Doing More with Less: Characterizing Dataset Downsampling for AutoML

Fatjon Zogaj
ETH Zurich
Zürich, Switzerland
fzogaj@ethz.ch

José Pablo Cambronero
Massachusetts Institute of Technology
Cambridge, U.S.A.
jcamsan@mit.edu

Martin C. Rinard
Massachusetts Institute of Technology
Cambridge, U.S.A.
rinard@csail.mit.edu

Jürgen Cito
TU Wien
Vienna, Austria
Massachusetts Institute of Technology
Cambridge, U.S.A.
juergen.cito@tuwien.ac.at

ABSTRACT

Automated machine learning (AutoML) promises to democratize machine learning by automatically generating machine learning pipelines with little to no user intervention. Typically, a search procedure is used to repeatedly generate and validate candidate pipelines, maximizing a predictive performance metric, subject to a limited execution time budget. While this approach to generating candidates works well for small tabular datasets, the same procedure does not directly scale to larger tabular datasets with 100,000s of observations, often producing fewer candidate pipelines and yielding lower performance, given the same execution time budget. We carry out an extensive empirical evaluation of the impact that downsampling – reducing the number of rows in the input tabular dataset – has on the pipelines produced by a genetic-programming-based AutoML search for classification tasks.

PVLDB Reference Format:
Fatjon Zogaj, José Pablo Cambronero, Martin C. Rinard, and Jürgen Cito. Doing More with Less: Characterizing Dataset Downsampling for AutoML. PVLDB, 14(11): 2059 - 2072, 2021.
doi:10.14778/3476249.3476262

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/ipa-lab/autoML-sampling-public.

1 INTRODUCTION

Building and tuning well performing machine learning systems is a difficult task that benefits from domain and specialized data science knowledge [11, 14, 29, 61]. Developing a machine learning pipeline requires users to identify the relevant algorithms, decide how to compose these, choose the key hyperparameters and their values for each algorithm, and then implement this pipeline (typically) using a third-party library [15, 49]. To further increase the complexity of the task at hand, the user will likely need to change their choices depending on the dataset they are working with, *and* empirically validate the performance of their pipeline candidates. The complexity of this task has motivated the use of automated systems which can generate pipelines, validate them, and identify the best performing pipeline within a given execution time budget. These systems have been broadly termed automated machine learning (AutoML) [27].

Due to the size of the pipeline search space, many AutoML search procedures traditionally require significant computational resources and time, up to the order of days [25, 32]. These requirements increase further when AutoML is applied to a dataset with more than a few thousand observations. The difficulty of scaling existing AutoML search procedures to large datasets has been documented in both the literature [35, 45] and user reports [3–5].

Many popular machine learning algorithms display high runtime complexity as a function of the number of observations in the dataset. For example, the training time complexity of decision trees in scikit-learn [46], the popular Python machine learning library, is $O(n \log(n) m)$ [1] in the number of observations (n) and the number of features (m), while the time complexity for support vector machines (with non-linear kernels) is $O(n^3 m)$ (or $O(n^2 m)$ if efficient caching is used) [2]. While such complexities may not be onerous when training a single model, in the context of AutoML we are interested in training (and evaluating) 1,000s to 10,000s of models, which becomes increasingly difficult with a growing number of observations.

1.1 Impact of Downsampling on GP-Based AutoML for Classification

We empirically investigate the use of a preprocessing step to improve the scalability of genetic programming-based AutoML to large classification datasets: downsampling the training data. By executing the main search loop on a substantially smaller amount of data, the search procedure can generate and evaluate more pipelines, and explore the use of more computationally expensive operators, given the same execution time budget. Importantly, downsampling is a conceptually simple and non-intrusive strategy, which is compatible with existing AutoML systems.

Proceedings of the VLDB Endowment, Vol. 14, No. 11 ISSN 2150-8097.
doi:10.14778/3476249.3476262

To motivate the use of downsampling, we focus our study on the impact that downsampling has on TPOT [43], a popular GP-based AutoML tool. We carry out this study by collecting a benchmark set of 20 datasets from OpenML [55], DARPA D3M [18], and Criteo [34]. We focus our data collection on medium to large classification datasets, a subset of dataset sizes that has traditionally been missing from AutoML benchmarking suites [12] due to the associated computational burden. Medium to larger datasets are particularly important as they more closely reflect the increasingly large amounts of data in industry [42], and they are a challenge for existing AutoML systems. We use these 20 datasets to explore the impact of varying downsampling ratios and execution time budgets, the two main hyperparameters associated with downsampling.

Our results show that, for genetic programming-based AutoML, downsampling provided significant benefits for large classification datasets. In particular, we found that for 18 of 20, the search procedure produced a *higher* scoring pipeline when we downsampled the training data. For 14 out of 16 of the larger datasets, the optimal downsampling ratio in our evaluation ranged between 0.01 and 0.2, a significant reduction in the number of observations used during the search for candidate pipelines. Our results show that when executing the search with the dataset-optimal downsampling ratio, the search procedure evaluated up to 4.5 times more pipelines than when the procedure is carried out on the original dataset. We also evaluated the impact that longer execution time budgets (up to 10 hours) can have on the four datasets where downsampling provided the largest performance improvement. We found that for these datasets running up to 10 hours provided some performance improvements when we downsampled, and larger improvements for the setting where we use the full dataset. However, for all four datasets we found better performing pipelines when searching for 5 minutes on the optimally downsampled dataset compared to 240 minutes on the full dataset. This still holds true for three out of four datasets when extending the execution time budget to 600 minutes.

We carried out a qualitative analysis of the pipelines produced when downsampling and compared these to those produced when executing the search on the original dataset. Our analysis considers 480,000 pipelines generated, which consist of over 920,000 individual operators. We found that 8 different API components constitute the top 5 most frequently occurring operators across all our sampling ratios. Interestingly, the relative frequency of these operators varies substantially (by up to 10.1 percentage points) for lower sampling ratios, while these same components appear with a more similar relative frequency (differing by only 1.8 percentage points) when we consider the full dataset. We believe this shift in relative frequency indicates that for smaller sampling ratios pipeline performance is particularly dependent on operator choice, and the higher number of pipelines generated when downsampling allows the search procedure to cover more such choices. As the dataset size scales back up, the choice of operators seems to become increasingly less important (in terms of predictive performance) during the search process, potentially reflecting the benefit of larger data over particular algorithm properties.

