarks with the high-dimensional hyperparameter space, closely followed by TPE. In this experiment, we evaluated the performance of both the proposed method and SMAC on the CASH problem.

High-dimensional Hyperparameter Space. Here we demonstrated that the proposed method still works well when the hyperparameter space becomes large. We evaluated the following three methods on OpenML *PC4* dataset with 500 trials: average policy (AVG), SMAC and our approach (OURS). The hyperparameter space of CASH problem is gradually augmented by adding more and more ML algorithms into the algorithm candidate \mathcal{A} with $|\mathcal{A}| = K$. The performance of each method is tested with different K s: $K = [1, 2, 4, 8, 12, 16]$. When K equals to 1, the hyperparameter space only includes the hyperparameters of the optimal algorithm; if K is set to 16, the hyperparameter space contains the hyperparameters of all ML algorithms and the algorithm selection hyperparameter. As illustrated in Table 1, SMAC suffers from the low-efficiency issue. With the increase of K , it is infeasible for BO methods to learn a surrogate model that models this huge optimization space accurately within 500 trials. Consequently, the validation accuracy drops from 95.02% to 93.63%. In contrast, the proposed method utilizes the elimination criterion to dismiss the poor-performing algorithms from the candidate set, thus decreasing the dimension of hyperparameter space automatically. Hence our method still can achieve the best accuracy - 95.02% when K equals to 16.

K	AVG	SMAC	OURS
1	95.02	95.02	95.02
2	94.68	94.79	95.01
4	94.31	94.06	95.02
8	93.91	93.60	95.02
12	93.50	93.48	95.01
16	93.39	93.63	95.02

Table 1: The validation accuracy (%) with different K s in the CASH problem.

Resource Allocation Figure 2 (a) depicts the validation accuracy of three methods across trials, where 500 trials are used to solve the CASH problem with $K = 16$. In the first 100 trials, SMAC and the proposed method behave similarly, and both of them explore the performance distribution over the optimization space. Then our method starts to identify and dismiss the poor-performing algorithms by leveraging the known HPO results. More resources (trials) are allocated to the more promising algorithms, and this procedure brings significant performance improvement. Due to the huge hyperparameter space, SMAC cannot model the performance for each ML algorithm effectively. Therefore,

