The VLDB Journal (2024) 33:833–857
https://doi.org/10.1007/s00778-024-00842-3

REGULAR PAPER

How good are machine learning clouds? Benchmarking two snapshots over 5 years

Jiawei Jiang[1] · Yi Wei[1] · Yu Liu[2] · Wentao Wu[3] · Chuang Hu[1] · Zhigao Zheng[1] · Ziyi Zhang[1] · Yingxia Shao[4] · Ce Zhang[2]

Received: 14 January 2023 / Revised: 1 October 2023 / Accepted: 19 December 2023 / Published online: 15 March 2024

Abstract

We conduct an empirical study of machine learning functionalities provided by major cloud service providers, which we call *machine learning clouds*. Machine learning clouds hold the promise of hiding all the sophistication of running large-scale machine learning: Instead of specifying *how* to run a machine learning task, users only specify *what* machine learning task to run and the cloud figures out the rest. Raising the level of abstraction, however, rarely comes free—a performance penalty is possible. *How good, then, are current machine learning clouds on real-world machine learning workloads?* We study this question by conducting benchmark on the mainstream machine learning clouds. Since these platforms continue to innovate, our benchmark tries to reflect their evolvement. Concretely, this paper consists of two sub-benchmarks—`mlbench` and `automlbench`. When we first started this work in 2016, only two cloud platforms provide machine learning services and limited themselves to model training and simple hyper-parameter tuning. We then focus on binary classification problems and present `mlbench`, a novel benchmark constructed by harvesting datasets from Kaggle competitions. We then compare the performance of the top winning code available from Kaggle with that of running machine learning clouds from both Azure and Amazon on `mlbench`. In the recent few years, more cloud providers support machine learning and include automatic machine learning (AutoML) techniques in their machine learning clouds. Their AutoML services can ease manual tuning on the whole machine learning pipeline, including but not limited to data preprocessing, feature selection, model selection, hyper-parameter, and model ensemble. To reflect these advancements, we design `automlbench` to assess the AutoML performance of four machine learning clouds using different kinds of workloads. Our comparative study reveals the strength and weakness of existing machine learning clouds and points out potential future directions for improvement.

Keywords Machine learning cloud · Automatic machine learning · Benchmark

1 Introduction

In spite of the recent advancement of machine learning research, modern machine learning systems are still far from easy to use, at least from the perspective of business users

✉ Jiawei Jiang
jiawei.jiang@whu.edu.cn

Yi Wei
weiyi96@whu.edu.cn

Yu Liu
yu.liu@inf.ethz.ch

Wentao Wu
wentao.wu@microsoft.com

Chuang Hu
handc@whu.edu.cn

Zhigao Zheng
zhengzhigao@whu.edu.cn

Ziyi Zhang
ziyi_zhang@whu.edu.cn

Yingxia Shao
shaoyx@bupt.edu.cn

Ce Zhang
ce.zhang@inf.ethz.ch

[1] School of Computer Science, Wuhan University, Wuhan, China

[2] Department of Computer Science, ETH Zürich, Zurich, Switzerland

[3] Redmond, Microsoft Research, Washington, USA

[4] Beijing University of Posts and Telecommunications, Beijing, China

or even scientists without a computer science background [50]. Recently, there is a trend toward pushing machine learning onto the cloud as a "service," a.k.a. *machine learning clouds*. By putting a set of machine learning primitives on the cloud, these services significantly raise the level of abstraction for machine learning. For example, with Amazon Machine Learning, users only need to upload the dataset and specify the type of task (classification or regression). The cloud will then automatically train machine learning models without any user intervention.

From a data management perspective, the emergence of machine learning clouds represents an attempt toward *declarative machine learning*. Instead of relying on users to specify *how* a machine learning task should be configured, tuned, and executed, machine learning clouds manage all these physical decisions and allow users to focus on the logical side: *what* tasks they want to perform with machine learning.

Raising the level of abstractions and building a system to automatically manage all physical decisions, however, rarely comes free. In the context of a data management system, a sophisticated query optimizer is responsible for generating good physical execution plans. Despite the intensive and extensive research and engineering effort that has been put into building capable query optimizers in the past four decades, query optimizers still often make mistakes that lead to disastrous performance.

In the context of declarative machine learning, things become even more subtle. A bad choice of "physical plan" may result in not only suboptimal performance but also suboptimal quality (e.g., accuracy). In this paper, we investigate the usability of state-of-the-art machine learning clouds. Specifically, we ask the following question:

> *To what extent can existing declarative machine learning clouds support real-world machine learning tasks?*

To answer this question, we conduct an empirical study with `mlbench` and `automlbench`, two novel benchmarks consisting of real-world datasets *and* best-effort solutions harvested from humans.

One characteristic of commercial machine learning clouds is their ever-lasting evolvement over time. When we started this study at 2016, there were only two popular machine learning clouds—Azure Machine Learning Studio and Amazon Machine Learning, and they only provided *training service* of a chosen model and simple hyper-parameter tuning via grid search. After our first phase of benchmarking [34],[1] more machine learning clouds emerged, accommodating *AutoML services* for pipeline execution, e.g., data cleaning, feature selection, feature transformation, model selection, hyper-parameter tuning, and model ensemble. These new arising features motivate our second phase of benchmarking that focuses on the effectiveness of AutoML functionalities. In summary, this work consists of the above two sub-benchmarks—(1) `mlbench`: benchmark of model training services in pioneering machine learning clouds, and (2) `automlbench`: benchmark of AutoML services in recent machine learning clouds.

[1] Refer to https://dl.acm.org/doi/pdf/10.14778/3231751.3231770.

1.1 Benchmark of model training services

In the evaluation of model training services, we study *what would users lose by resorting to declarative machine learning clouds instead of using a best-effort, non-declarative machine learning system?* To answer this question, we run collected real-world datasets over two machine learning clouds and compare them with best-effort solutions harvested from Kaggle competitions. We use a novel methodology that allows us to separate measuring the performance of the machine learning clouds themselves from other external factors that may have significant impact on quality, such as feature selection and hyper-parameter tuning. Moreover, we use a novel performance metric that measures the strength and weakness of current machine learning clouds by comparing their relative performance with top-ranked solutions in Kaggle competitions.

Technical Contributions in this Benchmark.

- *C1* We present the `mlbench` benchmark. One prominent feature of `mlbench` is that each of its datasets comes with a best-effort baseline of both feature engineering and selection of machine learning models.
- *C2* We propose a novel performance metric based on the notion of "quality tolerance" that measures the performance gap between a given machine learning cloud and top-ranked Kaggle competition performers.
- *C3* We evaluate two popular machine learning clouds, Azure Machine Learning Studio and Amazon Machine Learning, using `mlbench`. Our experimental result reveals interesting strengths and limitations of both clouds. Detailed analysis of the results further points out promising future directions to improve both machine learning clouds.

1.2 Benchmark of AutoML services

To catch up with the development of AutoML techniques, the existing machine learning clouds mostly, if not all, resort to including AutoML services. In this way, they extend their capabilities from mere model training in their original version to pipelined machine learning workloads. In addition to simple hyper-parameter tuning, the current machine learning clouds provision AutoML assists in almost every aspect of machine learning, e.g., data preprocessing, data

cleaning, feature engineering, model training, and model ensemble. The abundance of AutoML services can significantly enhance the usability of machine learning clouds and save tremendous human efforts on tuning proper knobs. Following this trend in the machine learning clouds, our second sub-benchmark in this work evaluates both effectiveness and efficiency of AutoML services in the recent machine learning clouds.

Technical Contributions in this Benchmark

- *C1* We present the `automlbench` benchmark. To fully assess the capabilities of AutoML services, `automlbench` considers a broad scope of datasets and workloads.
- *C2* By comparing four mainstream machine learning clouds that offer different scopes of AutoML, we are able to study the importance of each AutoML component and draw insights to understand the landscape of AutoML services in the cloud.
- *C3* We compare the models trained by cloud platforms with those submitted by top-ranked Kaggle solutions. The empirical results show the limitation of AutoML offerings on the current machine learning clouds.

2 The `mlbench` benchmark

2.1 Methodology

Benchmarking systems fairly is not an easy task. Three key aspects came to mind when designing a benchmark for machine learning clouds:
(1) We need to measure not only the *performance* (speed) but also the *quality* (precision). The two are coupled, and their relative importance changes with respect to the user's budget and tolerance for suboptimal quality.
(2) The quality of an application depends on both *feature engineering* and the *machine learning model*. If these two factors are not decoupled, our result will be unfair to most machine learning clouds, as they usually do not provide an efficient mechanism for automatic feature engineering.
(3) To compare declarative machine learning clouds with the best effort of using non-declarative machine learning systems, we need to construct a *strong baseline* for the latter. If this baseline is not strong enough, our result may be overly optimistic regarding machine learning clouds.

Starting from these principles, we made a few basic decisions that we shall present next.
Scope of Study We restrict ourselves to *binary classification*, one of the most popular machine learning tasks. As we will see, even with this constrained scope, there is no simple, single answer to the main question we aim to answer.
Protocol We collect top winning code for all binary classification competitions on Kaggle. We then filter them to select a subset to include in `mlbench` with the following protocol. For the code that we are able to install and finish running within 24 h, we further collect features extracted by the winning code. The features are then used for training and testing models provided by both the machine learning cloud and the Kaggle winning solution. We also include datasets constructed using raw features (see Sect. 2.2.2).
Discussion The intuition behind our methodology is simple: The top winning code of Kaggle competitions represents the arguably best effort among existing machine learning solutions. Of course, it is biased by the competitions published on Kaggle and the solutions provided by the participants. Nonetheless, given the high impact of Kaggle competitions, we believe that using the winning code as a performance baseline significantly raises the bar compared with using standard libraries and therefore reduces the risk that we might be overly optimistic about the machine learning clouds. Moreover, given the "crowdsourcing" nature of Kaggle, the baseline will keep up with the advancement of machine learning research and practice, perhaps at a much faster pace than standard libraries can.

2.1.1 Quality metric

Our methodology of adopting Kaggle winning code as a baseline raises the question of designing a reasonable quality metric. To measure the quality of a model deployed on machine learning clouds, we introduce the notion of "quality tolerance" (of a user).

Definition 1 The *quality tolerance* of a user is τ if she/he can be satisfied only by being ranked among the top $\tau\%$, assuming that she/he uses a model M provided by the cloud to participate in a Kaggle competition.

Of course, the "user" in Definition 1 is just hypothetical. Essentially, quality tolerance measures the performance gap between the machine learning cloud and the top-ranked code of a Kaggle competition. A lower quality tolerance suggests a more stringent user requirement and therefore a more capable machine learning cloud if it can meet that quality tolerance.

Based on the notion of quality tolerance, we are mainly interested in two performance metrics of a model M:

- *Capacity*, the minimum quality tolerance $\tau_{\min}$ that M can meet for a given Kaggle competition T;
- *Universality*, the number of Kaggle competitions that M can achieve a quality tolerance of τ.

Intuitively, capacity measures how high M can be ranked in a Kaggle competition, whereas universality measures in how many Kaggle competitions M can be ranked that high.

We use $c(M, T)$ and $u(M, \tau)$ to denote the capacity and τ-universality of M. Moreover, we use $\mathcal{K}(M, \tau)$ to denote the set of Kaggle competitions whose quality tolerance τ have been reached by $u(M, \tau)$:

$$u(M, \tau) = |\mathcal{K}(M, \tau)|.$$

Similarly, if a machine learning cloud $\mathcal{M}$ provides n models $\{M_1, \ldots, M_n\}$ $(n \geq 1)$, we can define the capacity of $\mathcal{M}$ with respect to a Kaggle competition T as

$$c(\mathcal{M}, T) = \min_{M_i \in \mathcal{M}} c(M_i, T), \quad 1 \leq i \leq n,$$

and define the τ-universality of $\mathcal{M}$ as

$$u(\mathcal{M}, \tau) = |\bigcup_{i=1}^{n} \mathcal{K}(M_i, \tau)|.$$

Clearly, the capacity of $\mathcal{M}$ over T is the capacity of the best model that $\mathcal{M}$ provides for T, whereas the τ-university of $\mathcal{M}$ is the number of Kaggle competitions in which $\mathcal{M}$ can meet quality tolerance τ (with the best model it can provide).

Finally, if there are m Kaggle competitions $\mathcal{T} = \{T_1, \ldots, T_m\}$, we define the capacity of $\mathcal{M}$ over $\mathcal{T}$ as

$$c(\mathcal{M}, \mathcal{T}) = \max_{T_j \in \mathcal{T}} c(\mathcal{M}, T_j), \quad 1 \leq j \leq m.$$

It measures the *uniformly* best quality tolerance that $\mathcal{M}$ can meet for any of the competitions in $\mathcal{T}$.

In the rest of this paper, we will use the notation $c(M)$, $u(M)$, $c(\mathcal{M})$, and $u(\mathcal{M})$ whenever the corresponding quality tolerance and Kaggle competition(s) are clear from the context.

2.1.2 Limitations and discussion

Our motivation of using ranking as performance metric is to provide a normalized score *across* all datasets. However, ranking itself does not tell the full story. One caveat is that ranking measures the *relative* performance and may be sensitive to the change in the underlying, *absolute* metric, such as the "area under curve" (AUC) score that is commonly used by Kaggle competitions. To illustrate this, Fig. 1 presents the histograms (i.e., distributions) of the AUC scores in two Kaggle competitions (see Sect. 2.2.3 for the details of the competitions). The red, green, and blue lines correspond to the teams ranked at the top 95%, 50%, and 5%. The distance between the scores of top 50% (green) and top 5% (blue) shows the sensitivity—for D-PHY, ranking is quite sensitive to small changes in AUC as most of the teams have similar scores. Therefore, when benchmarking machine learning clouds, it is important to look at *both* ranking and absolute quality. In this benchmark, our analysis will always base on both.

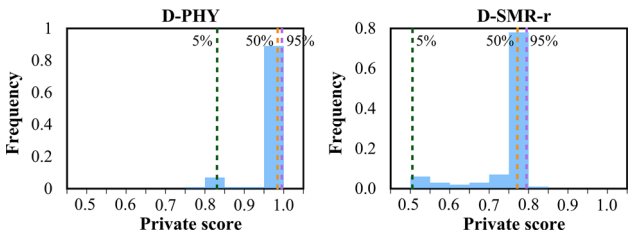

Fig. 1 Histograms of the AUC scores on private leader board for two example datasets D-PHY and D-SMR-r

2.2 Kaggle competitions and datasets

2.2.1 Kaggle competitions

Kaggle hosts various types of competitions for data scientists. There are seven different competition categories, and we are particularly interested in the category "Featured" that aims to solve commercial, real-world machine learning problems. For each competition, Kaggle provides a necessary description of its background, training and testing datasets, evaluation metric, and so on. These competitions are online only for a while, and Kaggle allows participants to submit multiple times during that period. Kaggle evaluates and provides a score for each submission (shown as a public leader board). Participants can specify their final submissions before a competition ends, and a final, private leader board is available after the competition ends. The winners are determined based on their rankings on the private leader board. In this paper, we treat the top ten on the private leader board as "winning code," and we look for the one ranked the highest among the top ten.

2.2.2 Overview

`mlbench` is curated from Kaggle competitions with or without winning code. We describe the protocol of curating as follows.