We evaluated the extent to which downsampling impacts pipelines that contain gradient boosting classifiers. To do so, we restricted the classifier search space to two gradient boosting classifiers: scikit-learn's `GradientBoostingClassifier` and xgboost's `XGBClassifier`. We found that when carrying out the search on the original datasets, the AutoML search failed to produce a pipeline successfully, for both a 5 minute and 60 minute search budget. In contrast, downsampling allowed the search to evaluate substantially more candidates and successfully generated candidate pipelines even in the case of a 5 minute execution budget.

We also carried out experiments to compare the impact of the search space explored by the downsampled search versus the search space explored by the original search. We generated candidate pipelines using a downsampled search and then ranked these using the original dataset. Analogously, we generated candidate pipelines using the original dataset and then ranked them using a downsample of the data. We find that ranking pipelines using a downsampled dataset and ranking pipelines using the original dataset produces top candidates that perform similarly. In contrast, the way the pipelines were generated plays a bigger role. Pipelines generated by a search on the optimally downsampled datasets displayed higher predictive performance than pipelines generated by a search on the full dataset. This observation, combined with the observed ranking effects, lends support to our hypothesis that downsampling allows the AutoML search to explore a fundamentally different part of the pipeline search space for large datasets.

Carrying out our full set of experiments represented a significant computational cost, totaling approximately 8 weeks of wall clock time on a well resourced server. To enable others to carry out additional analyses on our dataset, we have released a reproducibility package. The package contains the raw outputs of our experiments, including pipelines generated, in a queryable form. The package includes the code used in our experimental framework, which runs our experiments from scratch, as well as utilities for data consumers, including functions to compute common summary statistics and compare visualizations across sampling ratios and datasets.

1.2 Contribution

To summarize, the contributions of this work are:

- We propose and perform a rigorous empirical investigation into the use of downsampling as a non-intrusive way to scale genetic-programming-based AutoML to larger classification datasets.
- We collected a benchmark suite of 20 primarily medium to large datasets, and evaluated the impact of downsampling on TPOT, a popular genetic programming-based AutoML tool. We focus our investigation on two key hyperparameters in a downsampling algorithm: downsampling ratio and execution time budget.
- We packaged and released our code and experimental outputs to enable future analysis by others, while mitigating the high computational cost.

The remaining sections are structured as follows. Section 2 provides an overview of machine learning pipelines in our context, as well as a formal description of AutoML. Section 3 presents the use of downsampling and introduces it into the standard formulation of AutoML. Section 4 presents the details of our experimental methodology, including dataset criteria and evaluation setup. We present the results of these experiments and their implications in detail in

```
pipeline = Pipeline(steps=[
  ('simpleImputer',
SimpleImputer(strategy=median)),
  ('bernoullinb',
BernoulliNB(binarizer=0.5, alpha=3.7, fit_prior=True))
])
```

Figure 1: An example pipeline generated using TPOT. The pipeline first imputes values and then learns classification labels using a naive bayes classifier. In addition to choosing components, the user must choose hyperparameter values for each.

Section 5. Section 6 provides an overview of the publicly-shared experimental results data. In Section 7 we present possible threats to validity and mitigants. Section 8 presents existing literature and its relation to our contribution. Finally, Section 9 provides closing remarks.

2 BACKGROUND

We now present background topics referenced within the paper.

2.1 Machine Learning Pipelines

We define a machine learning pipeline to be a composition of zero or more data preparation algorithms, for example feature preprocessing, extraction, and transformation, and a final estimator, such as one of various regression or classification algorithms. Typically such a pipeline is implemented by composing operators in a domain specific library, such as scikit-learn [46], a popular Python machine learning library. To fully define the pipeline, a developer must choose values for hyperparameters (or opt to use defaults) for each component in the pipeline. This process grows in complexity as the developer must empirically validate many pipelines, often tuning choices and revisiting prior experiments [56]. Figure 1 presents an example ML pipeline. Despite having just two components, this pipeline requires that the developer appropriately set 4 hyperparameters, and doing so requires that the developer empirically evaluate different choices. The core goal of AutoML is to automatically generate a fully configured pipeline, reducing (or when possible, eliminating) the need for this developer to manually explore the space of pipelines.

2.2 AutoML for Classification

To formalize the AutoML problem, we follow the formulation presented in [15]. While a similar formulation is possible for regression tasks, we focus this work on classification as it has been a popular task for AutoML research to date [20, 22, 43, 58].

Let $d \in \mathcal{D}$ be a dataset consisting of n observations, where each observation consists of a set of covariates and an output value (i.e. string, integer, or boolean label). We denote the training and test splits with d_{train} and d_{test}, respectively. Let $p \in \mathcal{P}$ be a pipeline, drawn from the space of possible pipelines, and $p(d_{\text{train}})$ be shorthand for *training* (i.e. fitting parameter values) on a dataset d. Let $\mathcal{S} \subset \mathcal{P}$ be the subset of pipelines in the AutoML tool's search space. Let e be a scoring function that measures the predictive performance of a pipeline, such as cross-validated macro-averaged F1 score. Let $c : \mathcal{P} \times \mathcal{D} \times \mathcal{D} \to \mathbb{R}$ be a cost function that evaluates the time it takes to construct a pipeline, fit it on training data, and evaluate it on test data. Let $b \in \mathbb{R}$ be the search time budget, which limits how long the AutoML system can spend searching for an optimal pipeline. The AutoML problem can be defined as

$$\underset{p \in \mathcal{S}}{\text{argmax}}\; e(p(d_{\text{train}}), d_{\text{test}}) \text{ s.t. } \sum_{p \in \mathcal{S}} c(p, d_{\text{train}}, d_{\text{test}}) \leq b \qquad (1)$$

Given that the search space $\mathcal{P}$ grows exponentially with the number of available operations and their hyperparameter choices the procedure cannot simply evaluate all pipelines and instead must rely on pruning or prioritization techniques (manually defined or learned) to efficiently navigate the space of possible pipelines [27, 60].

