# Efficient Identification of Approximate Best Configuration of Training in Large Datasets

**Silu Huang,[1*] Chi Wang,[2] Bolin Ding,[3†] Surajit Chaudhuri[2]**
[1]University of Illinois, Urbana-Champaign, IL
[2]Microsoft Research, Redmond, WA
[3]Alibaba Group, Bellevue, WA
shuang86@illinois.edu, {wang.chi, surajitc}@microsoft.com, bolin.ding@alibaba-inc.com

## Abstract

A configuration of training refers to the combinations of feature engineering, learner, and its associated hyperparameters. Given a set of configurations and a large dataset randomly split into training and testing set, we study how to efficiently identify the best configuration with approximately the highest testing accuracy when trained from the training set. To guarantee small accuracy loss, we develop a solution using confidence interval (CI)-based progressive sampling and pruning strategy. Compared to using full data to find the exact best configuration, our solution achieves more than two orders of magnitude speedup, while the returned top configuration has identical or close test accuracy.

## Introduction

Increasing the productivity of data scientists has been a target for many machine learning service providers, such as Azure ML, DataRobot, Google Cloud ML, and AWS ML. For a new predictive task, a data scientist usually spends a vast amount of time to train a good ML solution. A proper *configuration*, i.e., the combination of preprocessing, feature engineering, learner (i.e., training algorithm) and the associated hyperparameters, is critical to achieving good performance. It usually takes tens or hundreds of trials to select a suitable configuration.

There are AutoML tools like auto-sklearn (Feurer et al. 2015) to automate these trials, and output a configuration with highest evaluated performance. However, both the manual and AutoML approaches have become increasingly inefficient as the available ML data volume grows to millions or more. Even the trial for a single configuration can take hours or days for such large-scale datasets. Motivated by this efficiency issue, we propose a module called *approximate best configuration* (ABC). Given a set of configurations, it outputs the approximate best configuration, such that the accuracy loss to the best configuration is below a threshold. Our goal is to *efficiently* identify the approximate best configuration.

The intuition behind ABC is that the ML model trained over a sampled dataset can be used to approximate the model trained over the full dataset. However, the optimal sample size to determine the best configuration up to an accuracy loss threshold is unknown. We develop a novel confidence interval (CI)-based progressive sampling and pruning solution, by addressing two questions: *(a) CI estimator:* given a sampled training dataset, how to estimate the confidence interval of a configuration's real performance with full training data? *(b) scheduler:* as the optimal sample size is unknown *a priori*, how to allocate appropriate sample size for each configuration?

Our contributions are summarized as the following.

- We develop an ABC framework using progressive sampling and CI-based pruning. It ensures finding an approximate best configuration while reducing the running time.
- We present and prove bounds for the real test accuracy when the ML model is trained using full data, based on the model trained with sampled data.
- Within ABC, we design an approximately optimal scheduling scheme based on the confidence interval, for allocating sample size among different configurations.
- We conduct experiments with large datasets. We demonstrate that our ABC solution is tens to hundreds of times faster, while returning top configurations with no more than 1% accuracy loss.

## Problem Formulation

**Notions and Notations.** In this paper, we focus on classification tasks with a large set of labeled data $\mathcal{D}$. In order for reliable evaluation of a trained classifier, data scientists usually split the available data *randomly* into training and testing set $\mathcal{D}_{tr}$ and $\mathcal{D}_{te}$. After that, they specify a number of configurations of the ML workflow and try to identify the best configuration. Let $\mathcal{C}$ be the candidate configuration set and $C_i$ be the $i^{th}$ configuration in $\mathcal{C}$. We further let $n$ be the number of configurations, i.e., $n = |\mathcal{C}|$. Using terminology from learning theory, each configuration $C_i$ defines a *hypothesis space* $\mathcal{H}_i$, where each *hypothesis* $H \in \mathcal{H}_i$ is a possible classifier trained under this configuration. Given a training dataset $\mathcal{D}_{tr}$, the learner in $C_i$ will output a hypothesis $H^i_{tr} \in \mathcal{H}_i$ as the trained classifier. The quality of the classifier is measured against the heldout testing data $\mathcal{D}_{te}$. In this paper, we focus on *accuracy* as the quality metric. We denote the accuracy of hypothesis $H \in \mathcal{H}_i$ on dataset $D$ as

[*]Work done while visiting Microsoft Research
[†]Work done while working in Microsoft Research

| | $\lvert\mathcal{D}\rvert$ | $\lvert\mathcal{F}\rvert$ | Origin |
|---|---|---|---|
| Twitter | 1.4M | 9866 | Twitter, Stanford |
| FlightDelay | 7.3M | 630 | U.S. Department of Transportation |
| NYCTaxi | 10M | 21 | NYC Taxi & Limousine Commission |
| HEPMASS | 10M | 28 | UCI |
| HIGGS | 10.6M | 28 | UCI |

Table 1: Dataset Description

$\mathcal{A}(H, D)$. In particular, given a configuration $C_i$, we define its *real test accuracy* as $\mathcal{A}_i = \mathcal{A}(H^i_{tr}, \mathcal{D}_{te})$.

**Problem Definition.** A standard practice to select the best configuration from a configuration set $\mathcal{C}$ is to train with each configuration using full training data, and then pick the one with the highest test accuracy, i.e., $C_{i^*} = \arg\max_{C_i \in \mathcal{C}}\{\mathcal{A}_i\}$. Note that an implicit assumption made here is that the returned classifier with full training data has equal or higher test accuracy than the classifier trained with sampled training data. We call this *exploitativeness* assumption and follow it in this paper. From a user's perspective, if there are multiple configurations with nearly identical highest real test accuracy, then it would suffice to return any of them as the best configuration. So we introduce a new problem *approximate best configuration* identification, as formalized in Problem 1.

**Problem 1** (Approximate Best Configuration Identification)**.** *Given a configuration candidate set $\mathcal{C}$ and an accuracy loss tolerance $\epsilon$, identify a configuration $C_{i'}$ whose real test accuracy is within $\epsilon$ away from that of the best configuration $C_{i^*}$, i.e., $\mathcal{A}_{i^*} - \mathcal{A}_{i'} \leq \epsilon$, and minimize the total running time.*

## CI-based Framework

Before introducing our framework, we first describe some insights based on simple observations. We experiment on the *FlightDelay* dataset published in Azure Machine Learning gallery (Mund 2015) with five learners (as five configurations). Readers can refer to Table 1 for detailed statistics of this dataset, where $\lvert\mathcal{D}\rvert$ and $\lvert\mathcal{F}\rvert$ is the number of records and features respectively. The *learning curve* for each configuration is depicted in Figure 1, where x-axis is the training sample size in log-scale and y-axis is the test accuracy on $\mathcal{D}_{te}$. In general, the test accuracy approaches the real test accuracy with the increase of the training sample size. When the sample size is large enough (>2M), the configuration with the highest test accuracy is LightGBM – the true best configuration.

