Complexity vs. Performance: Empirical Analysis of Machine Learning as a Service

Yuanshun Yao
ysyao@cs.uchicago.edu
University of Chicago

Zhujun Xiao
zhujunxiao@cs.uchicago.edu
University of Chicago

Bolun Wang
bolunwang@cs.ucsb.edu
UCSB/University of Chicago

Bimal Viswanath
viswanath@cs.uchicago.edu
University of Chicago

Haitao Zheng
htzheng@cs.uchicago.edu
University of Chicago

Ben Y. Zhao
ravenben@cs.uchicago.edu
University of Chicago

ABSTRACT

Machine learning classifiers are basic research tools used in numerous types of network analysis and modeling. To reduce the need for domain expertise and costs of running local ML classifiers, network researchers can instead rely on centralized Machine Learning as a Service (MLaaS) platforms.

In this paper, we evaluate the effectiveness of MLaaS systems ranging from fully-automated, turnkey systems to fully-customizable systems, and find that with more user control comes greater risk. Good decisions produce even higher performance, and poor decisions result in harsher performance penalties. We also find that server side optimizations help fully-automated systems outperform default settings on competitors, but still lag far behind well-tuned MLaaS systems which compare favorably to standalone ML libraries. Finally, we find classifier choice is the dominating factor in determining model performance, and that users can approximate the performance of an optimal classifier choice by experimenting with a small subset of random classifiers. While network researchers should approach MLaaS systems with caution, they can achieve results comparable to standalone classifiers if they have sufficient insight into key decisions like classifiers and feature selection.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Applied computing**;

1 INTRODUCTION

Machine learning (ML) classifiers are now common tools for data analysis. They have become particularly indispensable in the context of networking, where characterizing the behavior of protocols, services, and users often requires large scale data mining and modeling. ML tools have pervaded problems from all facets of networking, with examples ranging from network link prediction [43, 54], network localization [41, 77], user behavior analysis [71, 72], congestion control protocols [59, 74], performance characterization [8, 76], botnet detection [31], and network management [1].

As ML tools are increasingly commoditized, most network researchers are interested in them as black box tools, and lack the resources to optimize their deployments and configurations of ML systems. Without domain experts or instructions on building custom-tailored ML systems, some have tried developing automated or "turnkey" ML systems for network diagnosis [42]. A more mature alternative is ML as a Service (MLaaS), with offerings from Google, Amazon, Microsoft and others. These services run on the cloud, and provide a query interface to an ML classifier trained on uploaded datasets. They simplify the process of running ML systems by abstracting away challenges in data storage, classifier training, and classification.

Given the myriad of decisions in designing any ML system, it is fitting that MLaaS systems cover the full spectrum between extreme simplicity (turn-key, nonparametric solutions) and full customizability (fully tunable systems for optimal performance). Some are simple black-box systems that do not even reveal the classifier used, while others offer users choice in everything from data preprocessing, classifier selection, feature selection, to parameter tuning.

MLaaS today are opaque systems, with little known about their efficacy (in terms of prediction accuracy), their underlying mechanisms and relative merits. For example, how much freedom and configurability do they give to users? What is the difference in potential performance between fully configurable and turnkey, "black-box" systems? Can MLaaS providers build in better optimizations that outperform hand-tuned user configurations? Do MLaaS systems offer enough configurability to match or surpass the performance of locally tuned ML tools?

In this paper, we offer a first look at empirically quantifying the performance of 6 of the most popular MLaaS platforms across a large number (119) of labeled datasets for binary classification. Our goals are three-fold. *First*, we seek to understand how MLaaS systems compare in performance against each other, and against a fully customized and tuned local ML library. Our results will shed light on the cost-benefit tradeoff of relying on MLaaS systems instead of locally managing ML systems. *Second*, we wish to better understand the correlations between complexity, performance and performance variability. Our results will not only help users choose between MLaaS providers based on their needs, but also guide companies in traversing the complexity and performance tradeoff when

IMC '17, November 1–3, 2017, London, United Kingdom

ACM ISBN 978-1-4503-5118-8/17/11…$15.00
https://doi.org/10.1145/3131365.3131372

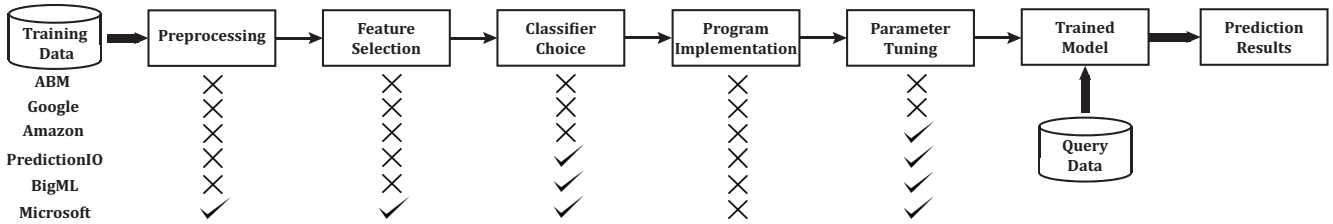


Figure 1: Standard ML pipeline and the steps that can be controlled by different MLaaS platforms.

building their own local ML systems. *Third*, we want to understand which key knobs have the biggest impact on performance, and try to design generalized techniques to optimize those knobs.

Our analysis produces a number of interesting findings.

- *First*, we observe that current MLaaS systems cover the full range of tradeoffs between ease of use and user-control. Our results show a clear and strong correlation between increasing configurability (user control) and both higher optimal performance and higher performance variance.
- *Second*, we show that classifier choice accounts for much of the benefits of customization, and that a user can achieve near-optimal results by experimenting with a small random set of classifiers, thus dramatically reducing the complexity of classifier selection.
- *Finally*, our efforts find clear evidence that fully automated (black-box) systems like Google and ABM are using server-side tests to automate classifier choices, including differentiating between linear and non-linear classifiers. We note that their mechanisms occasionally err and choose suboptimal classifiers. As a whole, this helps them outperform other MLaaS systems using default settings, but they still lag far behind tuned versions of their competitors. Most notably, a heavily tuned version of the most customizable MLaaS system (Microsoft) produces performance nearly-identical to our locally tuned ML library (scikit-learn).

To the best of our knowledge, this paper is the first effort to empirically quantify the performance of MLaaS systems. We believe MLaaS systems will be an important tool for network data analysis in the future, and hope our work will lead to more transparency and better understanding of their suitability for different network research tasks.

2 UNDERSTANDING MLAAS PLATFORMS

MLaaS platforms are cloud-based systems that provide machine learning as a web service to users interested in training, building, and deploying ML models. Users typically complete an ML task through a web page interface. These platforms simplify and make ML accessible to even non-experts. Another selling point is the affordability and scalability, as these services inherit the strengths of the underlying cloud infrastructure.

For our analysis, we choose 6 mainstream MLaaS platforms, including Amazon Machine Learning (Amazon[1]), Automatic Business Modeler (ABM[2]), BigML[3], Google Prediction API (Google[4]), Microsoft Azure ML Studio (Microsoft[5]), and PredictionIO[6]. These are the MLaaS services widely available today.

The MLaaS Pipeline. Figure 1 shows the well-known sequence of steps typically taken when using any user-managed ML software. For a given ML task, a user first preprocesses the data, and identifies the most important features for the task. Next, she chooses an ML model (*e.g.* a classifier for a predictive task) and an appropriate implementation of the model (since implementation difference could cause performance variation [9]), tunes parameters of the model and then trains the model. Specific MLaaS platforms can simplify this pipeline by only exposing a subset of the steps to the user while automatically managing the remaining steps. Figure 1 also shows the steps exposed to users by each platform. Note that some (ABM and Google) expose none of the steps to the user but provide a "1-click" mode that trains a predictive model using an uploaded dataset. At the other end of the spectrum, Microsoft provides *control* for nearly every step in the pipeline.

Control and Complexity. It is intuitive that more control over each step in the pipeline allows knowledgeable users to build higher quality models. Feature, model, and parameter selection can have significant impact on the performance of an ML task (*e.g.* prediction). However, successfully optimizing each step requires overcoming significant *complexity* that is difficult without in-depth knowledge and experience. On the other hand, when limiting control, it is unclear whether services can perform effective automatic management of the pipeline and parameters, *e.g.* in the case of ABM and Google. Current MLaaS systems cover the whole gamut in terms of user control and complexity and provide an opportunity to investigate the impact of complexity on performance.

We summarize the controls available in the pipeline for classification tasks in each platform. More details are available in Section 3.