Datasets from Winning Code As shown in Fig. 2, we collected 267 Kaggle competitions in total and found winning code for 41 of these competitions. We are unable to find winning code for the remaining 226 competitions. Fortunately, the 41 competitions with available winning code already exhibit sufficient diversity to evaluate various aspects of machine learning clouds. Figure 3 further summarizes the types of

Statistics of Kaggle Competitions	Number
Total Competitions	267
Competitions with Winning Code	41
Competitions without Winning Code	226

Fig. 2 Statistics of Kaggle competitions

Tasks of Kaggle Competitions	Number
Binary Classification	13
Multi-class Classification	14
Regression	9
Others	5

Fig. 3 Kaggle competitions with winning code

machine learning tasks covered by these 41 competitions with winning code. Given the scope of study we stated in Sect. 2.1, the 13 competitions that are binary classification tasks are the focus of our evaluation.

We then ran the winning code of the 13 competitions on Microsoft Azure for the purpose of extracting the features used by the winning code (recall Sect. 2.1). We failed to run the winning code for "Avito Context Ad Clicks." For "Santander Customer Satisfaction" and "Higgs Boson Machine Learning Challenge," the code cannot be finished on an Azure machine with a 16-core CPU and 112GB memory. Therefore, there were 10 competitions for which we finished running the winning code successfully. We further removed datasets whose outputs are either three-dimensional features that cannot be supported by the machine learning clouds we studied or features that cannot be extracted and saved successfully. Moreover, the winning code of "KDD Cup 2014" generated two sets of features—it uses the ensemble method with two models. This results in 7 datasets with features extracted by the winning code.

Beyond Winning Code We also constructed datasets using the raw features from Kaggle (details in Sect. 2.2.3), which results in 11 additional datasets. Specifically, we include all binary competitions ended by July 2017 that (1) use AUC as evaluation metric, (2) can be joined by new users, (3) have datasets available for download, (4) still allow for submission and scoring, (6) do not contain images, videos, and HTML files, and (5) whose total size does not exceed Azure's limitation.

In total, `mlbench` contains 18 datasets with 7 datasets having *both* features produced by winning code and the raw features provided by the competition. We summarize the statistics of the datasets in Fig. 4. We can see a reasonable diversity across the datasets in terms of the size of the training set, the size of the testing set, and the number of features. Moreover, the ratio between the sizes of the training set and testing set varies as well. For example, D-VP has a testing set 10 times larger than the training set, which is quite different from the vanilla setting, where the training set is much larger.

Dataset	Training Set	Test Set	# Features	Training Size	Test Size
D-SCH-r	86	119,748	410	0.3MB	488MB
D-PIS-r	5,500	5,952	22	1.2MB	1.29MB
D-EG-r	7,395	3,171	124	21MB	9MB
D-VPr-r	10,506	116,293	53	2MB	19MB
D-AEA-r	32,769	58,921	9	1.94MB	3.71MB
D-SCS-r	76,020	75,818	369	56.6MB	56.3MB
D-SMR-r	145,231	145,232	1,933	921MB	921MB
D-GMC-r	150,000	101,503	10	7.21MB	4.75MB
D-HQC-r	260,753	173,836	297	198MB	131MB
D-KDD-r	619,326	44,772	139	571MB	40.7MB
D-PBV-r	2,197,291	498,687	52	181MB	76MB
D-SCH	86	119,748	410	0.3MB	488MB
D-EG	7,395	3,171	29	2.59MB	1.35MB
D-VP	10,506	116,293	17	0.8MB	9.1MB
D-AEA	32,769	58,921	135	54MB	97MB
D-PHY	38,012	855,819	74	28.9MB	859MB
D-KDD2	131,329	44,772	100	105MB	36MB
D-KDD1	391,088	44,772	190	282MB	32MB

Fig. 4 Statistics of datasets. The "-r" in the dataset names indicates raw feature (see Sect. 2.2.3)

2.2.3 Dataset details

We present more details about the datasets listed in Fig. 4 in supplemental materials.[2] For each dataset, we introduce the background of the corresponding Kaggle competition and describe the features used by the winning code, as well as the models and algorithms it adopts. *Note that the winning code may apply feature engineering techniques to transform "raw features" provided by Kaggle.*

2.3 Experimental settings

We evaluate the declarative machine learning services provided by two major cloud vendors: Microsoft Azure Machine Learning Studio and Amazon Machine Learning. We will use **Azure** and **Amazon** as shorthand.

We first introduce the current APIs of **Azure** and **Amazon** and then all machine learning models they provide.

2.3.1 Existing cloud API

Both **Azure** and **Amazon** start by asking users to upload their data, which can be in the form of CSV files. Users then specify the machine learning tasks they want to run on the cloud. However, **Azure** and **Amazon** offer different APIs, as illustrated below.

[2] https://drive.google.com/file/d/1lXC47nBjDyfqrNyUIC9xv-SEIuks44Gg/view.

– **Azure** provides an API using which users specify the *types* of *machine learning models*, such as (1) logistic regression, (2) support vector machine, and (3) decision tree. For each type of model, **Azure** provides a set of default hyper-parameters for users to use in an out-of-the-box manner. **Azure** also supports different ways of automatic hyper-parameter tuning and provides a default range of values to be searched for.
– **Amazon** provides an API by which users specify the *types* of *machine learning tasks*, namely (1) binary classification, (2) multiclass classification, and (3) regression. For each type, **Amazon** automatically chooses the type of machine learning models. **Amazon** always runs a logistic regression for binary classification [3]. **Amazon** further provides a set of default hyper-parameters for logistic regression, but users can also change these default values.

2.3.2 Machine learning models

We give a brief description of the machine learning models provided by **Azure** and **Amazon**.

– *Two-Class Averaged Perceptron (C-AP)* It is a linear classifier and can be thought of as a simplified neural network: There is only one layer between input and output.
– *Two-Class Bayes Point Machine (C-BPM)* This is a Bayesian classification model, which is not prone to overfitting. The Bayes point is the average classifier that efficiently approximates the theoretically optimal Bayesian average of several linear classifiers [25] (in terms of generalization performance).
– *Two-Class Boosted Decision Tree (C-BDT)* Boosting is a well-known ensemble algorithm that combines weak learners to form a stronger learner (e.g., AdaBoost [19]). The boosted decision tree is an ensemble method that constructs a series of decision trees [41]. Except for the first tree, each of the remaining trees is constructed by correcting the prediction error of the previous one. The final model is an ensemble of all constructed trees.
– *Two-Class Decision Forests (C-DF)* This classifier is based on random decision forests [26]. Specifically, it constructs multiple decision trees that vote on the most popular output class.
– *Two-class Decision Jungle (C-DJ)* This is an ensemble of rooted decision directed acyclic graphs (DAGs). In conventional decision trees, only one path is allowed from the root to a leaf. In contrast, a DAG in a decision jungle allows multiple paths from the root to a leaf [43].
– *Two-Class Logistic Regression (C-LR)* This is a classic classifier that predicts the probability of an instance by fitting a logistic function. It is also the only classifier that **Amazon** supports.
– *Two-Class Neural Network (C-NN)* Neural networks are bio-inspired algorithms that are loosely analogous to the observed behavior of a biological brain's axons [23]. Specifically, the input layer (representing input data) and the output layer (representing answers) are connected by layers of weighted edges and nodes, which encode the so-called activation functions.
– *Two-Class Support Vector Machine (C-SVM)* SVM [12] is another well-known classifier. It works by separating the data with the "maximum-margin" hyperplane.

2.3.3 Hyper-parameter tuning

Each machine learning algorithm consists of a set of hyper-parameters to tune. The methodology we use in this paper is to rely on the *default* tuning procedure provided by the machine learning cloud. Figure 5 summarizes the hyper-parameters provided by the machine learning clouds. For each machine learning model, we conduct an exhaustive grid search on all possible parameter combinations.

Because **Amazon** only has the option of logistic regression and automatically tunes the learning rate, we tuned hyper-parameters for models provided by **Azure**. We performed hyper-parameter tuning in an exhaustive manner: For each combination of hyper-parameter values in the whole search space, we ran the model based on that setting. The best hyper-parameter is then selected based on the AUC score

Platform	Model	# Combinations	Tuned Parameters
Azure	C-AP	6	Learning Rate Maximum # iterations
	C-BPM	1	# iterations
	C-BDT	180	Maximum # leaves per tree Minimum # samples per leaf learning rate # trees
	C-DF	81	# trees={1, **8**, 32} Maximum depth # Random splits per note Minimum # samples per leaf
	C-DJ	81	# Decision DAGs Max depth of decision DAGs Max width of decision DAGs # optimizations per layer
	C-NN	12	Learning Rate # iterations
	C-SVM	15	# iterations λ
	C-LR	72	Optimization tolerance L1 regularization weight L2 regularization weight Memory size for L-BFGS
Amazon	C-LR	N/A	Amazon tunes the learning rate

Fig. 5 Hyper-parameters tuned for each model. The default parameters are in bold font

Data Set	C-AP	C-BPM	C-BDT	C-DF	C-DJ	C-LR	C-NN	C-SVM
D-SCH-r	6	9	280	12	130	15	11	11
D-PIS-r	6	5	639	178	551	16	30	16
D-EG-r	28	15	10,165	260	530	145	3,548	271
D-VP-r	32	8	3,105	821	3,342	135	824	304
D-AEA-r	33	22	25,240	2,028	3,968	230	690	229
D-SCS-r	126	286	6,198	2,816	7,119	791	1,583	1,433
D-SMR-r	18,315	3,116	NA	15,853	19,332	NA	NA	NA
D-GMC-r	27	17	4,146	4,425	10,635	206	501	208
D-HQC-r	2,697	454	47,502	22,023	NA	12,404	29,617	27,341
D-KDD-r	4,290	906	NA	40,816	63,185	37,328	NA	39,716
D-PBV-r	11,450	877	NA	NA	NA	84,290	NA	NA
D-SCH	6	9	280	12	130	15	11	11
D-EG	6	13	837	227	690	17	39	22
D-VP	13	14	1,546	422	2,383	55	216	100
D-AEA	25	241	3,495	780	3,413	210	323	193
D-PHY	21	114	2,901	644	3,167	132	262	144
D-KDD2	550	1,450	365,256	6,675	9,339	7,228	NA	6,191
D-KDD1	310	3,814	40,364	24,796	88,044	4,455	4,465	3,074

Fig. 6 Total training time (s) on Azure with hyper-parameter tuning (HPT)

obtained with fivefold cross-validation. AUC is the evaluation metric used by all Kaggle competitions we included in `mlbench`. The third column in Fig. 5 presents the number of hyper-parameter combinations in the search space for each model. For example, C-AP employs two hyper-parameter knobs, "learning rate" and "maximum number of iterations," with three and two alternative values. As a result, there are six combinations in total. For completeness, we report the time spent on tuning hyper-parameters for **Azure** models in Fig. 6.

2.4 Results on winning features

We first evaluated the performance of **Azure** and **Amazon** *assuming users have already conducted feature engineering and only use the machine learning cloud as a declarative execution engine of machine learning models*. Our analysis in this section will mainly focus on the seven datasets where winning code is available (i.e., the datasets in Fig. 4 without the "-r" suffix). We will discuss the cases when raw features are used in Sect. 2.5. For each dataset and model, we run **Azure** and **Amazon** for at most 24 h.

2.4.1 Capability and universality

We first report the performance of **Azure** and **Amazon**, based on the *capacity* and *universality* metrics defined in Sect. 2.1.1. Figure 7 presents the result.

In Fig. 7a, the x-axis represents the quality tolerance, whereas the y-axis represents the number of models required if a machine learning cloud can achieve a certain tolerance level τ for all seven datasets (i.e., a τ-university of seven). The minimum τ shown in Fig. 7a then implies the capacity of a machine learning cloud. We observe that the capacity of **Azure** is around 31 (i.e., $c(\text{Azure}) = 31$), whereas the capacity of **Amazon** is around 83 (i.e., $c(\text{Amazon}) = 83$). Under this measurement, state-of-the-art machine learning clouds are far from competitive than deliberate machine learning models designed manually: With the goal of meeting a τ-university of seven, τ can only be as small as 31 for **Azure** (and 83 for **Amazon**). In other words, in at least one Kaggle competition (among the seven), **Azure** is ranked outside the top 30%, whereas **Amazon** is ranked outside the top 80% on the leader board.

However, we note that this might be a distorted picture given the existence of "outliers." In Fig. 7b, c, we further present results by excluding the datasets D-VP and D-KDD2. Although the capacity of **Amazon** remains the same, the capacity of **Azure** improves dramatically: $c(\text{Azure})$ drops to 7 by excluding D-VP and further drops to 5 by excluding D-KDD2, which suggests that **Azure** can be ranked within the top 10% or even the top 5% in most of the Kaggle competitions considered.

2.4.2 Breakdown and analysis

We next take a closer look at how the machine learning clouds perform in individual competitions. Figure 8 reports the details of the AUC of different models on different datasets. The number in parentheses next to the AUC is the rank (of this AUC) on the leader board. We note that not every winning code we found is top-ranked. Often, the top-ranked code is not available, and in this case, we seek the next available winning code (among the top 10) on the leader board. We have several interesting observations.

Diversity of models is beneficial An obvious difference between **Azure** and **Amazon** is that **Azure** provides more alternative models than **Amazon**. While the reason for **Amazon** to provide only logistic regression is unclear, the results presented in Fig. 8 suggest that the additional models provided by **Azure** do help. In more detail, Fig. 9 compares the capacity of **Azure** and **Amazon** on different datasets. We observe that **Azure** wins over **Amazon** in terms of capacity, often by a large margin. The capacity of **Azure** over all the datasets is 31.24 (6.99 if excluding D-VP and 2.54 if further excluding D-KDD2) versus 84.34 of **Amazon**, as shown in Fig. 7.

Model selection is necessary For a given dataset, the variation in terms of prediction quality is quite large across different models. For example, by using the models provided by **Azure** on the dataset "D-SCH," the rank varies from 3 (as good as the winning code) to 281 (ranked at the bottom 10% of 313 Kaggle competition participants). This makes model selection a difficult job for **Azure** users. (**Amazon** users do not have this problem, as logistic regression is their only option.)