Furthermore, the optimal sample size to minimize the running time could vary for different configurations. If we magically know that we should use 2M training samples for LightGBM and 16K training samples for all the other configurations, we can save even more time and still identify the correct best configuration. Unfortunately, the optimal sample size for each configuration is unknown. A natural idea is to increase the sample size gradually, until a plateau is reached in the learning curve. However, a naive plateau estimator based on the learning curve is error-prone. As shown from

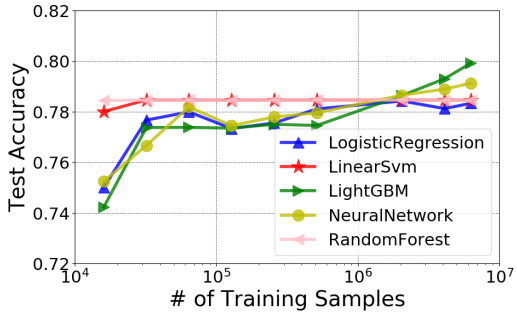



Figure 1: Learning Curve

Figure 1, LightGBM's learning curve is flat from 32K to 128K. If we stop increasing the sample size for it, it will be mis-pruned. Therefore, a more robust strategy is needed.

**Overview.** The main idea is to estimate the *confidence interval* (CI) of each configuration's real test accuracy with sampled data, instead of simply using a point estimation of the real test accuracy. In each round, we train the classifier for a selected configuration on some sampled training data. We call such a round of training a *probe*. After a probe, we update the confidence interval for the configuration. As the sample size increases, the confidence interval shrinks, and the badly-performing configurations can be pruned based on the CIs. The pruning based on CI is more robust than based on random observations from the learning curve.

**Algorithm 1:** ABC

1 **Input:** configuration set $\mathcal{C}$, accuracy loss threshold $\epsilon$;
2 **Output:** the approximate best configuration;
3 Initialization: $C_{prob} \leftarrow C_1$, $C_{i'} \leftarrow C_1$, $\Omega \leftarrow \mathcal{C}$;
4 **while** $|\Omega| > 1$ **do**
5  PROBE($C_{prob}$);
6  $[C_{prob}.l, C_{prob}.u] \leftarrow$CIESTIMATOR ($C_{prob}$) ;
7  **if** $C_{prob}.l > C_{i'}.l$ **then** $C_{i'} \leftarrow C_{prob}$;
8  **for** $C \in \Omega$ **do** // pruning
9  **if** $C.u - C_{i'}.l \leq \epsilon$ **then** $\Omega \leftarrow \Omega - C$;
10  **if** *Pruning happens* **then**
11  **for** $C \in \Omega$ **do**
12  $C.u_{old} \leftarrow C.u$; $C.l_{old} \leftarrow C.l$;
13  $C_{prob} \leftarrow$ SCHEDULER ($\Omega$)
14 **return** $C_{i'}$;

**Detailed Algorithm.** ABC proceeds round by round as shown in Algorithm 1, where each configuration $C_i$ is annotated with its current sample size ($C_i.s$), current lower bound ($C_i.l$), current upper bound ($C_i.u$), and the snapshot of lower bound ($C_i.l_{old}$) and upper bound ($C_i.u_{old}$) in the last pruning round. In each round within the while loop (line 4), it first probes the configuration $C_{prob}$ (line 5). Then it calls a CIESTIMATOR subroutine to quickly estimate the confidence interval for $\mathcal{A}_{prob}$ (line 6). Next, it prunes badly-performing configurations (line 7-9). Line 7 identifies the configuration $C_{i'}$ with the largest lower bound. Line 8-9 prunes an configuration if its upper bound is within $\epsilon$ away from the largest

lower bound. If there exists configuration being pruned (line 10), we call this iteration a *snapshot* and will update $C.l_{old}$ and $C.u_{old}$ for each configuration $C$ in this snapshot (line 11-12). At last, it calls a SCHEDULER subroutine to determine which configuration to probe next as well as its sample size (line 10).

We describe CIESTIMATOR and SCHEDULER in the next two sections.

## CI Estimator

In this section, we will derive a CIESTIMATOR for each configuration's real test accuracy, based on the probe over sampled data. For configuration $C_i$, the confidence interval $[l_i, u_i]$ needs to contain the real test accuracy $\mathcal{A}_i$ with high probability. The computation of $l_i$ and $u_i$ needs to be efficient, i.e., no slower than the probe. In the following, we assume $i$ is fixed and omit it in the notations.

At the first glance, the CI estimation may remind readers of the generalization error bounds (e.g., VC-bound). The generalization error bound is a universal bound of the difference between each hypothesis's accuracy in training data and its accuracy in infinite data following the same distribution. Nevertheless, the confidence interval we need is the range of the real test accuracy of the hypothesis $H_{tr}$ trained from full training data, while we only have the hypothesis $H_{S_{tr}}$ trained from a sample $S_{tr} \subset D_{tr}$. Therefore, we cannot apply generalization error bound to obtain our confidence interval.

We use Figure 2 to summarize the notations and their relationships which are important for understanding the theoretical results. $H_{tr}$, $H_{S_{tr}}$, and $H^*$ correspond to the returned hypothesis after training a fixed configuration with full training dataset $\mathcal{D}_{tr}$, the sampled training dataset $S_{tr}$, and the full data $\mathcal{D}$ respectively. For instance, Figure 2(a) shows the overall derivation relationships among $\mathcal{D}$, $\mathcal{D}_{tr}$, $\mathcal{D}_{te}$, $S_{tr}$, $S_{te}$, and $H_{S_{tr}}$. First, the full training data $\mathcal{D}_{tr}$ and the full testing data $\mathcal{D}_{te}$ are randomly split from the whole data $\mathcal{D}$. Second, the sampled training data $S_{tr}$ and the sampled testing data $S_{te}$ are randomly drawn from the full training data $\mathcal{D}_{tr}$ and full testing data $\mathcal{D}_{te}$. Last, $H_{S_{tr}}$ is trained from the sampled training data $S_{tr}$. Note that the CI estimator only has access to $H_{S_{tr}}$, $S_{tr}$ and $S_{te}$. Though $H_{tr}$ and $H^*$ are not accessible, they are useful in our analysis.

