MLBench: Benchmarking Machine Learning Services Against Human Experts

Yu Liu[†] Hantian Zhang[†] Luyuan Zeng[†] Wentao Wu[‡] Ce Zhang[†]
[†]ETH Zurich [‡]Microsoft Research
{yu.liu, hantian.zhang, ce.zhang}@inf.ethz.ch wentao.wu@microsoft.com

ABSTRACT

Modern machine learning services and systems are complicated data systems — the process of designing such systems is an art of compromising between *functionality*, *performance*, and *quality*. Providing different levels of system supports for different functionalities, such as automatic feature engineering, model selection and ensemble, and hyperparameter tuning, could improve the quality, but also introduce additional cost and system complexity. In this paper, we try to facilitate the process of asking the following type of questions: *How much will the users lose if we remove the support of functionality x from a machine learning service?*

Answering this type of questions using existing datasets, such as the UCI datasets, is challenging. The main contribution of this work is a novel dataset, `MLBench`, harvested from Kaggle competitions. Unlike existing datasets, `MLBench` contains not only the raw features for a machine learning task, but also those used by the winning teams of Kaggle competitions. The winning features serve as a baseline of best human effort that enables multiple ways to measure the quality of machine learning services that cannot be supported by existing datasets, such as relative ranking on Kaggle and relative accuracy compared with best-effort systems.

We then conduct an empirical study using `MLBench` to understand example machine learning services from Amazon and Microsoft Azure, and showcase how `MLBench` enables a comparative study revealing the strength and weakness of these existing machine learning services quantitatively and systematically. The full version of this paper can be found at arxiv.org/abs/1707.09562

PVLDB Reference Format:
Yu Liu, Hantian Zhang, Luyuan Zeng, Wentao Wu, and Ce Zhang. MLBench: Benchmarking Machine Learning Services Against Human Experts. *PVLDB*, 11 (10): 1220-1232, 2018.
DOI: https://doi.org/10.14778/3231751.3231770

1. INTRODUCTION

Modern machine learning services and toolboxes provide functionalities that surpass training a single machine learning model. Instead, these services and systems support a range of different machine learning models, different ways of training, and auxiliary utilities such as model selection, model ensemble, hyperparamter tuning, and feature selection. Azure Machine Learning Studio [1] and Amazon Machine Learning [2] are prominent examples.

Proceedings of the VLDB Endowment, Vol. 11, No. 10

DOI: https://doi.org/10.14778/3231751.3231770

Motivating Questions: How good is good? The ongoing advancement of machine learning services raises some natural questions, from the perspectives of both service users and providers. We are currently hosting a similar service as Azure Machine Learning Studio, namely `ease.ml` [31], to our local users at ETH Zurich, and this work is motivated by the following two questions.

• *User: How good are machine learning services such as Azure Machine Learning Studio and `ease.ml`?* We were unable to provide our user (a biologist) with a good answer, because we did not have a reasonable measure of "good" that is easy to convey.

• *Provider (Us): Do we need to provide more machine learning models on `ease.ml` (than Azure provides) to our local user?* We were unable to answer this question because we did not know how to reasonably measure the potential improvement on accuracy for a given machine learning task.

Baseline Approaches. We are not satisfied with two strategies: (1) Run a standard dataset from the UCI repository [21] and report the absolute accuracy, or (2) run an existing system (e.g., scikit-learn) on a standard dataset and report the difference (i.e., relative accuracy) between it and Azure Machine Learning Studio. Neither the absolute accuracy nor the relative accuracy over a baseline system reveals the full picture of a machine learning service as they rely on the hardness of the given machine learning task and the maturity of the baseline system, which are often opaque to our users.

MLBench. In this paper, we present `MLBench`, a novel dataset that we constructed to benchmark machine learning systems and services. The goal of `MLBench` is to provide a quantitative quality measure to answer questions such as *what would users lose compared with a best-effort baseline?* To construct `MLBench`, we first collected real-world datasets *and* best-effort solutions harvested from Kaggle competitions, and then developed multiple performance metrics by comparing relative performance of machine learning services with top-ranked solutions in Kaggle competitions.

An Empirical Study. We present an example use case of `MLBench` by conducting an empirical study of two online machine learning services, i.e., Azure Machine Learning Studio and Amazon Machine Learning. As we will see, `MLBench` provides a way for us to decouple multiple factors having impact on the quality: each individual machine learning model and other external factors such as feature engineering and model ensemble.

Premier of Kaggle Competitions. Kaggle [3] is a popular platform hosting a range of machine learning competitions. Companies or scientists can host their real-world applications on Kaggle; Each user has access to the training and testing datasets, submits their solutions to Kaggle, and gets a quality score on the test set. Kaggle motivates users with prizes for top winning entries. This "crowdsourcing" nature of Kaggle makes it a representative sample of real-world machine learning workloads.

Summary of Technical Contributions.
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 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 system and top-ranked Kaggle performers.
C3. We showcase one application of MLBench by evaluating two online machine learning services: Azure Machine Learning Studio and Amazon Machine Learning. 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 services.

Scope of Study. We study three most popular machine learning tasks: *binary classification*, *multi-class classification*, and *regression*. Due to space limitation, we focus our discussion on binary classification. As we will see, even with this constrained scope, there is no simple, single answer to the main question we aim to answer. We also briefly cover multi-class classification and regression but leave the complete details to the full version of this paper [32].

Reproducibility. We made MLBench publicly available at the following website: http://ds3lab.org/mlbench.

Overview. The rest of the paper is organized as follows. We start by having an in-depth discussion on why existing datasets such as the UCI repository are not sufficient. We then present our methodology and the MLBench benchmark in Section 3. We next present experimental settings and evaluation results in Section 4-6. We summarize related work in Section 8 and conclude in Section 9.

2. MOTIVATION

Over the years, researchers have applied many techniques on the UCI datasets [21] and shown incremental improvements. Unfortunately, using UCI datasets was insufficient for the questions from both our users and ourselves as machine learning service providers.

We have two goals for our benchmark in this work:

- It should allow users and service providers to fairly compare different machine learning services;
- It should allow users and service providers to understand the limitations of their services.

A benchmark like the UCI repository can satisfy the first goal but not the second. To see this more clearly, consider the case that we simply use n datasets D_1, ..., D_n from the UCI database. For a machine learning service that provides m models M_1, ..., M_m, we are able to evaluate the accuracy of each model on each dataset: $a(D_i, M_j)$, for $1 \leq i \leq n$ and $1 \leq j \leq m$. Based on these accuracy numbers, we are able to compare the *relative* performance of the models on different datasets. Similarly, we can make this *relative* comparison between different machine learning services.