Hyper-parameter tuning makes a difference for a single model Both **Azure** and **Amazon** provide logistic regres-

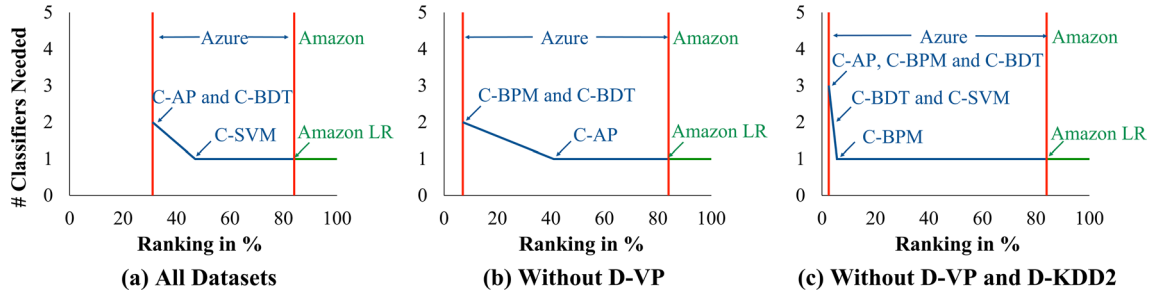

Fig. 7 Capacity and universality of machine learning clouds as quality tolerance varies

Dataset	Leader board	Winning Code	Azure								Amazon
			C-AP	C-BPM	C-BDT	C-DF	C-DJ	C-LR	C-NN	C-SVM	
D-SCH-r	0.93 (1)	0.91 (3)	0.88 (37)	0.92 (3)	0.69 (279)	0.68 (281)	0.82 (165)	0.89 (22)	0.88 (30)	0.90 (8)	0.74 (264)
D-PIS-r	0.88 (1)		0.76 (127)	0.78 (118)	0.87 (27)	0.86 (62)	0.87 (48)	0.78 (118)	0.80 (114)	0.77 (120)	0.86 (66)
D-EG-r	0.89 (1)	0.89 (4)	0.86 (388)	0.86 (381)	0.86 (374)	0.86 (382)	0.86 (388)	0.86 (388)	0.86 (388)	0.85 (400)	NA (NA)
D-VP-r	0.86 (1)	0.86 (1)	0.63 (1105)	0.64 (1083)	0.69 (840)	0.63 (1100)	0.60 (1137)	0.67 (928)	0.66 (1023)	0.57 (1157)	0.66 (989)
D-AEA-r	0.92 (1)	0.92 (2)	0.87 (834)	0.84 (932)	0.87 (802)	0.86 (889)	0.79 (1049)	0.88 (737)	0.87 (798)	0.86 (869)	0.85 (901)
D-SCS-r	0.83 (1)	0.83 (3)	0.75 (4128)	0.77 (4038)	0.82 (3084)	0.82 (3432)	0.82 (3240)	0.78 (4019)	0.79 (3916)	0.71 (4302)	0.81 (3691)
D-SMR-r	0.80 (1)		0.74 (1826)	0.70 (1898)	NA (NA)	0.73 (1842)	0.71 (1888)	NA (NA)	NA (NA)	NA (NA)	NA (NA)
D-GMC-r	0.87 (1)		0.74 (810)	0.70 (820)	0.87 (137)	0.86 (505)	0.87 (139)	0.70 (837)	0.83 (713)	0.81 (750)	0.86 (544)
D-HQC-r	0.97 (1)		0.94 (1431)	0.93 (1476)	0.97 (977)	0.96 (1206)	NA (NA)	0.94 (1429)	0.95 (1335)	0.94 (1463)	0.96 (1294)
D-KDD-r	0.68 (1)	0.67 (2)	0.59 (130)	0.55 (314)	NA (NA)	0.56 (297)	0.54 (373)	0.60 (77)	NA (NA)	0.54 (363)	0.58 (152)
D-PBV-r	1.00 (1)		0.95 (1651)	0.98 (1498)	NA (NA)	NA (NA)	NA (NA)	0.97 (1529)	NA (NA)	NA (NA)	0.96 (1612)
D-SCH	0.93 (1)	0.91 (3)	0.88 (37)	0.92 (3)	0.69 (279)	0.68 (281)	0.82 (165)	0.89 (22)	0.88 (30)	0.90 (8)	0.74 (264)
D-EG	0.89 (1)	0.89 (4)	0.89 (2)	0.89 (3)	0.88 (54)	0.88 (278)	0.88 (291)	0.89 (2)	0.89 (4)	0.89 (2)	0.88 (326)
D-VP	0.86 (1)	0.86 (1)	0.65 (1046)	0.70 (734)	0.76 (408)	0.74 (473)	0.72 (568)	0.71 (648)	0.74 (496)	0.72 (613)	0.68 (923)
D-AEA	0.92 (1)	0.92 (2)	0.91 (75)	0.91 (94)	0.92 (29)	0.90 (147)	0.91 (63)	0.91 (72)	0.91 (72)	0.90 (401)	0.90 (361)
D-PHY	1.00 (1)	1.00 (2)	NA (NA)	NA (NA)	NA (NA)	NA (NA)	NA (NA)	NA (NA)	NA (NA)	NA (NA)	NA (NA)
D-KDD2	0.68 (1)	0.67 (2)	0.58 (192)	0.51 (399)	0.62 (33)	0.61 (53)	0.60 (72)	0.58 (198)	NA (NA)	0.57 (206)	0.60 (65)
D-KDD1	0.68 (1)	0.67 (2)	0.65 (12)	0.64 (25)	0.64 (26)	0.62 (40)	0.64 (25)	0.65 (12)	0.60 (68)	0.64 (20)	0.63 (29)

Fig. 8 Area under curve (AUC) and rankings on the private leader board of Kaggle for various datasets and models. The results for public leader board are similar. The winning code of KDD is an ensemble of the classifiers trained from both D-KDD1 and D-KDD2

Dataset	**Azure** (Model)	**Amazon**	Winning
D-SCH (313)	0.96 (C-BPM)	84.34	0.96
D-EG (625)	0.32 (C-AP)	52.16	0.64
D-VP (1306)	31.24 (C-BDT)	70.67	0.08
D-AEA (1687)	1.72 (C-BDT)	21.40	0.12
D-KDD2 (472)	6.99 (C-BDT)	13.77	0.42
D-KDD1 (472)	2.54 (C-LR)	6.14	0.42

Fig. 9 Capacity of **Azure** and **Amazon** on different datasets (i.e., Kaggle competitions)

Dataset	**Azure** (C-LR)	**Amazon**	Winning
D-SCH (313)	7.03	84.34	0.96
D-EG (625)	0.32	52.16	0.64
D-VP (1306)	49.62	70.67	0.08
D-AEA (1687)	4.27	21.40	0.12
D-KDD2 (472)	41.95	13.77	0.42
D-KDD1 (472)	2.54	6.14	0.42

Fig. 10 Capacity of the logistic regression model (C-LR) from **Azure** and **Amazon** on different datasets

sion. The difference is that **Azure** provides more knobs for hyper-parameter tuning (recall Fig. 5). Figure 10 compares the capacity of the logistic regression model ("C-LR") provided by **Azure** and **Amazon**. **Azure** wins on most of the datasets, perhaps due to more systematic hyper-parameter tuning. However, there is no free lunch: Hyper-parameter tuning is time-consuming (recall Fig. 6).

2.4.3 Model selection

The previous section gives an overview of the performance of **Azure** and **Amazon** in terms of their capacity and universality. However, although we observe that the additional models provided by **Azure** significantly improve performance, model selection and hyper-parameter tuning become new challenges.

From the user's perspective, there is then a natural question: *Given a machine learning task, which model should a user choose (for good performance)?* The answer depends on (1) the capacity of the models, (2) the time the user is willing to spend on parameter tuning and training, and (3) the user's quality tolerance level.

In the following, we study the trade-off between these factors. Our goal is not to give a definitive conclusion, which is in general impossible given the variety of machine learning

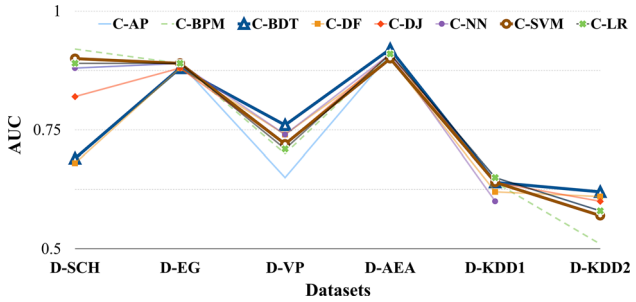

Fig. 11 Quality of different models

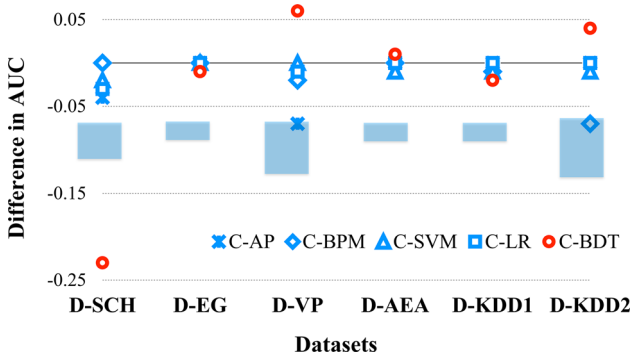

Fig. 12 BDT versus linear classifiers

tasks and models. Rather, by presenting the results observed in our study, we hope we can give some insights into what is going on in reality to help users come up with their own recipes.

2.4.4 Linear versus nonlinear models

In Fig. 7, we have incrementally noted the models we need to include to improve the capacity of **Azure** (with respect to a given universality). Clearly, we find that nonlinear classifiers (e.g., C-BDT, C-NN, etc.) are the driving force that propels the improvement. It is then an interesting question to investigate where the improvement indeed comes from. We further compare the AUC of the models over different datasets in Fig. 11, based on the raw data in Fig. 8.

We observe that nonlinear models (e.g., C-BDT) dominate linear models (e.g., C-SVM) as the dataset size increases. This is intuitive: Nonlinear models are more complicated than linear models in terms of the size of hypothesis space. However, nonlinear models are more likely to suffer from overfitting on small datasets (e.g., the smallest dataset D-SCH in Fig. 11).

Figure 12 further presents a zoomed-in comparison between C-BDT, the dominant nonlinear classifier, and the linear models C-AP, C-BPM, C-SVM, and C-LR. The y-axis represents the difference in terms of AUC between a model and the best linear model. For example, the best linear model on the dataset D-SCH is C-BPM with an AUC of 0.92, whereas the best linear model on the dataset D-EG is C-SVM with an AUC of 0.89 (see Fig. 8). Linear models often perform similarly regardless of dataset size: There is apparently a hit-or-miss pattern for linear models; namely, either the actual hypothesis falls into the linear space or it does not. As a result, there is often no big difference in terms of prediction quality between linear models: If users believe that linear models are sufficient for a learning task, they can focus on reducing the training time rather than picking which model to use.

Observation 1 *To maximize quality on Kaggle, use a nonlinear model whenever it can scale and the dataset is not too small.*

2.4.5 Training time versus prediction quality

As we have mentioned, there is an apparent trade-off between the prediction quality of a model and the training time required for that model. More sophisticated models usually have more knobs to tune and therefore need more time for training. Given that nonlinear models in general outperform linear models on large datasets, it is worth further investigating the trade-off between their training time and prediction quality.

We summarize the comparison result in Fig. 13. Figure 13 presents the trade-off between the prediction quality and the total training time on hyper-parameter tuning. For ease of exposition, we order the models by their training time along the x-axis. We also include linear models in our comparison for completeness.

In each plot of Fig. 13, the blue horizontal line represents the AUC of the winning code and the red horizontal line represents the AUC of (the logistic regression model provided by) **Amazon**, whereas the scattered points present the AUC of **Azure** models. We have noted that the choice of linear versus nonlinear models can make a difference. However, one interesting phenomenon we observe is that the choice within each category seems not so important, i.e., the prediction quality of different nonlinear models is similar. Although this is understandable for linear models, it is a bit surprising for nonlinear models. One reason for this is that most of the nonlinear models provided by **Azure** are based on decision trees (C-BDT, C-DJ, and C-DF). Moreover, more training time does not always lead to better prediction quality.

Based on these observations, our second empirical rule for model selection on machine learning clouds is:

Observation 2 *To maximize efficiency, within each category (linear or nonlinear), pick the one with the shortest training time.*

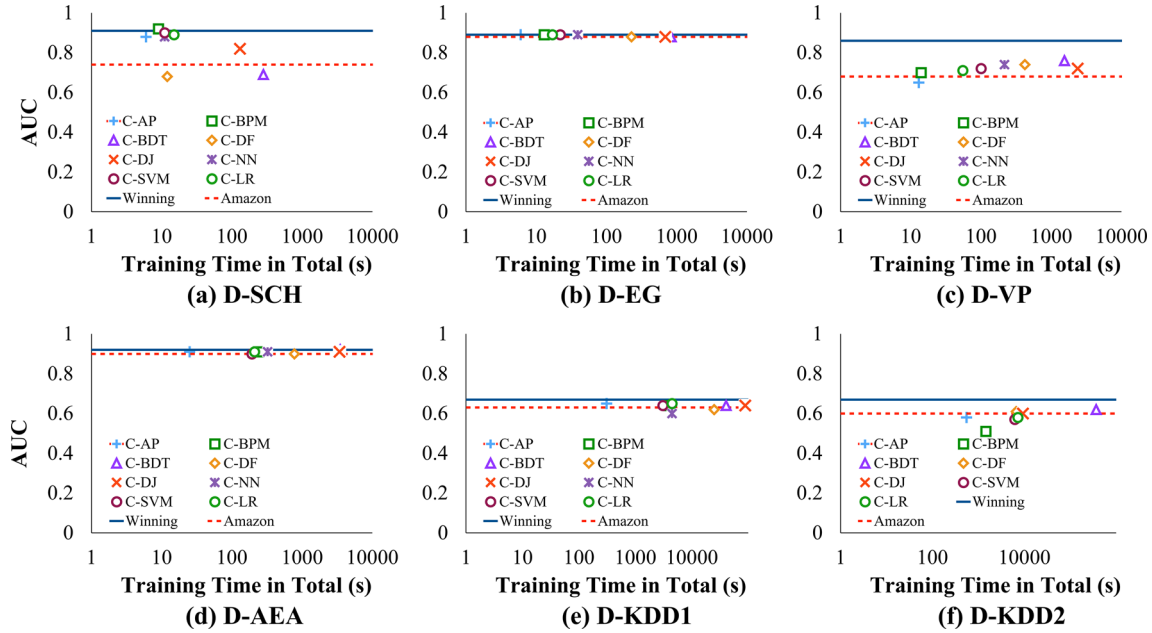

Fig. 13 Trade-off between prediction quality (AUC) and *total* training time. The blue line represents the AUC of the winning code, and the red line represents the AUC of logistic regression (C-LR) on **Amazon** (color figure online)

2.4.6 Quality tolerance regime

We emphasize that the rules we presented in Observations 1 and 2 are purely empirical. So far, we have looked at the model selection problem from only two of the three respects, i.e., the capacity of the models and the training time they require. We now investigate the third respect: the user's quality tolerance. This is a more fundamental and subtle point: A certain quality tolerance may not even be achievable for a machine learning task given the current capacity of machine learning clouds. (For example, we have seen that neither **Azure** nor **Amazon** can achieve even a quality tolerance of 30 on D-VP.)

To avoid oversimplifying the problem, we define the concept of *quality tolerance regime* based on the capacity of the machine learning clouds we currently observe:

- *Low tolerance regime*. Corresponds to the case when the quality tolerance is below 1.[3]
- *Middle tolerance regime*. Corresponds to the case when the quality tolerance is between 1 and 5.
- *High tolerance regime*. Corresponds to the case when the quality tolerance is between 5 and 10.

To give some sense of how well a model must perform to meet the tolerance regimes, in Fig. 14, we present the AUC that a model has to achieve to meet the high tolerance regime,

Dataset	Winning AUC	10% AUC
D-SCH	0.91	0.88
D-EG	0.89	0.88
D-VP	0.86	0.79
D-AEA	0.92	0.90
D-KDD2	0.67	0.62
D-KDD1		

Fig. 14 AUC corresponding to the high tolerance regime of different Kaggle competitions

the loosest criterion in our definition, in different Kaggle competitions. This is a way to measure the intensity of a competition: The smaller the gap is between the winning AUC and the top 10% AUC, the more intense the competition is. Some competitions are highly competitive: The gap is merely 0.01 on D-EG.

Of course, one can change the thresholds in the above definition and therefore shift the regimes to different regions of the tolerance range $(0, 100]$. Based on our definition and Fig. 9, **Azure** can meet the low tolerance regime for the datasets D-SCH and D-EG, the middle tolerance regime for the datasets D-AEA and D-KDD1, and the high tolerance regime for the dataset D-KDD2. In contrast, **Amazon** only meets the high tolerance regime on the dataset D-KDD1.

To better understand the performance of **Azure** with respect to different quality tolerance regimes, we further present in Fig. 15 a "heat map" that indicates the quality tolerance levels met by different **Azure** models on different datasets (or, in terms of the capacity of models, the heat map

[3] That is, when users can only be satisfied by winning the Kaggle competition or being ranked among the top 1%.