**Upper bound.** The intuition behind the confidence interval estimation is that we need to relate the two hypotheses $H_{S_{tr}}$ and $H_{tr}$, and use the information we have on $H_{S_{tr}}$ to infer the performance of $H_{tr}$. To upper bound the accuracy of $H_{tr}$, we leverage a *fitness* condition: the training process produces a hypothesis that fits the training data. When the configuration is fixed, the accuracy in a dataset $D$ of the hypothesis trained on $D$ should be no worse than the hypothesis trained on a different dataset $D' \neq D$. It is the only assumption we need to prove the upper bound, no matter what training algorithm is used. Under this condition, we found an inequality chain to connect the training accuracy $\mathcal{A}(H_{S_{tr}}, S_{tr})$ to the real test accuracy of $H_{tr}$.

**Theorem 1** (Upper Bound)**.** *Under the fitness condition, with probability at least* $1 - \frac{\delta}{2n^2}$, $\mathcal{A}(H_{tr}, \mathcal{D}_{te}) \leq u$, *where*

$$u \triangleq \mathcal{A}(H_{S_{tr}}, S_{tr}) + (\frac{1}{2|S_{tr}|} \ln \frac{4n^2}{\delta})^{\frac{1}{2}} + (\frac{1}{2|\mathcal{D}_{te}|} \ln \frac{4n^2}{\delta})^{\frac{1}{2}}$$

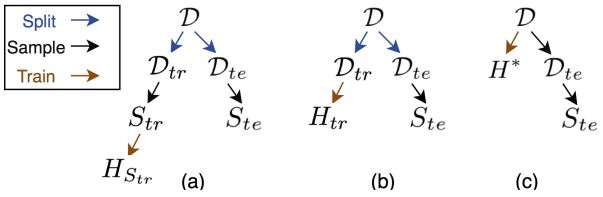



Figure 2: Notations Used in CI Estimation and Analysis

*Proof.* Let us first recall the fitness condition. Given a fixed configuration $C_i$, let $H$ and $H'$ be the hypothesis returned by training on two different sample sets $D$ and $D'$, respectively. Note that $H$ and $H'$ are both from the same fixed hypothesis space $\mathcal{H}_i$. Our assumption is that $H$ *has no lower accuracy on* $D$ *than* $H'$. Similarly, $H'$ *has no lower accuracy on* $D'$ *than* $H$.

First, let us break down $\mathcal{A}(H_{tr}, \mathcal{D}_{te}) - \mathcal{A}(H_{S_{tr}}, S_{tr})$ into four clauses, as shown in Equation (1).

$$\begin{aligned}&\mathcal{A}(H_{tr}, \mathcal{D}_{te}) - \mathcal{A}(H_{S_{tr}}, S_{tr})\\ =&[\mathcal{A}(H_{tr}, \mathcal{D}_{te}) - \mathcal{A}(H^*, \mathcal{D}_{te})] + [\mathcal{A}(H^*, \mathcal{D}_{te}) - \mathcal{A}(H^*, \mathcal{D})] +\\ &\mathcal{A}(H^*, \mathcal{D}) - \mathcal{A}(H^*, S_{tr})] + [\mathcal{A}(H^*, S_{tr}) - \mathcal{A}(H_{S_{tr}}, S_{tr})]\end{aligned} \tag{1}$$

Since $\mathcal{D}_{tr}$ and $\mathcal{D}_{te}$ are randomly split from $\mathcal{D}$, we have $\mathcal{D} = \mathcal{D}_{tr} \cup \mathcal{D}_{te}$. Let $x = \frac{|\mathcal{D}_{te}|}{|\mathcal{D}|}$ be the hold-out ratio. For any $H \in \mathcal{H}$, we have:

$$\begin{aligned}&x\mathcal{A}(H, \mathcal{D}_{te}) + (1-x)\mathcal{A}(H, \mathcal{D}_{tr}) = \mathcal{A}(H, \mathcal{D})\\ \Rightarrow\ &\mathcal{A}(H, \mathcal{D}_{te}) = \frac{1}{x}[\mathcal{A}(H, \mathcal{D}) - (1-x)\mathcal{A}(H, \mathcal{D}_{tr})]\end{aligned} \tag{2}$$

Next, apply Equation (2) to the first clause in Equation (1):

$$\begin{aligned}&\mathcal{A}(H_{tr}, \mathcal{D}_{te}) - \mathcal{A}(H^*, \mathcal{D}_{te})\\ =&\frac{1}{x}[\mathcal{A}(H_{tr}, \mathcal{D}) - \mathcal{A}(H^*, \mathcal{D})] - \frac{1-x}{x}[\mathcal{A}(H_{tr}, \mathcal{D}_{tr})\\ &- \mathcal{A}(H^*, \mathcal{D}_{tr})] \leq 0\end{aligned} \tag{3}$$

The inequality is derived from the fitness assumption (recall from Figure 2 $H_{tr}$ is trained from $\mathcal{D}_{tr}$ and $H^*$ is trained from $\mathcal{D}$).

Next, we bound the second clause in Equation (1) with Hoeffding inequality: With probability at least $(1 - \frac{\delta}{4n^2})$,

$$\mathcal{A}(H^*, \mathcal{D}_{te}) - \mathcal{A}(H^*, \mathcal{D}) \leq (\frac{1}{2|\mathcal{D}_{te}|} \ln \frac{4n^2}{\delta})^{\frac{1}{2}} \tag{4}$$

Similarly, with probability at least $(1 - \frac{\delta}{4n^2})$,

$$\mathcal{A}(H^*, \mathcal{D}) - \mathcal{A}(H^*, S_{tr}) \leq (\frac{1}{2|S_{tr}|} \ln \frac{4n^2}{\delta})^{\frac{1}{2}} \tag{5}$$

Please note that Equation (4) and (5) will not hold if we replace $H^*$ with $H_{S_{tr}}$. This is because for hypothesis $H_{S_{tr}}$,

$\mathcal{D}_{te}$ and $S_{tr}$ cannot be regarded as random samples, since $H_{S_{tr}}$ is tailored to the sample set $S_{tr}$. Therefore, introducing $H^*$ is necessary in our analysis.