However, we remain clueless about the *absolute* performance of the machine learning services. Consider two machine learning services $\mathcal{M}_1$ and $\mathcal{M}_2$. Given a dataset D, let $a_1 = 0.82$ and $a_2 = 0.81$ be the best accuracy $\mathcal{M}_1$ and $\mathcal{M}_2$ can achieve over D. What we can conclude based on a_1 and a_2 is that $\mathcal{M}_1$ outperforms $\mathcal{M}_2$ over D. Nonetheless, there are several natural questions that both service users and providers may further ask here:

- How good is the accuracy a_1 (resp. a_2) observed by $\mathcal{M}_1$ (resp. $\mathcal{M}_2$)? Is a_1 a really good number over D or is it far behind the state of the art?
- How much improvement over a_1 (resp. a_2) can we get by including more models in $\mathcal{M}_1$ (resp. $\mathcal{M}_2$)?
- How significant is the gap between a_1 and a_2? Essentially, how difficult is the learning task defined over D? (An accuracy difference of 0.01 may mean a lot if D is a tough case.)

None of the above questions can be answered satisfactorily without a comparison against the *frontier* of current machine learning techniques. As far as we know, MLBench is the first, *rudimentary* attempt to provide a baseline for reasoning and understanding the *absolute* performance of machine learning services.

One alternative approach to MLBench could be to stay with an existing benchmark such as the UCI dataset but try to find the *best* features and models – not only those models that have been provided by machine learning services – and run them for each dataset. We see at least two potential shortcomings of this approach. First, it is costly to engineering features to achieve the best accuracy for each UCI dataset. Second, it is expensive to collect all existing models — there are just too many and one may have to resort to a shortened list (e.g., models that have been implemented by Weka [27]). Moreover, it remains elusive that how representative the datasets in UCI and the models in Weka are — they are definitely quite different from the datasets and models that people have used in Kaggle competitions. It is definitely important to explore alternatives to MLBench, but it is beyond the scope of this paper.

3. THE MLBENCH BENCHMARK

In this section, we present details of the MLBench benchmark constructed by harvesting winning code of Kaggle competitions.

3.1 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.

3.2 Methodology

Benchmarking systems fairly is not an easy task. Three key aspects came to mind when designing a benchmark for machine learning services:
(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*. These two factors need to be decoupled to understand the quality provided by a given machine learning service.
(3) To compare machine learning services with the best effort of a given machine learning task, 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 services.

Starting from these principles, we made a few basic decisions that we shall present next.

3.2.1 Methodology

We collect top winning code for all 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 hours, we further collect features extracted by the winning code. The features are then used for training and testing models provided by the machine learning services and the Kaggle winning solution. We also include datasets constructed using raw features (see Section 3.3).

Discussion. At first glance, our methodology is quite trivial. Indeed, there is little novelty in the procedure itself, though the engineering effort involved is substantial. (It took us more than nine months to finish the experimental evaluation presented in Section 5.) On second thought, one may wonder what the point is of spending so much effort.

To see the subtlety here, consider an alternative approach that is much easier to implement: Take one well-known dataset (or several datasets) such as those from the UCI Machine Learning Repository, run a standard feature selection algorithm, and compare the performance of a machine learning service with that of standard machine learning libraries (e.g., Weka [27]) on this dataset. There are, however, a couple of caveats in this approach. First, it is unclear how challenging the learning problem (associated with the dataset) is. There may be *subjective* justification but no *objective* metric of the difficulty. Second, it is questionable whether the models covered by standard libraries represent the state of the art. Depending on the popularity and maturity of the libraries, coverage may vary dramatically. Third, feature engineering and model selection are more of an art mastered only by human experts. If we ignore both, our result might be overly optimistic or overly pessimistic for machine learning service. (Section 2 provides more detailed analysis.)

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 services. 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.

3.2.2 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 services, we introduce the notion of "quality tolerance" (of a user).

DEFINITION 1. *The* quality tolerance *of a user is τ if s/he can be satisfied only by being ranked among the top τ%, assuming that s/he uses a model M provided by the service 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 service 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 service 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:

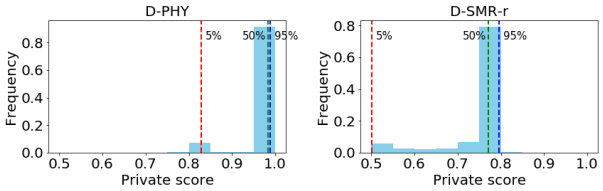


Figure 1: Histograms of the AUC scores on private leader board for two example datasets D-PHY and D-SMR-r.

- *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 $u(M,\tau)$, namely,

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

Similarly, if a machine learning service $\mathcal{M}$ provides n models $\{M_1, ..., M_n\}$ ($n \geq 1$), 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

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

Clearly, the capacity of $\mathcal{M}$ over T is the capacity of the best model that $\mathcal{M}$ provides for T, whereas the τ-universality 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 $\mathcal{T} = \{T_1, ..., T_m\}$, 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.

3.2.3 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.[1] To illustrate this, Figure 1 presents the histograms (i.e., distributions) of the AUC scores in two Kaggle competitions (see Section 3.4 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)

[1] AUC computes the area under the "Receiver Operating Characteristic" (ROC) curve. A perfect model will have an AUC score of 1, whereas random guessing will result an AUC around 0.5 [14].

Table 1: Statistics of Kaggle Competitions

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

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 services, it is important to look at *both* ranking and absolute quality. In this paper, our analysis will always base on both.

Notes on Alternative Metrics. There can be various other choices except for the *capacity* and *universality* metrics we chose. Indeed, we never claimed that our choice is the best or should be unique, and it is actually our intention to encourage people to come up with their own performance metrics on top of `MLBench`, customized for their own situations and purposes. Our contribution here is to give a concrete example of how such a "fair" comparison could look like.

Having said this, we do think the *capacity* and *university* metrics we used have some advantages over other metrics, such as those based on relative performance ratios over Kaggle winning code, which are probably more intuitive.