- *Preprocessing:* The first step involves dataset processing. Common preprocessing tasks include *data cleaning* and *data transformation*. Data cleaning typically involves handling missing feature values, removing outliers, removing incorrect or duplicate records. None of the 6 systems provides any support for automatic data cleaning and expects the uploaded data to be already sanitized with errors removed. Data transformation usually involves normalizing or scaling feature values to lie within

[1] https://aws.amazon.com/machine-learning

[2] http://e-abm.com
[3] https://bigml.com
[4] https://cloud.google.com/prediction
[5] https://azure.microsoft.com/en-us/services/machine-learning
[6] http://predictionio.incubator.apache.org

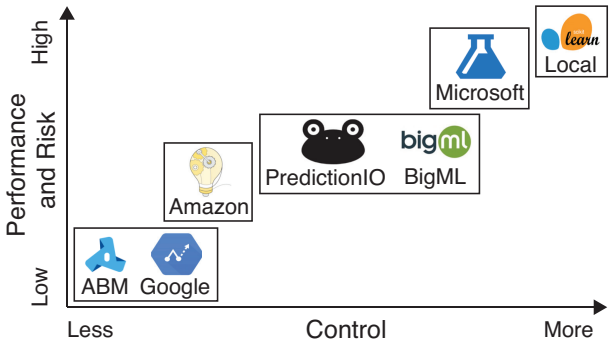


Figure 2: Overview of control vs. performance/risk tradeoffs in MLaaS platform.

certain ranges. This is particularly useful when features lie in different ranges, where it becomes harder to compare variations in feature values that lie in a large range with those that lie in a smaller range. Microsoft is the only platform that provides support for data transformation.

- *Feature selection:* This step selects a subset of features most relevant to the ML task, *e.g.* those that provide more predictive power for the task. Feature selection helps improve classification performance, and also simplifies the problem by eliminating irrelevant features. A popular type of feature selection scheme is *Filter method*, where a statistical measure (independent of the classifier choice) is used to rank features based on their class discriminatory power. Only Microsoft supports feature selection and provides 8 Filter methods. Some platforms, *e.g.* BigML, provide user-contributed scripts for feature selection. We exclude these cases since they are not officially supported by the platform and require extra effort to integrate them into the ML pipeline.
- *Classifier selection:* Different classifiers can be chosen based on the complexity of the dataset. An important complexity measure is the linearity (or non-linearity) of the dataset, and classifiers can be chosen based on their capability of estimating a linear or non-linear decision boundary. Across all platforms, we experiment with 10 classifiers. ABM and Google offer no user choices. Amazon only supports Logistic Regression[7]. BigML provides 4 classifiers, PredictionIO provides 8, while Microsoft gives the largest number of choices: 9.
- *Parameter tuning:* These are parameters associated with a classifier and they must be tuned for each dataset to build a high quality model. Amazon, PredictionIO, BigML, and Microsoft all support parameter tuning. Usually each classifier allows users to tune 3 to 5 parameters. We include detailed information about classifiers and their parameters in Section 3.

Key Questions. To help understand the relationships between complexity, performance, and transparency in MLaaS platforms, we focus our analysis around three key questions and briefly summarize our findings. Figure 2 provides a simple visualization to aid our discussion.

- *How does the complexity (or control) of ML systems correlate with ideal model accuracy?* Assuming we cover the available configuration space, how strongly do constraints in complexity limit model accuracy in practice? How do different controls compare in relative impact on accuracy?
  Answer: Our results show a clear and strong correlation between increasing complexity (user control) and higher optimal performance. Highly tunable platforms like Microsoft outperform others when configurations of the ML model are carefully tuned. Among the three control dimensions we investigate, classifier choice accounts for the most benefits of customization.
- *Can increased control lead to higher risks (of building a poorly performing ML model)?* Real users are unlikely to fully optimize each step of the ML pipeline. We quantify the likely performance variation at different levels of user control. For instance, how much would a poor decision in classifier cost the user in practice on real classification tasks?
  Answer: We find higher configurability leads to higher risks of producing poorly performing models. The highest levels of performance variation also come from choices in classifiers. We also find that users only need to explore a small random subset of classifiers (3 classifiers) to achieve near-optimal performance instead of experimenting with an entire classifier collection.
- *How much can MLaaS systems optimize the automated portions of their pipeline?* Despite their nature as black boxes, we seek to shed light on hidden optimizations at the classifier level in ABM and Google. Are they optimizing classifiers for different datasets? Do these internal optimizations lead to better performance compared to other MLaaS platforms?
  Answer: We find evidence that black-box platforms, *i.e.* Google and ABM, are making a choice between linear and non-linear classifiers based on characteristics of each dataset. Results show that this internal optimization successfully improves these platforms' performance, when compared to other MLaaS platforms (Amazon, PredictionIO, BigML and Microsoft) without tuning any available controls. However, in some datasets, a naive optimization strategy that we devised makes better classifier decisions and outperforms them.

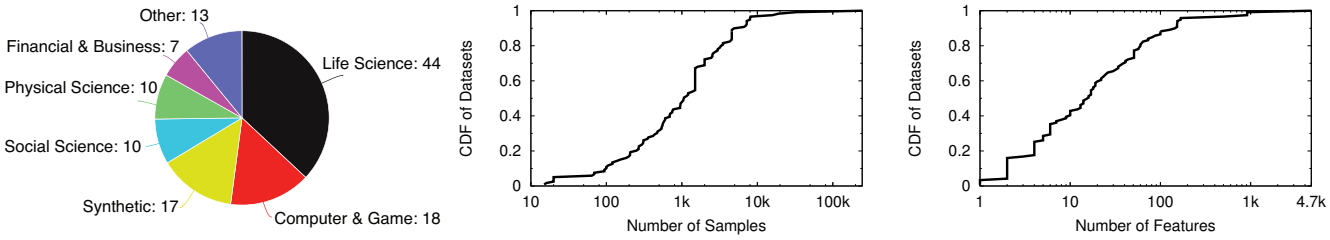
3 METHODOLOGY

We focus our efforts on binary classification tasks, since that is one of the most common applications of ML models in deployed systems. Moreover, binary classification is one of the two learning tasks (the other being regression) that are commonly supported by all 6 ML platforms. Other learning tasks, *e.g.* clustering and multi-class classification, are only supported by a small subset of platforms.

3.1 Datasets

We describe the datasets we used for training ML classifiers. We use 119 labeled datasets from diverse application domains such as life science, computer games, social science, and finance *etc.* Figure 3(a)

[7] Amazon does not specify which classifier is used during the model training, but this information is claimed in its documentation page: http://docs.aws.amazon.com/machine-learning/latest/dg/types-of-ml-models.html.



(a) Breakdown of application domains. (b) Distribution of sample numbers. (c) Distribution of feature numbers.

Figure 3: Basic characteristics of datasets used in our experiments.

shows the detailed breakdown of application domains. The majority of datasets (94 out of 119) are from the popular UCI machine learning repository [3], which is widely adopted for benchmarking ML classifiers. The remainder include 16 popular synthetic datasets from scikit-learn[8], and 9 datasets used in other applied machine learning studies [5, 13, 17, 18, 32, 33, 70, 73][9]. It is also important to highlight that our datasets vary widely in terms of the number of samples and number of features, as shown in Figure 3(b) and Figure 3(c). Datasets vary in size from 15 samples to 245, 057 samples, while the dimensionality of datasets ranges from 1 to 4, 702. Note that we limit the number of extremely large datasets (with size over 100k) due to the high computational complexity incurred in using them on MLaaS platforms. We include complete information about all datasets separately[10]. As none of the MLaaS platforms provides any support for data cleaning, we perform the following data preprocessing steps locally before uploading to MLaaS platforms. Our datasets include both numeric and categorical features. Following prior conventions [23], we convert all categorical features to numerical values by mapping $\{C_1, ..., C_N\}$ to $\{1, ..., N\}$. We acknowledge that this may impact performance of some classifiers, *e.g.* distance-based classifiers like kNN [45]. But since our goal is to compare performance across different platforms instead of across classifiers, this preprocessing is unlikely to change our conclusions. For datasets with missing values, we replace missing fields with median values of corresponding features, which is a common ML preprocessing technique [62]. Finally, for each dataset, we randomly split data samples into training and test set by 70%–30% ratio. We train classifiers on each MLaaS platforms using the same training and held-out test set. We report classification performance on the test set.

3.2 MLaaS Platform Measurements

In this section, we describe our methodology for measuring classification performance of MLaaS platforms when we vary available controls.

Choosing Controls of an ML System. As mentioned in Section 2, we break down an ML system into 5 dimensions of control. In this paper, we consider 4 out of 5 dimensions by excluding Program Implementation which is not controllable in any platform. The remaining dimensions are grouped into three categories, Preprocessing (data transformation) and Feature Selection (FEAT), Classifier Choice (CLF), and Parameter Tuning (PARA). Note that we combine Preprocessing with Feature Selection to simplify our analysis, as both controls are only available in Microsoft. In the rest of the paper, we interchangeably use the term Feature Selection and FEAT to refer to this combined category. Overall, these three categories of control present the easiest and most impactful options for users to train and build high quality ML classifiers. As baselines for performance comparison, we use two reference points that represent the extremes of the complexity spectrum, one with no user-tunable control, and one where users have full control over all control dimensions. To simulate an ML system with no control, we set a default choice for each control dimension. We refer to this configuration as **baseline** in later sections. Since not all of the 6 platforms we study have a default classifier, we use Logistic Regression as the baseline, as it is the only classifier supported by all 4 platforms (where the control is available). All MLaaS platforms select a default set of parameters for Logistic Regression (values and parameters vary across platforms), and we use them for the baseline settings. We perform no feature selection for the baseline settings. To simulate an ML system with full control, we use a local ML library, scikit-learn, as this library allows us to tune all control dimensions. We refer to this configuration as **local** in later sections.