Last, since $H_{S_{tr}}$ is trained from $S_{tr}$, by the fitness assumption we have:

$$\mathcal{A}(H^*, S_{tr}) - \mathcal{A}(H_{S_{tr}}, S_{tr}) \leq 0 \tag{6}$$

By substituting the four clauses in Equation (1) with Equation (3)-(6), we obtain Theorem 1 using union bound. □

Note that the computation of $\mathcal{A}(H_{S_{tr}}, S_{tr})$ is no slower than the probing (i.e., training with sampled data). In fact, the evaluation is usually much more efficient than training for the same scale of dataset.

**Lower Bound.** The lower bound is easier due to the *exploitativeness* presumption discussed in the problem formulation: Full training data produce better hypothesis than sampled training data for a fixed configuration. The real test accuracy of $H_{tr}$ can then be lower bounded by $\mathcal{A}(H_{S_{tr}}, D_{te})$. However, the computation of $\mathcal{A}(H_{S_{tr}}, D_{te})$ can be slower than probing, if $|D_{te}| \gg |S_{tr}|$. To make the CI estimation efficient, we also sample the testing data. We denote the sampled testing data as $S_{te}$. We can then lower bound $\mathcal{A}(H_{S_{tr}}, D_{te})$ by $\mathcal{A}(H_{S_{tr}}, S_{te})$ minus a variation term.

**Theorem 2** (Lower Bound)**.** *Under the exploitativeness assumption, with probability at least* $1 - \frac{\delta}{2n^2}$*,*

$$\mathcal{A}(H_{tr}, \mathcal{D}_{te}) \geq l \triangleq \mathcal{A}(H_{S_{tr}}, S_{te}) - (\frac{1}{2|S_{te}|} \ln \frac{2n^2}{\delta})^{\frac{1}{2}}$$

*Proof.* First, in the problem formulation we have assumed

$$\mathcal{A}(H_{tr}, \mathcal{D}_{te}) \geq \mathcal{A}(H_{S_{tr}}, \mathcal{D}_{te}) \tag{7}$$

Next, based on Hoeffding inequality, with probability at least $1 - \frac{\delta}{2n^2}$,

$$\mathcal{A}(H_{S_{tr}}, S_{te}) - \mathcal{A}(H_{S_{tr}}, \mathcal{D}_{te}) \leq (\frac{1}{2|S_{te}|} \ln \frac{2n^2}{\delta})^{\frac{1}{2}} \tag{8}$$

Combining Equation (7) and (8), we have $\mathcal{A}(H_{tr}, \mathcal{D}_{te}) \geq l$ with probability at least $1 - \frac{\delta}{2n^2}$. □

CIESTIMATOR **in Algorithm 1.** With Theorem 1 and 2, we can now estimate the current lower bound and upper bound of the probing configuration $C_{prob}$. As shown in Algorithm 2, we first initialize the current lower bound $C_{prob}.l$ and upper bound $C_{prob}.u$ according to Theorem 1 and 2. Furthermore, we add a constraint that the current CI must be contained in the CI of the last snapshot where pruning happens, i.e., $[C_{prob}.l, C_{prob}.u] \subset [C_{prob}.l_{old}, C_{prob}.u_{old}]$. Thus, if $C_{prob}.l < C_{prob}.l_{old}$, we replace $C_{prob}.l$ with $C_{prob}.l_{old}$ (line 4). Similar for $C_{prob}.u$ (line 5). In this way, we can guarantee that the CI for each configuration shrinks from one snapshot to another snapshot where pruning happens. Recall that $C_{prob}.l_{old}$ and $C_{prob}.u_{old}$ get updated in each snapshot.

**Algorithm 2:** CIESTIMATOR

1 **Input:** $C_{prob}, \mathcal{A}_{prob}(H_{S_{tr}}, S_{tr}), \mathcal{A}_{prob}(H_{S_{tr}}, S_{te})$ ;
2 **Output:** $[C_{prob}.l, C_{prob}.u]$;
3 $[C_{prob}.l, C_{prob}.u] \leftarrow$ Theorem 1 and 2 ;
4 **if** $C_{prob}.l < C_{prob}.l_{old}$ **then** $C_{prob}.l \leftarrow C_{prob}.l_{old}$;
5 **if** $C_{prob}.u > C_{prob}.u_{old}$ **then** $C_{prob}.u \leftarrow C_{prob}.u_{old}$;
6 **return** $[C_{prob}.l, C_{prob}.u]$ ;

## Correctness of Algorithm 1

Using Algorithm 2, we can estimate the confidence interval for the real test accuracy, based on each probe over the sampled training data $S_{tr}$ and the sampled testing data $S_{te}$. Next, Theorem 3 shows that Algorithm 1 can successfully return an approximate best configuration with high probability.

**Corollary 1** (Confidence Interval)**.** *With probability at least* $1 - \frac{\delta}{n^2}$*,* $\mathcal{A}_i \in [l_i, u_i]$.

**Theorem 3** (Correctness)**.** *With probability at least* $1 - \delta$*, Algorithm 1 returns the approximate best configuration* $C_{i'}$ *with* $\mathcal{A}_{i^*} - \mathcal{A}_{i'} \leq \epsilon$.

*Proof.* Without loss of generality, we assume $C_1$ is the returned configuration $C_{i'}$ by Algorithm 1. We denote the iterations in Algorithm 1 with at least one configuration pruned, i.e., snapshots, as set $R$. We will prove that when all the confidence intervals at iterations $R$ correctly bound the real test accuracy (denoted as event $E$), the algorithm returns correct approximate configuration. We show that when $E$ happens, for each pruned configuration $2 \leq i \leq n$, $\mathcal{A}_i \leq \mathcal{A}_1 + \epsilon$.