First, both *capacity* and *university* are based on *ranking* on the Kaggle leaderboard, which is the criterion used by a Kaggle competition when deciding the winners. This is a more *stringent* criterion compared with ones based on relative performance ratios. Because the competitiveness of Kaggle competitions varies vastly (some are very challenging whereas the others are relatively easy – see Table 6(a)), a machine learning service that achieves 90% of AUC scores of the winning codes in two binary classification competitions can be ranked quite differently.

Second, Kaggle uses different performance evaluation metrics for different types of competitions, or even within the same type of competitions. For example, while binary classification problems typically use AUC score as the metric, regression problems use metrics varying from RMSE, MAE, Gini, and so on. A machine learning service that achieves 90% of the AUC score of the winning code in a binary classification competition and that achieves 90% of the RMSE of the winning code in a regression competition then mean very differently. Reporting these relative performance numbers to users of machine learning services, many of whom are not machine learning experts, may mislead them. Rankings, on the other hand, are semantically consistent regardless of the type of competition: A machine learning service ranks within the top 10% of the participants on the leaderboard means exactly what it literally means, regardless of whether it is a binary classification, multi-class classification, or regression competition.

Of course, there are limitations of the ranking-based metrics we used. As we have mentioned, there is an apparent robustness issue given that rankings depend on particular participants of Kaggle competitions. We admit that this is certainly problematic for a competition if there are only a few participants. Nevertheless, the competitions we have included in `MLBench` did involve a fairly reasonable number of participating teams. However, even if the number of participants is large, there is still a chance that most of the participants are incompetent — outperforming those participants is fairly easy but can lead to a high ranking in the leaderboard. We currently have no way to distinguish such cases, and we have implicitly relied on Kaggle competition organizers/publishers to ensure the quality/attractiveness of their competitions.

Table 2: Kaggle Competitions with Winning Code

Tasks of Kaggle Competitions	Number
Binary Classification	13
Multi-class Classification	9
Regression	9
Others	10

3.3 MLBench Overview

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

Datasets from Winning Code. As shown in Table 1, 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 services. Table 2 further summarizes the types of machine learning tasks covered by these 41 competitions with winning code. We next focus on the 13 binary classification competitions due to space constraint.

Binary Classification Competitions. We ran the winning code of the 13 binary classification competitions on Microsoft Azure for the purpose of extracting the features used by the winning code. 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 excluded datasets whose inputs are either three-dimensional features (e.g., 3D medical images) or features that cannot be extracted and saved successfully.[2] 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 Section 3.4), 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.[3]

Statistics of Datasets. In total, `MLBench` contains 18 datasets for binary classification 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 Table 3. 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.

3.4 Dataset Details

We present details about the datasets listed in Table 3. For each dataset, we first introduce the background of the corresponding

[2]We excluded those datasets with 3D features because we cannot directly use the raw data to train models provided by machine learning services — additional feature engineering is mandatory.

[3]https://docs.microsoft.com/en-us/azure/machine-learning/studio/faq

Table 3: Statistics of datasets for binary classification ("-r" indicates datasets with raw feature). See Section 3.4 for details.

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

Kaggle competition. We then describe the features used by the winning code we found, which characterize the datasets themselves, as well as the models and algorithms it adopts.

• **MLSP2014-Schizophrenia Classification Challenge (D-SCH and D-SCH-r)**: In this competition, multimodal features derived from brain magnetic resonance imaging (MRI) scans and labels of the training data are provided. The goal of this competition is to build machine learning models that can predict if a person is a "healthy control" or "schizophrenic patient" in the testing data.[4] We use the winning code from Karolis Koncevicius [4]. Interestingly, the winning code uses the same features as the raw data provided by Kaggle. The algorithm it uses is distance weighted discrimination [35]. We abbreviate the constructed dataset as **D-SCH**. Another dataset is constructed using the raw data provided by Kaggle and is referred to as **D-SCH-r**, although **D-SCH** and **D-SCH-r** contain the same features in this particular case.

• **Influencers in Social Networks (D-PIS-r)**: In this competition, each data point describes the features extracted based on the Twitter activities of two individuals. In the training dataset, the data points are labelled to indicate which one of the two individuals is more influential on a social network. The goal of this competition is to predict the more influential individual from the given features of 2 individuals.[5] No winning code is available for this competition. We take the raw data provided by Kaggle and construct the dataset **D-PIS-r**.

• **StumbleUpon Evergreen Classification Challenge (D-EG and D-EG-r)**: In this competition, URLs and their corresponding raw contents are given. The goal of this competition is to build a classifier that can label a URL (i.e., Web page) as either "ephemeral" (i.e., pages that are only relevant for a short period of time, such as news articles) or "evergreen" (i.e., pages that maintain a timeless quality, such as Wikipedia articles). [6] We use the winning code from Marco Lui [5]. It extracts features from raw HTML documents and uses only text-based features. The most important step is a stacking-based approach that combines the generated features [33]. The algorithm the winning code uses is logistic regression. Features used by this winning code are stored by extracting the input to the logistic regression classifier. We abbreviate this constructed dataset as **D-EG**. Again, we construct another dataset **D-EG-r** using the raw features.

[4] https://www.kaggle.com/c/mlsp-2014-mri

[5] https://www.kaggle.com/c/predict-who-is-more-influential-in-a-social-network.

[6] https://www.kaggle.com/c/stumbleupon

• **West-Nile Virus Prediction (D-VP and D-VP-r)**: In this competition, the participants are given weather, location, testing, and spraying data to predict whether or not West Nile Virus is present. [7] We use the winning code from [6], which added three new features to the raw dataset. The predictions are initialized according to a normal distribution. Each prediction is then multiplied by various coefficients obtained from other information related to the target (e.g., geographical information). The predictions are then normalized. We abbreviate this dataset as **D-VP**. The corresponding dataset using raw features is denoted as **D-VP-r**.

• **Amazon.com-Employee Access Challenge (D-AEA and D-AEA-r)**: The goal of this competition is to create an algorithm that can predict approval/denial for an unseen employee, based on historical records about approval/denial of employee's requests to access certain resources.[8] We use the winning code from Owen Zhang [7]. It first converts the original categorical features to numerical features. It then builds six models from subsets of the features as well as features obtained via post-processing (e.g., aggregation). The final prediction is generated by an ensemble of predictions from individual models. The algorithms it uses are GBM (generalized boosted regression modeling) [37], random forest [15], extremely randomized trees [26], and glmnet (lasso and elastic-net regularized generalized linear models) [25]. Features used by this winning code are stored by unioning all features used by the models. We abbreviate this constructed dataset as **D-AEA**. Correspondingly, the dataset containing only the raw data from Kaggle is denoted as **D-AEA-r**.