2.3 TPOT and Genetic Programming

TPOT is a popular AutoML system that generates tree-shaped pipelines by carrying out a genetic programming-based search. Genetic programming (GP) refers to a general evolutionary framework for searching/optimizing in a space where individuals are represented as structured programs [31]. In particular, TPOT's individuals correspond to pipeline definitions. Like traditional evolutionary methods, individuals in the population are *evolved* to maximize some metric – termed the fitness of an individual. Fitness for pipelines, for example, can be defined as their cross-validated performance on the data provided to the search procedure. The search initializes with a randomly generated population of individuals, and at each *generation* it produces new individuals through mutation and cross-over operations. Mutation operations can extend, shrink, or modify pipeline definitions, while cross-over operations combine two pipelines to produce a new variant. Finally, like many evolutionary methods, there is a stochastic selection procedure to choose the subset of individuals that move on to the next generation. We point the interested reader to the original TPOT paper for further details on the exact GP configuration used [43].

2.4 Scope and Audience

We focus our experiments on a popular search procedure for AutoML: genetic programming (GP). In our implementation, we use TPOT, a popular open-source GP-based AutoML system, with over 8,000 stars on GitHub[1] and over 1,400 forks. The goal of our experiments is to provide empirical evidence of the impact that downsampling may have on GP-based AutoML. We investigate key properties such as predictive performance, runtime performance, fundamental pipeline properties such as their length and distribution of operators, as well as the impact of different search spaces. To the best of our knowledge, this is the first extensive evaluation of the impact of downsampling in this setting. Insights gained from these experiments are useful to two key groups: AutoML designers, who may be interested in incorporating downsampling into their own tools in a principled fashion, and AutoML users, who may *already* be employing downsampling and would like a clear understanding of the potential implications of their downsampling.

[1]As of April 2021

3 DOWNSAMPLING AUTOML

Large datasets pose a particular challenge for AutoML, as the time complexity of many key operators grows with the number of observations in a dataset. We systematically evaluate one approach to alleviate this challenge: downsampling the dataset prior to executing the AutoML search. We formalize this approach by making a slight modification to Equation (1). Let $s : \mathcal{D} \times \mathbb{R} \rightarrow \mathcal{D}$ be a sampling function, which takes a dataset d, a real-valued sampling ratio $r \in \mathbb{R}$, and returns a dataset $d' \subset d$, such that $\frac{|d'|}{|d|} \approx r$, where $|x|$ is the number of rows in a table x.

The downsampled AutoML problem is now:

$$\underset{p \in \mathcal{S}, r \in (0,1)}{\operatorname{argmax}} \; e(p(s(d_{\text{train}}, r)), d_{\text{test}}) \text{ s.t. } \sum_{p \in \mathcal{S}} c(p, s(d_{\text{train}}, r), d_{\text{test}}) \leq b \tag{2}$$

The main search loop operates on the downsampled dataset to identify the most promising candidate pipeline within the subset $\mathcal{S}$ of the pipeline search space $\mathcal{P}$. This pipeline is *then* fit on the entire training dataset, prior to returning to the user, making the downsampling process transparent. Algorithm 1 illustrates this process.

Algorithm 1 Downsampled AutoML

```
INPUT: A training dataset $d_{\text{train}}$, an AutoML search procedure SEARCH, a
  sampling function $s$, a sampling ratio $r$, and an execution time budget $b$
OUTPUT: An output pipeline
  procedure DOWNSAMPLEANDSEARCH
    $d'_{\text{train}} \leftarrow s(d_{\text{train}}, r)$            ▷ Sample down to ratio $r$
    $p \leftarrow$ SEARCH($d'_{\text{train}}, b$)          ▷ Generate best pipeline
    return $p(d_{\text{train}})$              ▷ Fit best on full training set
```

Downsampling raises the following key questions:

How does sampling affect predictive performance? Given that the AutoML system is trying to maximize performance on the original dataset, we naturally want to understand the impact that the choice of sampling ratio r may have on the optimal pipeline's performance. In particular, will varying values of r result in different output pipelines with different performances?

How does sampling affect runtime performance? A key motivation for introducing r was to enable AutoML use with larger datasets, which typically require significantly larger execution time budgets. A lower r produces a smaller dataset for the search, which should intuitively result in faster pipeline evaluation.

Does sampling impact the traditional relationship between longer execution budgets and better predictive performance? Traditionally, AutoML systems have been evaluated (and used) with large execution budgets. Given that we have introduced another search hyperparameter (r), we ask whether the traditional relationship between execution budget and predictive performance (i.e. longer budgets lead to better predictive performance) still holds.

Does sampling impact pipeline characteristics? When using AutoML across different datasets, we often obtain pipelines with varying components, lengths, and architectures. With the introduction of downsampling, we ask whether fundamental properties of the pipeline, such as its length and the operators included, vary as a function of the downsampling ratio r.

How does downsampling impact popular gradient boosting classifiers? Gradient boosting classifiers, in particular XGBoost [17], are popular due to their high performance and ease of use. We restrict the space of classifiers explored by the AutoML system to gradient boosting classifiers and evaluate the impact of downsampling on this class of popular classifiers.

What is the impact of the pipeline search space? We consider the set of pipelines that are generated and evaluated during a search with downsampled data versus the original dataset. We distinguish between the task of *generating* versus *ranking* pipelines and use this distinction to disentangle the impact of varying search spaces. In particular, we compare the candidates generated during a downsampled search and a search on the original dataset.

4 METHODOLOGY

We now provide a detailed description of our experimental methodology including dataset selection and implementation.

4.1 Large Classification Datasets

A first challenge in evaluating the impact of downsampling on the effectiveness of AutoML is the dataset collection task. Existing benchmark suites, such as OpenML100 and OpenML-CC18[2] focus on small to medium sized datasets (up to 100,000 observations). To address the need for larger tabular classification datasets, we narrowed our search to datasets that: have a varied number of features (i.e. columns); small, medium, *and large* numbers of observations; have a varied number of target classes; are not easily solved benchmarks; do not require sophisticated preprocessing or cleaning, a standard criteria in AutoML benchmark collection; and belong to different domains, to avoid domain-specific effects.