Consider iteration $r \in R$ and let $l_{i_r}^{(r)}$ be the the highest lower bound in this iteration, and $C_{p_r}$ be a pruned configuration in iteration $r$. When $E$ happens, $\mathcal{A}_{p_r} \leq u_{p_r}$. And $u_{p_r} \leq l_{i_r}^{(r)} + \epsilon$ according to line 9 in Algorithm 1. So the pruned configuration must satisfy $\mathcal{A}_{p_r} \leq l_{i_r}^{(r)} + \epsilon$. Furthermore, as Algorithm 1 proceeds to iteration $r+1$, $l_{i_r}^{(r)} \leq l_{i_{r+1}}^{(r+1)}$. This is because the lower bound of configuration $C_{i_r}$ does not decrease from snapshot $r$ to snapshot $r+1$ according to Algorithm 2, i.e., $l_{i_r}^{(r)} \leq l_{i_r}^{(r+1)}$.In addition, $l_{i_r}^{(r+1)} \leq l_{i_{r+1}}^{(r+1)}$ in snapshot $r+1$, according to line 7 in Algorithm 1. Thus, $\mathcal{A}_{p_r} \leq l_1 + \epsilon$ by induction on the iteration number $r$. Also, since $l_1 + \epsilon \leq \mathcal{A}_1 + \epsilon$, we have $\mathcal{A}_{p_r} \leq \mathcal{A}_1 + \epsilon$.

Next, let $\bar{E}_r$ be the event that all the confidence intervals at iteration $r$ correctly bound the real test accuracy where $r \in R$. We can decompose event $\bar{E}$ into non-overlapping sub-events $\{\bar{E}_1, \bar{E}_2|E_1, \bar{E}_3|E_2, \cdots\}$, where $\bar{E}$ (resp. $\bar{E}_r$) is the opposite event of $E$ (resp. $E_r$). According to Corollary 1, the derived confidence interval $[l_i, u_i]$ is correct with probability at least $1 - \frac{\delta}{n^2}$ for any configuration $C_i$. Hence, the probability of $\bar{E}_{r+1}|E_r$ is at most $\frac{\delta}{n}$ according to Algorithm 2 and the union bound. Furthermore, we have $n-1$ pruned configurations across all iterations, so $|R| < n$. Consequently, the probability of $\bar{E}$ is at most $\delta$ by summing up the probability of non-overlapping sub-events $\bar{E}_{r+1}|E_r$. That is, event $E$ happens with probability at least $1 - \delta$. Thus, we have $\mathcal{A}_{i^*} - \mathcal{A}_{i'} \leq \epsilon$ with probability at least $1 - \delta$. □

**Discussion.** We have used the exploitativeness assumption in deriving the lower bound: $\mathcal{A}_i(H_{S_{tr}}, D_{te}) \leq \mathcal{A}_i(H_{tr}, D_{te})$ for any $C_i$. We argue that even though this assumption is not exactly satisfied in practice, it holds closely enough to provide useful results. That is, in most cases this assumption holds, and even when this assumption is violated, we can perform a post-processing step after the algorithm finishes. If there exists $\mathcal{A}_{i'}(H_{S_{tr}}, D_{te}) > \mathcal{A}_{i'}(H_{tr}, D_{te})$ for the selected configuration $C_{i'}$, the user could use $H_{S_{tr}}$ instead of $H_{tr}$ as the final classifier. First, that satisfies users' preference in finding a more accurate classifier. Second, it holds the $\epsilon$-guarantee, because the lower bound $l_{i'}$ holds for $H_{S_{tr}}$ of configuration $C_{i'}$, and it is no lower than the pruning lower bounds $l_{i_r}, \forall r \in R$.

The correctness of our algorithm is independent of the choice of the scheduler.

## Scheduler

Now we have shown that our proposed ABC can identify the approximate best configuration with high probability. This section focuses on the optimization part in Problem 1, i.e., how to minimize the total running time. Let $T_i(s)$ be the probing time with a sampled training dataset size $s$ for configuration $C_i$, and $t_i$ be the accumulated running time for probing configuration $C_i$ in Algorithm 1. Also, let $l_i$ and $u_i$ be the lower bound and upper bound respectively for configuration $C_i$ when the algorithm terminates. With these notations, the design of SCHEDULER in ABC can be expressed as a constrained optimization problem. Without loss of generality, assume $C_1$ is returned by Algorithm 1.

**Problem 2** (Scheduling)**.** *Design a scheduler to minimize* $\mathcal{T} = \sum_i t_i$*, subject to:*

$$u_2 \leq l_1 + \epsilon, u_3 \leq l_1 + \epsilon, \ldots, u_n \leq l_1 + \epsilon$$

The objective function in Problem 2 is the time taken to identify the approximate best configuration. Since probing dominates the running time in each iteration, we use the total time of all probes as the proxy of the identification time. The constraints in Problem 2 ensure that all the configurations $C_i$ are pruned except $C_1$, and are necessary for the termination of Algorithm 1.

To solve Problem 2, we begin with studying the properties of the 'oracle' optimal scheduling scheme when it has access to $t_i$ as a function of $l_i$ and $u_i$ respectively, i.e., $t_i = f_i(l_i)$ and $t_i = g_i(u_i)$, after the samples are drawn. We claim that the optimal scheduling scheme with this oracle access probes each configuration uniquely once, since otherwise we can always reduce the total running time by only keeping the last probe. Our objective function can be rewritten as $f_1(l_1) + g_2(u_2) + \cdots + g_n(u_n)$. Furthermore, by applying the method of Lagrange multipliers, we obtain the conditions the optimal solution must satisfy:

$$\begin{cases} \frac{df_1}{dl_1} = -(\frac{dg_2}{du_2} + \cdots + \frac{dg_n}{du_n}) \\ l_1 + \epsilon = u_2 = \cdots = u_n \end{cases} \tag{9}$$

Now, since we do not have oracle access to $f_i$ and $g_i$, there is no closed-form formula to decide the optimal sample size $s_i^*$ for configuration $C_i$. To solve this challenge, We propose a scheduling scheme GRADIENTCI with two parts.

First, we use the gradient of the running time with respect to the confidence interval to determine the configuration to probe next. We depict this strategy in Algorithm 3. GRADIENTCI first sorts the remaining configuration set $\Omega$ in descending order of the upper bound (line 3), and make a guess ($\Omega_1$) on the best configuration $C_1$. Next, it compares the gradient $\frac{\Delta T_1}{\Delta l_1}$ with $|\frac{\Delta T_2}{\Delta u_2} + \cdots + \frac{\Delta T_n}{\Delta u_n}|$: If $\frac{\Delta T_1}{\Delta l_1}$ is smaller, then configuration $\Omega_1$ with the largest upper bound is picked for the next probe (line 4); otherwise, configuration $\Omega_2$ with the second largest upper bound is picked (line 5). Here, $\Delta T_i$ denotes the running time difference between the recent two consecutive probes on $C_i$, and $\frac{\Delta T_i}{\Delta l_i}$ serves as the proxy of $\frac{\Delta f_i}{\Delta l_i}$ (similar for $\frac{\Delta g_i}{\Delta u_i}$). The choice between $\Omega_1$ and others is based on the first condition in Equation (9). Intuitively, if the lower bound of $\Omega_1$ grows faster (per time spent) than all the other configurations' upper bounds' decrease, then we opt to probe $\Omega_1$. The choice of $\Omega_2$ among $\Omega_2$ to $\Omega_n$ is based on the second condition in Equation (9), towards attaining the same upper bound for them.