• **Santander Customer Satisfaction (D-SCS-r)**: In this competition[9], the objective is to identify if a customer is unsatisfied with their experience in dealing with the Santander bank. A list of numeric features as well as a label are provided to the participants. There is no winning code available for this competition. We use the raw data provided by Kaggle to construct the dataset **D-SCS-r**.

• **Springleaf Marketing Response (D-SMR-r)**: A large set of anonymized features describing a customer are provided in each entry of the training dataset. The goal of this competition is to use the features of the customer to predict whether s/he will respond to a direct mail offer.[10] No winning code is available for this competition. We therefore construct the dataset **D-SMR-r** using the raw data from Kaggle.

• **Give me some credit (D-GMC-r)**: In this competition[11], the participants are asked to help a bank to predict the probability that a client will experience financial distress in the next two years. No winning code is available. We take the raw data from Kaggle and denote the dataset as **D-GMC-r**.

• **Homesite Quote Conversion (D-HQC-r)**: In this competition[12], the participants are asked to predict whether or not a customer will purchase the quoted product from an insurance company. The training data includes anonymized features covering information about the product, the client, the property going to be assured, and the location. We use the raw data to create the dataset **D-HQC-r**.

• **KDD Cup 2014-Predicting Excitement (D-KDD1, D-KDD2 and D-KDD-r)**: In this competition, the participants are asked to help DonorsChoose.org identify projects that are exceptionally exciting to the business, given all the related data about projects, do-

[7] https://www.kaggle.com/c/predict-west-nile-virus

[8] https://www.kaggle.com/c/amazon-employee-access-challenge

[9] https0.://www.kaggle.com/c/santander-customer-satisfaction

[10] https://www.kaggle.com/c/springleaf-marketing-response

[11] https://www.kaggle.com/c/GiveMeSomeCredit

[12] https://www.kaggle.com/c/homesite-quote-conversion

nations, and so on. [13] We use the winning code from [8]. It builds two diverse feature sets based on raw features and generated features. These two diverse feature sets are then used to train two different models: gradient boosting regressor and gradient boosting machine. The final result is based on the ensemble of the two. We abbreviate these two constructed datasets as **D-KDD1** and **D-KDD2**. As before, we also create a dataset that only contains the raw data, denoted as **D-KDD-r**.

• **Predicting Red Hat Business Value (D-PBV-r)**: The goal of this competition[14] is to identify customers with potential business value. To achieve this goal, records of customer activities are provided to the participants. In addition, each customer is associated with a set of features. There is no winning code for this competition. We construct the dataset **D-PBV-r** by joining the tables containing raw data describing the activities and the features of the customers.

• **Flavours of Physics: Finding $\tau \rightarrow \mu\mu\mu$ (D-PHY)**: In this competition, participants are given a list of collision events (of two high-energy particle beams travelling at close to the speed of light inside the Large Hadron Collider — the worlds largest and most powerful particle accelerator) and their properties to predict whether $\tau \rightarrow 3\mu$ decay happens in a collision or not.[15] We use the winning code from Gramolin [9]. It designs new features based on original features. One original feature is not used because it prevents passing of the agreement test.[16] In addition, the winning code does not use all the training data. Regarding the algorithm, it uses only XGBoost [17]. It trains two different XGBoost models on different sets of features. The final result is an ensemble of results obtained by the two models. The combination of two independent classifiers enables it to pass the correlation test.[17] Features used by this winning code are stored by extracting the input to the models. Then the features taken into different models are merged and duplicated features are dropped. We abbreviate it as **D-PHY**.

There are missing values in the datasets D-PHY and D-KDD2. We replace the missing values in these two datasets with the average values of corresponding features and "N/A", respectively, for our experiments on Azure and Amazon. We refer the readers to the full version of this paper for details of the datasets in `MLBench` for multi-class classification and regression problems [32].

4. AN EXAMPLE MLBENCH USECASE

To showcase an application of `MLBench`, 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. The goal here is *not* to compare between these two services, instead, to illustrate the process of using `MLBench` and the kinds of insights one can get with this novel dataset. Again, we focus on binary classification problems. We first introduce the current APIs of **Azure** and **Amazon** and then all machine learning models they provide.

4.1 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.

• **Azure** provides an API using which users specify the *types* of *machine learning models*, such as (1) logistic regression, (2) support vector machine, (3) decision tree, etc. 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. For now, **Amazon** always runs a logistic regression for binary classification [10]. **Amazon** further provides a set of default hyper-parameters for logistic regression, but users can also change these default values.

4.2 Machine Learning Models

In the following, 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.[18]

• **Two-Class Bayes Point Machine (C-BPM)**: It 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 (in terms of generalization performance) [29].

• **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 [24]). The boosted decision tree is an ensemble method that constructs a series of decision trees [36]. 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)**: It is based on random decision forests [30]. Specifically, it constructs multiple decision trees that vote on the most popular output class.[19]

• **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 [38].

• **Two-Class Logistic Regression (C-LR)**: This is a classic classifier that predicts the probability of an instance by fitting a logistic function.[20] 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 [28]. 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.[21]

• **Two-Class Support Vector Machine (C-SVM)**: SVM is another well-known classifier [19]. It works by separating the data with the "maximum-margin" hyperplane.[22]

[13] https://www.kaggle.com/c/kdd-cup-2014-predicting-excitement-at-donors-choose
[14] https://www.kaggle.com/c/predicting-red-hat-business-value
[15] https://www.kaggle.com/c/flavours-of-physics
[16] https://www.kaggle.com/c/flavours-of-physics/details/agreement-test
[17] https://www.kaggle.com/c/flavours-of-physics/details/correlation-test
[18] https://msdn.microsoft.com/en-us/library/azure/dn906036.aspx
[19] https://msdn.microsoft.com/en-us/library/azure/dn906008.aspx
[20] https://msdn.microsoft.com/en-us/library/azure/dn905994.aspx
[21] https://msdn.microsoft.com/en-us/library/azure/dn905947.aspx
[22] https://msdn.microsoft.com/en-us/library/azure/dn905835.aspx

4.3 Hyper-parameter Tuning

Each machine learning algorithm consists of a set of hyperparameters to tune. The methodology we use in this paper is to rely on the *default* tuning procedure provided by the machine learning service. In the full version of this paper [32], we summarize the hyper-parameters provided by the machine learning services. 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 (for binary classification) and automatically tunes the learning rate, we only 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 obtained with five-fold cross validation. (AUC is the evaluation metric used by all binary-classification competitions we included in `MLBench`.)