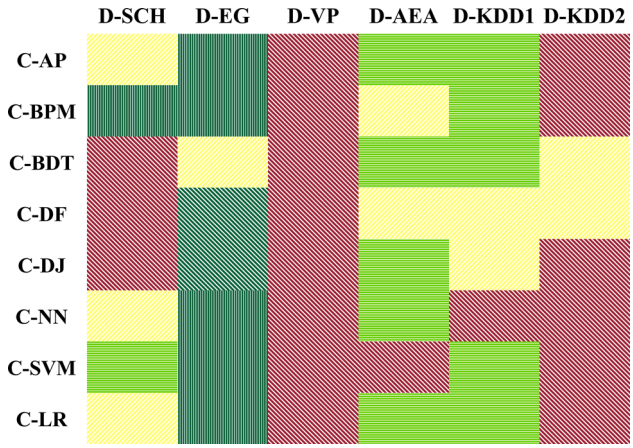

Fig. 15 A heat map that represents the capacity of different models on different datasets. Dark green represents low tolerance, light green represents middle tolerance, yellow represents high tolerance, and red regions are out of the tolerance regimes we defined (color figure online)

represents model capacity across different Kaggle competitions).

The dark green regions correspond to the low tolerance regime, the light green regions correspond to the middle tolerance regime, and the yellow regions correspond to the high tolerance regime. The other regions are outside the tolerance regimes we defined. We find that **Azure** actually only meets the low tolerance regime on small datasets, where linear models work well. **Azure** can meet the middle and high tolerance regimes on large datasets, thanks to the inclusion of nonlinear models.

2.4.7 Summary and discussion

Given the previous analysis, there is an obvious trade-off between the efficiency and effectiveness of machine learning clouds from a user's perspective. The more alternative models a machine learning cloud provides, the more likely it is that a better model can be found for a particular machine learning task. However, model selection becomes more challenging and users may spend more time (and money) on finding the most effective model.

Meanwhile, we also find that there is a gap between the best available model on machine learning clouds and the winning code available on Kaggle for certain machine learning tasks. It is then natural to ask the question of how to narrow the gap to *further improve machine learning clouds.* Of course, there is no reason to disbelieve that there is a possibility. For example, one can simply provide more models to increase the chance of finding a better model, though this may make model selection even harder. It is also not so clear which models should be included, given the trade-off between the capacity of a model and the training time required to tune the model.

Dataset	Best **Azure**	Winning	Quality Gap	Ranking Gap (%)
D-EG	C-AP	LR	-0.01	-0.3 (No. 4 → 2)
D-SCH	C-BPM	DWD	-0.01	0
D-KDD1	C-LR	Ensemble	0.02	2.12
D-AEA	C-BDT	Ensemble	0.01	1.6
D-KDD2	C-BDT	Ensemble	0.05	6.57
D-VP	C-BDT	NA	0.11	31.16

Fig. 16 Gaps between **Azure** and Kaggle winning code on different datasets (DWD is shorthand for "distance weighted discrimination")

We investigated this question from a different viewpoint by looking into the gap itself. Instead of asking how to make the gap narrower, we ask *why* there is a gap. Figure 16 compares the best performing **Azure** model with the winning code from Kaggle. Again, we separate small datasets (D-EG and D-SCH), where linear models outperform nonlinear models, from large datasets, where nonlinear models are better. The "Quality Gap" column presents the difference in AUC between the winning code and the **Azure** model, and the "Ranking Gap" column shows the corresponding movement in rankings on the Kaggle leader board. For example, on D-EG, the winning code is actually slightly worse than C-AP from **Azure**, with a quality gap of -0.01 and a ranking gap of -0.32%: The winning code is ranked fourth (i.e., top 0.64%), whereas C-AP could be ranked second (i.e., top 0.32%). The larger the quality gap and ranking gap are, the more potential improvement there is. One prominent observation from Fig. 16 is that the winning code on the large datasets leverages ensemble methods (details in Sect. 2.2.3), whereas the best nonlinear models from **Azure** (C-BDT, C-DJ, C-DF) more or less leverage ensemble methods as well. Therefore, it seems that **Azure** is moving in the right direction by supporting more ensemble methods, though it needs to further improve their performance. **Amazon** may need to incorporate ensemble methods (as well as nonlinear models).

2.5 Results on all datasets

So far, our study has been focused on Kaggle competitions whose winning code is available. We now discuss the insights we got by analyzing all 18 datasets.

The Importance of Feature Engineering

As most of the winning code spends significant effort on feature engineering, there is a clear gap from the typical way that people use machine learning clouds in practice, where feature engineering may not be at the level achieved by the winning code. Consequently, our previous results for the machine learning clouds may be over-optimistic. In practice, neither **Azure** or **Amazon** provides feature engineering functionality. We assess the impact of feature engineering and make the case for a potentially promising research direction of "declarative feature engineering on the cloud."

Dataset	C-AP	C-BPM	C-BDT	C-DF	C-DJ	C-LR	C-NN	C-SVM
D-SCH	0.00 (0.00%)	0.00 (0.00%)	0.00 (0.00%)	0.00 (0.00%)	0.00 (0.00%)	0.00 (0.00%)	0.00 (0.00%)	0.00 (0.00%)
D-EG	0.03 (3.81%)	0.03 (3.36%)	0.02 (2.04%)	0.02 (2.33%)	0.02 (2.63%)	0.03 (3.82%)	0.03 (3.83%)	0.04 (4.48%)
D-VP	0.02 (3.89%)	0.06 (9.75%)	0.07 (10.71%)	0.11 (17.64%)	0.12 (20.66%)	0.04 (5.41%)	0.08 (12.25%)	0.15 (27.24%)
D-AEA	0.04 (5.14%)	0.07 (8.42%)	0.05 (5.73%)	0.04 (5.26%)	0.12 (15.31%)	0.03 (3.88%)	0.04 (4.46%)	0.04 (4.74%)
D-KDD2	-0.01 (-1.31%)	-0.04 (-8.08%)	NA (NA)	0.05 (9.07%)	0.06 (11.80%)	-0.02 (-3.06%)	NA (NA)	0.03 (4.76%)
D-KDD1	0.06 (10.60%)	0.09 (15.35%)	NA (NA)	0.06 (10.86%)	0.10 (19.25%)	0.05 (8.64%)	NA (NA)	0.10 (17.63%)

Fig. 17 Improvement on private AUC score attributed to feature engineering. Green indicates quality increase after feature engineering; red otherwise. All numbers are with hyper-parameter tuning (color figure online)

Dataset	C-AP	C-BPM	C-BDT	C-DF	C-DJ	C-LR	C-NN	C-SVM
D-SCH	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
D-EG	386 (61.76%)	378 (60.48%)	320 (51.20%)	104 (16.64%)	97 (15.52%)	386 (61.76%)	384 (61.44%)	398 (63.68%)
D-VP	59 (4.52%)	349 (26.72%)	432 (33.08%)	627 (48.01%)	569 (43.57%)	280 (21.44%)	527 (40.35%)	544 (41.65%)
D-AEA	759 (44.99%)	838 (49.67%)	773 (45.82%)	742 (43.98%)	986 (58.45%)	665 (39.42%)	726 (43.03%)	468 (27.74%)
D-KDD2	-62 (-13.14%)	-85 (-18.01%)	NA (NA)	244 (51.69%)	301 (63.77%)	-121 (-25.64%)	NA (NA)	157 (33.26%)
D-KDD1	118 (25.00%)	289 (61.23%)	NA (NA)	257 (54.45%)	348 (73.73%)	65 (13.77%)	NA (NA)	343 (72.67%)

Fig. 18 Improvement on private ranking attributed to feature engineering. Green indicates quality increase after feature engineering; red otherwise. All numbers are with hyper-parameter tuning (color figure online)

We consider an extreme case where we do not perform feature engineering, and ask the question: *If we use raw features instead of features constructed by the winning code, how will it impact the performance of machine learning clouds?* In Figs. 17 and 18, we present the improvement in terms of the AUC score and the ranking on the private leader board by using the features from the winning code versus using the raw features (without feature engineering). Hyper-parameter tuning was turned on for each run. A negative value here indicates a drop in performance. Not surprisingly, in most cases using well engineered features helps boost performance significantly, though it is not always the case. For instance, for C-LR on D-KDD2, using features from the winning code decreases the AUC score by 0.03, and the corresponding ranking on the private leader board drops by 129. The last columns in Figs. 17 and 18 further show the improvement by the best model using engineered features versus that using raw features. Even under this best-versus-best comparison, the benefit of feature engineering is significant.

We also should not be overly pessimistic by the results, though. After all, in practice it is rare for people to completely give up feature engineering, given the intensive and extensive research on feature selection in the literature. Consequently, our comparison on using only raw features should be understood as a worst-case study for the performance of machine learning clouds. Meanwhile, it is interesting to further explore the "gray areas" between the two extremes that we have studied in terms of feature engineering: Instead of using either fine-tuned features or just raw features, how will machine learning clouds perform when combined with an automatic feature learning procedure? There is apparently a broad spectrum regarding the abundance of feature learning algorithms. After we finished `mlbench`, many major clouds providers include feature engineering in their services. This phenomenon motivates our second benchmark in Sect. 3 that study the effectiveness of feature learning offerings on the cloud.

The Importance of Hyper-parameter Tuning

We assess the importance of hyper-parameter tuning in a similar way and the result for all 18 dataset is in Fig. 19. We see that hyper-parameter tuning has significant impact on each individual algorithm, and most of the time, it improves the quality. One interesting observation is that, under the best-versus-best comparison (last column), the impact of hyper-parameter tuning drops significantly. This shows an interesting trade-off between *model selection* and *hyper-parameter tuning*—disabling model selection hurts the quality more significantly than disabling hyper-parameter tuning. This opens another interesting future research question: *Given limited computation budget, how should one balance between model selection and hyper-parameter tuning to maximize the final quality?* Similar to the previous challenge on feature engineering, there are lots of trade-offs. A naive approach is to perform hyper-parameter tuning for all models and then pick the model with the best performance, exactly as what we have done in our experimental evaluation. This brute-force approach is perhaps unacceptable in a situation with restrictive resource access. As another extreme approach, one can first find a model using model selection without any hyper-parameter tuning and then focus on hyper-parameter tuning for this particular model. However, there might be little guarantee on the performance of the model selected. Clearly, there are numerous hybrid strategies in between, where one can first decide on a set of candidate models and then perform hyper-parameter tuning for each candidate. A declarative machine learning service should hide these details from the user.

Dataset	C-AP	C-BPM	C-BDT	C-DF	C-DJ	C-LR	C-NN	C-SVM
D-SCH-r	-0.03 (-3.60%)	NA (NA)	0.04 (6.79%)	0.02 (2.39%)	0.11 (15.87%)	0.03 (3.92%)	-0.01 (-0.81%)	0.04 (4.47%)
D-PIS-r	0.00 (0.00%)	NA (NA)	0.01 (1.53%)	0.03 (3.93%)	0.01 (0.89%)	0.05 (6.49%)	0.01 (0.77%)	0.06 (8.17%)
D-EG-r	0.00 (0.16%)	NA (NA)	0.01 (1.12%)	0.01 (0.81%)	0.03 (3.25%)	-0.00 (-0.04%)	0.00 (0.16%)	0.02 (2.65%)
D-VP-r	0.00 (0.00%)	NA (NA)	0.02 (3.66%)	0.02 (2.84%)	-0.00 (-0.32%)	-0.02 (-3.40%)	0.02 (3.07%)	-0.02 (-2.94%)
D-AEA-r	-0.00 (-0.08%)	NA (NA)	0.04 (4.20%)	0.10 (13.74%)	0.06 (8.68%)	0.03 (3.22%)	0.01 (1.34%)	0.06 (8.09%)
D-SCS-r	0.00 (0.18%)	NA (NA)	0.01 (1.00%)	0.08 (10.65%)	0.01 (1.57%)	0.01 (0.80%)	0.01 (1.45%)	-0.01 (-1.56%)
D-SMR-r	0.00 (0.66%)	NA (NA)	NA (NA)	0.05 (7.99%)	0.07 (10.89%)	NA (NA)	NA (NA)	NA (NA)
D-GMC-r	0.04 (5.77%)	NA (NA)	0.00 (0.15%)	0.09 (11.39%)	0.00 (0.30%)	0.02 (2.56%)	0.00 (0.08%)	-0.00 (-0.09%)
D-HQC-r	0.00 (0.00%)	NA (NA)	0.00 (0.18%)	0.03 (3.37%)	NA (NA)	0.00 (0.00%)	0.01 (1.05%)	0.01 (0.61%)
D-KDD-r	0.00 (0.43%)	NA (NA)	NA (NA)	0.05 (10.68%)	0.03 (6.83%)	0.01 (1.26%)	NA (NA)	-0.01 (-0.95%)
D-PBV-r	0.00 (0.00%)	NA (NA)	NA (NA)	NA (NA)	NA (NA)	0.00 (0.13%)	NA (NA)	NA (NA)
D-SCH	-0.03 (-3.60%)	NA (NA)	0.04 (6.79%)	0.02 (2.39%)	0.11 (15.87%)	0.03 (3.92%)	-0.01 (-0.81%)	0.04 (4.47%)
D-EG	0.00 (0.42%)	NA (NA)	0.01 (0.82%)	0.03 (3.37%)	0.00 (0.53%)	0.00 (0.22%)	0.01 (0.88%)	0.00 (0.41%)
D-VP	-0.05 (-6.70%)	NA (NA)	0.04 (6.18%)	0.09 (14.48%)	0.05 (7.88%)	0.00 (0.63%)	0.05 (7.56%)	0.01 (2.10%)
D-AEA	0.00 (0.03%)	NA (NA)	0.01 (1.06%)	0.07 (8.76%)	0.01 (1.28%)	-0.00 (-0.15%)	0.04 (4.43%)	0.00 (0.40%)
D-PHY	NA (NA)	NA (NA)	NA (NA)	NA (NA)	NA (NA)	NA (NA)	NA (NA)	NA (NA)
D-KDD2	0.00 (0.66%)	NA (NA)	0.00 (0.15%)	0.07 (13.96%)	0.09 (17.19%)	-0.00 (-0.26%)	NA (NA)	-0.00 (-0.75%)
D-KDD1	0.03 (5.44%)	NA (NA)	-0.02 (-3.07%)	0.04 (6.27%)	0.06 (10.96%)	0.02 (3.87%)	0.00 (0.34%)	0.03 (4.84%)

Fig. 19 Improvement on private AUC score attributed to hyper-parameter tuning. Green indicates quality increase after feature engineering; red otherwise (color figure online)

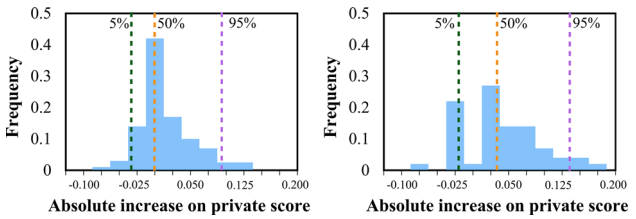

Fig. 20 AUC improvement attributed to (left) hyper-parameter tuning and (right) feature engineering

We can also compare the impact of hyper-parameter tuning and feature engineering by plotting the histograms for the performance improvement for all individual algorithms. In Fig. 20, the red, green, and blue vertical dashed lines in each figure indicate the 5th, 50th, and 90th percentiles, respectively. We see that the majority of the improvements falls between 0 and 10% in terms of AUC score. Comparing all these three lines "shifted to the right direction" for the feature engineering plot (i.e., the right histogram), indicating a more significant impact attributed to feature engineering than hyper-parameter tuning.