During our benchmark dataset collection task we noted that there is a scarcity of datasets with 1 million rows or more that satisfy the criteria outlined above. We now provide a summary of the sources and types of datasets that we were able to collect.

OpenML. We collected 16 datasets from OpenML – which has previously been used for AutoML benchmarking datasets [6, 22, 64] – aiming to cover the criteria outlined previously.

DARPA D3M:. We collected 3 additional datasets available through the DARPA Data-Driven Discovery of Models (D3M) publicly released datasets [18]. D3M is a program focused on developing new AutoML methods, and has participants from a wide range of academic and industrial organizations. The downloaded datasets are also available through OpenML, and we note that in total 7 of the OpenML datasets are also cross-listed as DARPA D3M datasets.

Criteo. We collected a large sample from the popular Criteo dataset. We sampled 1,144,307 rows (approximately 2.5%) of the original 1 TB (uncompressed) advertising dataset. This sample size is in line with that used in other benchmarking work on model selection [41] and was compatible with our single-server evaluation hardware setup and our computation time budgets.

[2]For more information about OpenML100 and OpenML-CC18, see: https://openml.github.io/OpenML/benchmark

Table 1: A summary of our benchmark suite of classification datasets. Numerical IDs are available through OpenML [55].

ID	Name	#Classes	#Features/Columns	#Instances/Rows
1468	cnae-9	9	857	1080
12	mfeat-factors	10	217	2000
3	kr-vs-kp	2	37	3196
1489	phoneme	2	6	5404
40668	connect-4	3	43	67557
41138	APSFailure	2	171	76000
41168	jannis	4	55	83733
23517	numerai28.6	2	22	96320
23512	higgs	2	29	98050
41150	MiniBooNE	2	51	130064
1483	ldpa	11	8	164860
1503	spoken-arabic-digit	10	15	263256
1219	Click-prediction-small	2	12	399482
1113	KDDCup99	23	42	494020
1169	airlines	2	8	539383
1596	covertype	7	55	581012
23397	COMET-MC-SAMPLE	2	6	761940
42468	hls4ml-lhc-jets-hlf	5	17	830000
354	poker	2	11	1025010
criteo	Criteo-sample	2	40	1144307

Table 1 presents a summary of the datasets used, ranging from small (1,000s of rows) to large (over a million rows).

4.2 Downsampling, Splitting, and Fitting

During preliminary analysis, we evaluated the impact of uniformly downsampling the training data with r ranging from 0.1 to 0.9 in 0.1 increments. We found that classification performance, as measured by macro-averaged F1 score, underwent the largest changes when r was drawn from a smaller range of $[0.1, 0.3]$, and conversely that sampling ratios between $[0.5, 1.0]$ did not induce significant variation. Based on these observations, we evaluated the following set of sampling ratios in our experiments: $r \in (0.0001, 0.001, 0.01, 0.05, 0.15, 0.2, 0.3, 0.5, 1.0)$.

We used k-fold cross-validation (CV) [30] to carry out our evaluation. When splitting the dataset to produce different folds, we performed a stratified splitting[3] to preserve target label ratios (i.e. if a dataset has an imbalanced set of labels, that imbalance is preserved in the splits). We set $k = 5$ for all datasets to balance the desire for lower-variance performance estimation and computational burden.

Figure 2 illustrates how we use data folds during evaluation. Namely, for a given iteration of k-fold cross-validation, we downsample the training fold, run the AutoML search on the downsampled fold to obtain a final candidate pipeline, fit the final candidate pipeline on the original training fold (without downsampling) to estimate any free parameters, and then evaluate the fitted pipeline on the full test fold. This process is repeated for each of the k folds. For example, with $k = 5$ and $r = 0.1$, for a given fold iteration the AutoML search is carried out on $0.8 * 0.1$ of the original dataset (0.8 constitutes the fraction of the dataset in the training fold, and 0.1 reflects the downsampling), the pipeline produced is then fit on the full training fold (0.8 of the original dataset), and evaluated on the remaining (disjoint) 0.2 of the original dataset.

Implementation. We implemented our benchmarking methodology by building on the experimental framework used in [15]. We used scikit-learn's implementations of standard utilities such as k-fold cross validation splitting, F1-score, and yellowbrick's [8] learning curve visualization. AutoML systems are known [6, 64] to occasionally crash or freeze during benchmarking. To mitigate this issue, we relied on a mixed use of job queuing[4] and intermittent task monitoring, with associated job restarts when necessary for benchmarking completion.

[3]We use Scikit-Learn's StratifiedKFold.

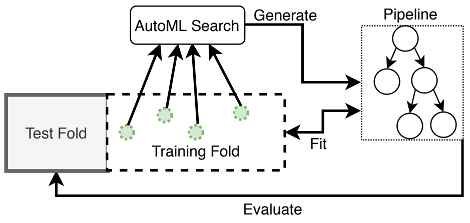


Figure 2: For a given iteration in our k-fold cross-validation procedure, we take the training fold (in white, dashed lines), and downsample to obtain a smaller set of observations (in green, dashed lines). The AutoML search is run using this downsampled set of observations. After a final pipeline has been generated, we fit this pipeline on the entire training fold, and then evaluate it on the test fold (in grey, solid lines).

Computational Resources. To carry out our experiments, we used a well-provisioned shared compute server. Our server provided four 14 core Intel Xeon E5-2697 v3 CPUs (2.60 GHz) and 256GB RAM. This setup reflects the resources that a well provisioned data scientist might have to apply an off-the-shelf AutoML tool on commodity hardware. For reference, recent AutoML benchmarking research, such as Gijsbers et al [22] used AWS instances with 32GB RAM and 8 vCPUs (Intel Xeon Platinum 8000 Skylake up to 3.1 GHz) and Shang et al [52] used a similar setup to ours: a single-machine with a 40-core Intel Xeon CPU E5-2660 v2 (2.20 GHz) and 256GB RAM. During execution of our experiments, attention was placed to monitor CPU usage across the machine and avoid significant changes in processing power when running across different datasets. Running all experiments from scratch required approximately 8 weeks of wall-clock time.