5. 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 service 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 Table 3 without the '-r' suffix). We will discuss the cases when raw features are used in Section 6.

5.1 Capacity and Universality

We first report the performance of **Azure** and **Amazon**, based on the *capacity* and *universality* metrics defined in Section 3.2.2. Figure 2 presents the result.

In Figure 2(a), the x-axis represents the quality tolerance, and the y-axis represents the minimum number of models required if a machine learning service can achieve a certain tolerance level τ (i.e., a ranking of top $\tau\%$) for all seven datasets (i.e., a τ-universality of seven). Figure 2(a) shows that, by only using C-SVM, **Azure** can be ranked within top 50%. In addition, if we further include C-AP and C-BDT, **Azure** can be ranked within top 31%. Moreover, including more models cannot improve the bottom line anymore. Note that this does not mean that the other models are useless — they can improve on some datasets but not the one where **Azure** observes the largest τ (i.e., the worst ranking).

The minimum τ shown in Figure 2(a) then implies the capacity of a machine learning service. We observe that the capacity of **Azure** is 31 (i.e., $c(\text{Azure}) = 31$), and the capacity of **Amazon** is 83 (i.e., $c(\text{Amazon}) = 83$). Under this measurement, state-of-the-art machine learning services are far from competitive than deliberate machine learning models designed manually: With the goal of meeting a τ-universality 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, **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 Figure 2(b) and 2(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.

Table 4: Capacity of **Azure** and **Amazon** on different datasets.

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

Table 5: Capacity of the logistic regression model (C-LR) from **Azure** and **Amazon** on different datasets.

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

5.1.1 Breakdown and Analysis

We next take a closer look at how the machine learning services perform in individual competitions. We note that not every winning code we found is top-ranked. If the top-ranked code is not available, 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 as the available model is unclear, the results presented in Figure 2 do suggest that the additional models provided by **Azure** do help. In more detail, Table 4 compares the capacity of **Azure** and **Amazon** on different datasets. We observe that **Azure** always 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 Figure 2.

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 we found) 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.)

Hyperparameter tuning makes a difference. Both **Azure** and **Amazon** provide logistic regression for binary classification. However, **Azure** provides more knobs for hyper-parameter tuning. Table 5 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. (We do not know how **Amazon** tunes the learning rate for logistic regression.) However, there is no free lunch: Hyper-parameter tuning is time-consuming (see Figures 5 and 6).

5.2 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

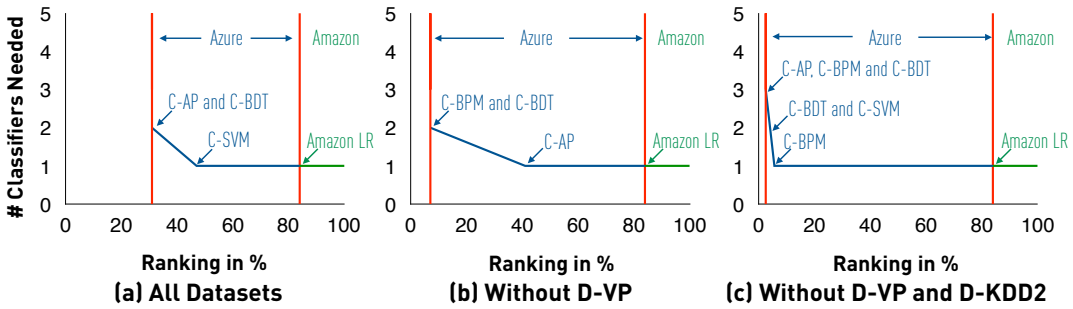


Figure 2: Capacity and universality of machine learning services as quality tolerance various.

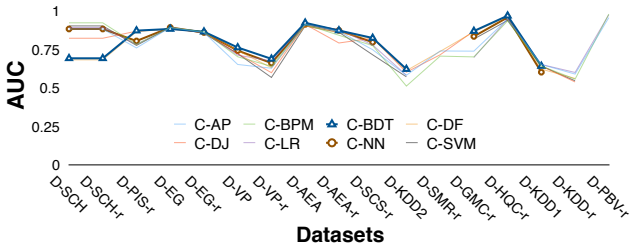


Figure 3: Quality of different models.

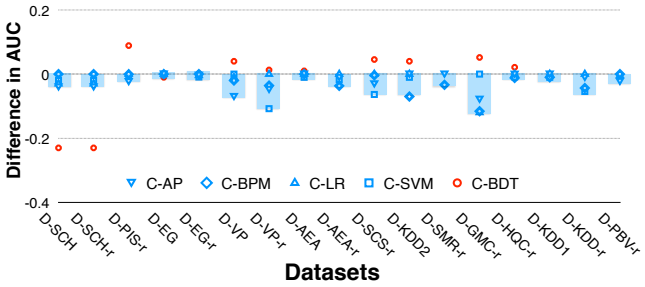


Figure 4: BDT vs. Linear Classifiers.

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 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.

5.2.1 *Linear vs. Nonlinear Models*

In Figure 2, 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 Figure 3.

We observe that nonlinear models (e.g., C-BDT) can outperform linear models (e.g., C-SVM) as the dataset size increases. (The x-axes of Figures 3 and 4 represent datasets ordered by their training sizes.) This is not surprising: 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 dataset D-SCH in Figure 3).

Figure 4 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. 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.