3 The `automlbench` benchmark

In recent years, machine learning cloud has been under rapid development with functionalities going beyond model training and hyper-parameter tuning. Major cloud platforms offer (optional) AutoML components in the training, e.g., feature preprocess, feature selection, model selection, hyper-parameter tuning, and model ensemble. These AutoML techniques could potentially free users from manual tuning of both algorithm and system knobs, which entails a high barrier for non-experts and causes expensive costs even for experienced practitioners. *How do machine learning clouds perform today, five years after mlbench*? Specifically:

> How far away today's AutoML systems are from experienced data scientists (e.g., top-ranked Kaggle solutions)? What is the main shortcoming of the current AutoML offerings?

To answer this question, as a follow up to `mlbench`, we conduct, in 2021, the `automlbench` benchmark to compare the AutoML capabilities of several popular machine learning clouds.

Baselines We use four commercial machine learning clouds in this benchmark—Google Cloud, Amazon AWS, Microsoft Azure, and Oracle DataScience.

Workloads The evaluated datasets are presented in Fig. 21.

- *Classification dataset* The details of the evaluated Kaggle datasets have been described in Sect. 2.2.3. We use their raw features to train classification models.
- *Noisy dataset* To assess how the baselines handle noisy input, we choose two noisy datasets, KDD and EEG. These two datasets are from our previous benchmark, i.e., CleanML [32]. This contains a raw version and a clean version for each dataset. In KDD-missing,[4] 75.9% of the rows have missing values. KDD-clean removes rows with missing values from KDD-missing. EEG-outlier is an

[4] https://www.kaggle.com/c/kdd-cup-2014-predicting-excitement-at-donors-choose.

Datasets	# instance	# features	# class
D-PIS-r	5,500	22	2
D-VP-r	10,506	11	2
D-AEA-r	32,769	9	2
D-SCS-r	76,020	369	2
D-PBV-r	2,197,291	12	2
D-KDD-r	619,326	139	2
D-GMC-r	150,000	10	2
D-EG-r	7,395	124	2
D-HQC-r	260,753	297	2
D-SMR-r	145,231	1,933	2
KDD-missing	131,329	100	2
KDD-clean	31,709	100	2
EEG-outlier	14,980	14	2
EEG-clean	14,980	14	2
MNIST	70,000	784	10
HIGGS	11 million	28	2
RCV	82,853	47,236	2
BikeSharing	17,379	16	NA
BlogFeadback	59,961	280	NA
MSD	381,746	90	NA
CT	53,500	385	NA
SuperConductivity	15,767	81	NA
UJIndoorLoc	19,937	524	NA

Fig. 21 Datasets used in `automlbench`

electroencephalography dataset that contains outliers,[5] and EEG-clean replaces the outliers via SD criterion and mean imputation according to [32].
- *Image dataset* MNIST is chosen as an example to evaluate the baselines over image dataset.
- *Large-scale dataset* We use two large-scale datasets, Higgs and RCV, from UCI Machine Learning Repository [1] to test the scalability of the baseline.
- *Regression dataset* Except for classification tasks, we also run regression tasks over six regression datasets collected from UCI Machine Learning Repository—BikeSharing, BlogFeadback, MSD, CT, SuperConductivity, and UJIndoorLoc.

Principles Throughout this benchmark, we follow the *best vs. best* principle. For each baseline, we select all the provided AutoML components, as shown in Fig. 22. The chosen platforms provide various scopes of AutoML components, giving us possibility to assess the importance of each AutoML component. Some of the baseline allow user to set the task parallelism (usually has a maximal limitation), while some others do not. To be fair, we use the maximal allowed parallelism for each platform.

Protocols If the evaluated dataset does not originally provide a train subset and a test subset, we split the dataset by 0.9/0.1, in which 90% is used for training and 10% for testing. For each task, we run fivefold cross-validation over the train subset to assure the generality of the results. The platforms

[5] https://archive.ics.uci.edu/ml/datasets/EEG+Eye+State.

Platform	Data Cleaning	Feature Preprocess	Feature Selection	Model Selection	Model Tuning	Model Ensemble
Google	✓	✓		✓	✓	✓
Amazon	✓	✓		✓	✓	
Microsoft	✓	✓		✓	✓	✓
Oracle	✓	✓	✓	✓	✓	

Fig. 22 AutoML components in machine learning clouds

offer two stopping conditions—(1) the trained model is considered as converged (e.g., loss does not significantly change within a few iterations), and (2) reaching the time budget. To assure a fair comparison, we set a maximal time budget for all the platforms, as we will specify in each experiment.

Metrics We report both efficiency metric (run time) and statistical metric (validation/test accuracy or AUC). Considering that the baselines have different pricing models, we also report the cost (in dollar) for each task. Note that the cost may not be a scientific metric since the cloud provider may decide their pricing policies according to commercial purposes.

3.1 Results on classification datasets

We run the classification datasets on four platforms. We vary the maximal time budget from 30 min to 3 h, and present the AUC results in Figs. 23, 24, and 25. Our results are organized into two orthogonal dimensions:

- *Smaller datasets and larger datasets* Among these datasets, D-PVB-r and D-KDD-r have relatively more data examples, whereas other datasets are relatively smaller.
- *Holdout testset and Kaggle testset* In Figs. 23, 24 and 25, the testsets are generated as a holdout set of the original training set, whereas in Figs. 45 and 46 we use the private testsets on Kaggle.

We obtain different observations for different regimes along these two dimensions. Figure 30 illustrates the output pipelines of different platforms.

For **smaller datasets** and **holdout testset**, ORACLE DATASCIENCE performs relatively well—not only it achieves the best or second-best test scores on six out of ten datasets (Fig. 25), the fact that it applies data sampling and feature selections (see output pipelines in Fig. 26) also means that it is the fastest (cheapest) (see Figs. 27, 29, 30) among all four platforms. On the other hand, GOOGLE CLOUD achieves the highest validation AUC over eight datasets, however, does seem to overfit when it comes to the testset. Particularly, it encounters prediction error given a time budget of 3 h, as GOOGLE CLOUD outputs a complex model that cannot handle unseen features in the holdout testsets. GOOGLE CLOUD generates an ensemble model for each workload,

Platform		D-PIS-r	D-VP-r	D-AEA-r	D-SCS-r	D-PBV-r	D-KDD-r	D-GMC-r	D-EG-r	D-HQC-r	D-SMR-r
Google	val.	- (-)	- (-)	- (-)	- (-)	- (-)	- (-)	- (-)	- (-)	- (-)	- (-)
	test	- (-)	- (-)	- (-)	- (-)	- (-)	- (-)	- (-)	- (-)	- (-)	- (-)
Amazon	val.	- (-)	0.9097 (-0.0296)	0.9187 (-0.0074)	- (-)	- (-)	- (-)	0.8669 (0.0)	0.8631 (-0.0075)	- (-)	- (-)
	test	- (-)	0.9240 (0.0198)	0.9261 (-0.0296)	- (-)	- (-)	- (-)	0.8684 (-0.0016)	0.8676 (-0.0135)	- ()	- ()
Microsoft	val.	0.9364 (0.0)	0.9351 (-0.0042)	0.8932 (-0.0329)	- (-)	- (-)	- (-)	0.8651 (-0.0018)	0.8706 (0.0)	- (-)	- (-)
	test	0.9286 (-0.0312)	0.9381 (-0.0057)	0.8831 (-0.0726)	- (-)	- (-)	- (-)	0.8700 (0.0)	0.8811 (0.0)	- (-)	- (-)
Oracle	val.	0.9278 (-0.0086)	0.9393 (0.0)	0.9261 (0.0)	0.8841 (0.0)	- (-)	0.8064 (0.0)	0.8638 (-0.0031)	0.758 (-0.1126)	0.9601 (0.0)	- (-)
	test	0.9598 (0.0)	0.9438 (0.0)	0.9557 (0.0)	0.9008 (0.0)	- (-)	0.8078 (0.0)	0.8693 (-0.0007)	0.7208 (-0.1603)	0.9644 (0.0)	- (-)

Fig. 23 Validation and test AUC of classification datasets in `automlbench` (time budget = 30 min). Green numbers indicate the best validation AUC for each dataset, blue numbers indicate the best test AUC, and "–" means time out. The numbers in brackets indicate the difference compared with the best results (color figure online)

Platform		D-PIS-r	D-VP-r	D-AEA-r	D-SCS-r	D-PBV-r	D-KDD-r	D-GMC-r	D-EG-r	D-HQC-r	D-SMR-r
Google	val.	0.863 (-0.091)	0.956 (0.0)	0.962 (0.0)	0.983 (0.0)	0.998 (0.0)	0.976 (0.0)	0.979 (0.0)	0.851 (-0.0204)	0.98 (0.0)	- (-)
	test	0.8831 (-0.0767)	0.8929 (-0.0519)	0.9035 (-0.0522)	0.8782 (-0.0457)	0.9999 (0.0)	0.8244 (0.0)	0.8699 (-0.0003)	0.8739 (-0.0056)	0.9657 (-0.001)	- (-)
Amazon	val.	0.9469 (-0.007)	0.9170 (-0.039)	0.9333 (-0.0287)	0.9101 (-0.0729)	0.9617 (-0.0363)	0.8059 (-0.1701)	0.8691 (-0.1099)	0.8689 (-0.0026)	0.9640 (-0.016)	- ()
	test	0.9355 (-0.0243)	0.9448 (0.0)	0.8993 (-0.0564)	0.9125 (-0.0114)	0.9614 (-0.0385)	0.8041 (-0.0203)	0.8676 (-0.0026)	0.8720 (-0.0075)	0.9667 (0.0)	- ()
Microsoft	val.	0.9539 (0.0)	0.9375 (-0.0185)	0.919 (-0.043)	0.9111 (-0.0719)	0.9539 (-0.0441)	0.8048 (-0.1712)	0.8658 (-0.1132)	0.8714 (0.0)	0.9646 (-0.0154)	- (-)
	test	0.9461 (-0.0137)	0.9429 (-0.0019)	0.9316 (-0.0241)	0.9239 (-0.2984)	0.9537 (-0.0462)	0.8054 (-0.019)	0.8702 (0.0)	0.8795 (0.0)	0.8733 (-0.0934)	- (-)
Oracle	val.	0.9278 (-0.0261)	0.9393 (-0.0167)	0.9261 (-0.0359)	0.8843 (-0.0989)	0.9944 (-0.0036)	0.8064 (-0.1696)	0.8638 (-0.1152)	0.758 (-0.1134)	0.9601 (-0.0199)	- (-)
	test	0.9598 (0.0)	0.9438 (-0.001)	0.9557 (0.0)	0.9076 (-0.0163)	0.9899 (-0.01)	0.8079 (-0.0165)	0.8693 (-0.0009)	0.7208 (-0.1587)	0.9644 (-0.0023)	- (-)

Fig. 24 Validation and test AUC of classification datasets in `automlbench` (time budget = 1 h). Green numbers indicate the best validation AUC for each dataset, blue numbers indicate the best test AUC, and "–" means time out. The numbers in brackets indicate the difference compared with the best results (color figure online)

Platform		D-PIS-r	D-VP-r	D-AEA-r	D-SCS-r	D-PBV-r	D-KDD-r	D-GMC-r	D-EG-r	D-HQC-r	D-SMR-r
Google	val.	0.863 (-0.091)	0.956 (0.0)	0.965 (0.0)	0.984 (0.0)	0.999 (0.0)	0.976 (0.0)	0.978 (0.0)	0.881 (0.0)	0.981 (0.0)	- -
	test	0.8831 (-0.0767)	0.8929 (-0.0519)	0.8995 (-0.0562)	0.8752 (-0.0685)	0.9998 (0.0)	0.8247 (0.0)	NA NA	NA NA	0.9662 (-0.0014)	- -
Amazon	val.	0.9469 (-0.007)	0.9170 (-0.039)	0.9333 (-0.0317)	0.9101 (-0.0739)	0.9978 (-0.0012)	0.8089 (-0.1671)	0.8696 (-0.1084)	0.8727 (-0.0083)	0.9653 (-0.0157)	0.7772 (-0.0065)
	test	0.9355 (-0.0243)	0.9448 (0.0)	0.8993 (-0.0564)	0.9125 (-0.0312)	0.9977 (-0.0021)	0.8077 (-0.017)	0.8689 (-0.0007)	0.8772 (-0.0017)	0.9676 (0.0)	0.7855 (-0.0026)
Microsoft	val.	0.9539 (0.0)	0.9375 (-0.0185)	0.919 (-0.046)	0.9165 (-0.0675)	0.9632 (-0.0358)	0.8066 (-0.1694)	0.8648 (-0.1132)	0.8715 (-0.0095)	0.9648 (-0.0162)	0.7837 (0.0)
	test	0.9461 (-0.0137)	0.9429 (-0.0019)	0.9316 (-0.0241)	0.9315 (-0.0122)	0.9638 (-0.036)	0.8054 (-0.0193)	0.8696 (0.0)	0.8789 (0.0)	0.9028 (-0.0648)	0.7881 (0.0)
Oracle	val.	0.9278 (-0.0261)	0.9393 (-0.0167)	0.9261 (-0.0389)	0.9063 (-0.0777)	0.9975 (-0.0015)	0.8064 (-0.1696)	0.8638 (-0.1142)	0.758 (-0.123)	0.9601 (-0.0209)	0.7735 (-0.0102)
	test	0.9598 (0.0)	0.9438 (-0.001)	0.9557 (0.0)	0.9437 (0.0)	0.9848 (-0.015)	0.8079 (-0.0168)	0.8693 (-0.0003)	0.7208 (-0.1581)	0.9644 (-0.0032)	0.7805 (-0.0076)

Fig. 25 Validation and test AUC of classification datasets in `automlbench` (time budget = 3 h). Green numbers indicate the best validation AUC for each dataset, blue numbers indicate the best test AUC, "–" means time out, and "NA" means prediction error on the test-set. The numbers in brackets indicate the difference compared with the best results (color figure online)

Platform	D-PIS-r	D-VP-r	D-AEA-r	D-SCS-r	D-PBV-r	D-KDD-r
Google	VotingEnsemble	VotingEnsemble	VotingEnsemble	VotingEnsemble	VotingEnsemble	VotingEnsemble
Amazon	OneHotEncoder → StandardScalar → XGBoost	OneHotEncoder → TF-IDF → StandardScalar → XGBoost	StandardScalar → XGBoost	LogExtreme → PCA → StandardScalar → LinearLearner	OneHotEncoder → PCA → StandardScalar → XGBoost	QuantileExtreme → OneHotEncoder → PCA → StandardScalar → LinearLearner
Microsoft	VotingEnsemble	MaxAbsScalar → LightGBM	StackEnsemble	StackEnsemble	VotingEnsemble	MaxAbsScalar → LightGBM
Oracle	DataSampling → FeatureSelection → ExtraTrees	DataSampling → FeatureSelection → KNeighbors	DataSampling → FeatureSelection → AdaBoost	DataSampling → FeatureSelection → AdaBoost	DataSampling → FeatureSelection → RandomForest	DataSampling → FeatureSelection → AdaBoost

Platform	D-GMC-r	D-EG-r	D-HQC-r	D-SMR-r
Google	VotingEnsemble	VotingEnsemble	VotingEnsemble	VotingEnsemble
Amazon	RobustImputer → OneHotEncoder → StandardScaler XGBoost	RobustImputer → OneHotEncoder → TfidfVectorizer → StandardScaler XGBoost	RobustImputer → OneHotEncoder → PCA → StandardScaler XGBoost	RobustImputer → OneHotEncoder → TfidfVectorizer → StandardScaler XGBoost
Microsoft	VotingEnsemble	VotingEnsemble	VotingEnsemble	StackEnsemble
Oracle	DataSampling → FeatureSelection → XGBClassifier	DataSampling → FeatureSelection → LGBMClassifier	DataSampling → FeatureSelection → LGBMClassifier	DataSampling → FeatureSelection → XGBClassifier

Fig. 26 Output ML pipeline of classification datasets in `automlbench`

Platform	D-PIS-r	D-VP-r	D-AEA-r	D-SCS-r	D-PBV-r	D-KDD-r	D-GMC-r	D-EG-r	D-HQC-r	D-SMR-r
Google	56min $18.14	44min $14.17	1.6h $30.76	3h $57.96	3.5h $67.62	3.5h $67.62	2.8h $54.10	2.5h $48.3	3.4h $65.69	-
Amazon	1h $1.08	40min $0.72	47min $0.84	53min $0.95	3h $3.23	3h $3.23	2h $2.15	2.5h $2.69	3h $3.23	3h $3.23
Microsoft	1h $0.76	46min $0.58	45min $0.57	2h $1.52	4.6h $3.5	3.5h $2.66	4.3h $3.27	3.3h $2.51	4.6h $3.50	4.8h $3.65
Oracle	8.5min $0.018	1min $0.002	5.7min $0.012	1.7h $0.213	1.61h $0.82	36min $0.31	5.6min $0.011	3min $0.006	23min $0.20	3.8h $1.94

Fig. 27 Runtime and cost of classification datasets in `automlbench`. Note that some executions are longer than the time budget (3 h) because the platforms need to finish the existing trials. Green numbers indicate the shortest runtime for each dataset, blue numbers indicate the lowest cost, and "–" means time out (color figure online)

which prefers DNNs, over the original features. We hypothesize that this preference of DNN model makes the overfitting problem more serious on smaller datasets—in fact, as we will see, when the dataset gets larger, the relative performance of GOOGLE CLOUD and ORACLE DATASCIENCE will also change.