5 RESULTS

We now present our experimental results. We report macro-averaged F1 test score, averaged over the k folds in our cross-validation. When no pipeline is generated, we report an empty entry in the corresponding table. A pipeline may fail to be produced in a variety of cases, for example when the input data provided to the AutoML search is too small (raising repeated exceptions in underlying operators), or when the dataset is too large and the AutoML search fails to produce enough candidate pipelines during the execution budget given. Unless otherwise specified, we execute the AutoML search with an execution budget of 5 minutes.

[4]We use task-spooler, a simple task management system for single machines.

Table 2: F1 score under varying sampling ratios. The header row indicates the sampling ratio, while the ID column indicates the dataset. Datasets are sorted in ascending order of number of rows, with those above the horizontal line having at most 10,000 observations. The bolded numbers correspond to the best performance for that dataset. We find that, with the exception of dataset 3 and 23517, downsampling resulted in higher performance.

ID	0.0001	0.001	0.01	0.05	0.1	0.15	0.2	0.3	0.5	1.0
1468					0.945	0.94	0.95	0.947	**0.952**	0.944
12				0.971	0.793	0.974	0.968	0.974	**0.978**	0.975
3			0.957	0.993	0.992	0.992	0.993	0.991	0.994	**0.995**
1489			0.722	0.841	0.858	0.85	0.879	0.884	**0.887**	0.884
40668		0.476	0.64	0.651	0.69	0.7	**0.709**	0.674	0.66	0.582
41138	0.74	0.713	0.862	0.864	0.884	**0.909**	0.905	0.871	0.87	0.854
41168		0.398	0.521	**0.552**	0.53	0.543	0.533	0.531	0.514	0.487
23517		0.509	0.493	0.514	0.514	0.512	0.517	0.516	0.516	**0.518**
23512		0.688	0.714	0.71	0.716	**0.721**	0.717	0.718	0.716	0.699
41150		0.9	0.925	0.929	**0.929**	0.929	0.925	0.921	0.908	0.895
1483			0.703	**0.935**	0.917	0.796	0.788	0.675	0.688	0.577
1503		0.108	0.187	0.258	0.259	**0.261**	0.245	0.213	0.192	0.103
1219	0.494	0.54	0.562	0.582	**0.598**	0.595	0.568	0.566	0.555	0.536
1113	0.483		**0.968**		0.965	0.967	0.937	0.839	0.941	0.939
1169	0.439	0.617	0.651	0.653	0.652	**0.656**	0.653	0.648	0.631	0.626
1596		0.671	0.938	0.941	0.944	**0.944**	0.944	0.871	0.807	0.64
23397	0.516	0.89	0.879	0.914	0.911	0.914	0.91	**0.917**	0.878	0.728
42468	0.69	0.76	**0.764**	0.764	0.762	0.753	0.752	0.746	0.74	0.693
354	0.573	0.868	0.792	**0.947**	0.833	0.782	0.658	0.613	0.596	0.574
criteo	0.613	0.637	**0.655**	0.616	0.609	0.609	0.591	0.605	0.586	0.572

Table 3: Difference in F1 test score, scaled by 100 for clarity, for various sampling ratios when compared to the original data for datasets with more than 10,000 observations. We obtain higher performance for all datasets when using downsampling, with the exception of dataset 23517, which performs slightly better using the full dataset.

ID	0.01	0.05	0.1	0.15	0.2	0.3	0.5	Original
40668	5.9	6.9	10.8	11.8	**12.7**	9.3	7.9	0.582
41138	0.8	1.0	3.0	**5.4**	5.0	1.6	1.5	0.854
41168	3.5	**6.6**	4.3	5.6	4.7	4.5	2.7	0.487
23517	-2.5	-0.4	-0.4	-0.6	**-0.2**	-0.2	-0.3	0.518
23512	1.6	1.1	1.7	**2.3**	1.8	1.9	1.8	0.699
41150	3.0	3.4	**3.5**	3.4	3.1	2.6	1.4	0.895
1483	12.7	**35.8**	34.0	22.0	21.1	9.9	11.1	0.577
1503	8.4	15.5	15.6	**15.7**	14.2	11.0	8.9	0.103
1219	2.6	4.6	**6.2**	5.9	3.2	3.0	1.9	0.536
1113	**2.9**		2.6	2.8	-0.2	-9.9	0.2	0.939
1169	2.5	2.7	2.6	**2.9**	2.7	2.1	0.5	0.626
1596	29.8	30.0	30.4	**30.4**	30.4	23.1	16.7	0.64
23397	15.1	18.6	18.4	18.7	18.3	**19.0**	15.1	0.728
42468	**7.1**	7.1	6.9	6.0	5.9	5.3	4.7	0.693
354	21.8	**37.2**	25.9	20.8	8.3	3.9	2.1	0.574
criteo	**8.2**	4.4	3.7	3.7	1.9	3.2	1.4	0.572

5.1 RQ1: Predictive performance

While downsampling allows us to scale search to larger datasets, there could be a potential trade-off in terms of predictive performance. In particular, it may be that the pipeline that is optimal in the downsampled training data does not generalize to the larger dataset. This potential bias is an increasingly important concern as the dataset size decreases. To address this question, we consider the observed pipeline performance as a function of sampling ratio. Table 2 shows the F1 score (averaged over test folds) for each dataset, across different sampling ratios. For clarity, we have sorted datasets in increasing order of their original number of observations (with those above the horizontal line having at most 10,000 observations), and we bold the best performance for each dataset. Our results show that downsampling can actually *improve* performance, with respect to performing AutoML search on the full dataset size. Only two of 20 datasets obtained their best performance when using the original full training fold. We see a clear divide between smaller datasets (less than 10,000 observations) and larger datasets (more than 50,000 observations). For smaller datasets, 3 out of 4 performed better with a sampling ratio of 0.5. For larger datasets, 14 of 16 datasets obtained their best performance when we downsampled the training folds to 0.01 to 0.2 of the original size. Note that for extremely small sampling ratios (e.g. 0.001), the search procedure may fail to generate a pipeline. It is also important to note that we cannot immediately compare the performance of the pipelines generated to those that might be obtained with an exhaustive procedure such as grid search, as the space TPOT explores is unbounded due to the abililty to compose (through cross-over and mutation) an arbitrary number of pipeline operators.