Performing Measurements by Varying Controls. We evaluate performance of MLaaS platforms on each dataset by varying available controls. Table 1 provides detailed information about available choices for each control dimension. We vary the FEAT and CLF dimensions by simply applying all available choices listed for each system in Table 1. It is interesting to note that the CLF choices vary across platforms even though all platforms are competing to provide the same service, *i.e.* binary classification. For example, Random Forests and Boosted Decision Tree, best performing classifiers based on prior work [14, 15], are only available on Microsoft. The PARA dimension is varied by applying grid search. We explore all possible options for categorical parameters. For example, we include both L1 and L2 in regularization options from Logistic Regression. For numerical parameters, we start with the default value provided by platforms and scan a range of values that are two orders of magnitude lower and higher than the default. In other words, for each numerical parameter with a default value of D, we investigate

[8] http://scikit-learn.org
[9] There are two datasets used in [73].
[10] http://sandlab.cs.uchicago.edu/mlaas

Platform	FEAT	CLF (# of parameter tuned: parameter list (PARA))
Amazon	×	**Logistic Regression** (3: maxIter, regParam, shuffleType)
PredictionIO	×	**Logistic Regression** (3: maxIter, regParam, fitIntercept), **Naive Bayes** (1:lambda), **Decision Tree** (2: numClasses, maxDepth),
BigML	×	**Logistic Regression** (3: regularization, strength, eps), **Decision Tree** (3: node threshold, ordering, random candidates), **Bagging** [11] (3: node threshold, number of models, ordering), **Random Forests** [12] (3: node threshold, number of models, ordering)
Microsoft	**Fisher LDA**, **Filter-based** (using Pearson, Mutual, Kendall, Spearman, Chi, Fisher, Count)	**Logistic Regression** (4: optimization tolerance, L1 regularization weight, L2 regularization weight, memory size for L-BFGS), **Support Vector Machine** (2: # of iterations, Lambda), **Averaged Perceptron** [27] (2: learning rate, max. # of iterations), **Bayes Point Machine** [34] (1: # of training iteration), **Boosted Decision Tree** [28] (4: max. # of leaves per tree, min. # of training instances per leaf, learning rate, # of trees constructed), **Random Forests** (5: resampling method, # of decision trees, max. depth of trees, # of random splits per node, min. # of samples per leaf), **Decision Jungle** [58] (5: resampling method, # of DAGs, max. depth of DAGs, max. width of DAGs, # of optimization step per DAG layer),
scikit-learn	**FClassif**, **MutualInfoClassif**, **GaussianNorm**, **MinMaxScaler**, **MaxAbsScaler**, **L1Normalization**, **L2Normalization**, **StandardScaler**	**Logistic Regression** (3: penalty, C, solver), **Naive Bayes** (1: prior), **Support Vector Machine** (3: penalty, C, loss), **Linear Discriminant Analysis** (2: solver, shrinkage), **k-Nearest Neighbor** (3: n_neighbors, weights, p), **Decision Tree** (2: criterion, max_features), **Boosted Decision Tree** (3: n_estimators, criterion, max_features), **Bagging** (2: n_estimators, max_features), **Random Forests** (2: n_estimators, max_features), **Multi-Layer Perceptron** [52] (3: activation, solver, alpha)

Table 1: Detailed configurations for MLaaS platforms and local library measurement experiments. For each control dimension, we list available configurations (feature selection methods, classifiers, and tunable parameters).

Platform	# Feature Selections	# Classifiers	# Parameters	# Measurements
ABM	-	1 (1)	-	119
Google	-	1 (1)	-	119
Amazon	-	1 (1)	3 (3)	4,284
PredictionIO	-	3 (8)	6 (25)	3,719
BigML	-	4 (4)	12 (46)	12,838
Microsoft	8 (8)	7 (9)	23 (34)	1,728,791
scikit-learn	8 (14)	10 (14)	32 (111)	2,137,410

Table 2: Scale of the measurements. The last column shows total number of configurations we tested on each platform. Numbers in parenthesis in column #2 to #4 show the number of available options shown to users on each platform, while numbers outside parenthesis show the number of options we explore in experiments.

three values: $\frac{D}{100}$, D, and $100 \times D$. For example, we explore 0.0001, 0.01 and 1 for the regularization strength parameter in Logistic Regression, where the default value is 0.01. We also manually examine the parameter type and its acceptable value range to make sure the parameter value is valid.

Table 2 shows the total number of measurements we perform for each platform and the number of choices for each control dimension. All experiments were performed between October 2016 and February 2017. For platforms with no control, we perform one measurement per dataset, giving us 119 prediction results (ABM and Google). At the other extreme, Microsoft requires over 1.7M measurements, given the large number of available controls. Note that numbers in the last column is much larger than the product of numbers in previous columns, because for each parameter we tune, we explore multiple values, resulting in a larger number of total measurements. To set up experiments, we leverage web APIs provided by the platforms, allowing us to automate experiments through scripts. Unfortunately, Microsoft only provides an API for using preconfigured ML models on different datasets, and there is no API for configuring ML models. Hence, in the case of Microsoft, we manually configure ML models (over 200 model configurations) using the web GUI, and then automate the application of the models to all datasets.

Evaluation Metrics. We measure the performance of a platform by computing the *average F-score across all datasets*. F-score is a better metric compared to accuracy as many of our datasets have imbalanced class distributions. It is defined as the harmonic mean of *precision* and *recall*. Precision is the fraction of samples predicted to be positive that are truly positive and recall is the fraction of positive samples that are correctly predicted. Note that other metrics like Area Under Curve or Average Precision are also not biased by imbalanced datasets, but unfortunately cannot be applied, as PredictionIO and several classifiers on BigML do not provide a prediction score.

To validate whether a single metric (Average F-score) is representative of performance across all the datasets, we compute the *Friedman ranking* [55] of platforms across all the datasets. Friedman ranking statistically ranks platforms by considering a given metric (*e.g.* F-score) across all datasets. A platform with a higher Friedman rank exhibits statistically better performance when considering all datasets, compared to a lower ranked platform. We observe that the platform ranking based on average F-score is consistent with the Friedman ranking (using F-score), suggesting that average F-score is a representative metric. In the rest of the paper, the *performance* of a platform refers to the *average F-score across datasets*.

4 COMPLEXITY VS. PERFORMANCE

We have shown that MLaaS platforms represent ML system designs with different levels of complexity and user control. In this section, we try to answer our first question: *How does the performance of ML systems vary as we increase their complexity?*

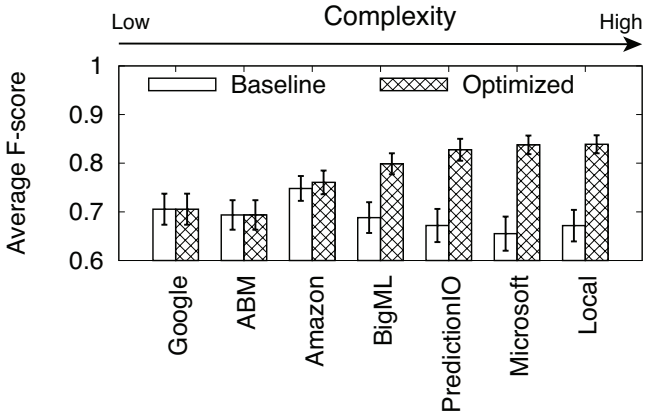


Figure 4: Optimized and baseline performance (F-score) of platforms and local library.

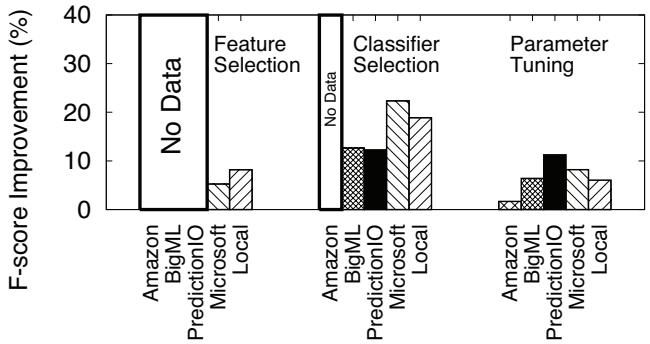


Figure 5: Relative improvement in performance (F-score) over baseline as we tune individual controls (white boxes indicate controls not supported).

(a) Baseline performance.

Platform	Avg. Fried. Ranking	Avg. F-score	Avg. Accuracy	Avg. Precision	Avg. Recall
Amazon	253.7	0.748 (250.5)	0.850 (269.5)	0.782 (298.0)	0.755 (196.7)
Google	267.7	0.706 (261.4)	0.851 (217.7)	0.751 (261.4)	0.711 (330.4)
ABM	344.5	0.694 (285.8)	0.833 (366.5)	0.738 (359.3)	0.691 (366.6)
BigML	348.1	0.688 (326.8)	0.822 (347.2)	0.741 (335.7)	0.688 (385.6)
PredictionIO	379.5	0.672 (389.2)	0.818 (432.6)	0.682 (387.6)	0.741 (308.9)
Local	388.8	0.672 (411.9)	0.832 (401.8)	0.668 (419.4)	0.723 (322.1)
Microsoft	424.3	0.655 (477.3)	0.833 (391.9)	0.715 (370.5)	0.659 (457.6)

(b) Optimized performance.