For **larger datasets** and **holdout testset** (D-PBV-r and D-KDD-r), however, we obtain quite different observations. In this case, GOOGLE CLOUD outperforms all other platforms. Later, we will see similar behavior on other large-scale datasets (see Higgs in Fig. 39, 1.2 AUC points compared with the second-best platform). We hypothesize that, on these large-scale datasets, the benefit of DNNs starts to show up.

In the previous experiments, we set a time budget and the platforms encounter time out on some datasets. To show the model quality when the model is converged, we use more time budget for these datasets and run these workloads until convergence. As shown in Fig. 28, GOOGLE CLOUD achieves slight improvements given a larger time budget. Overall, GOOGLE CLOUD outperforms other platforms on larger datasets but still suffers from the overfitting problem. AMAZON AWS, MICROSOFT AZURE, and ORACLE DATASCIENCE have higher model qualities, at the expense of significantly more time. However, it is the user's choice whether it is beneficial to obtain a better model with more time consumption and monetary cost.

When the testset is from **Kaggle leaderboard** (see Fig. 45 shows all the results), however, the observations are also different. AMAZON AWS outperform significantly over GOOGLE CLOUD on smaller datasets, by a margin of up to 3 AUC points, while the observation is opposite on larger datasets. AMAZON AWS and MICROSOFT AZURE outperform ORACLE DATASCIENCE significantly, by a margin of up to 13 AUC points. We hypothesize that the unknown private testsets on Kaggle might be from more different data distributions than the holdout testsets. Therefore, the strategies

Platform	Dataset	val. AUC	test AUC	time
Google	D-SCS-r	0.985 (0.0)	0.8748 (-0.0689)	3.8h
	D-PBV-r	0.999 (0.0)	0.9998 (0.0)	4.6h
	D-KDD-r	0.977 (0.0)	0.8248 (0.0)	6.2h
	D-HQC-r	0.983 (0.0)	0.966 (-0.0016)	5.3h
Amazon	D-PBV-r	0.9983 (-0.001)	0.998 (-0.0018)	7.3h
	D-KDD-r	0.8096 (-0.1664)	0.8089 (-0.0158)	4.9h
	D-HQC-r	0.9668 (-0.0142)	0.9682 (0.0)	8.2h
	D-SMR-r	0.7826 (-0.0026)	0.7876 (-0.0015)	12.3h
Microsoft	PBV	0.9798 (-0.0192)	0.9795 (-0.0203)	8.7h
	D-KDD-r	0.8085 (-0.1675)	0.8079 (-0.0168)	5.1h
	D-GMC-r	0.8656 (-0.1124)	0.8699 (0.0)	7.7h
	D-EG-r	0.8717 (-0.0093)	0.8796 (0.0)	4.8h
	D-HQC-r	0.965 (-0.016)	0.9035 (-0.0641)	6.5h
	D-SMR-r	0.7852 (0.0)	0.7891 (0.0)	8.1h
Oracle	D-SMR-r	0.7847 (-0.0005)	0.7863 (-0.0028)	7.4h

Fig. 28 Validation AUC, test AUC, and runtime until convergence. These workloads encounter time out given a time budget of 3 h. The numbers in brackets indicate the difference compared with the best results

that take into consideration all data examples and features, without sampling, might be able to construct a more robust estimator.

Observation 3 *(Classification Dataset)* ***Platforms with different design principles perform well in different scenarios.*** *Overall, appropriate data subsampling and feature selection (*ORACLE DATASCIENCE*) are great ways to making training significantly faster; deep learning-based methods (*GOOGLE CLOUD*) are great for complex, large-scale tasks when overfitting is less of a concern; and* AMAZON AWS *and* MICROSOFT AZURE*, which do not rely on data subsampling and feature selection, seem to be more robust in the presence of data drifting during test time.*

(Comparison with Kaggle Winner) As the continuous development of AutoML techniques, one question is frequently raised:

> Can a pure AutoML solution achieve compare accuracy as models delicately tuned by experienced data scientists?

We choose three datasets and use the models generated by these platforms to predict the stand-alone test datasets on Kaggle. We then upload the predictions to Kaggle and report the results returned by Kaggle in Fig. 31. Unfortunately, the AutoML platforms significantly lag behind the Kaggle winners. The AutoML platforms often rank between 50 and 80%. In the solutions submitted by the high-ranked users, most of them did a lot of work on feature engineering and knowledge integration. In addition to traditional feature transformations, they also created new features by combining raw features, which is not considered in the existing AutoML platforms. This kind of high-level feature engineering operator requires an in-depth understanding of raw features, both contextually

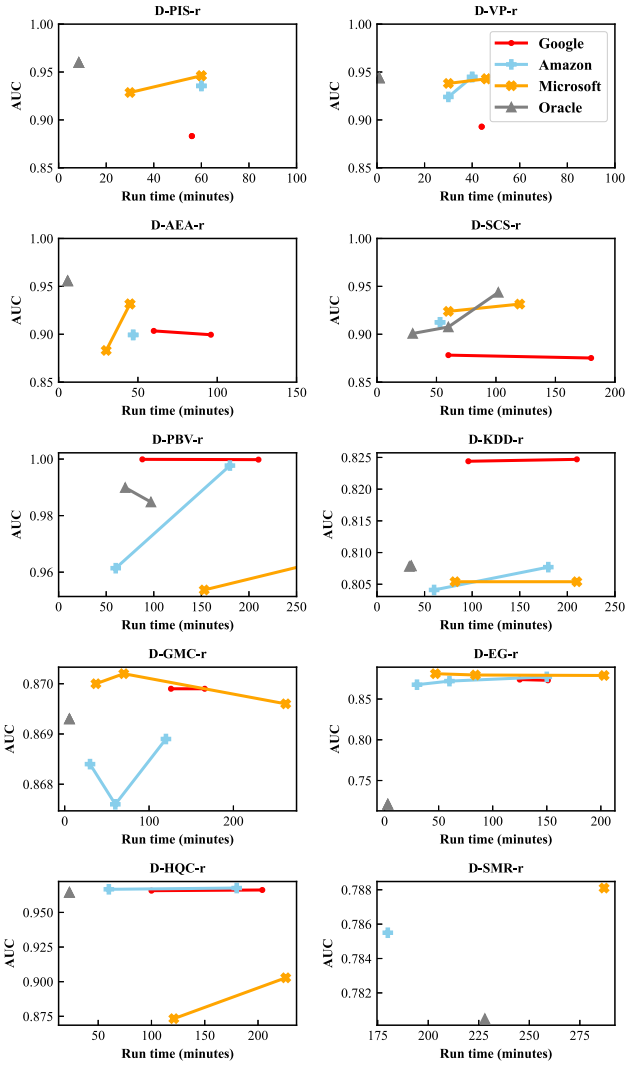

Fig. 29 Test AUC versus time budget on classification datasets. We set three time budgets (30 min, 1 h, 3 h) for all platforms

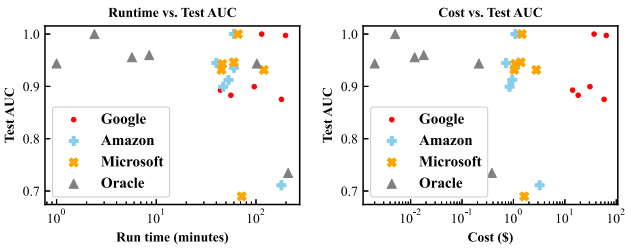

Fig. 30 Runtime (cost) versus test AUC of AutoML platforms on classification datasets

and physically, which is yet difficult for current AutoML techniques. Nevertheless, this does not mean AutoML techniques are not useful—as we have shown in the results of `mlbench`, given high-quality features, AutoML techniques can tune a model comparable with top-ranked Kaggle solutions and save considerable human efforts.

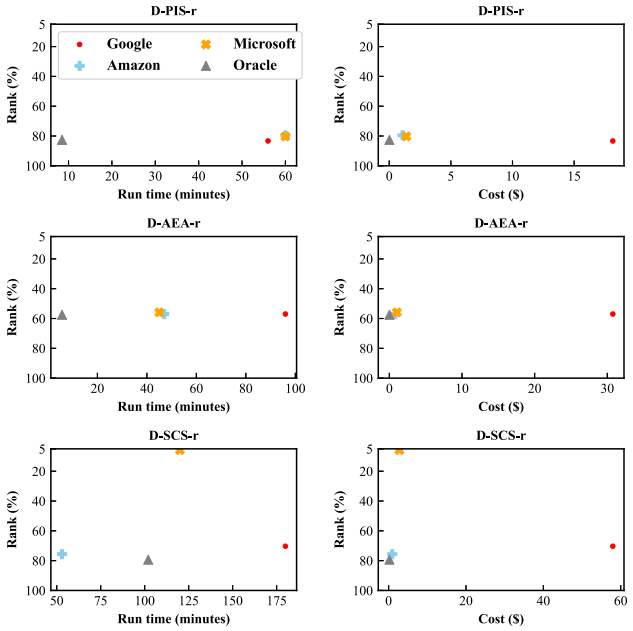

Fig. 31 Comparison with Kaggle winners on three representative classification datasets. The *x*-axis refers to runtime or cost of the platforms, and the *y*-axis refers to the ranking in the Kaggle leaderboard

Observation 4 *(Classification Dataset)* ***Machine learning problems are often data problems and knowledge problems.*** *It is important for future ML platforms to better support the human-in-the-loop development process to facilitate knowledge integration and data wrangling and transformations.*

3.2 Results on noisy dataset

The two noisy datasets represent different scenarios—KDD contains missing values, and EEG has outliers. We treat and discuss them separately.

– *Missing values* We train KDD-missing with the AutoML platforms and calculate the AUC over the validation dataset and the test dataset. The maximal time budget is set as 3 h. Then, we do the same for the KDD-clean dataset in which the rows with missing values are removed. The results are shown in Figs. 32, 33 and 34. Figure 32 shows the validation and test AUC on the noisy datasets. If we compare the results of KDD-missing and KDD-clean, the model accuracy is significantly higher on KDD-clean dataset in which we remove training data with missing values. For instance, MICROSOFT AZURE's test AUC on KDD-clean is 0.7239, while that on KDD-missing is only 0.7030. This proves that missing values pose negative impact on the trained model and the existing platforms cannot effectively handle missing values. On each of KDD-missing and KDD-clean, the models trained by different platforms perform similarly. Platforms with overfitting problems can suffer in this scenario. On the other hand, platforms that include imputer and feature processing in the pipeline perform better. It shows that that fixing missing values, even automatically, can help improve model accuracy. Figure 35 summarizes the combination of statistical metric (test AUC) and efficiency metrics (runtime and cost).

– *Outliers* On the raw dataset EEG-outlier, platforms that support model ensemble output better models, as shown in Fig. 32, by training ensemble models. EEG-clean is provided by CleanML [32] which manually modifies EEG-outlier by detecting outliers with SD criterion and imputing them with mean imputation. On this modified version, the validation AUC and test AUC of the evaluated platforms barely change. If we look at the output pipelines in Fig. 34, some platforms output ensemble models, while other platforms prefer tree models. The consistency of model qualities shows that these models are robust to outliers.

Platform		KDD-missing	KDD-clean	EEG-outlier	EEG-clean
Google	val.	0.928 (0.0)	0.941 (0.0)	0.998 (0.0)	0.998 (0.0)
	test	0.7071 (-0.0073)	0.7233 (-0.0012)	0.9988 (0.0)	0.9985 (0.0)
Amazon	val.	0.7078 (-0.2202)	0.9093 (-0.0317)	0.9504 (-0.0476)	0.9485 (-0.0495)
	test	0.7144 (0.0)	0.7245 (0.0)	0.9874 (-0.0113)	0.9965 (-0.002)
Microsoft	val.	0.7174 (-0.2106)	0.7261 (-0.2149)	0.9866 (-0.0114)	0.9841 (-0.0139)
	test	0.7030 (-0.0114)	0.7239 (-0.0006)	0.9930 (-0.0057)	0.9921 (-0.0064)
Oracle	val.	0.7063 (-0.2217)	0.7164 (-0.2246)	0.9669 (-0.0311)	0.9652 (-0.0328)
	test	0.7137 (-0.0007)	0.7200 (-0.0045)	0.9919 (-0.0068)	0.9915 (-0.007)

Fig. 32 Validation and test AUC of noisy datasets in `automlbench`. The numbers in brackets indicate the difference compared with the best results

Platform	KDD-missing	KDD-clean	EEG-outlier	EEG-clean
Google	3h $57.96	3h $57.96	60min $19.32	1.39h $26.85
Amazon	1.1h $1.18	45min $0.48	1.5h $1.61	1.5h $1.61
Microsoft	1.1h $1.53	1.2h $1.67	46min $1.06	39min $0.9
Oracle	18min $0.15	6.6min $0.06	2min $0.02	1.8min $0.02

Fig. 33 Runtime and cost of noisy datasets in `automlbench`. Green numbers indicate the shortest runtime for each dataset, and blue numbers indicate the lowest cost. The pricing is $19.32/h for GOOGLE CLOUD, $1.075/h for AMAZON AWS, $1.388/h for MICROSOFT AZURE, and $0.501 for ORACLE DATASCIENCE. We use a better instance for ORACLE DATASCIENCE according to the platform's recommendation (color figure online)

Platform	KDD-missing	KDD-clean	EEG-outlier	EEG-clean
Google	VotingEnsemble	VotingEnsemble	VotingEnsemble	DNNLinear
Amazon	RobustImputer → ThresholdOneHotEncoder → RobustStandardScaler → XGBoost	RobustImputer → RobustMissingIndicator → LogEXtremeValuesTransformer → ThresholdOneHotEncoder → TfidfVectorizer → RobustPCA → RobustStandardScaler → Linear-Learner	RobustImputer → RobustStandardScaler → XGBoost	RobustImputer → RobustStandardScaler → XGBoost
Microsoft	StandardScalerWrapper → ExtremeRandomTrees	MaxAbsScaler → VotingEnsemble	SparseNormalizer → StackEnsemble	StandardScalerWrapper → StackEnsemble
Oracle	DataSampling → FeatureSelection → LightGBM	DataSampling → FeatureSelection → AdaBoostClassifier	DataSampling → FeatureSelection → LightGBM	DataSampling → FeatureSelection → LightGBM

Fig. 34 Output ML pipeline of noisy datasets in `automlbench`

Similar to our previous discussions, we observe the overfitting behavior on GOOGLE CLOUD; however, all platforms have similar test AUC on these four datasets. In terms of speed, ORACLE DATASCIENCE is the fastest (cheapest) in this experiment because of data subsampling and feature selection. This is consistent with our previous observation (similar to the "smaller dataset, holdout test" case)

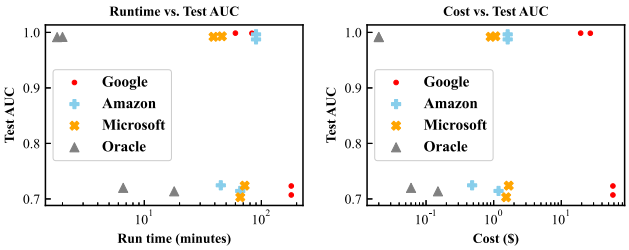

Fig. 35 Runtime (cost) versus test AUC of AutoML platforms on noisy datasets

3.3 Results on small image dataset

Figures 36, 37 and 38 illustrate the results on MNIST dataset giving 3 h of maximal time. Platforms that trains a tree model achieve the best test accuracy, but the accuracy gap is not significant. Here, platforms that conduct aggressive feature selection performs sub-optimally because this causes the loss of relations between neighboring pixels. GOOGLE CLOUD chooses DNN models in the ensemble model, while the other three platforms choose tree models. We set the maximal execution time as 3 h, and all the platforms use up the time budget. This indicates that training an image dataset is often more time-consuming than normal classification datasets due to higher dimensionality and larger model size.