For clarity, Table 3 focuses on the 16 datasets with more than 10,000 observations. The table presents the F1 score obtained when using the original training fold, along with the score difference (scaled by 100) when the dataset is downsampled with varying ratios. We bold the entry that results in the largest increase (or smallest decrease) in performance. Generally, we see that there were improvements across downsampling ratios for all datasets but 23517, which performs best when we use the full dataset.

Figure 3 provides a more detailed view of the performance changes for the 4 datasets where we observed the largest improvement in F1 score when searching on a downsampled dataset. We present the training F1 score, test F1 score without fitting the final pipeline on the full training fold, and the test F1 score after fitting the final pipeline on the full training fold.

Our analysis shows that for three of these datasets, with the exception of 1483, the training F1 score decreases substantially as the size of the dataset grows. In particular, note that at low sampling ratios, the search procedure is able to find a pipeline that achieves near-perfect training score. A reduction in training score may indicate that the pipelines produced with the larger set of observations lack the model capacity to tackle the task. Conversely, for small datasets, the pipelines considered are able to sufficiently capture variations in the dataset.

Next, we note that for 3 of the 4 datasets there is a significant gap in the test F1 score between the setting where we *do not* fit the final pipeline on the full training fold and the setting where we *do*. This emphasizes a key takeaway: while we can generate the final candidate pipeline by searching on a downsampled dataset, we should always re-fit this pipeline on the full training set (a step noted in Algorithm 1).

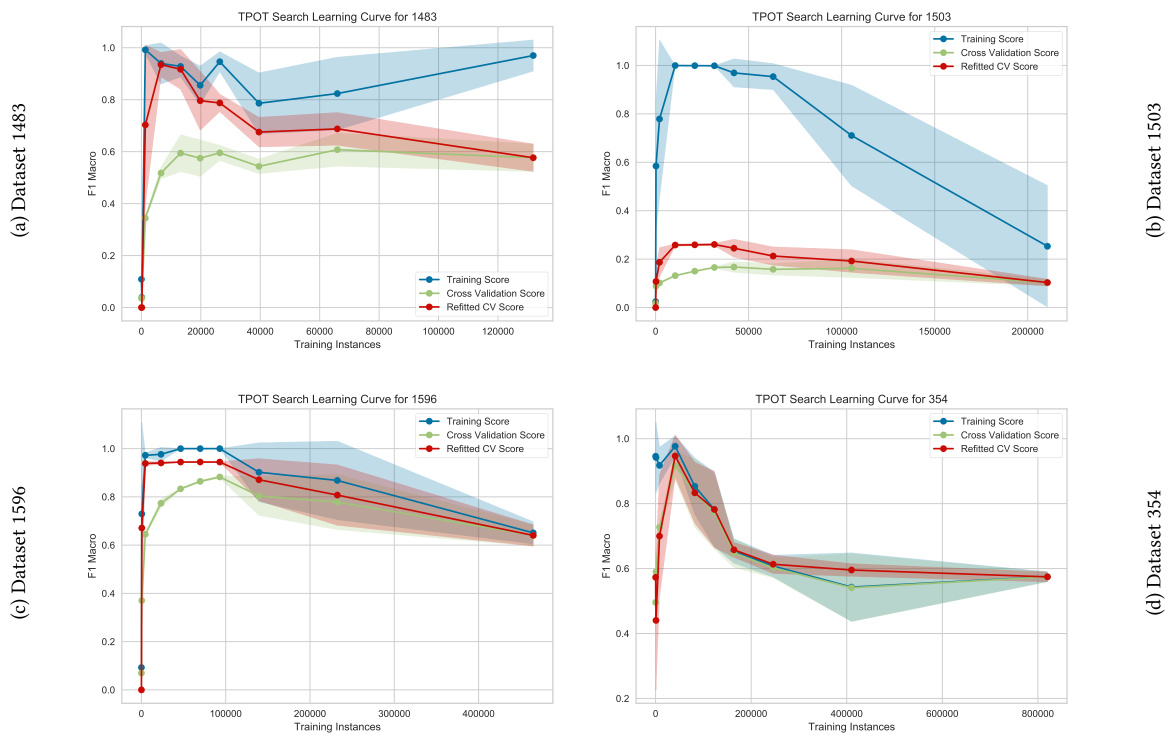


Figure 3: F1 score over different downsampling ratios for the datasets with the largest performance increase when using their optimal downsampling ratio. We show training F1 score (blue), test F1 score without refitting on the full training fold (green), and the final test F1 score when refitted on the training fold (red, see Algorithm 1 for details). We show that performance in these datasets degrades as a function of the dataset size. We also see that for 3 of the 4 dataset refitting the final pipeline on the full training fold (rather than the downsampled version) is critical to improve performance.

5.2 RQ2: Runtime performance

To evaluate the impact of sampling on runtime performance, we considered the number of pipelines that the AutoML search procedure is able to generate and evaluate during a fixed amount of time. We ran the AutoML search using a time-budget of 5 minutes.

Table 4 shows the average (over CV iterations, and rounded to nearest integer) number of pipelines explored for each of the downsampling ratios and the full dataset. We highlight in bold the ratio which results in the largest number of pipelines evaluated for each dataset. As expected, we found that downsampling more aggressively allows the search procedure to generate and evaluate more pipelines, subject to the same execution time budget. This in particular is a key advantage as the system can explore a substantially larger portion of the search space. For some datasets this amounts to 1.3 to 22.6 more pipelines than when the search is carried out on the full dataset. We also see that this effect is increasingly stark as we increase the size of the original dataset. For example, for dataset 354 – our second largest dataset with over 1 million observations – the search procedure generates and validates 95 pipelines when using the original dataset, while the most extreme downsampling ($r = 0.0001$) results in 1973 pipelines. The best performing sampling ratio for dataset 354 ($r = 0.05$) results in 3 times as many pipelines when compared to searching on the original dataset.