Platform	Avg. Fried. Ranking	Avg. F-score	Avg. Accuracy	Avg. Precision	Avg. Recall
Local	190.1	0.839 (179.4)	0.916 (184.2)	0.984 (201.3)	0.990 (195.5)
Microsoft	211.1	0.837 (186.5)	0.914 (190.3)	0.954 (231.3)	0.863 (236.3)
PredictionIO	318.6	0.828 (245.7)	0.886 (238.7)	0.779 (478.4)	0.852 (311.5)
BigML	365.9	0.789 (307.5)	0.876 (281.7)	0.880 (287.9)	0.802 (351.4)
Amazon	446.7	0.761 (545.3)	0.863 (524.3)	0.826 (398.2)	0.795 (318.9)
Google	641.9	0.706 (692.6)	0.853 (606.7)	0.744 (605.5)	0.704 (662.9)
ABM	758.8	0.694 (784.3)	0.834 (774.1)	0.735 (747.7)	0.684 (729.1)

Table 3: Baseline and optimized performance of MLaaS platforms. The Friedman ranking of each metric is included in the parenthesis. Lower Friedman ranking indicates consistently higher performance across all datasets.

4.1 Optimized Performance

First we evaluate the optimized performance each MLaaS platform can achieve by tuning all possible controls provided by the platform, *i.e.* FEAT, CLF, and PARA. In this process, we train individual models for all possible combinations of the 3 controls (whenever available) and use the best performing model for each dataset. We report the average F-score across all datasets for each platform as its performance. We refer to these results as **optimized**. Note that the optimized performance is simply the highest performance on the test set that is obtained by training different models using all available configurations. We do not optimize the model on test set.

We also generate the corresponding reference points, *i.e.* **baseline** and **local**. For **local**, we compute the highest performance on our local ML library by tuning all 3 control dimensions. For baseline, we measure the performance of "fully automated", zero-control versions of all systems (MLaaS and our local library), by using the baseline configurations for each platform. As mentioned earlier, these reference points capture performance at two ends of the complexity spectrum (no control vs. full control).

Figure 4 shows the optimized average F-score for each MLaaS platform, together with the optimized results. Platforms are listed on the x-axis based on increasing complexity. We observe a strong correlation between system complexity and the optimized classification performance. The platform with highest complexity (Microsoft) shows the highest performance (0.83 average F-score), and performance decreases as we consider platforms with lower complexity/control (Google and ABM), with ABM showing the lowest performance (0.71 F-score). As expected, the local library outperforms all MLaaS platforms, as it explores the largest range of model configurations (most feature selections techniques, classifiers, and parameters). Note that the performance difference between local and MLaaS platforms with high complexity is smaller, suggesting that adding more complexity and control beyond Microsoft brings diminishing returns. In addition, when we compare the baseline performance with the optimized performance for platforms with high complexity (Microsoft), the difference is significant, with up to 26.7% increase in F-score, further indicating that higher complexity provides room for more performance improvement. Lastly, the error bars show the standard error of the measured performance, and we observe that the statistical variation of performance measures for different platforms is not large.

For completeness, we include the detailed baseline and optimized performance of MLaaS platforms in Table 3. We include *F-score*, and other 3 metrics, *accuracy*, *precision*, and *recall*. We also compute the *Friedman ranking* of each metric across datasets [55]. A lower Friedman ranking indicates consistently higher performance over all datasets. Platforms in both tables are ordered based on average Friedman ranking over 4 evaluation metrics in ascending order. We can see that average F-score is a representative metric, because the ranking based on F-score values matches the ranking induced by the Friedman metric.

4.2 Impact of Individual Controls

We have shown that higher complexity in the form of more user control contributes to higher optimized performance. Now we breakdown the potential performance gains from baseline configurations, and investigate the potential gains contributed by each type of control. In the collection of tunable controls and design decisions, answering this question would tell us which decisions have the most impact on the final performance. We start by tuning only one dimension of control while leaving others at baseline settings. Figure 5 shows the percentage improvement in performance from the baseline setting for each platform and control dimension. Note that Google and ABM are not included in this analysis. In addition, we have 3 platforms (Amazon, BigML, PredictionIO) missing in the Feature Selection column, one (Amazon) missing in the Classifier Selection column. These are the platforms that do not support tuning those respective control dimensions. We observe the largest performance improvement of 14.6% (averaged across all platforms) when giving users the ability to select specific ML classifiers. In fact, in the case of Microsoft, F-score improves by 22.4% which is the highest among all platforms when we optimize the classifier choice for each dataset. After the classifier dimension, feature selection provides the next highest improvement in F-score (6.1%) across all platforms, followed by the classifier parameter dimension (3.4% improvement in F-score). The above results show that classifier is the most important control dimension that significantly impacts the final performance. To shed light on the general performance of different classifiers, we analyze classifier performance with default parameters and with optimized parameter configurations. Table 4(a) shows the top 4 classifiers when using baseline (default) parameters. It is interesting to note that no single classifier dominates in terms of performance over all the datasets. Table 4(b) shows a similar trend even when we optimize the parameters. This suggests that we need a mix of multiple linear (*e.g.* LR, NB) and non-linear (RF, BST, DT) classifiers to achieve high performance over all datasets.

Summary. Our results clearly show that platforms with higher complexity (more dimensions for user control) achieve better performance. Among the 3 key dimensions, classifier choice provides the largest performance gain. Just by optimizing the classifier alone, we can already achieve close to optimized performance. Overall, Microsoft provides the highest performance across all platforms, and a highly tuned Microsoft model can produce performance identical to that of a highly-tuned local scikit-learn instance.

5 RISKS OF INCREASING COMPLEXITY

Our experiments in Section 4 assumed that users were experts on each step in the ML pipeline, and were able to exhaustively search for the optimal classifier, parameters, and feature selection schemes to maximize performance. For example, for Microsoft, we evaluated over 17k configurations to determine the configuration with optimized performance. In practice, users may have less expertise, and are unlikely to experiment with more than a small set of classifiers or available parameters. Therefore, our second question is: *Can increased control lead to higher risks (of building poorly performing ML models)?* To quantify risk of generating poorly performing models, we use performance variation as the metric, and compute variation on each platform as we tune available controls.

5.1 Performance Variation across Platforms

First we measure the performance variation of each MLaaS platform across a range of system configurations (of `CLF`, `PARA`, and `FEAT`) described in Section 3. For each configuration and platform, we compute average performance across all datasets. Then we iterate through all configurations, and obtain a range of performance scores which capture the performance variation. Each configuration would generate a single point in the range of performance scores. Higher variation means a single poor decision in design could produce a significant performance loss. We plot performance variation results for each platform in Figure 6. As before, platforms on the x-axis are ordered based on increasing complexity. First, we observe a positive correlation between complexity of an MLaaS platform and higher performance variation. Among MLaaS platforms, Microsoft shows the largest variation, followed by less complex platforms like PredictionIO and Amazon. For Microsoft, F-score ranges from 0.49 to 0.75. Also as expected, our local ML library has the highest performance variation. The takeaway is that even though more complex platforms have the potential to achieve higher performance, there are higher risks of building a poorly configured (and poorly performing) ML model.

(a) Ranking of classifiers using baseline parameters

Rank	BigML	PredictionIO	Microsoft	Local
1	LR (34.5%)	LR (42.9%)	BST (50.4%)	BST (24.4%)
2	RF (26.1%)	DT (38.7%)	AP (16.8%)	KNN (12.6%)
3	DT (24.4%)	NB (18.5%)	BPM (10.9%)	DT (10.9%)
4	BAG (15.1%)		RF (7.6%)	RF (10.9%)

(b) Ranking of classifiers using optimized parameters

Rank	BigML	PredictionIO	Microsoft	Local
1	RF (32.8%)	LR (48.7%)	BST (43.7%)	MLP (32.8%)
2	BAG (30.3%)	DT (36.1%)	DJ (17.6%)	BST (27.7%)
3	LR (27.7%)	NB (16.0%)	AP (16.0%)	RF (9.2%)
4	DT (9.2%)		RF (13.4%)	KNN (6.7%)

Table 4: Top four classifiers in each platform using baseline/optimized parameters. Number in parenthesis shows the percentage of datasets where the corresponding classifier achieved highest performance. LR=Logistic Regression, BST=Boosted Decision Trees, RF=Random Forests, DT=Decision Tree, AP=Average Perceptron, KNN=k-Nearest Neighbor, NB=Naive Bayes, BPM=Bayes Point Machine, BAG=Bagged Trees, MLP=Multi-layer Perceptron, DJ=Decision Jungle.

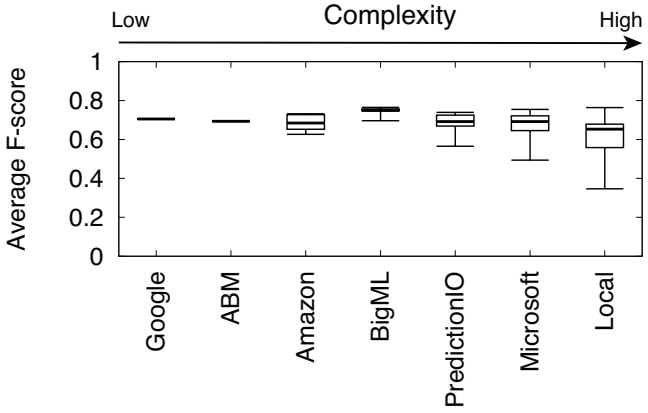


Figure 6: Performance variation in MLaaS platforms when tuning all available controls.