5.2.2 *Training Time vs. 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 Figures 5 and 6. Figure 5 presents the trade-off between the prediction quality and the total training time on hyper-parameter tuning, whereas Figure 6 presents the trade-off in the average sense (i.e., with respect to the average time spent on training a model under a specific hyper-parameter setting). 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 Figures 5 and 6, 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. For example, in Figure 6, the average training time of C-DJ is significantly longer than the

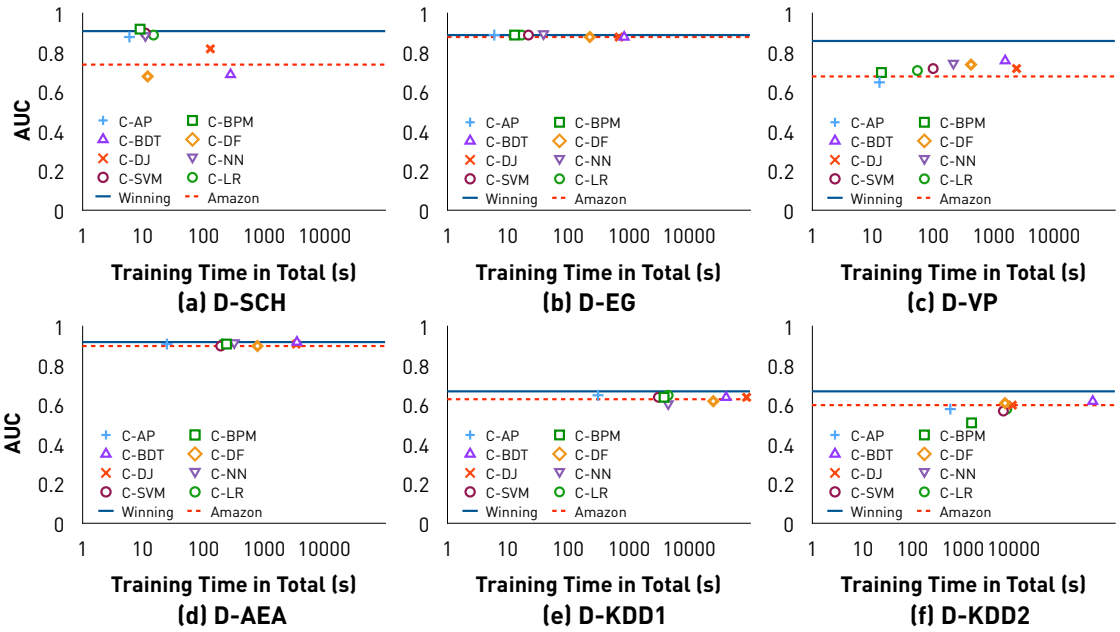
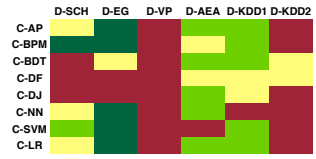


Figure 5: Tradeoff between prediction quality (AUC) and *total* training time. The blue line represents the AUC of the winning code, and the red line represents the AUC of logistic regression (C-LR) on **Amazon**.

Table 6: (a) AUC corresponding to the high tolerance regime of different Kaggle competitions. (b) 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.

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 D-KDD1	0.67	0.62

(a)



(b)

others over the dataset D-KDD1. (Note that the x-axis is at the logarithmic scale.) However, it is outperformed by even linear models such as C-AP, and its prediction quality is very close to that of C-BDT. Considering the average training time presented in Figure 6, C-AP is the choice among linear models, whereas C-BDT is the choice among nonlinear models.

5.2.3 *Quality Tolerance Regime*

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 services. (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 services we currently observe:

- *Low tolerance regime*. Corresponds to the case when the quality tolerance is below 1.[23]
- *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 Table 6(a), we present the AUC that a model has to achieve to meet the high tolerance regime, 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 Table 4, **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 but fails on the others.

To better understand the performance of **Azure** with respect to different quality tolerance regimes, we further present in Table 6(b) 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 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

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

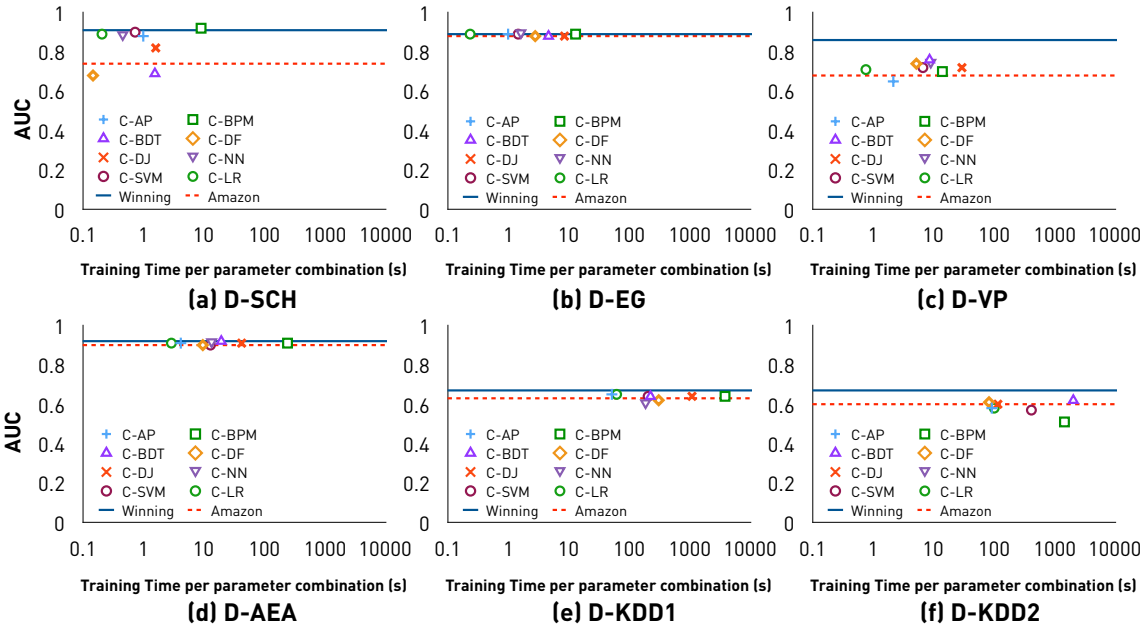


Figure 6: Tradeoff between prediction quality (AUC) and *average* training time per parameter. The blue line represents the AUC of the winning code and the red line represents the AUC of logistic regression (C-LR) on **Amazon**.

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.

This is also a summary that covers many observations that we have so far. In view of the Kaggle competitions (by reading the heat map vertically), some are more challenging than the others. For example, none of the models can meet even the high tolerance regime on the dataset D-VP, and only C-BDT can meet the high tolerance regime on the dataset D-KDD2. In view of the models (by reading the heat map horizontally), there is not a one-size-fits-all solution: No one can dominate the others across the datasets. Moreover, there is apparently a separation between the "comfortable zones" of the models: Linear models are more capable on small datasets, whereas nonlinear models are more capable on large datasets.

5.3 Summary and Discussion

Given the previous analysis, there is an obvious trade-off between the efficiency and effectiveness of machine learning services from a user's perspective. The more alternative models a machine learning service 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 services 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 services*. 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.