5.3 RQ3: Time budget and predictive performance

As discussed earlier, it is traditionally the case that longer AutoML search procedure runs are more likely to find a better performing pipeline. A natural question is whether this relationship holds true when we use a downsampled dataset for searching.

To explore this question, we ran the following experiment. We took the optimal downsampling ratio, identified from earlier experiments using 5 minutes, and we ran the downsampled search procedure for 60 minutes. We then took the original dataset (without any form of downsampling) and ran the search procedure for 60 minutes as well. As in prior experiments, we use the same approach with k-fold cross validation. We compute the average test F1 score and the number of pipelines evaluated (averaged over folds, and rounded to the nearest integer).

Table 5 presents these results. We omit dataset 12, which failed to complete after multiple execution attempts. We find that when using the optimal downsampling ratio, running the search procedure for 60 minutes can improve scores slightly, but with few exceptions (4 datasets: 41168, 1483, 1219, 354) these improvements are small. Note that when we run the search procedure for 60 minutes, the number of pipelines explored increases substantially (up to 14x).

Table 4: Average number of generated pipelines under different sampling ratios during cross-validation, subject to a five minute execution time budget. Lower sampling ratios can result in 1.3 to 22.6 times more pipelines generated when compared to the full dataset.

ID	0.0001	0.001	0.01	0.05	0.1	0.15	0.2	0.3	0.5	1.0
1468					397	**420**	399	403	301	323
12				**420**	359	363	361	341	363	288
3			**2584**	1438	1420	1220	1143	1096	774	581
1489			**2747**	1755	1512	1221	1206	1102	855	619
40668		**1809**	1044	535	481	402	381	383	383	261
41138	186	**2058**	635	400	363	250	306	227	302	173
41168		**1419**	525	387	367	269	329	248	245	134
23517		**2627**	1272	599	481	481	423	384	363	187
23512		**1691**	917	401	388	382	386	382	285	112
41150		**1186**	600	403	365	384	349	284	209	170
1483			**538**	385	382	362	361	306	285	151
1503		**942**	382	346	324	208	228	169	190	112
1219	**2318**	1565	691	423	438	364	324	322	280	171
1113	**2100**		419		206	171	171	169	132	93
1169	**3055**	1869	919	422	367	288	307	206	228	190
1596		**827**	386	343	231	169	152	133	94	94
23397	**3107**	3038	775	483	404	348	250	284	283	165
42468	**1299**	711	384	325	230	172	188	132	113	113
354	**1973**	965	429	325	285	243	240	202	164	95
criteo	**2032**	658	423	250	188	133	131	139	94	95

Table 5: Average F1 test score and number of pipelines explored (#PL), when running for 5 and 60 minutes. We compare the case where we use the optimal downsampling ratio (Best Ratio), and the case where we run on the originally-sized dataset (Full Data). Datasets that performed best with the full data were run with their respective second best ratio (marked with a star *). We find that when downsampling we can obtain slightly higher performance when increasing the execution time budget, but these improvements are modest in all but 4 datasets (41168, 1483, 1219, and 354). For 15 datasets the performance of a 5 minute downsampled search outperformed carrying out the search on the full dataset for 60 minutes.

	Best Ratio					Full Data				
ID	Ratio	5min	#PL	60min	#PL	Ratio	5min	#PL	60min	#PL
1468	0.50	0.952	301	0.955	2761	1.0	0.944	324	0.957	2639
3	0.50*	0.994	774	0.993	5574	1.0	0.995	581	0.996	4898
1489	0.50	0.887	855	0.889	7736	1.0	0.884	619	0.896	5930
40668	0.20	0.709	382	0.715	2931	1.0	0.582	262	0.645	1874
41138	0.15	0.909	250	0.902	1834	1.0	0.854	173	0.849	696
41168	0.05	0.552	388	0.577	2931	1.0	0.487	134	0.496	598
23517	0.20*	0.517	423	0.516	4652	1.0	0.518	187	0.520	1621
23512	0.15	0.721	383	0.721	3163	1.0	0.699	113	0.699	770
41150	0.10	0.929	365	0.931	3307	1.0	0.895	170	0.903	1066
1483	0.05	0.935	386	0.976	3075	1.0	0.577	151	0.610	573
1503	0.15	0.261	208	0.266	2865	1.0	0.103	113	0.106	457
1219	0.10	0.598	439	0.634	3232	1.0	0.536	172	0.565	792
1113	0.01	0.968	419	0.968	4637	1.0	0.939	93	0.938	418
1169	0.15	0.656	289	0.649	2716	1.0	0.626	190	0.632	558
1596	0.15	0.944	170	0.940	1307	1.0	0.640	95	0.603	220
23397	0.30	0.917	285	0.924	2255	1.0	0.728	165	0.841	867
42468	0.01	0.764	384	0.760	3343	1.0	0.693	114	0.686	269
354	0.05	0.947	326	0.991	3104	1.0	0.574	95	0.582	704
criteo	0.01	0.655	424	0.653	3966	1.0	0.572	96	0.595	269

When using the original dataset, without downsampling, we find that running for 60 minutes can increase scores slightly as well, however the best score still underperforms the score obtained after searching for 5 minutes with the optimal downsampling ratio in 15 of these 19 datasets. Of the remaining 4 datasets, 3 have less than 10,000 samples. Additionally the increase in number of pipelines explored when running for 60 minutes is on average smaller.

For 15 datasets, we found that executing a 5 minute search at the optimal downsampling ratio produced a higher performing pipeline than a 60 minute search using the full dataset. Interestingly, we find some cases where this is true even when the 60 minute search on the full dataset results in more candidate pipelines evaluated. For example, we see that for dataset 1483 a 0.05 downsampling ratio evaluated 386 pipelines and produced a final pipeline with a score of 0.935. Meanwhile, a 60 minute search on the full dataset evaluated 573 pipelines (almost 50% more pipelines) but the final pipeline achieved a lower score of 0.61. We believe such cases provide support to a hypothesis that the effects of downsampling do not only stem from the increased number of explored pipelines, but are rooted in deeper dataset characteristics. Analyzing such characteristics remains a question for future work, which we believe will be facilitated by our reproducibility package.