Second, we design the sample size sequence within each configuration. As shown in line 6 of Algorithm 3, we utilize a common trick called *geometric scheduling*, which was used in prior work to increase the sample size for a single configuration (Provost, Jensen, and Oates 1999). We further derive the closed-form for the optimal step size $c$, when $T_i(s)$ is a power function over the sample size, i.e., $T_i(s) = s^\alpha$ where $\alpha$ is a real number. The optimal step size follows $c = 2^{\frac{1}{\alpha}}$. Details can be found in our technical report (Huang et al. 2018).

**Algorithm 3:** SCHEDULER–GRADIENTCI

1 **Input:** Remaining configurations $\Omega$;
2 **Output:** Configuration for next probe $C_{prob}$;
3 sort_by_upper_bound($\Omega$);
4 **if** $\frac{\Delta T_1}{\Delta l_1} \leq |\frac{\Delta T_2}{\Delta u_2} + \cdots + \frac{\Delta T_n}{\Delta u_n}|$ **then** $C_{prob} \leftarrow \Omega_1$ ;
5 **else** $C_{prob} \leftarrow \Omega_2$ ;
6 $C_{prob}.s \leftarrow c \times C_{prob}.s$;
7 **return** $C_{prob}$ ;

**Performance Analysis.** In practice, ABC is used in two scenarios. Scenario (i): during exploration, users want to try a few configurations (e.g., verifying usefulness of a few new features) as an intermediate step. The identification result will decide the follow-up trials, but it does not serve as the final configuration, and does not require full training of $C_{i'}$. Scenario (ii): at the end of the exploration, users need to get the trained classifier corresponding to the selected configuration $C_{i'}$. In scenario (ii), the total running time involves not only the time to identify the approximate best configuration, but also the time taken to train the classifier on full data with $C_{i'}$. When $C_{i'}$ is fixed, both scenario (i) and (ii) share the same optimal scheduler. Furthermore, under certain conditions, we are able to prove a 4-approx guarantee for GRADIENTCI. Please find details in our technical report (Huang et al. 2018).

## Experiments

This section evaluates the efficiency and effectiveness of our ABC module. First, we evaluate whether ABC successfully identifies top configuration and meanwhile reduce the total running time. Second, we compare the CI-based pruning with an existing pruning algorithm based on point estimation. We also compare different scheduling schemes in our technical report (Huang et al. 2018).

### Experimental Setup

**Configurations.** We focus on the task of classifying featurized data in our evaluation. Specifically, we choose five widely used and high-performance learners: *LogisticRegression, LinearSVM, LightGBM, NeuralNetwork,* and *RandomForest*. Each classifier is associated with various hyperparameters, e.g., the number of trees in RandomForest and the penalty coefficient in LinearSVM. In total they have 29 discrete or continuous hyperparameters. In our experiments, we use random search to generate each hyperparameter value from its corresponding domain.

**Datasets.** We evaluate with five large-scale machine learning benchmarks that are publicly available. As discussed in introduction, the motivation of ABC is to handle large datasets and quickly identify the approximate best configuration. Thus, the datasets evaluated in our experiments are all at the scale of millions of records ($|\mathcal{D}|$) and with up to 10K features ($|\mathcal{F}|$). We do not use the AutoML benchmarks such as HPOlib (Eggensperger et al. 2013) or OpenML (Vanschoren et al. 2013), which mainly contain small or median-sized datasets (up to 50K records). The statistics of each dataset are depicted in Table 1. We used min-max normalization for all datasets, and n-gram extraction as well as model-based top-K feature selection for Twitter.

**Algorithms.** We compare our proposed ABC with the standard approach named *Full-run*. For each configuration, Full-run first trains the classifier with full training data, and then tests it on the full testing data. Afterwards, it returns the configuration with the highest testing accuracy. This method is supported in mature tools like scikit-learn and Azure ML. Existing approaches to best configuration identification, such as DAUB (Sabharwal, Samulowitz, and Tesauro 2016) or successive-halving (Jamieson and Talwalkar 2016), are heuristics without accuracy guarantee. Our solution and such heuristics are not apple-to-apple comparison, as they cannot ensure $\epsilon$-approximation guarantee on accuracy. Nevertheless, we conduct a best effort comparison with Successive-halving.

**Setup.** We conducted our evaluation on a VM with 8 cores and 56 GB RAM. The initial training sample size and testing sample size are 1000 and 2000 respectively. The geometry step size is set to be $c = 2$. $\epsilon = 0.01, \delta = 0.5$. Since $\delta$ is under the $\log$ term, the result is not sensitive to $\delta$. We also conduct experiments with varying $\epsilon$, as shown in our technical report (Huang et al. 2018).

We use the same set of sampled configurations for both Full-run and ABC. We vary the number of input configurations from 5 to 80. Since we focus on large datasets, it already takes a day or half to finish Full-run with 80 configurations for a single dataset. So unlike the case of small datasets, 80-100 is a realistic number because that is how many configurations a user can try with Full-run within a reasonable time.

### ABC vs. Full-run

We compare ABC against Full-run from two perspectives, running time and accuracy. We first compute the *speedup* achieved by ABC, where speedup is defined as the ratio between Full-run's total running time and ours. Next, we compare the configuration $C_{i'}$ returned by our ABC with the best configuration $C_{i^*}$ provided by Full-run in terms of real test accuracy.