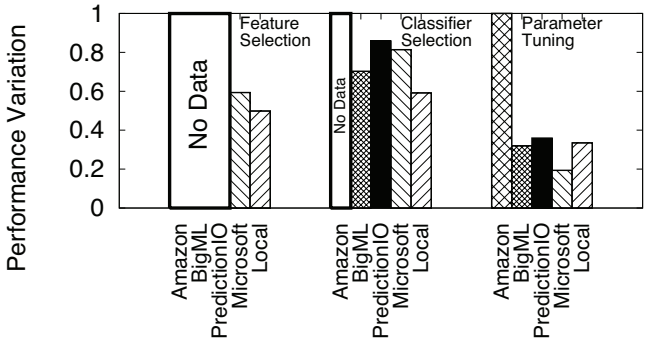


Figure 7: Performance variation when tuning CLF, PARA and FEAT individually, normalized by overall variation (white boxes indicate controls not supported).

5.2 Variation from Tuning Individual Controls

Next we analyze the contribution of each control dimension towards the variation in performance. When we tune a single dimension, we keep the other controls at their default values set by the platform, *i.e.* use the **baseline** settings. Figure 7 shows the *portion* of performance variation caused by each control dimension, *i.e.* a ratio normalized by the overall variation measured in our previous experiment. We observe that classifier choice (CLF) is the largest contributor to variation in performance. For example, in the case of Microsoft and PredictionIO (both exhibiting large variation), over 80% of the variation is captured by just tuning CLF. Thus, it is important to note that even though CLF can provide the largest improvement in performance (Section 4), if not carefully chosen, can lead to significant performance degradation. On the other hand, for all platforms, except Amazon, tuning the PARA dimension results in the least variation in performance. We are unable to verify the reason for the high variation in the case of Amazon (for PARA), but suspect it is due to either implementation or default parameter settings.

Partial Knowledge about Classifiers. Given the disproportionally large impact classifier choice has on performance and performance variation, we want to understand how users can make better decisions without exhaustively experimenting over the entire gamut of ML classifiers. Instead, we simulate a scenario where the user experiments with (and chooses the best out of) a randomly chosen subset of k classifiers from all available classifiers in a platform. We measure the highest F-score possible in each k-classifier subset. Next, we average the highest F-score across all possible subsets of size k. Results are shown in Figure 8 with all platforms supporting classifier selection. We observe a trend of rapidly improving performance as users try multiple classifiers. We observe that just trying a randomly chosen subset of 3 classifiers often achieves performance that is close to the optimal found by experimenting with all classifiers. In the case of Microsoft, we observe an F-score of 0.76 which is only 5% lower than the F-score we can obtain by trying all 8 classifiers. Performance variation also decreases significantly once a user explores 3 or more classifiers in these platforms.

Summary. Our results show that increasing platform complexity leads to better performance, but also leads to significant performance penalties for poor configuration decisions. Our results suggest that much/most of the gains can be achieved by focusing on classifier choice, and that experimenting with a random subset of 3 classifiers often achieves performance and lowers variation close to optimal.

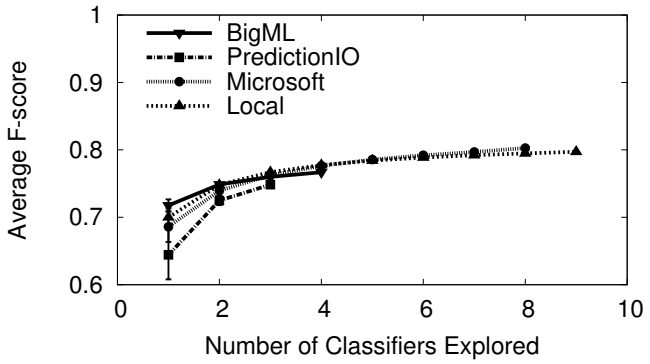
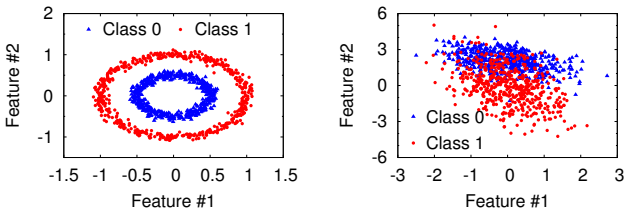


Figure 8: Average performance vs. number of classifiers explored.



(a) Visualization of CIRCLE. (b) Visualization of LINEAR.

Figure 9: Visualization of two datasets: synthetic non-linearly-separable dataset (CIRCLE) and synthetic linearly-separable dataset (LINEAR).

Category	Classifiers
Linear	LR, NB, Linear SVM, LDA
Non-Linear	DT, RF, BST, KNN, BAG, MLP

Table 5: Assignment of classifiers available on local library into linear vs. non-linear categories.

6 HIDDEN OPTIMIZATIONS

In the final part of our analysis, we seek to shed light on any platform-specific optimizations outside of user-visible configurations or controls. More specifically, we focus on understanding hidden optimizations used by fully automated black-box platforms, Google and ABM. These platforms have the most leeway to implement internal optimizations, because their entire ML pipeline is fully automated. In Section 4.1 (Figure 4), we observe that Google and ABM outperform many other platforms when applying default configurations. This suggests that their hidden configurations are generally better than alternative default settings.

Among the countless potential options for optimization, we focus on a simple yet effective technique: optimizing classifier choices based on dataset characteristics [46]. We raise the question: *Are black-box platforms automatically selecting a classifier based on the dataset characteristics?* Note that our usage of the phrase "selecting a classifier" should be broadly interpreted as covering different possible implementation scenarios (for optimization). For example, optimization can be implemented by switching between distinct classifier instances, or a single classifier implementation that internally alters decision characteristics depending on the dataset. While our analysis cannot infer such implementation details, we provide evidence of internal optimization in black-box platforms (Section 6.1). We further quantitatively analyze their optimization strategy (Section 6.2), and finally examine the potential for improvement (Section 6.3)

6.1 Evidence of Internal Optimizations

We select two datasets from our collection, a non-linearly-separable synthetic dataset, which we call CIRCLE[11], and a linearly-separable synthetic dataset, referred to as LINEAR[12]. Figure 9(a) and Figure 9(b) show visualizations of the two datasets. Both datasets have only two features. Given the contrasting characteristics (linearity) of the two datasets, our hypothesis is that they would help to differentiate between linear and non-linear classifier families based on prediction performance.

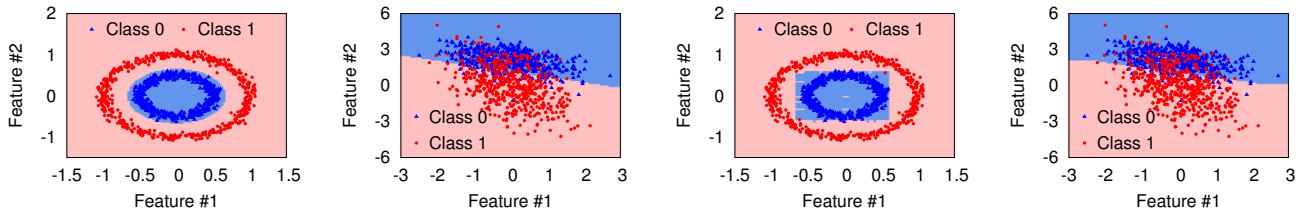
We examine Google and ABM's prediction results on CIRCLE and LINEAR to infer their classifier choices. Since we have no ground-truth information here, we resort to analyzing decision boundaries generated by the two platforms. The decision boundary is visualized by querying and plotting the predicted classes of a 100×100 mesh grid. Figure 10(a) and Figure 10(b) illustrate Google's decision boundary on CIRCLE and LINEAR, respectively. It is very clear that Google's decision boundary on CIRCLE forms a circle, indicating Google is using a non-linear classifier, or a non-linear kernel, *e.g.* RBF kernel [67]. On LINEAR, Google's decision boundary matches a straight line. It shows Google is using a linear classifier. Experiments on ABM also show similar results. Figure 10(c) and Figure 10(d) show the decision boundaries of ABM on CIRCLE and LINEAR, respectively. Thus, both platforms are optimizing and switching classifier choices for the two datasets. Additionally, Google's decision boundary on CIRCLE (circular shape) is different from ABM (rectangular shape), indicating that they selected different non-linear classifiers. Based on the shape of decision boundaries, it is likely that Google used a non-linear kernel based classifier while ABM chose a tree-based classifier.

6.2 Predicting Classifier Family

In this section, we present a method to automatically predict the classifier used by a platform using just two pieces of information—knowledge of the training dataset and prediction results from the platform. Such a method would help us automatically find instances where a black-box platform would change classifiers depending on the dataset characteristics.

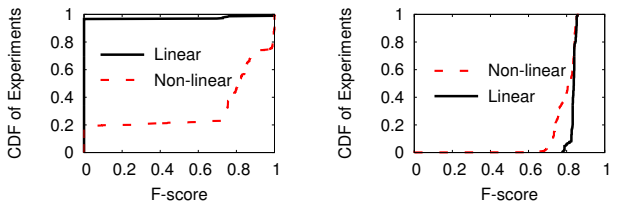
At a high level, we observe that it is hard to pin-point the specific classifier used by a platform, using just the dataset and the prediction results. This is because prediction results of different

[11] http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_circles.html
[12] http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html



(a) Google's decision boundary on CIRCLE. (b) Google's decision boundary on LINEAR. (c) ABM's decision boundary on CIRCLE. (d) ABM's decision boundary on LINEAR.