(Comparing Platforms) Although four platforms perform similarly (w.r.t. accuracy) on small image dataset, we still have several interesting observations. GOOGLE CLOUD obtains the best accuracy over the training set but still suffers from overfitting. AMAZON AWS achieves the best test accuracy, followed by MICROSOFT AZURE. We observe that both AMAZON AWS and MICROSOFT AZURE choose a scaler operator, a standard data preprocessing in image processing. ORACLE DATASCIENCE outputs a comparable, but slightly less accurate, model than others. Apart from the lack of a scaler operator, we hypothesize that data sampling and feature selection in ORACLE DATASCIENCE might hurt slightly for image data.

Platform	Validation Accuracy	Test Accuracy	Time	Cost
Google	0.9829 (0.0)	0.9741 (-0.0068)	3h	$57.96
Amazon	0.9812 (-0.0017)	0.9809 (0.0)	3h	$3.23
Microsoft	0.9717 (-0.0112)	0.9748 (-0.0061)	3h	$4.17
Oracle	0.9621 (-0.0208)	0.9730 (-0.0079)	3h	$1.5

Fig. 36 Validation accuracy, test accuracy, runtime and cost of MNIST dataset in `automlbench`. The numbers in brackets indicate the difference compared with the best results

Platform	Output Pipeline
Google	VotingEnsemble
Amazon	RobustImputer → RobustStandardScaler → XGBoost
Microsoft	MaxAbsScaler → StackEnsemble
Oracle	DataSampling → FeatureSelection → LightGBM

Fig. 37 Output ML pipeline of MNIST dataset in `automlbench`

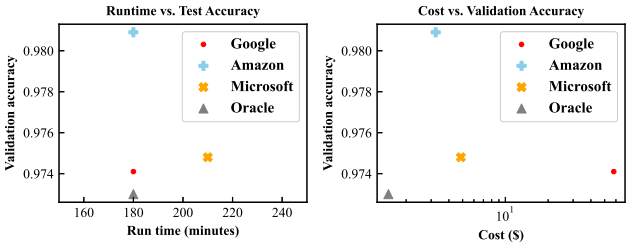

Fig. 38 Runtime (cost) versus accuracy of AutoML platforms on MNIST

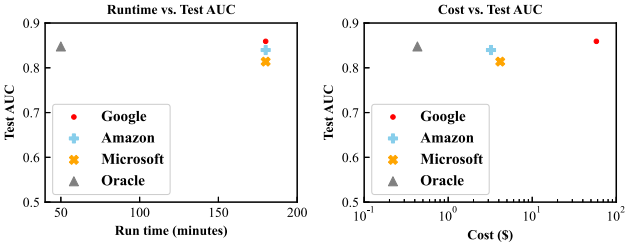

Fig. 41 Runtime (cost) versus test AUC of AutoML platforms on large-scale datasets

Platform		Higgs (1h)	Higgs (3h)	RCV
Google	val.	0.847	0.861 (0.0)	NA
	test	0.845	0.8591 (0.0)	
	time	1.5h	3h	
	cost	$29	$58	
Amazon	val.	-	0.8387 (-0.0223)	NA
	test	-	0.8397 (-0.0194)	
	time	-	3h	
	cost	-	$3.23	
Microsoft	val.	0.8119	0.8139 (-0.0471)	NA
	test	0.8116	0.8138 (-0.0453)	
	time	1.5h	3h	
	cost	$1.04	$2.08	
Oracle	val.	0.8328	0.8328 (-0.0282)	NA
	test	0.8477	0.8477 (-0.0114)	
	time	50min	50min	
	cost	$0.43	$0.43	

Fig. 39 Validation AUC, test AUC, runtime and cost of large-scale datasets in `automlbench`. The numbers in brackets indicate the difference compared with the best results. "–" means time out, and NA means the system cannot run the dataset

Platform	Higgs	RCV
Google	VotingEnsemble	NA
Amazon	OneHotEncoder → StandardScaler → XGBoost	NA
Microsoft	MaxAbsScaler → StackEnsemble	NA
Oracle	DataSampling → FeatureSelection → LightGBM	NA

Fig. 40 Output ML pipeline of large-scale datasets in `automlbench`. NA means the system cannot run the dataset

3.4 Results on large-scale dataset

We choose two large-scale datasets in this experiment. Higgs contains 11 million instances, and RCV contains more than 47,000 features. The maximal time budget for all the platforms is set to be 3 h. As the results in Figs. 39, 40 and 41 illustrate, all the baselines cannot handle RCV dataset since they restrict the dimension of input dataset.

On Higgs, GOOGLE CLOUD generates the most accurate model on both validation set and testset, implying that DNN model can outperform traditional models given enough input data. Due to data subsampling and feature selection, ORACLE DATASCIENCE is significantly faster, while 1.2 AUC points lower than GOOGLE CLOUD. The relative importance between this quality difference and the additional runtime/cost depends on applications.

Observation 5 *(Large-scale Dataset)* ***The support of high-dimensional large-scale datasets could be significantly improved in today's AutoML services.***

3.5 Results on regression dataset

We next run six regression datasets using the baseline platforms. We calculate the mean-squared error (MSE) over the validation dataset and test dataset as the statistical loss (the lower the better). The maximal time budget is 3 h for all the platforms. Figures 42 and 43 give the loss, runtime, cost and output pipeline. Overall, four platforms obtain comparable loss across different datasets. In terms of model generalization, all the platforms generalize relatively well on the first five datasets. However, on the last dataset UJIndoorLoc, only GOOGLE CLOUD performs well on the test dataset. We find that the other three platforms put more weights on some categorical indicator features (e.g., building id) which are closely related to the target (longitude). Their trained tree models are not suitable for these categorical features and thereby encounter overfitting. Consequently, the trained models cannot generalize well to unseen indicators. GOOGLE CLOUD, however, prefers DNN models and does not have this problem. Figure 44 summarizes the combinations of loss and efficiency metrics (runtime and cost).

(Comparing Platforms) Overall, the evaluated platforms obtain comparable statistical loss on the chosen regression datasets, except for one dataset on which GOOGLE CLOUD is significantly better than the other platforms. The execution strategies are similar as those on other types of datasets—GOOGLE CLOUD trains an ensemble DNN model, AMAZON

Platform		BikeSharing	BlogFeadback	MSD	CT	SuperConductivity	UJIndoorLoc
Google	val. loss	0.0	612.81	79.85	0.18	97.32	4.11
	test loss	4.78×10^{-5}	354.46	80.49	26.57	84.42	0.47
	time	1.64h	3h	3h	3h	1.49h	3h
	cost	$31.68	$58	$58	$58	$28.79	$58
Amazon	val. loss	7×10^{-6}	447.23	76.30	0.40	84.01	2.17
	test loss	4.25×10^{-6}	329.35	75.61	18.99	90.18	742.36
	time	1.6h	2h	3h	2h	1.5h	2.7h
	cost	$1.72	$2.15	$3.23	$2.15	$1.61	$2.90
Microsoft	val. loss	8×10^{-6}	612.36	81.28	5.56	103.94	8.85
	test loss	1.07×10^{-5}	332.73	80.42	52.57	100.40	1021.39
	time	41min	3h	3h	3h	1.1h	2.2h
	cost	$0.95	$4.16	$4.16	$4.16	$1.53	$3.05
Oracle	val. loss	1.58×10^{-26}	618.5	82.75	0.077	91.71	69.20
	test loss	1.28×10^{-26}	327.83	80.93	44.54	167.76	1132.55
	time	1.8min	56min	1.1h	1.3h	17.8min	1.1h
	cost	$0.02	$0.48	$0.56	$0.66	$0.15	$0.56

Fig. 42 Validation loss, test loss, runtime and cost on regression datasets in `automlbench`

Platform	BikeSharing	BlogFeadback	MSD	CT	SuperConductivity	UJIndoorLoc
Google	VotingEnsemble	VotingEnsemble	VotingEnsemble	VotingEnsemble	VotingEnsemble	VotingEnsemble
Amazon	RobustImputer → TfidfVectorizer → StandardScaler → LinearLearner	RobustImputer → OneHotEncoder → StandardScaler → XGBoost	RobustImputer → StandardScaler → XGBoost	RobustImputer → OneHotEncoder → StandardScaler → XGBoost	RobustImputer → QuantileValues → RobustPCA → StandardScaler → XGBoost	RobustImputer → OneHotEncoder → StandardScaler → XGBoost
Microsoft	StandardScaler → ElasticNet	MinMaxScaler → StackEnsemble	RobustScaler → StackEnsemble	RobustScaler → StackEnsemble	MinMaxScaler → VotingEnsemble	MaxAbsScaler → StackEnsemble
Oracle	DataSampling → FeatureSelection →LinearRegression	DataSampling → FeatureSelection → RandomForest	DataSampling → FeatureSelection → LGBMRegressor	DataSampling → FeatureSelection → KNeighbors	DataSampling → FeatureSelection → ExtraTrees	DataSampling → FeatureSelection → RandomForest

Fig. 43 Output ML pipeline of regression datasets in `automlbench`

AWS performs complex feature preprocessing and trains a single model (often tree model), MICROSOFT AZURE runs simple feature scaling and trains an ensemble model, and ORACLE DATASCIENCE selects important features and prefers tree models.

3.6 Comparing `automlbench`, `mlbench` and Kaggle winners

`automlbench` and `mlbench` run on two snapshots of ML clouds, five years apart. *How do these two snapshots compare?*

We submit the output models of `mlbench` and `automlbench` to Kaggle competitions and compare them with top-ranking solutions. We do not include D-VP-r and D-PHY because their unlabeled testsets on Kaggle lack some features in the provided labeled dataset. To keep consistent with the protocol of `mlbench`, we use all labeled dataset for training and submit output model to predict unlabeled testset on Kaggle.

(Comparing `mlbench` and `automlbench`) We observe some interesting points by comparing `mlbench` and `automlbench`. Back in 2017, for Azure in `mlbench`, we run every combination of hyper-parameter for each model to tune the optimal configuration. And for Amazon in `mlbench`, we run logistic regression using its default hyper-parameter tuning mechanism. Today, all AutoML platforms in `automlbench` have a much larger search space because there are more operators (e.g., preprocessing, feature transformation, model selection, etc.) and each operator has several hyper-parameters to tune.

Looking at the accuracy in Fig. 45 and runtime in Fig. 46, we see that

1. A single logistic regression model and automatic hyper-parameter tuning (Amazon in `mlbench`) is a strong baseline—in fact, it is competitive to many AutoML systems being offered today, at least on these three datasets.
2. A systematic search over a larger search space of alternative models (Azure in `mlbench`) does pay off significantly. In comparison, today's AutoML systems are

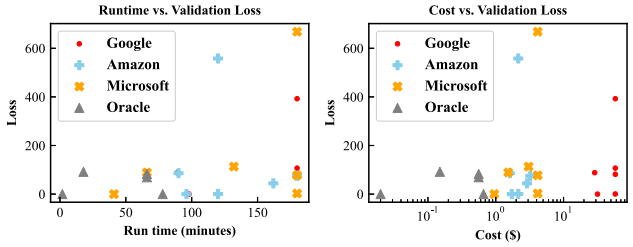

Fig. 44 Runtime (cost) versus loss of AutoML platforms on regression datasets

faster, but often cannot reach the quality of such a systematic search.

(Comparing with Kaggle Winner) When placed in the Kaggle leaderboard, we observe that all platforms cannot beat data science experts. This is unsurprising since the Kaggle winners designed specific-purpose feature engineering techniques for each individual dataset, which is beyond the scope and capability of these platforms. Nevertheless, for users who are non-experts on ML, these platforms can save tremendous time with their easy-to-use interfaces and rich libraries and produce moderately good models.

3.7 Comparing different platforms

We observe that different platforms make different design decisions, and ***none of them dominates other platforms***. Instead, they outperform others in different regimes. Specifically, we observed five representative regimes:

1. **Regime 1 (7 Datasets, D-PIS-r, D-AEA-r, D-SCS-r, D-GMC-r, D-EG-r, D-HQC-r, D-SMR-r)**, in which datasets are more "under-determined" (smaller in terms of # rows) and testset is of the same distribution as the training set (generated via random holdout).
2. **Regime 2 (3 Datasets, D-PBV-r, D-KDD-r, Higgs)**, in which datasets are more "over-determined" (larger in terms of # rows) and testset is of the same distribution as the training set (generated via random holdout).
3. **Regime 3 (All Datasets)**, in which the testset is from Kaggle's private leaderboard. This private testset might come from a different distribution as the training set.
4. **Regime 4 (Multimedia Dataset)**, in which datasets are collected from multimedia sources (e.g., images).
5. **Regime 5 (Sparse Dataset)**, in which datasets are high-dimensional but sparse (i.e., with a small fraction of nonzero features).

Considering the data statistics and machine learning tasks, our observations are as follows:

1. Feature selection and data sampling (ORACLE DATA-SCIENCE) are powerful techniques to improve the resource efficiency, whereas in *Regime 1* they can achieve comparable and sometimes better quality as other platforms.
2. Deep learning-based models (GOOGLE CLOUD) are powerful in dealing with complex datasets and multimedia datasets and can outperform other systems significantly, in terms of quality, in *Regime 2* and *Regime 4*. Of course, these techniques can be slower and have the risk of overfitting to smaller datasets.
3. Platforms without aggressive feature selection and data sampling (AMAZON AWS and MICROSOFT AZURE) can be more robust when given a new testset and thus outperform other platforms in *Regime 3*.
4. The existing platforms cannot work on high-dimensional and sparse datasets in *Regime 5*, which incur both expensive computation and overwhelming storage cost.