**Efficiency Comparison.** As discussed, ABC is used in two scenarios in practice. During exploration, users only need the identification result to decide the follow-up trials, but do not require full training on the selected configuration. At the end of the exploration, users need to train the classifier with the selected configuration in the full data. Thus, we evaluate the running time speedup in these two scenarios: *(i)* we first compare the identification time between our ABC and Full-run as depicted in Figure 3a(i); *(ii)* we then compare the total running time including the time to train the final classifier, in Figure 3a(ii). Our solution is on average 190× faster than Full-run in scenario (i), and is on average 60× faster than Full-run in scenario (ii). Furthermore, 23 out of 25 experiments (i.e., 5 different datasets times 5 different configuration set size) has at least $|\mathcal{C}|\times$ speedup in scenario (i), and 22 out of 25 experiments achieve at least $|\mathcal{C}|\times$ speedup in scenario (ii). This means that the running time of ABC is even faster than fully evaluating one average configuration in most cases, which further means even a perfectly distributed Full-run can't beat the non-distributed ABC.

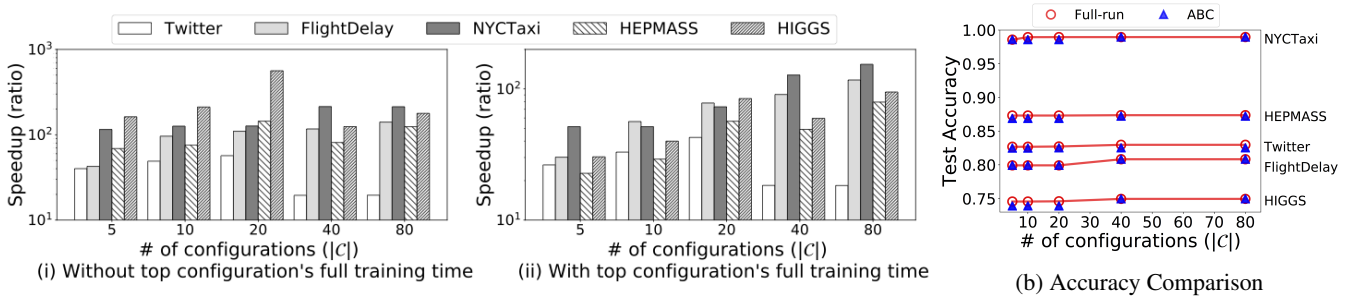
The speedup on dataset Twitter is consistently lower than other datasets. This is mainly because Twitter is one order of magnitude smaller than the other datasets. With the same sample size, the sampling ratio is higher than the other datasets, which causes lower speedup.

**Effectiveness Comparison.** As illustrated in Figure 3b, ABC successfully identifies the configuration whose real test accuracy is within 0.01 from the best configuration's real test accuracy in all of our experiments. In particular, when $|\mathcal{C}| = 40$ or 80, ABC successfully identifies the exact best configuration for FlightDelay, NYCTaxi, and HIGGS. The largest deviation is around 0.0068 when $|\mathcal{C}| = 20$ for HIGGS.

**Takeaway.** Compared to Full-run, our proposed ABC can successfully identify a competitive or identical best configuration but with much less time.

### CI-based pruning vs. Successive-halving

Next, we compare our proposed CI-based pruning with Successive-halving. Successive-halving was proposed as a pruning strategy to evaluate iterative training configurations with a resource budget of the total number of iterations of all configurations. We modify it to use the total sample size as the resource budget. In each round, it trains a classifier with the sampled data for each remaining configuration, and then eliminates the half of the low-performing configurations. This pruning is based on point estimation, i.e., they directly use $\mathcal{A}(H_{S_{tr}}, D_{te})$ to approximate $\mathcal{A}(H_{tr}, D_{te})$, in contrast



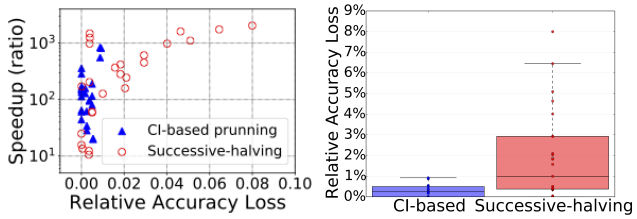

(a) Speedup Comparison

(b) Accuracy Comparison

Figure 3: Comparison Between Full-run and ABC

to using confidence interval as ABC. It repeats until there is only one remaining configuration.

Since the two solutions are designed to satisfy different constraints (accuracy loss and resource), they are not directly comparable. We do our best to make a fair comparison. In this section, we run Successive-halving with identical sample size sequence as ABC, to compare the CI-based pruning and the halving strategy based on point estimation. We perform the same set of experiments as in the main experiment section for Successive-halving. We introduce a metric, called *relative accuracy loss*, to measure the difference between the returned configuration $C_{i'}$ and the best configuration $C_{i^*}$ in terms of the test accuracy: $\Delta_{rel} = \frac{|\mathcal{A}_{i^*} - \mathcal{A}_{i'}|}{\mathcal{A}_{i^*}}$. The smaller $\Delta_{rel}$ is, the better.




(a) Speedup vs. $\Delta_{rel}$

(b) Boxplot of $\Delta_{rel}$

Figure 4: ABC vs. Successive-halving

We depict the comparison between Successive-halving and our ABC in Figure 4. The x-axis in Figure 4a refers to the relative accuracy loss compared to the best configuration by Full-run, y-axis is the speedup compared to the running time of Full-run, and each point corresponds to a specific experiment with a certain dataset and $\mathcal{C}$. We can see that Successive-halving has a similar speedup as ABC over Full-run. However, the relative accuracy loss can be an order of magnitude larger than that of ABC, e.g., $8\%$ vs. $0.8\%$. This is because the pruning performed in Successive-halving is based on the ranking of the current test accuracy. On the contrary, ABC uses confidence interval of the real test accuracy to perform safer pruning. Figure 4b presents a boxplot summarizing the relative accuracy loss for our solution and Successive-halving respectively. On average, the relative accuracy loss for our CI-based solution is $0.24\%$ (all below $1\%$), and $2\%$ for Successive-halving (up to $8\%$), which is nearly ten times larger.