Dataset ID	Validation Performance (%)						Test Performance (%)					
	TPE	SMAC	UCB	CMAB	SFMX	OURS	TPE	SMAC	UCB	CMAB	SFMX	OURS
1049	94.02	93.85	94.27	94.20	94.10	95.26	90.42	90.64	90.75	90.92	90.98	91.13
917	94.62	95.00	94.38	94.81	94.19	95.00	84.35	84.25	84.50	84.50	84.35	85.40
847	87.42	87.41	87.43	87.39	87.38	87.49	86.27	86.23	86.20	86.20	86.36	86.30
54	86.10	85.96	86.03	86.03	85.81	86.18	86.00	85.7	86.00	86.06	86.06	86.47
31	79.88	79.94	79.81	79.94	80.06	80.06	72.95	73.65	73.45	73.45	74.35	74.35
181	57.18	56.93	57.02	56.72	56.85	57.23	60.03	59.93	59.83	59.33	59.56	59.63
40670	97.76	97.73	97.90	97.98	97.84	98.10	96.55	96.60	96.63	96.69	96.68	96.77
40984	99.20	99.16	99.19	99.22	99.08	99.24	96.80	97.25	96.80	97.14	97.23	97.14
46	97.63	97.48	97.24	97.32	97.36	97.44	95.44	95.27	95.11	95.56	95.44	95.44
772	60.95	60.20	60.49	61.03	60.46	61.20	53.19	53.76	54.06	54.36	54.01	53.85
310	99.00	98.97	98.97	98.98	99.00	99.02	98.67	98.71	98.75	98.65	98.67	98.71
40691	70.76	70.66	71.05	71.02	70.74	71.95	66.50	66.09	65.03	65.97	66.00	66.66
1501	95.25	95.22	94.98	94.86	95.02	95.33	96.71	95.49	96.43	96.30	96.43	96.80
1557	67.49	67.52	67.37	67.58	67.22	67.85	61.71	62.05	62.09	61.99	61.99	62.68
182	91.99	92.14	92.03	91.95	91.90	92.04	91.33	91.40	91.25	91.52	91.32	91.50
823	98.53	98.50	98.56	98.54	98.55	98.60	98.08	98.04	98.01	98.03	98.04	98.10
1116	99.75	99.73	99.72	99.51	99.72	99.87	99.36	98.98	99.36	99.44	99.44	99.50
151	93.51	93.41	93.28	93.42	93.31	94.01	93.31	93.32	93.26	93.27	93.08	93.95
1430	85.72	85.69	85.75	85.62	85.69	85.85	85.09	85.02	85.13	85.01	85.06	85.17
32	99.55	99.53	99.45	99.42	99.47	99.63	99.30	99.25	99.55	99.41	99.34	99.60
354	84.80	84.95	79.18	80.80	79.06	87.93	85.00	80.87	80.98	79.12	79.37	87.99
60	86.81	86.88	86.74	86.65	86.76	86.90	86.54	86.52	86.55	86.44	86.28	86.65
846	90.14	90.12	90.15	90.05	90.16	90.19	89.01	89.00	88.74	88.90	89.04	89.07
28	98.85	98.81	98.77	98.59	98.78	98.87	98.84	98.73	98.84	98.84	98.81	98.85
1471	97.99	97.93	97.84	97.50	97.93	98.28	97.75	97.38	98.08	97.83	97.74	97.61
9976	87.02	87.02	86.54	86.97	85.82	86.83	85.85	86.62	85.65	85.46	85.58	86.60
23512	72.96	73.12	72.96	72.80	72.90	73.20	72.60	72.29	72.55	72.46	72.60	72.86
41082	97.89	97.74	97.65	97.10	97.74	98.10	97.54	97.10	97.56	97.54	97.55	97.62
389	87.73	86.60	86.80	86.66	86.60	87.70	87.56	86.37	86.98	87.22	87.38	87.51
184	89.33	89.12	89.23	89.17	89.19	89.65	88.34	88.20	88.21	88.18	88.22	88.78

Table 2: Average validation accuracy and test accuracy for all considered methods on 30 OpenML datasets.

its performance improves very slowly, and it is even worse than the average policy. To further compare our method with SMAC, Figure 2 (b) illustrates their percentages of the HPO trials for each ML algorithm respectively. In this problem (dataset), Adaboost is the optimal algorithm. As can be seen, our method identifies and terminates 13 unpromising ML algorithms by using 20% trials. Another 30% of trials are used to dismiss the left two algorithms that have a near-optimal performance. Almost 50% of trials are spent on tuning the optimal algorithm – Adaboost. In contrast, most of the trials in SMAC are used to tune the poor-performing algorithms.

Speedups We evaluated the achievable speedups of our method against the baseline - SMAC on 10 OpenML datasets. Continued with the previous settings, 5000 trials in total are given to SMAC. The speedup is measured with the number of trials (#) that each method needs to reach the same validation accuracy (%). Table 3 depicts the speedup results. As can be seen, our method is more efficient than SMAC in terms of the number of trials one needs to reach the same validation accuracy. To derive a more clear illustration about this, we plotted the validation accuracy curve of these two methods across trials on the PC4 dataset. As shown in Figure 2 (c), the final validation accuracy of SMAC is still worse than the one that our approach achieves within 250 trials. The empirical results demonstrate that the proposed method can outperform the existing CASH algorithm - SMAC by over an order of magnitude speedups.

Dataset ID	Val Acc	#SMAC	#OURS	Speedups
1049	94.81	5000	250	20.0x
40691	71.38	5000	395	12.7x
40670	97.86	5000	230	21.7x
847	87.48	5000	480	10.4x
32	99.61	5000	450	11.1x
151	93.94	5000	350	14.3x
184	89.63	4000	500	8.00x
354	87.53	5000	427	11.7x
1471	98.20	5000	500	10.0x
41082	98.10	3000	500	6.20x
Average	-	-	-	12.6x

Table 3: Speedup results on 10 OpenML datasets.