Table 7: Gaps between **Azure** and Kaggle winning code (DWD is shorthand for "distance weighted discrimination").

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

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.

Table 7 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 scores 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 Table 7 is that the winning code on the large datasets leverages ensemble methods (details in Section 3.4), 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 more work towards that to incorporate ensemble methods (as well as nonlinear models).[24]

[24]Very recently, Amazon launched a new machine learning service called **SageMaker** [11]. We have further evaluated these models

Table 8: Improvement on private AUC score attributed to feature engineering. All numbers are with hyperparameter tuning.

Dataset	C-AP	C-BPM	C-BDT	C-DF	C-DJ	C-LR	C-NN	C-SVM	BestVsBest
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%)	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%)	0.03 (3.20%)
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%)	0.07 (10.71%)
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%)	0.04 (5.03%)
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%)	0.02 (3.63%)
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%)	0.05 (8.64%)

Table 9: Improvement on private ranking attributed to feature engineering. All numbers are with hyperparameter tuning.

Dataset	C-AP	C-BPM	C-BDT	C-DF	C-DJ	C-LR	C-NN	C-SVM	BestVsBest
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%)	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%)	372 (59.52%)
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%)	432 (33.08%)
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%)	708 (41.97%)
D-KDD2	-62 (-13.14%)	-85 (-18.01%)	NA (NA)	244 (51.69%)	301 (63.77%)	-121 (-25.64%)	NA (NA)	157 (33.26%)	44 (9.32%)
D-KDD1	118 (25.00%)	289 (61.23%)	NA (NA)	257 (54.45%)	348 (73.73%)	65 (13.77%)	NA (NA)	343 (72.67%)	65 (13.77%)

6. FEATURE ENGINEERING

So far, our study has been focused on Kaggle competitions whose winning codes are available. That is, the datasets we have discussed consist of well-engineered features generated by winning codes. In practice, feature engineering is a critical factor that can significantly impact machine learning results. Unfortunately, feature engineering is often more of an art than a science. To better understand the impact of feature engineering on our results, we conduct a comparative study using the datasets with only raw features.

6.1 Feature Engineering In Winning Code

We start by investigating in more detail the feature engineering techniques explored by Kaggle winning codes. In general, the techniques are pretty diverse, many of which rely on specific semantics of the raw features. For example, in the dataset D-EG, the raw features represent Web page contents (i.e., text) and the winning code further constructs text-based features such as tf-idf scores of the Web pages. However, we also identify some techniques that are commonly used by multiple winning codes:

• (*Feature Transformation*) Most feature transformations performed by winning codes are rather simple. Common techniques include converting categorical features to numerical ones, standardizing feature representations such as converting dates to triples representing all date, month, and year information, and so on.

• (*Feature Aggregation*) The most common, aggregated features generated by winning codes are the *counting* features. That is, for each raw-feature column in a dataset, they generate additional features that represent the frequency of each distinct value in that column. Another type of aggregated feature widely exploited by winning codes is about *statistic* properties of raw-feature columns, such as the max, min, mean, median, and standard deviation of a column.

• (*Feature Selection Using Random Forests*) The most common feature selection method leveraged by winning codes is using influence scores generated by *random forests* [30] to select the most influential features. This is not surprising, though, as random forests are popular for feature ranking.

More advanced but less frequent feature engineering techniques used by winning codes include using singular-value decomposition (SVD), subsampling, fitting a (e.g., normal) distribution based on training data, etc. We refer the readers to the full version of this paper [32] for the complete details.

6.2 Impact 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."

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 Figures 8 and 9, 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). Hyperparameter 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 Figures 8 and 9 further show the improvement by the best model using engineered features versus the best model 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. One challenge here is to decide appropriate feature learning procedures for a given learning problem. Since this is orthogonal (but complementary) to the current work, we leave it as one of the future directions for exploration. Ideally, this should be integrated into machine learning services as part of the declarative service, therefore it might be a promising aspect for machine learning service service providers to consider as well.

over our datasets in `MLBench` [32]. We find that **SageMaker** outperforms **Amazon** significantly, due to the addition of new models.

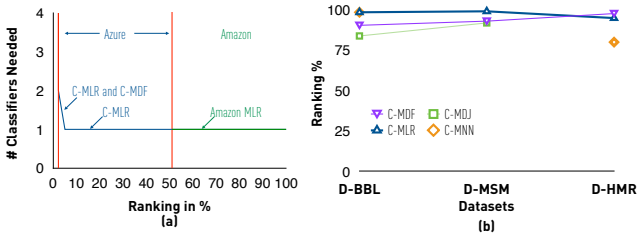


Figure 7: (a) Capacity and universality: The multi-class case. (b) Quality of different models: The multi-class case.

Table 10: Capacity of C-MLR from **Azure** and **Amazon**.

Dataset	**Azure** (C-MLR)	**Amazon**	Winning
D-BBL (211)	1.42	47.39	0.47
D-MSM (321)	0.80	NA	0.31
D-HMR (377)	4.98	3.18	0.27

7. BEYOND BINARY CLASSIFICATION

We have conducted similar case studies for multiclass classification and regression problems on **Azure** and **Amazon**. In this section, we briefly summarize our observations and leave the details to the full version of this paper [32].

7.1 Multi-Class Classification Competitions

Datasets and models. Using a protocol similar to that described in Section 3.3, we obtained ten datasets harvested from multi-class classification competitions on Kaggle, five with features extracted by winning codes whereas the other five with raw features. **Azure** supports four classifiers, namely, C-MDF (multiclass decision forest), C-MDJ (multiclass decision jungle), C-MLR (multiclass logistic regression), and C-MNN (multiclass neural network), whereas **Amazon** only supports C-MLR. Again, **Azure** provides more knobs for hyperparameter tuning — **Amazon** again automatically tunes the learning rate for its C-MLR.

Capacity and universality. Figure 7(a) shows the capacity of **Azure** and **Amazon** over all datasets with winning features (i.e., with universality of 3). Like the binary-classification case, **Azure** exhibits a much better capacity than **Amazon** because of the additional models it supports and more opportunities for hyperparameter tuning. (The capacity of **Azure** is 2 whereas the capacity of **Amazon** is 52.) While both **Azure** and **Amazon** support C-MLR, their performances differ dramatically as shown in Table 10. We speculate this is due to hyperparameter tuning of C-MLR on **Azure**.