## Related Work

**AutoML.** AutoML has gained increasing attention in the past few years. The scope of AutoML includes automated feature engineering, model selection, and hyperparameter tuning process. Some prevailing AutoML tools are Auto-sklearn for Python (Feurer et al. 2015) and Auto-Weka for Java (Thornton et al. 2013). Most research focus is devoted to the search strategy, i.e., which configurations to evaluate. The strategies can be broadly categorized as grid search (Pedregosa et al. 2011), random search (Bergstra and Bengio 2012), spectral search (Hazan, Klivans, and Yuan 2018), Bayesian optimization (Hutter, Hoos, and Leyton-Brown 2011; Bergstra et al. 2011; Snoek, Larochelle, and Adams 2012), meta-learning (Feurer et al. 2015), and genetic programming (Olson et al. 2016). Few studies address the efficiency issue in ranking these configurations on large datasets. TuPAQ (Sparks et al. 2015) and HyperDrive (Rasley et al. 2017) are two systems which focus on hyperparameter tuning when all the configurations correspond to iterative training processes. They distribute the configurations into multiple machines, and use heuristic early stopping rules for training iterations. (Jamieson and Talwalkar 2016) further models this problem as a non-stochastic multi-armed bandit process, where each arm corresponds to a configuration, each pull corresponds to a few training iterations, and the reward is the intermediate accuracy on the test data. Recognizing the difference with the stochastic bandit process, they propose a Successive-halving pruning strategy in the fixed budget setting. They focus on this setting because they have found it difficult to derive the confidence bounds of real test accuracy based on limited training iterations. Hyperband (Li et al. 2017) uses Successive-halving as a building block and tries to vary the number of random configurations under the same budget. While Hyperband suggests that the notion of resource can be generalized from training iterations to sample size of training data, we should notice that it is now possible to derive confidence bounds of real test accuracy based on sampled training data. Therefore, our ABC can be used to replace Successive-halving in this scenario to achieve lower accuracy loss. In the Bayesian optimization frame-

work, RoBO (Klein et al. 2017) treats the sample size as a hyperparameter, and uses random sample size to evaluate each configuration and a kernel function to extrapolate the real test accuracy.

**Generalization Error Bounds.** Generalization error bound has been studied extensively (Zhou 2002; Koltchinskii et al. 2000; Bousquet and Elisseeff 2002), among which *VC-bound* (Vapnik 1999) is a well-known technique for bounding the generalization error. The main idea behind VC-bounds is to use *VC-dimension* to characterize the complexity of the hypothesis class. Besides VC-dimension, other existing techniques for deriving generalization error bounds include covering number (Zhou 2002), Rademacher complexity (Koltchinskii et al. 2000), and stability bound (Bousquet and Elisseeff 2002). While the definition of generalization error bounds is different from the confidence bound needed for ABC, they have been used in other work to guide progressive sampling for a *single* configuration (Elomaa and Kääriäinen 2002).

## Discussion and Conclusion

We studied the problem of efficiently finding approximate best configuration among a given set of training configurations for a large dataset. Our CI-based progressive sampling and pruning solution ABC can successfully identify a top configuration with small or no accuracy loss, in much less time than the exact approach. The CI-based pruning is more robust than pruning based on point estimates.

There are multiple use cases that can benefit from our proposed ABC. The input of ABC can be either specified by the users based on their domain knowledge, or generated from an AutoML search algorithm. Our ABC module can help data scientists identify a top configuration faster. As they iteratively refine it, they can use ABC to verify whether altering part of the configuration (such as changing features) boosts the performance, by invoking ABC with the old and new configurations. In addition, our confidence bounds can be potentially used to accelerate Bayesian optimization and spectral search in large datasets, which is interesting future work.

## References


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

Bergstra, J. S.; Bardenet, R.; Bengio, Y.; and Kégl, B. 2011. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*.

Bousquet, O., and Elisseeff, A. 2002. Stability and generalization. *Journal of machine learning research* 2(Mar):499–526.

Eggensperger, 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*.

Elomaa, T., and Kääriäinen, M. 2002. Progressive rademacher sampling. In *AAAI'02*, 140–145.

Feurer, M.; Klein, A.; Eggensperger, K.; Springenberg, J.; Blum, M.; and Hutter, F. 2015. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*.

Hazan, E.; Klivans, A.; and Yuan, Y. 2018. Hyperparameter optimization: A spectral approach. In *ICLR'18*.

Huang, S.; Wang, C.; Ding, B.; and Chaudhuri, S. 2018. Efficient identification of approximate best configuration of training in large datasets. *arXiv preprint arXiv:1811.03250*.

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*.

Jamieson, K., and Talwalkar, A. 2016. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial Intelligence and Statistics*.

Klein, A.; Falkner, S.; Mansur, N.; and Hutter, F. 2017. Robo: A flexible and robust bayesian optimization framework in python. In *NIPS 2017 Bayesian Optimization Workshop*.

Koltchinskii, V.; Abdallah, C. T.; Ariola, M.; Dorato, P.; and Panchenko, D. 2000. Improved sample complexity estimates for statistical learning control of uncertain systems. *IEEE Transactions on Automatic Control* 45(12):2383–2388.

Li, L.; Jamieson, K.; DeSalvo, G.; Rostamizadeh, A.; and Talwalkar, A. 2017. Hyperband: A novel bandit-based approach to hyperparameter optimization. In *ICLR'17*.

Mund, S. 2015. *Microsoft azure machine learning*. Packt Publishing Ltd.

Olson, R. S.; Bartley, N.; Urbanowicz, R. J.; and Moore, J. H. 2016. Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO'16.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine learning in python. *JMLR* 12:2825–2830.

Provost, F.; Jensen, D.; and Oates, T. 1999. Efficient progressive sampling. In *ACM SIGKDD international conference on Knowledge discovery and data mining*.

Rasley, J.; He, Y.; Yan, F.; Ruwase, O.; and Fonseca, R. 2017. Hyperdrive: Exploring hyperparameters with pop scheduling. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, 1–13.

Sabharwal, A.; Samulowitz, H.; and Tesauro, G. 2016. Selecting near-optimal learners via incremental data allocation. In *AAAI'16*.

Snoek, J.; Larochelle, H.; and Adams, R. P. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*.

Sparks, E. R.; Talwalkar, A.; Haas, D.; Franklin, M. J.; Jordan, M. I.; and Kraska, T. 2015. Automating model search for large scale machine learning. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*.

Thornton, C.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2013. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *ACM SIGKDD international conference on Knowledge discovery and data mining*.

Vanschoren, J.; van Rijn, J. N.; Bischl, B.; and Torgo, L. 2013. Openml: Networked science in machine learning. *SIGKDD Explorations* 15(2):49–60.

Vapnik, V. N. 1999. An overview of statistical learning theory. *IEEE transactions on neural networks* 10(5):988–999.

Zhou, D.-X. 2002. The covering number in learning theory. *Journal of Complexity* 18(3):739–767.