Comparison with All Considered Methods

In this experiment, we compared the proposed method with all considered baselines in terms of two perspectives: 1) final accuracy, and 2) the efficiency, i.e., the number of trials one needs to reach the same validation accuracy. In the first part, each method is given 500 trials, and the average accuracy across 10 runs is reported. Table 2 lists both the average validation accuracy and the average test accuracy on 30 OpenML datasets. In order to evaluate the generalization of the corresponding model, we also compared the accuracy on the test set. As can be seen, the proposed method achieves the best validation accuracy on 26 out of 30 datasets, and it also reaches the highest test accuracy on 20 out of 30 datasets. This gives that the ML models obtained by our method generalize well. Although our method does not get

Dataset ID	Heidari et al	BOHB	OURS	Speedups against BOHB
1049	94.26	94.31	95.25	8.0x
40691	71.05	71.06	71.95	7.5x
40670	97.82	97.79	98.10	8.6x
847	87.35	87.42	87.49	2.4x
32	99.42	99.52	99.63	3.9x
151	93.25	93.64	94.01	5.3x
184	89.18	89.40	89.65	4.5x
354	80.79	85.18	87.90	15.7x
1471	97.99	97.87	98.28	5.7x
41082	97.69	97.96	98.12	3.5x

Table 4: Average validation accuracy (%) and speedups compared with the considered methods.

the highest accuracy on a few datasets, its result is very close to the best one. It is worth noting that, on most datasets, our method outperforms both the existing bandit-based methods (CMAB, UCB, and Softmax) and BO-based methods in terms of the final accuracy in solving the CASH problem.

In the second part, we took another two related works into consideration: Heidari et al. (Heidari, Kearns, and Roth 2016) and BOHB (Falkner, Klein, and Hutter 2018). First we ran these two methods on 10 datasets with 500 trials, and the result is reported in Table 4. Although Heidari et al. (2016) leverage the concave reward function, this method cannot outperform the solution found by our approach because it tries to maximize the accumulated rewards. As mentioned previously, the objective in CASH focuses more on identifying the optimal arm, instead of optimizing the accumulated rewards. Similar to our approach, BOHB adopts an adaptive mechanism to conduct hyperparameter optimization. The reason why this method cannot beat our method is that it does not use the structure information about the concave rewards in CASH. By contrast, our method, with the Rising Bandits, absorbs the advantages of these two kinds of methods, and avoids their drawbacks successfully. Furthermore, similar to the last section about speedups, we gave the baseline - BOHB enough trials, enabling it to reach the same validation accuracy that our method gets within 500 trials (that is, the fourth column in Table 4). Finally, we obtained the speedups against BOHB, and illustrated the result in Table 4. It exhibits that the CASH-oriented Rising Bandits are more efficient than the existing adaptive method in solving the CASH problem.

Comparison with AutoML Systems

To investigate the practicality and effectiveness of our method in the AutoML systems, we implemented the proposed method based on the components of Auto-Sklearn and compared it with three popular AutoML systems. Each system is given 2 hours, and the average test accuracy across 10 runs is reported. The cost-aware variant of our method is used to solve the CASH problems. Because the three AutoML systems do not take the evaluation cost into account, they only optimize the performance, instead of optimizing both efficiency and performance together. As a result, given a limited time, these AutoML systems suffer from the low-efficiency problem. The empirical results in Table 5 demonstrate that the proposed method is more efficient than the existing AutoML systems on the 12 OpenML datasets.