Figure 10: Decision boundaries generated by Google and ABM on CIRCLE and LINEAR. Both platforms produced linear and non-linear boundaries for different datasets.



(a) CIRCLE. (b) LINEAR.

Figure 11: Performance of predicting local linear/non-linear classifier choices on CIRCLE and LINEAR datasets.

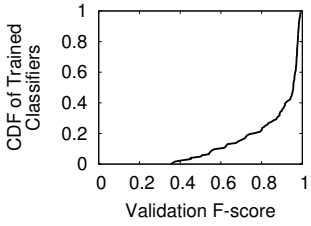


Figure 12: Validation performance of predicting linear/non-linear classifiers.

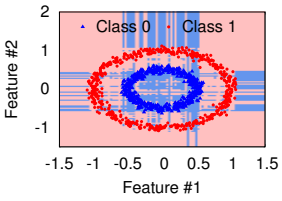


Figure 13: Amazon's decision boundary on CIRCLE.

classifiers tend to overlap. However, we find that it is possible to accurately infer the broad *classifier family*, more specifically, *linear* or *non-linear* classifiers.

Our key insight is that we can control datasets used for the inference and thus selectively choose datasets that elicit significant divergence between prediction results of linear and non-linear classifiers. To give an example, we examine the performance of the local library classifiers on the CIRCLE and LINEAR datasets. We categorize local classifiers into linear and non-linear families, as shown in Table 5. Figures 11(a) and 11(b) shows the performance (F-score) of the two categories of classifiers on the two datasets. As expected, we find that linear and non-linear classifiers produce very different F-scores on the two datasets, regardless of other configuration settings. Non-linear classifiers outperform linear classifiers when on CIRCLE. For LINEAR, linear classifiers outperform non-linear classifiers in many cases. This is because of the noisy nature of the dataset causing non-linear classifiers to overfit, and therefore produce lower performance compared to linear classifiers. Next, we present our methodology to accurately predict the classifier family by identifying more datasets that show divergence in prediction results of linear and non-linear classifiers.

Methodology. We build a supervised ML classifier for the prediction task. For training the classifier, we use prediction results, and ground-truth of classifier choices from the local library and the three platforms that allow user control of the classifier dimension (i.e. Microsoft, BigML and PredictionIO). Features used for training include aggregated performance metrics (F-score, precision, recall, accuracy), and the predicted labels. We train one classifier for each dataset in our collection. Each training sample is one ML experiment using a single configuration of the ML pipeline (*i.e.* choices of FEAT, CLF and PARA). Measurements are randomly split into training, validation, and test sets. Training and validation sets contain 70% of experiments, and test set contains the remaining 30% experiments. We train a Random Forests classifier with 5-fold cross-validation, and pick the best performing classifier based on validation performance. Based on prior work, Random Forests is one of the best performing classifiers for supervised binary classification tasks [15, 23]. Figure 12 shows the distribution of cross-validation performance of classifiers trained on all 119 datasets. Not all datasets could differentiate linear and non-linear classifiers. There are 64 datasets that produce classifiers achieving higher than 0.95 F-score. In other datasets, classifiers failed to separate linear and non-linear classifiers as they produce similar performance. Intuitively, we do not expect all datasets to perform well, and one goal of the training process is to identify datasets with high differentiating power. We select the 64 datasets where the trained classifiers achieve high performance (F-score > 0.95) on the validation set. To further test if they would generalize and accurately predict classifier choices, we apply them on the 30% held-out test set. All trained classifiers achieve F-score higher than 0.96. This further proves that the chosen classifiers can accurately predict the classifier family.

Classifier Choices of Google and ABM. We apply the selected 64 trained classifiers (covering 64 datasets) on Google and ABM and predict their classifier choices over linear and non-linear family. Results show Google uses linear classifiers on 39 out of 64 (60.9%) datasets, and non-linear classifiers for the remaining 25 (39.1%) datasets. ABM, on the other hand, uses linear classifiers on 44 (68.8%) out of 64 datasets, and non-linear on the remaining 20

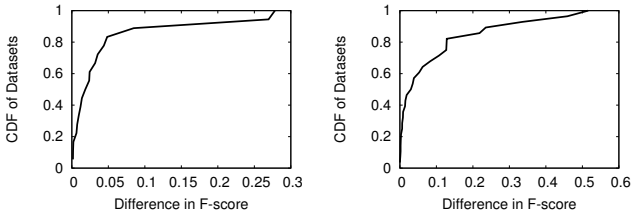
(a) Google vs. our naïve strategy.

Naïve \ Google	Linear	Non-linear
Linear	11 (25.5%)	5 (11.6%)
Non-linear	17 (39.5%)	10 (23.26%)

(b) ABM vs. our naïve strategy.

Naïve \ ABM	Linear	Non-linear
Linear	8 (16.7%)	3 (6.3%)
Non-linear	22 (45.8%)	15 (31.3%)

Table 6: Breakdown of datasets based on classifier choice when our naïve strategy outperforms black-box platforms.



(a) Google. (b) ABM.

Figure 14: Performance difference in datasets where naïve strategy outperforms Google/ABM using different classifier family.

(31.2%) datasets. If we compare Google and ABM, they pick the same classifier category on 49 (76.6%) datasets, but disagree on the remaining 15 (23.4%) datasets. The differences in the classifier choices could contribute to their overall performance difference in Figure 4. Overall, our results suggest that both platforms make different classifier choices (choosing linear or non-linear family) depending on the dataset.

Classifier Choices of Amazon. Recall that Amazon does not reveal any classifier information in their model training interface, but claims to use Logistic Regression on their documentation page. We apply our classifier prediction scheme on Amazon to investigate whether they are indeed using a single classifier for all tasks. Interestingly, 10 out of all 64 datasets have over 50% configurations that are predicted to be non-linear (the remaining are predicted to be linear). We also observe that Amazon produces a non-linear decision boundary when applied to the CIRCLE dataset (Figure 13). We suspect that Amazon uses non-linear techniques on top of Logistic Regression, *e.g.* non-linear kernel, or even uses other non-linear classifiers apart from Logistic Regression.

Unfortunately we are unable to corroborate our findings with the providers (Google, ABM, and Amazon), as all hidden optimizations are kept confidential and proprietary. However, our predictions demonstrate high accuracy in our validation and test datasets (where the underlying configuration is known).

6.3 Impact of Internal Optimizations

Previous experiments show that black-box platforms successfully choose classifier families with better performance when applied on the CIRCLE and LINEAR datasets. On these two datasets, Google and ABM would outperform a scheme that does not switch between classifier families. But are their strategies optimized for all other datasets? Are there cases where the two platforms make the wrong classifier choice?

To understand the potential for further improvement, we design a naïve classifier selection strategy using the local library, and compare its performance with Google and ABM. Our intuition is that if Google and ABM perform poorly when compared to our naïve strategy, there is potential for further improvement and we can understand the cases where classifier choices (*i.e.* linear vs non-linear) are potentially incorrect. We choose two widely used linear and non-linear classifiers, Logistic Regression and Decision Tree. These two classifiers are supported by most other platforms (Table 1). For each dataset, we train both classifiers and choose the one with higher performance. To further simplify the strategy and to avoid any impact of optimization from other control dimensions, we use the default parameter settings in Logistic Regression and Decision Tree, and perform no feature selection.

For our analysis, we again use the 64 datasets that can accurately predict choices of linear and non-linear classifiers. In 43 out of 64 datasets, our naïve strategy outperforms Google, and in 48 datasets, it outperforms ABM. This clearly indicates that Google and ABM have scope for further improvement.

We further compare the choices made by naïve strategy and black-box platforms. Table 6 shows the breakdown of the datasets by decisions when naïve strategy outperforms Google and ABM. In both platforms, in a majority of cases, the classifier choices do not match our simple strategy. In these cases, Google and ABM could increase their performance (F-score) by 20% and 34% on average, respectively, by choosing the other classifier family. Figure 14 shows a detailed breakdown (as a CDF) of performance difference between the black-box platforms and naïve strategy, when we outperform them. The potential performance improvement is significant in many cases.

When is switching classifier the best option for improvement? Although we show the potential performance improvement by switching classifiers, black-box platforms could use other methods to improve performance. For example, Google and ABM could perform better parameter tuning and feature selection to reduce the performance gap and justify their classifier choices. To identify cases where classifier switching is likely the best option, we compare our naïve strategy with the *optimal* performance of the other classifier family (*i.e.* not chosen). This means that when naïve strategy chooses a non-linear classifier, we compare its performance with the optimal linear classifier (across all configurations). If naïve strategy could still outperform Google and ABM under this scenario, it indicates that switching the classifier is likely the best way to further improve the performance. We find 3 datasets in the case of Google, and 4 for ABM, where changing the classifier is probably the best option to further improve performance. While our analysis

is limited to the 64 datasets where we can perform prediction, our finding highlights the existence of scenarios where Google and ABM clearly need to make better classifier choices.