3.8 Future directions

According to the previous results and discussions, the existing machine learning clouds all reveal shortcomings and limitations. To tackle these challenges, there are several possible solutions—(1) choose different categories of ML models considering the data characteristics (e.g., data type, data size, sparsity); (2) consolidate the advantages (e.g., feature selection, sampling, model ensemble) mentioned in the observations into a single system; (3) explicitly evaluate the model generalization by a stand-alone validation dataset from a different data distribution; (4) improve the support for high-dimensional and sparse datasets via techniques such as sparsity-aware optimization algorithm; (5) accelerate the hyper-parameter tuning process via techniques such as search tree construction [33], search space reduction [40], and successive halving [31]; and (6) provide easy-to-use programming interfaces for users to deploy their own AutoML techniques and extend the ecosystem of machine learning clouds.

4 Related work

Machine learning benchmarking There has been research on benchmarking different machine learning algorithms and comparing their relative quality on various datasets [8, 10, 16, 32, 47, 51]. Most of these efforts focus on benchmarking machine learning algorithms on "raw datasets" without much feature engineering, a key process for high-quality machine learning applications [14]. Our work is different in the sense that it consists of best-effort baselines for feature engineering and model selection. Another difference between our study and previous work is that, instead of benchmarking all exist-

Dataset	Leader board#1	mlbench		automlbench			
		Azure	Amazon	Google	Amazon	Microsoft	Oracle
D-PIS-r	0.88 (1/132)	0.87 (27), C-BDT	0.86 (66)	0.82 (110)	0.85 (105)	0.84 (106)	0.83 (109)
D-AEA-r	0.92 (1/1688)	0.88 (737), C-LR	0.85 (901)	0.82 (998)	0.83 (962)	0.84 (942)	0.83 (969)
D-SCS-r	0.83 (1/5116)	0.82 (3084), C-BDT	0.81 (3691)	0.81 (3597)	0.80 (3870)	0.83 (297)	0.77 (4058)
D-PBV-r	1.00 (1/2261)	0.98 (1498), C-BPM	0.96 (1612)	0.99 (1085)	0.93 (1781)	0.92 (1813)	0.92 (1790)
D-KDD-r	0.68 (1/474)	0.60 (77), C-LR	0.58 (152)	0.63 (30)	0.61 (57)	0.56 (284)	0.60 (60)
D-GMC-r	0.87 (1/925)	0.87 (137), C-BDT	0.86 (544)	NA	0.86 (514)	0.86 (545)	0.86 (529)
D-EG-r	0.89 (1/625)	0.86 (374), C-BDT	NA	NA	0.87 (351)	0.88 (324)	0.75 (516)
D-HQC-r	0.97 (1/1757)	0.97 (977), C-BDT	0.96 (1294)	0.96 (1144)	0.97 (851)	0.90 (1506)	0.96 (1061)
D-SMR-r	0.80 (1/2222)	0.74 (1826), C-AP	NA	-	0.79 (500)	0.79 (445)	0.78 (845)

Fig. 45 Comparing `mlbench` and `automlbench`. We use these raw datasets from Kaggle without any additional modification. NA means the system cannot run the dataset. The numbers are the scores given by Kaggle and the rankings on the private leader board after we submit the predictions. The results for public leader board are similar

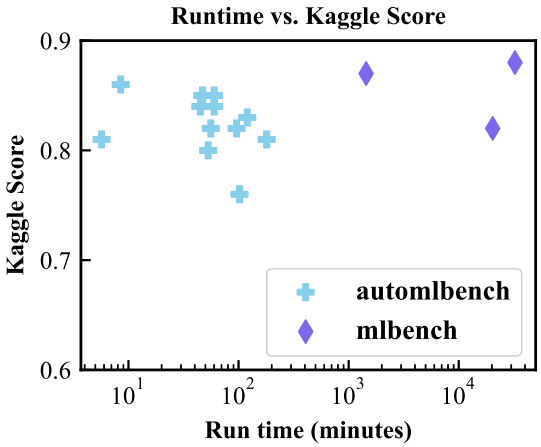

Fig. 46 Runtime versus Kaggle score of `mlbench` and `automlbench`

ing machine learning models, we focus on those provided by existing machine learning clouds and try to understand whether the current abstraction is enough to support users of these clouds.

Cloud benchmarking Benchmarking cloud services and more traditional relational databases have been an active research topic for decades. Famous benchmarks include the Wisconsin benchmark [13] and TPC benchmarks.[6] There are also benchmarks targeting clouds for different purposes, especially for data processing and management [11, 28, 35]. Our work is motivated by the success and impact of these benchmarks, and we hope to establish the first benchmark for declarative machine learning on the cloud.

AutoML techniques Automatic machine learning (AutoML) has a rich history in the previous decade [2, 24, 30, 46]. The existing works use diverse approaches to solve the optimization problems in AutoML, including Bayesian Optimization [9, 27, 29, 45, 46], recommendation-based methods [20, 36, 44] and genetic evolutionary [38]. A common problem of these methods is the cold-start problem, that is, how to predict the performance of a configuration on a new dataset. Meta-learning techniques are designed to tackle this cold-start issue [17, 21, 42, 49]. A range of AutoML toolkits have been developed by researchers [7], e.g., Auto-sklearn [18], auto_ml [39], Autogluon [15], TPOT [37] and H2O [30]. Unfortunately, the current machine learning clouds have not provided functionalities to deploy user-defined AutoML techniques in the clouds.

AutoML platforms Different from the above AutoML toolkits which mostly run on a single machine, several commercial AutoML products offered by major cloud providers enable users deploy end-to-end AutoML workloads in the cloud. Typical AutoML platforms include Google Cloud AutoML [22], Microsoft Azure AutoML [6], Amazon SageMaker Autopilot [4], Oracle AutoML [48] and IBM AutoAI [5]. Although all of them provision automatic training services, they automate different phases in machine learning pipeline, e.g., data cleaning, data sampling, feature selection, model selection and hyper-parameter tuning. Note that we do not choose IBM AutoAI since the processing of IBM AutoAI is similar to Amazon AWS, including feature preprocessing, model selection and hyper-parameter tuning. Therefore, we choose Amazon AWS as the representative platform.

5 Conclusion

In this paper, we presented an empirical study on the performance of state-of-the-art declarative machine learning clouds.

We first conducted our experiments based on `mlbench`, a dataset we constructed by collecting winning code from Kaggle competitions. We compared the performance of machine learning clouds with the Kaggle winning code we harvested. Our results show that there is an obvious gap between top-performing models on the cloud and Kaggle winning code in terms of low-quality tolerance regimes they can meet, though

[6] http://www.tpc.org/information/benchmarks.asp.

machine learning clouds do perform reasonably well when increasing the level of quality tolerance regimes. Detailed investigation further reveals that lack of adopting ensemble methods is perhaps one reason for the performance gap. A promising direction for improving the performance of machine learning clouds is therefore to incorporate more well-tuned models that leverage ensemble methods.

We next conducted another benchmark `automlbench` to compare AutoML services of machine learning clouds. We run different kinds of datasets on both classification and regression tasks. Through extensive evaluation, we understand the importance of different AutoML techniques since these machine learning clouds choose diverse AutoML scopes. We also identify the limitations of current clouds, e.g., feature engineering, by comparing Kaggle solutions.

(Limitations of this work) The machine learning clouds are evolving over time. Therefore, our benchmark is only an initial attempt to investigate the merits and shortcomings of machine learning clouds. We will keep updating our benchmark with the rise of new platforms and offerings.

Acknowledgements This work was sponsored by National Science and Technology Major Project (No. 2022ZD0116315) and Key R&D Program of Hubei Province (No. 2023BAB077). CZ and the DS3Lab gratefully acknowledge the support from the Swiss National Science Foundation (Project Number 200021_184628), Innosuisse/SNF BRIDGE Discovery (Project Number 40B2-0_187132), European Union Horizon 2020 Research and Innovation Programme (DAPHNE, 957407), Botnar Research Centre for Child Health, Swiss Data Science Center, Alibaba, Cisco, eBay, Google Focused Research Awards, Microsoft Swiss Joint Research Center, Oracle Labs, Swisscom, Zurich Insurance, Chinese Scholarship Council and the Department of Computer Science at ETH Zurich.

References

1. https://archive.ics.uci.edu/ml/datasets/
2. Aguilar Melgar, L., et al.: Ease.ml: a lifecycle management system for machine learning. In: 11th Annual Conference on Innovative Data Systems Research (CIDR 2021) (virtual). CIDR (2021)
3. Amazon: Amazon cloud. http://docs.aws.amazon.com/machine-learning/latest/dg/learning-algorithm.html (2021)
4. Amazon: Amazon sagemaker autopilot. https://aws.amazon.com/sagemaker/autopilot/ (2021)
5. Auto, I.: Ibm autoai. https://www.ibm.com/cloud/watson-studio/autoai (2021)
6. Azure, M.: Azure automated machine learning. https://aws.amazon.com/sagemaker/autopilot/ (2021)
7. Balaji, A., Allen, A.: Benchmarking automatic machine learning frameworks. arXiv:1808.06492 (2018)
8. Bauer, E., Kohavi, R.: An empirical comparison of voting classification algorithms: bagging, boosting, and variants. Mach Learn **36**, 105–139 (1998)
9. Bergstra, J., et al.: Hyperopt: a python library for optimizing the hyperparameters of machine learning algorithms. In: Proceedings of the 12th Python in science conference, vol. 13, p. 20. Citeseer (2013)
10. Caruana, R., et al.: An empirical comparison of supervised learning algorithms. In: ICML (2006)
11. Cooper, B.F., et al.: Benchmarking cloud serving systems with YCSB. In: SoCC (2010)
12. Cortes, C., Vapnik, V.: Support-vector networks. Mach Learn **20**, 273–297 (1995)
13. DeWitt, D.J.: The Wisconsin benchmark: past, present, and future. In: The Benchmark Handbook for Database and Transaction Systems (1993)
14. Domingos, P.: A few useful things to know about machine learning. In: CACM (2012)
15. Erickson, N., Mueller, J., Shirkov, A., Zhang, H., Larroy, P., Li, M., Smola, A.: Autogluon-tabular: robust and accurate automl for structured data. arXiv:2003.06505 (2020)
16. Fernández-Delgado, M., et al.: Do we need hundreds of classifiers to solve real world classification problems. In: JMLR (2014)
17. Feurer, M., et al.: Initializing Bayesian hyperparameter optimization via meta-learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 29 (2015)
18. Feurer, M., et al.: Auto-sklearn: efficient and robust automated machine learning. In: Automated Machine Learning, pp. 113–134 (2019)
19. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. In: JCSS (1997)
20. Fusi, N., et al.: Probabilistic matrix factorization for automated machine learning. Adv. Neural Inf. Process. Syst. **31**, 3348–3357 (2018)
21. Gomes, T.A., et al.: Combining meta-learning and search techniques to select parameters for support vector machines. Neurocomputing **75**(1), 3–13 (2012)
22. Google: Google cloud automl. https://cloud.google.com/automl (2021)
23. Haykin, S.: Neural Networks: A Comprehensive Foundation, 2nd edn. Prentice Hall PTR, Hoboken (1998)
24. He, X., et al.: Automl: a survey of the state-of-the-art. Knowl.-Based Syst. **212**, 106622 (2021)
25. Herbrich, R., et al.: Bayes point machines. In: JMLR (2001)
26. Ho, T.K.: Random decision forests. In: ICDAR (1995)
27. Hutter, F., et al.: Sequential model-based optimization for general algorithm configuration. In: International Conference on Learning and Intelligent Optimization, pp. 507–523. Springer (2011)
28. Jiang, J., Gan, S., Liu, Y., Wang, F., Alonso, G., Klimovic, A., Singla, A., Wu, W., Zhang, C.: Towards demystifying serverless machine learning training. In: Proceedings of the 2021 International Conference on Management of Data, pp. 857–871 (2021)
29. Kotthoff, L., et al.: Auto-weka: automatic model selection and hyperparameter optimization in weka. In: Automated Machine Learning, pp. 81–95. Springer, Cham (2019)
30. LeDell, E., Poirier, S.: H2o automl: scalable automatic machine learning. In: Proceedings of the AutoML Workshop at ICML (2020)
31. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: a novel bandit-based approach to hyperparameter optimization. J. Mach. Learn. Res. **18**(1), 6765–6816 (2017)
32. Li, P., et al.: Cleanml: a study for evaluating the impact of data cleaning on ml classification tasks. In: 36th IEEE International Conference on Data Engineering (ICDE 2020) (virtual) (2021)
33. Li, Y., Shen, Y., Zhang, W., Zhang, C., Cui, B.: Volcanoml: speeding up end-to-end automl via scalable search space decomposition. VLDB J. **32**(2), 389–413 (2023)
34. Liu, Y., et al.: MLbench: benchmarking machine learning services against human experts. Proc. VLDB Endow. **11**(10), 1220–1232 (2018)
35. Luo, C., et al.: Cloudrank-d: benchmarking and ranking cloud computing systems for data processing applications. Front. Comput. Sci. **6**, 347–362 (2012)
36. Mısır, M., et al.: Alors: an algorithm recommender system. Artif. Intell. **244**, 291–314 (2017)

37. Olson, R.S., Moore, J.H.: Tpot: a tree-based pipeline optimization tool for automating machine learning. In: Workshop on Automatic Machine Learning, pp. 66–74. PMLR (2016)
38. Olson, R.S., et al.: Evaluation of a tree-based pipeline optimization tool for automating data science. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016, pp. 485–492 (2016)
39. Parry, P., et al.: auto_ml. https://github.com/ClimbsRocks/auto_ml (2007)
40. Perrone, V., Shen, H., Seeger, M.W., Archambeau, C., Jenatton, R.: Learning search spaces for Bayesian optimization: another view of hyperparameter transfer learning. Adv. Neural Inf. Process. Syst. **32** (2019)
41. Quinlan, J.R.: Induction of decision trees. Mach. Learn. (1986)
42. Reif, M., et al.: Meta-learning for evolutionary parameter optimization of classifiers. Mach. Learn. **87**(3), 357–380 (2012)
43. Shotton, J., et al.: Decision jungles: compact and rich models for classification. In: NIPS (2013)
44. Sun-Hosoya, L., et al.: Activmetal: algorithm recommendation with active meta learning. In: IAL 2018 workshop, ECML PKDD (2018)
45. Thornton, C., et al.: Auto-weka: combined selection and hyperparameter optimization of classification algorithms. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 847–855 (2013)
46. Wong, C., et al.: Transfer learning with neural automl. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, pp. 8366–8375 (2018)
47. Wu, Z., Ramsundar, B., Feinberg, E.N., Gomes, J., Geniesse, C., Pappu, A.S., Leswing, K., Pande, V.: Moleculenet: a benchmark for molecular machine learning. Chem. Sci. **9**(2), 513–530 (2018)
48. Yakovlev, A., et al.: Oracle automl: a fast and predictive automl pipeline. Proc. VLDB Endow. **13**(12), 3166–3180 (2020)
49. Yogatama, D., Mann, G.: Efficient transfer learning method for automatic hyperparameter tuning. In: Artificial Intelligence and Statistics, pp. 1077–1085. PMLR (2014)
50. Zhang, C., et al.: An overreaction to the broken machine learning abstraction: the ease.ml vision. In: HILDA (2017)
51. Zöller, M.A., Huber, M.F.: Benchmark and survey of automated machine learning frameworks. J. Artif. Intell. Res. **70**, 409–472 (2021)