Dataset	ASK	HPSK	TPOT	OURS
AMAZON	72.33	73.67	75.45	82.60
POKER	84.91	84.83	81.59	85.92
WINE	65.69	65.61	65.54	66.76
FBIS-WC	86.17	86.21	86.61	87.30
OPTDIGITS	98.79	98.78	99.16	99.10
SEMEION	96.55	96.57	96.36	96.99
HIGGS	71.98	71.81	71.58	72.20
PC4	91.16	91.07	90.94	91.21
USPS	96.42	96.57	97.47	97.66
MUSK	99.29	99.20	99.63	99.73
ELEVATORS	88.64	88.71	88.86	89.01
ELECTRICITY	93.11	92.98	90.16	93.84

Table 5: Average test accuracy (%) of compared AutoML systems on 12 OpenML datasets.

Conclusion

In this paper, we proposed an alternating optimization framework to accelerate the CASH problem, where a novel MAB variant is introduced to conduct algorithm selection and the Bayesian optimization methods are used to conduct HPO for each ML algorithm individually. Moreover, we presented an online algorithm to solve the Rising Bandits problem with provably theoretical guarantees. We evaluated the performance of the proposed method on a number of AutoML tasks and demonstrated its superiority over the competitive baselines. In the future work, we plan to leverage the meta-learning techniques to speed up the CASH problem.

Acknowledgments

This work is supported by the National Key Research and Development Program of China (No.2018YFB1004403), NSFC (No.61832001, 61702015, 61702016, 61572039), Beijing Academy of Artificial Intelligence (BAAI), and Alibaba-PKU joint program. Jiawei Jiang is the corresponding author.

References

- Abdulrahman, S. M.; Brazdil, P.; Van Rijn, J. N.; and Vanschoren, J. 2015. Algorithm selection via meta-learning and sample-based active testing. *Algorithm Selection Workshop ECMLPKDD*.
- Bergstra, J. S.; Bardenet, R.; Bengio, Y.; and Kégl, B. 2011. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, 2546–2554.
- Besbes, O.; Gur, Y.; and Zeevi, A. 2014. Stochastic multi-armed-bandit problem with non-stationary rewards. In *Advances in neural information processing systems*, 199–207.
- Efimova, V.; Filchenkov, A.; and Shalammov, V. 2017. Fast automated selection of learning algorithm and its hyperparameters by reinforcement learning. In *International Conference on Machine Learning AutoML Workshop*.
- Eggenberger, K.; Feurer, M.; Hutter, F.; Bergstra, J.; Snoek, J.; Hoos, H.; and Leyton-Brown, K. 2013. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, volume 10, 3.