7 RELATED WORK

Analyzing MLaaS Platforms. There is very limited prior work focusing on MLaaS platforms. Chan, *et al.* and Ribeiro, *et al.* presented two different architecture designs of MLaaS platforms [16, 51]. Although we cannot confirm that these architectures are being used by any of the MLaaS platforms we studied, they shed light on potential ways MLaaS platforms operate internally. Other researchers investigated vulnerabilities of these platforms towards different types of attacks. This includes attacks that try to leak information about individual training records of a model [26, 57], and those aiming to duplicate the functionality of a black-box MLaaS model by querying the model [64]. While these studies are in general orthogonal to our work, there is scope for borrowing techniques from them that can help us better understand MLaaS platforms.

Empirical Analysis of ML Classifiers. Prior empirical analysis focused on determining the best classifier for a variety of ML problems using user-managed ML tools (*e.g.* Weka, R, Matlab) on a large number of datasets. Multiple studies conducted an exhaustive performance evaluation of up to 179 supervised classifiers using up to 121 datasets [15, 23, 66]. All studies observe that Random Forests, Boosted or Bagging Trees outperform other classifiers, including SVM, Naïve Bayes, Logistic Regression, Decision Tree, and Multi-layer Perceptron. Caruana *et al.* further studied classifier performance focusing on high-dimensional datasets [14]. They find that Random Forests perform better than Boosted Trees on high-dimensional data, and the relative performance difference between classifiers change as dimensionality increases. Other work also focused on evaluating performance of specific classifier families, for example tree-based classifiers [47, 50], rule-based classifiers [50], and ensemble methods [49].

In comparison, our work does not focus on a single step of the ML pipeline. Instead, we analyze end-to-end impact of complexity on classifier performance, through the lens of deployed MLaaS platforms. This allows us to understand how specific changes to the ML task pipeline impact actual performance in a real world system. Instead of focusing only on the best achievable performance for any classifier, we recognize the wide-spread use of ML by generalist users, and study the "cost" of suboptimal decisions in choosing and configuring classifiers in terms of degraded performance.

Automated Machine Learning. Many works focused on reducing human effort in ML system design by automating classifier selection and parameter tuning. Researchers proposed mechanisms to recommend classifier choices based on classifiers that are known to perform well on similar datasets [40]. Many mechanisms even use machine learning algorithms like collaborative filtering and k-nearest neighbor to recommend classifiers [4, 10, 60]. To perform automatic parameter optimization, methods have been proposed based on intuition-based Random Search [6, 7], and Bayesian optimization [7, 25, 37, 61]. These mechanisms have been shown to estimate suitable parameters with less computational complexity than brute-force methods like Grid Search [21]. Other works proposed techniques to automate the entire ML pipeline. For example, Auto-Weka [39, 63] and Auto-Sklearn [24] could search through the joint space of classifiers and their respective parameter settings and choose the optimal configuration.

Experimental Design for Evaluating ML Classifiers. The ML community has a long history on classifier evaluation using carefully designed benchmark tests [38, 53, 68]. Many studies proposed theoretical frameworks and guidelines for designing benchmark experiments [22, 36]. Dietterich used statistical tests to compare classifiers [20] and the methodology was later improved in follow-up work [19, 29]. Our performance evaluation using Friedman ranking is based on their methodology. Other work focused on comparing and benchmarking performance of popular ML software [30, 69], *e.g.* Weka [75], PRTools [65], KEEL [2] and, more recently, deep learning tools [56]. In addition, work has been done to identify and quantify the relationship between classifier performance and dataset properties [35, 46], especially dataset complexity [44, 48, 78]. Our work leverages similar insights about dataset complexity (linearity) to automatically identify classifier families based on prediction results.

8 LIMITATIONS

We point out three limitations of our study. *First*, we focus on 6 mainstream MLaaS platforms, covering services provided by both traditional Internet giants (Google, Microsoft, Amazon) and emerging startups (ABM. BigML, PredictionIO). We did not study other commercial MLaaS platforms because they either focus on highly specialized tasks (*e.g.* image/text classification), or does not support large scale measurements (*e.g.* posing strict rate limit). *Second*, we focus on binary classification tasks with three dimensions of control (`CLF`, `PARA`, and `FEAT`). We did not extend our analysis to other ML tasks and cover every configuration choice, *e.g.* more advanced classifiers. We leave these as future work. *Third*, we only study the classification performance of MLaaS platforms, which is one of the many aspects to evaluate MLaaS platforms. There are other dimensions, *e.g.* training time, cost, robustness to incorrect input. We leave further exploration of these aspects as future work.

9 CONCLUSIONS

For network researchers, MLaaS systems provide an attractive alternative to running and configuring their own standalone ML classifiers. Our study empirically analyzes the performance of MLaaS platforms, with a focus on understanding how user control impacts both the performance and performance variance of classification in common ML tasks.

Our study produced multiple key takeaways. First, as expected, with more control comes more potential performance gains, as well as greater performance degradation from poor configuration decisions. Second, fully automated platforms are optimizing classifiers using internal tests. While this greatly simplifies the ML process and helps them outperform other MLaaS platforms using default settings, their aggregated performance lags far behind well-tuned versions of more configurable alternatives (Microsoft, PredictionIO, local scikit-learn). Finally, much of the gains from configuration and tuning come from choosing the right classifier. Experimenting with a small random subset of classifiers is likely to achieve near-optimal results.

Our study shows that used correctly, MLaaS systems can provide networking researchers results comparable to standalone ML classifiers. While more automated "turnkey" systems are making some intelligent decisions on classifiers, they still have a long way to go. Thankfully, we show that for most classification tasks today, experimenting with a small random subset of classifiers will produce near-optimal results.

ACKNOWLEDGMENTS

We wish to thank our shepherd Chadi Barakat and the anonymous reviewers for their useful feedback. This project was supported by NSF grants CNS-1527939 and CNS-1705042. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of any funding agencies.

REFERENCES

[1] Bhavish Aggarwal, Ranjita Bhagwan, Tathagata Das, Siddharth Eswaran, Venkata N. Padmanabhan, and Geoffrey M. Voelker. 2009. NetPrints: Diagnosing home network misconfigurations using shared knowledge. In *Proc. of NSDI*.

[2] Jesús Alcalá-Fdez, Luciano Sánchez, Salvador Garcia, Maria Jose del Jesus, Sebastian Ventura, Josep M. Garrell, José Otero, Cristóbal Romero, Jaume Bacardit, Victor M. Rivas, et al. 2009. KEEL: A software tool to assess evolutionary algorithms for data mining problems. *Soft Computing-A Fusion of Foundations, Methodologies and Applications* 13, 3 (2009), 307–318.

[3] Arthur Asuncion and David Newman. 2007. UCI machine learning repository. http://archive.ics.uci.edu/ml. (2007).

[4] Rémi Bardenet, Mátyás Brendel, Balázs Kégl, and Michele Sebag. 2013. Collaborative hyperparameter tuning. In *Proc. of ICML*.

[5] Fabrício Benevenuto, Gabriel Magno, Tiago Rodrigues, and Virgílio Almeida. 2010. Detecting spammers on Twitter. In *Proc. of CEAS*.

[6] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, Feb (2012), 281–305.

[7] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *Proc. of NIPS*.

[8] Peter Bodik, Moises Goldszmidt, Armando Fox, Dawn B. Woodard, and Hans Andersen. 2010. Fingerprinting the datacenter: Automated classification of performance crises. In *Proc. of EuroSys*.

[9] Léon Bottou and Chih-Jen Lin. 2007. Support vector machine solvers. *Large scale kernel machines* (2007), 301–320.

[10] Pavel B. Brazdil, Carlos Soares, and Joaquim Pinto Da Costa. 2003. Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results. *Machine Learning* 50, 3 (2003), 251–277.

[11] Leo Breiman. 1996. Bagging predictors. *Machine learning* 24, 2 (1996), 123–140.

[12] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[13] Matthijs C. Brouwer, Allan R. Tunkel, and Diederik van de Beek. 2010. Epidemiology, diagnosis, and antimicrobial treatment of acute bacterial meningitis. *Clinical microbiology reviews* 23, 3 (2010), 467–492.

[14] Rich Caruana, Nikos Karampatziakis, and Ainur Yessenalina. 2008. An empirical evaluation of supervised learning in high dimensions. In *Proc. of ICML*.

[15] Rich Caruana and Alexandru Niculescu-Mizil. 2006. An empirical comparison of supervised learning algorithms. In *Proc. of ICML*.

[16] Simon Chan, Thomas Stone, Kit Pang Szeto, and Ka Hou Chan. 2013. PredictionIO: a distributed machine learning server for practical software development. In *Proc. of CIKM*.

[17] Helen Costa, Fabricio Benevenuto, and Luiz H.C. Merschmann. 2013. Detecting tip spam in location-based social networks. In *Proc. of SAC*.

[18] Helen Costa, Luiz Henrique de Campos Merschmann, Fabricio Barth, and Fabricio Benevenuto. 2014. Pollution, Bad-mouthing, and Local Marketing: The underground of location-based social networks. *Elsevier Information Sciences* (2014).

[19] Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research* 7, Jan (2006), 1–30.

[20] Thomas G. Dietterich. 1998. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation* 10, 7 (1998), 1895–1923.

[21] Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger Hoos, and Kevin Leyton-Brown. 2013. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *Proc. of NIPS*.

[22] Manuel J.A. Eugster, Torsten Hothorn, and Friedrich Leisch. 2016. Domain-based benchmark experiments: Exploratory and inferential analysis. *Austrian Journal of Statistics* 41, 1 (2016), 5–26.