Quality of models. Figure 7(b) further compares the qualities of the **Azure** models over different datasets. Unlike the binary classification competitions where we can directly compare the AUC scores, different multiclass classification competitions use different evaluation metrics such as *accuracy* and *log loss*. We therefore instead compare their rankings — this is one benefit of using ranking as our performance measure as we have discussed in Section 3.2.3. Interestingly, for the multiclass datasets `MLBench` includes, the (generalized) linear model C-MLR can outperform the other nonlinear models such as C-MDF and C-MNN. Meanwhile, the performance gap between **Azure** and the Kaggle winning codes is once again due to the lack of ensemble methods (see [32]).

7.2 Regression Competitions

`MLBench` includes 13 datasets harvested from regression competitions on Kaggle. Five of the datasets consist of winning features, whereas the others contain raw features. **Azure** supports seven models for regression problems, namely, C-BLR (Bayesian linear regression), C-BDR (boosted decision tree regression), C-DFR (decision forest regression), C-FFQ (fast forest quantile regression), C-LRR (linear regression), C-NNR (neural network regression), and C-PR (Poisson regression). All models come with knobs for hyperparameter tuning. In contrast, **Amazon** again only supports C-LRR and automatically tunes its learning rate.

Figure 8: Capacity and universality: The regression case.

Figure 8 presents the capacity of **Azure** and **Amazon** over all datasets as well as just the datasets D-LDP1, D-LDP2, and D-LDP3. Although **Azure** outperforms **Amazon** in both cases, both of them perform poorly when all datasets are involved (also see [32]).

8. RELATED WORK

There has been research on benchmarking machine learning algorithms and comparing their quality on various datasets [13, 16, 23]. 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 [22]. `MLBench` 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 existing machine learning models, we focus on those provided by existing machine learning services and try to understand whether the current abstraction is enough to support users of these services.

Benchmarking cloud services and relational databases have been an active research topic for decades. Famous benchmarks include the Wisconsin benchmark [20] and TPC benchmarks [12]. There are also benchmarks targeting clouds for different purposes, especially for data processing and management [18, 34]. Our work is motivated by the success and impact of these benchmarks, and we hope to establish the first benchmark for machine learning services.

9. CONCLUSION

In this paper, we presented `MLBench`, a dataset we constructed by collecting winning code from Kaggle competitions. Unlike existing datasets, `MLBench` contains not only the raw features for a machine learning task, but also those used by the winning teams of Kaggle competitions. `MLBench` then enables comparing quality of machine learning services against best-effort systems made by human experts, using novel measures such as capacity, universality, and quality regime.

As an example use case, we further conducted an empirical study on the performance of state-of-the-art declarative machine learning services using `MLBench`. Our results demonstrate the performance gap between machine learning services and Kaggle winning code. Detailed investigation further reveals that lack of adopting ensemble methods is one important reason.

(Acknowledgement) CZ and the DS3Lab gratefully acknowledge the support from the Swiss National Science Foundation NRP 75 407540_167266, IBM Zurich, Mercedes-Benz Research & Development North America, Oracle Labs, Swisscom, Zurich Insurance, Chinese Scholarship Council, and the Department of Computer Science at ETH Zurich, the GPU donation from NVIDIA Corporation, the cloud computation resources from Microsoft Azure for Research award program.

10. REFERENCES

[1] https://studio.azureml.net/.

[2] https://aws.amazon.com/aml/.

[3] https://www.kaggle.com/.

[4] https://github.com/KKPMW/Kaggle-MLSP-Schizo-3rd.

[5] https://github.com/saffsd/kaggle-stumbleupon2013.

[6] https://github.com/Cardal/Kaggle_WestNileVirus.

[7] https://github.com/owenzhang/Kaggle-AmazonChallenge2013.

[8] https://www.datarobot.com/blog/datarobot-the-2014-kdd-cup/.

[9] https://github.com/gramolin/flavours-of-physics.

[10] http://docs.aws.amazon.com/machine-learning/latest/dg/learning-algorithm.html.

[11] https://aws.amazon.com/sagemaker/.

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

[13] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine learning*, 36(1-2):105–139, 1999.

[14] A. P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159, 1997.

[15] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[16] R. Caruana et al. An empirical comparison of supervised learning algorithms. In *ICML*, pages 161–168, 2006.

[17] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *KDD*, pages 785–794, 2016.

[18] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with YCSB. In *SoCC*, pages 143–154, 2010.

[19] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[20] D. J. DeWitt. The Wisconsin benchmark: Past, present, and future. In *The Benchmark Handbook for Database and Transaction Systems*, pages 269–316. 1993.

[21] D. Dheeru and E. Karra Taniskidou. UCI machine learning repository, 2017.

[22] P. Domingos. A few useful things to know about machine learning. *CACM*, 55(10):78–87, 2012.

[23] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems. *JMLR*, 15(1):3133–3181, 2014.

[24] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *JCSS*, 1997.

[25] J. Friedman, T. Hastie, and R. Tibshirani. glmnet: Lasso and elastic-net regularized generalized linear models. *R package version*, 1(4), 2009.

[26] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[27] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[28] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, 2nd edition, 1998.

[29] R. Herbrich, T. Graepel, and C. Campbell. Bayes point machines. *JMLR*, 1(Aug):245–279, 2001.

[30] T. K. Ho. Random decision forests. In *ICDAR*, volume 1, pages 278–282, 1995.

[31] T. Li, J. Zhong, J. Liu, W. Wu, and C. Zhang. Ease.ml: Towards multi-tenant resource sharing for machine learning workloads. *PVLDB*, 11(5):607–620, 2018.

[32] Y. Liu, H. Zhang, L. Zeng, W. Wu, and C. Zhang. How good are machine learning clouds for binary classification with good features? *ArXiv*, 2017.

[33] M. Lui. Feature stacking for sentence classification in evidence-based medicine. In *ALTA*, pages 134–138, 2012.

[34] C. Luo et al. Cloudrank-d: benchmarking and ranking cloud computing systems for data processing applications. *Frontiers of Computer Science*, 6(4):347–362, 2012.

[35] J. S. Marron et al. Distance-weighted discrimination. *JASA*, 102(480):1267–1271, 2007.

[36] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[37] G. Ridgeway et al. gbm: Generalized boosted regression models. *R package version*, 1(3):55, 2006.

[38] J. Shotton, T. Sharp, P. Kohli, S. Nowozin, J. M. Winn, and A. Criminisi. Decision jungles: Compact and rich models for classification. In *NIPS*, pages 234–242, 2013.