- Falkner, S.; Klein, A.; and Hutter, F. 2018. Bohb: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, 1436–1445.
- Feurer, M.; Klein, A.; Eggenberger, K.; Springenberg, J.; Blum, M.; and Hutter, F. 2015. Efficient and robust automated machine learning. In *Advances in neural information processing systems*, 2962–2970.
- Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep learning*. MIT press.
- He, X.; Liao, L.; Zhang, H.; Nie, L.; Hu, X.; and Chua, T.-S. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, 173–182.
- Heidari, H.; Kearns, M. J.; and Roth, A. 2016. Tight policy regret bounds for improving and decaying bandits. In *IJCAI*, 1562–1570.
- Ho, Y.-C., and Pevney, D. L. 2001. Simple explanation of the no free lunch theorem of optimization. In *Proceedings of the 40th IEEE Conference on Decision and Control (Cat. No. 01CH37228)*, volume 5, 4409–4414. IEEE.
- Hu, Y.-Q.; Yu, Y.; Tu, W.-W.; Yang, Q.; Chen, Y.; and Dai, W. 2019. Multi-fidelity automatic hyper-parameter tuning via transfer series expansion. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*.
- Huang, S.; Wang, C.; Ding, B.; and Chaudhuri, S. 2019. Efficient identification of approximate best configuration of training in large datasets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 3862–3869.
- Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2011. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, 507–523. Springer.
- Jamieson, K., and Talwalkar, A. 2016. Non-stochastic best arm identification and hyperparameter optimization. In *AISTATS*, 240–248.
- Jiang, J.; Jiang, J.; Cui, B.; and Zhang, C. 2017. Tencentboost: a gradient boosting tree system with parameter server. In *2017 IEEE 33rd ICDE*, 281–284. IEEE.
- Jiang, J.; Cui, B.; Zhang, C.; and Fu, F. 2018. Dimboost: Boosting gradient boosting decision tree to higher dimensions. In *Proceedings of the 2018 International Conference on Management of Data*, 1363–1376. ACM.
- Kandasamy, K.; Dasarthy, G.; Schneider, J.; and Póczos, B. 2017. Multi-fidelity bayesian optimisation with continuous approximations. In *ICML*, 1799–1808.
- Klein, A.; Falkner, S.; Bartels, S.; Hennig, P.; and Hutter, F. 2017. Fast bayesian optimization of machine learning hyperparameters on large datasets. In *AISTATS*, 528–536.
- Komer, B.; Bergstra, J.; and Eliasmith, C. 2014. Hyperoptsklearn: automatic hyperparameter configuration for scikit-learn. In *ICML workshop on AutoML*, volume 9. Citeseer.
- Kotthoff, L.; Thornton, C.; Hoos, H. H.; Hutter, F.; and Leyton-Brown, K. 2017. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *The Journal of Machine Learning Research* 18(1):826–830.
- Levine, N.; Crammer, K.; and Mannor, S. 2017. Rotting bandits. In *Advances in NIPS*, 3074–3083.
- Liu, S.; Ram, P.; Bouneffouf, D.; Bramble, G.; Conn, A. R.; Samulowitz, H.; and Gray, A. G. 2019. Automated machine learning via ADMM. *CoRR* abs/1905.00424.
- Ma, J.; Wen, J.; Zhong, M.; Chen, W.; and Li, X. 2019. Mmm: Multi-source multi-net micro-video recommendation with clustered hidden item representation learning. *Data Science and Engineering* 4(3):240–253.
- Olson, R. S., and Moore, J. H. 2019. Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Automated Machine Learning*. Springer. 151–160.
- Poloczek, M.; Wang, J.; and Frazier, P. 2017. Multi-information source optimization. In *Advances in Neural Information Processing Systems*, 4288–4298.
- Quanming, Y.; Mengshuo, W.; Hugo, J. E.; Isabelle, G.; Yi-Qi, H.; Yu-Feng, L.; Wei-Wei, T.; Qiang, Y.; and Yang, Y. 2018. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306*.
- Russo, D. J.; Van Roy, B.; Kazerouni, A.; Osband, I.; Wen, Z.; et al. 2018. A tutorial on thompson sampling. *Foundations and Trends® in Machine Learning* 11(1):1–96.
- Sabharwal, A.; Samulowitz, H.; and Tesauro, G. 2016. Selecting near-optimal learners via incremental data allocation. In *Thirtieth AAAI Conference on Artificial Intelligence*.
- Shahriari, B.; Swersky, K.; Wang, Z.; Adams, R. P.; and De Freitas, N. 2015. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE* 104(1):148–175.
- Snoek, J.; Larochelle, H.; and Adams, R. P. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in NIPS*, 2951–2959.
- Sun, Q., and Pfahringer, B. 2013. Pairwise meta-rules for better meta-learning-based algorithm ranking. *Machine learning* 93(1):141–161.
- Sutton, R. S., and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.
- Swersky, K.; Snoek, J.; and Adams, R. P. 2013. Multi-task bayesian optimization. In *Advances in neural information processing systems*, 2004–2012.
- Wang, Z.; Zoghi, M.; Hutter, F.; Matheson, D.; and De Freitas, N. 2013. Bayesian optimization in high dimensions via random embeddings. In *Twenty-Third IJCAI*.
- Zhao, Y.; Shen, Y.; and Huang, Y. 2019. Dmdp: A dynamic multi-source default probability prediction framework. *Data Science and Engineering* 4(1):3–13.
- Zöller, M., and Huber, M. F. 2019. Survey on automated machine learning. *CoRR* abs/1904.12054.