[23] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. 2014. Do we need hundreds of classifiers to solve real world classification problems. *Journal of Machine Learning Research* 15, 1 (2014), 3133–3181.

[24] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and robust automated machine learning. In *Proc. of NIPS*.

[25] Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. 2015. Initializing bayesian hyperparameter optimization via meta-learning. In *Proc. of AAAI*.

[26] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proc. of CCS*.

[27] Yoav Freund and Robert E. Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine learning* 37, 3 (1999), 277–296.

[28] Jerome H. Friedman. 2002. Stochastic gradient boosting. *Computational Statistics and Data Analysis* 38, 4 (2002), 367–378.

[29] Salvador Garcia and Francisco Herrera. 2008. An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research* 9, Dec (2008), 2677–2694.

[30] Michael Goebel and Le Gruenwald. 1999. A survey of data mining and knowledge discovery software tools. *ACM SIGKDD explorations newsletter* 1, 1 (1999), 20–33.

[31] Peter Haider and Tobias Scheffer. 2014. Finding botnets using minimal graph clusterings. In *Proc. of ICML*.

[32] Frank E. Harrell. 2002. Very low birth weight infants dataset.

[33] Frank E. Harrell. 2006. VA lung cancer dataset.

[34] Ralf Herbrich, Thore Graepel, and Colin Campbell. 2001. Bayes point machines. *Journal of Machine Learning Research* 1, Aug (2001), 245–279.

[35] Robert C. Holte. 1993. Very simple classification rules perform well on most commonly used datasets. *Machine learning* 11, 1 (1993), 63–90.

[36] Torsten Hothorn, Friedrich Leisch, Achim Zeileis, and Kurt Hornik. 2005. The design and analysis of benchmark experiments. *Journal of Computational and Graphical Statistics* 14, 3 (2005), 675–699.

[37] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *Proc. of LION*.

[38] Eamonn Keogh and Shruti Kasetty. 2003. On the need for time series data mining benchmarks: a survey and empirical demonstration. *Data Mining and knowledge discovery* 7, 4 (2003), 349–371.

[39] Lars Kotthoff, Chris Thornton, Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. 2016. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research* 17 (2016), 1–5.

[40] Rui Leite, Pavel Brazdil, and Joaquin Vanschoren. 2012. Selecting classification algorithms with active testing. In *Proc. of MLDM*.

[41] Zhijing Li, Ana Nika, Xinyi Zhang, Yanzi Zhu, Yuanshun Yao, Ben Y. Zhao, and Haitao Zheng. 2017. Identifying value in crowdsourced wireless signal measurements. In *Proc. of WWW*.

[42] Dapeng Liu, Youjian Zhao, Haowen Xu, Yongqian Sun, Dan Pei, Jiao Luo, Xiaowei Jing, and Mei Feng. 2015. Opprentice: Towards practical and automatic anomaly detection through machine learning. In *Proc. of IMC*.

[43] Qingyun Liu, Shiliang Tang, Xinyi Zhang, Xiaohan Zhao, Ben Y. Zhao, and Haitao Zheng. 2016. Network growth and link prediction through an empirical lens. In *Proc. of IMC*.

[44] Julián Luengo and Francisco Herrera. 2015. An automatic extraction method of the domains of competence for learning classifiers using data complexity measures. *Knowledge and Information Systems* 42, 1 (2015), 147–180.

[45] Núria Macià and Ester Bernadó-Mansilla. 2014. Towards UCI+: A mindful repository design. *Information Sciences* 261 (2014), 237–262.

[46] Núria Macià, Ester Bernadó-Mansilla, Albert Orriols-Puig, and Tin Kam Ho. 2013. Learner excellence biased by data set selection: A case for data characterisation and artificial data sets. *Pattern Recognition* 46, 3 (2013), 1054–1066.

[47] Richard Maclin and David Opitz. 1997. An empirical evaluation of bagging and boosting. In *Proc. of AAAI*.

[48] Laura Morán-Fernández, Verónica Bolón-Canedo, and Amparo Alonso-Betanzos. 2016. Can classification performance be predicted by complexity measures? A study using microarray data. *Knowledge and Information Systems* (2016), 1–24.

[49] David Opitz and Richard Maclin. 1999. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research* 11 (1999), 169–198.

[50] Claudia Perlich, Foster Provost, and Jeffrey S. Simonoff. 2003. Tree induction vs. logistic regression: A learning-curve analysis. *Journal of Machine Learning Research* 4, Jun (2003), 211–255.

[51] Mauro Ribeiro, Katarina Grolinger, and Miriam A.M. Capretz. 2015. MLaaS: Machine learning as a service. In *Proc. of ICMLA*.

[52] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1988. Learning representations by back-propagating errors. *Cognitive modeling* 5, 3 (1988), 1.

[53] Steven L. Salzberg. 1997. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data mining and knowledge discovery* 1, 3 (1997), 317–328.

[54] Purnamrita Sarkar, Deepayan Chakrabarti, and Michael I. Jordan. 2012. Nonparametric link prediction in dynamic networks. In *Proc. of ICML*.

[55] David J. Sheskin. 2003. *Handbook of parametric and nonparametric statistical procedures*. CRC Press.

[56] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. 2016. Benchmarking state-of-the-art deep learning software tools. *International Conference on Cloud Computing and Big Data* (2016), 99–104.

[57] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *Proc. of IEEE S&P*.

[58] Jamie Shotton, Toby Sharp, Pushmeet Kohli, Sebastian Nowozin, John Winn, and Antonio Criminisi. 2013. Decision jungles: Compact and rich models for classification. In *Proc. of NIPS*.

[59] Anirudh Sivaraman, Keith Winstein, Pratiksha Thaker, and Hari Balakrishnan. 2014. An experimental study of the learnability of congestion control. In *Proc. of SIGCOMM*.

[60] Michael R. Smith, Logan Mitchell, Christophe Giraud-Carrier, and Tony Martinez. 2014. Recommending learning algorithms and their associated hyperparameters. In *Proc. of MLAS*.

[61] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Proc. of NIPS*.

[62] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. 2005. *Introduction to data mining*. Addison-Wesley Longman Publishing Co., Inc.

[63] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proc. of KDD*.

[64] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2016. Stealing machine learning models via Prediction APIs. In *Proc. of USENIX Security*.

[65] Ferdinand Van Der Heijden, Robert Duin, Dick De Ridder, and David MJ Tax. 2005. *Classification, parameter estimation and state estimation: an engineering approach using MATLAB*. John Wiley & Sons.

[66] Joaquin Vanschoren, Hendrik Blockeel, Bernhard Pfahringer, and Geoffrey Holmes. 2012. Experiment databases. *Machine Learning* 87, 2 (2012), 127–158.

[67] Jean-Philippe Vert, Koji Tsuda, and Bernhard Schölkopf. 2004. A primer on kernel methods. *Kernel Methods in Computational Biology* (2004), 35–70.

[68] Kiri Wagstaff. 2012. Machine learning that matters. In *Proc. of ICML*.

[69] Abdullah H. Wahbeh, Qasem A. Al-Radaideh, Mohammed N. Al-Kabi, and Emad M. Al-Shawakfa. 2011. A comparison study between data mining tools over some classification methods. *International Journal of Advanced Computer Science and Applications* (2011), 18–26.

[70] Gang Wang, Tristan Konolige, Christo Wilson, Xiao Wang, Haitao Zheng, and Ben Y. Zhao. 2013. You are how you click: Clickstream analysis for sybil detection. In *Proc. of Usenix Security*.

[71] Gang Wang, Bolun Wang, Tianyi Wang, Ana Nika, Haitao Zheng, and Ben Y. Zhao. 2014. Whispers in the dark: Analysis of an anonymous social network. In *Proc. of IMC*.

[72] Gang Wang, Xinyi Zhang, Shiliang Tang, Haitao Zheng, and Ben Y. Zhao. 2016. Unsupervised clickstream clustering for user behavior analysis. In *Proc. of CHI*.

[73] David J. Whellan, Robert H. Tuttle, Eric J. Velazquez, Linda K. Shaw, James G. Jollis, Wendell Ellis, Christopher M. O'connor, Robert M. Califf, and Salvador Borges-Neto. 2006. Predicting significant coronary artery disease in patients with left ventricular dysfunction. *American heart journal* 152, 2 (2006), 340–347.

[74] Keith Winstein and Hari Balakrishnan. 2013. TCP ex Machina: Computer-generated congestion control. In *Proc. of SIGCOMM*.

[75] Ian H. Witten, Eibe Frank, Leonard E. Trigg, Mark A. Hall, Geoffrey Holmes, and Sally Jo Cunningham. 1999. Weka: Practical machine learning tools and techniques with Java implementations.

[76] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I. Jordan. 2009. Detecting large-scale system problems by mining console logs. In *Proc. of SOSP*.

[77] Minhui Xue, Cameron Ballard, Kelvin Liu, Carson Nemelka, Yanqiu Wu, Keith Ross, and Haifeng Qian. 2016. You can Yak but you can't hide: Localizing anonymous social network users. In *Proc. of IMC*.

[78] Julian Zubek and Dariusz M. Plewczynski. 2016. Complexity curve: a graphical measure of data complexity and classifier performance. *PeerJ Computer Science* 2 (2016), e76.