5.3.1 Extended execution times. We repeated the experiments presented in Table 5 for the four datasets with the largest performance improvement under their best downsampling ratio. And we considered an even longer execution time budget: 10 hours, based on the methodology presented in [22]. Table 6 shows the average F1 test score and number of pipelines explored when the search runs for 5, 60, 240 and 600 minutes. For the 600 minute experiments we increase the per pipeline evaluation budget to 5 minutes instead of 1 minute. We find that extending execution time to 600 minutes in the downsampled datasets marginally improves performance compared to a search using a budget of 60 minutes. In contrast, when using the full dataset, we observed improvements across the board when using the longer execution time budget. For all four datasets we find that the pipeline produced by a downsampled search executed for 5 minutes outperformed that produced by a 240 minute search on the full dataset. For three out of four datasets we similarly observe that a downsampled 5 minute search outperformed a 600 minute search on the full dataset.

5.4 RQ4: Pipeline characteristics

As we have seen so far, data downsampling can impact the number of pipelines generated, as well as their predictive performance. In this section we investigate the extent to which pipeline definitions change as a result of downsampling, in particular we investigate pipeline length and operator choices. To do this, we analyzed over 480,000 pipelines with more than 920,000 operators, the result of searches carried out under a 5 minute execution time budget. For purposes of our analysis, we define an operator to be a single step in the pipeline, which can correspond to a data transformer or a predictor (a distinction presented in [49]).

We calculate the amount of operators per pipeline for different sampling ratios, analyzing all the pipelines evaluated during the search procedure to account for changes during evolutions. Our

Table 6: Average F1 test score and number of pipelines explored (#PL), when running for 5, 60 240 and 600 minutes on the 4 datasets that saw the biggest improvements from sampling. Searching for 5 minutes on the ideally-downsampled dataset produced higher performance than a 240 minute search on the original dataset. Underlined experiments were executed with a per pipeline evaluation budget of 5 minutes instead of 1 minute. We note two scores for entry (1483, Best Ratio, 240 min) and (1483, Best Ratio, 600 min), with and without CV fold outlier performances of 0.045.

	Best Ratio									Full Data								
ID	Ratio	5min	#PL	60min	#PL	240min	#PL	600min	#PL	Ratio	5min	#PL	60min	#PL	240min	#PL	600min	#PL
1483	0.05	0.935	386	0.976	3075	0.800/0.988	8246	0.797/0.985	9743	1.0	0.577	151	0.610	573	0.774	2600	0.746	5462
1503	0.15	0.261	208	0.266	2865	0.265	9484	0.267	9679	1.0	0.103	113	0.106	457	0.145	1416	0.265	2766
1596	0.15	0.944	170	0.940	1307	0.942	2160	0.959	9680	1.0	0.640	95	0.603	220	0.691	909	0.738	5263
354	0.05	0.947	326	0.991	3104	0.956	9502	0.996	9851	1.0	0.574	95	0.582	704	0.622	4880	0.662	4286

results indicate that in addition to exploring more pipelines, downsampling allows the search procedure to consider more complex pipeline architectures, as indicated by average pipeline length. In particular, we find that on average, smaller sampling ratios result in pipelines with more operators. For example, when we use a downsampling ratio of 0.0001 the average pipeline has 1.85 (0.30 sd) operators, while a full dataset results in an average pipeline with 1.60 (0.12 sd) operators. For context, a recent large scale pipeline analysis by Psallidas et al [49] found that most user-implemented scikit-learn (TPOT's target API) pipelines consist of 1 – 4 operators.

To evaluate the frequency of operator components, we consider the top five most frequent predictor components in the pipelines produced. This is done over all pipelines generated and evaluated during the search procedure's execution. We then take the union of these API components and plot their frequency across all downsampling ratios. Figure 4 shows the relative frequency of a corresponding API component as the count of that component normalized by the total number of pipelines produced for that downsampling ratio. Relative frequencies are computed within each dataset, and then averaged across datasets for each sampling ratio.

First, we note that the union of top five components corresponds to only 8 operators: decision trees, extra trees classifier, random forests, extreme gradient boosting classifier, multilayer perceptrons, gradient boosting classifier, stochastic gradient descent classifier and bernoulli naive bayes classifier.[5] Interestingly, our results show that the relative frequency for these operators ranges more widely when we use more aggressive downsampling ratios, while at higher sampling ratios, components appear with more similar frequency. For example, at a sampling ratio of 0.0001 relative frequencies range from 3.8–12.2%, while at a full dataset we have a relative frequency range of 6.5–8.3%. This may suggest that the choice of predictor becomes increasingly less important as the dataset size grows.

That fundamental properties of a pipeline, such as its length and its final predictor component, vary between extreme downsampling and full data regimes is a *key* insight provided by these experimental results. AutoML system developers who opt to incorporate downsampling as a step to scaling to larger datasets will need to be aware of these properties. To the best of our knowledge, this is the first detailed study that characterizes these important changes in AutoML pipelines as a result of downsampling.

[5]We plot results using the class names from the Scikit-Learn API for clarity.

5.5 RQ5: Gradient Boosting Classifiers

Gradient boosting classifiers, and in particular XGBoost [17], are popular machine learning models due to their effectiveness and ease of use for tabular datasets. We evaluated the extent to which downsampling observations impacts this subset of classifiers.

We restricted the set of classifiers available in TPOT's search space to gradient boosting classifiers, particularly scikit-learn's `GradientBoostingClassifier` and xgboost's `XGBClassifier`. Pipelines generated may include additional operators, such as feature preprocessors, but can only include the two gradient boosting classifiers specified. We ran the AutoML search with varying downsampling rates on the four datasets (1483, 1503, 1596, 354) for which we observed the largest performance improvement when downsampling in our prior experiments. We ran each search using the methodology described in Section 4.2 and time budgets of 5 minutes.

We present our results in Figure 5. At higher sampling rates, the AutoML search was unable to successfully produce an optimization result. This optimization failure is driven by two factors: (1) challenges in appropriately combining search parallelization across individuals in the GP-process and parallelization in the gradient boosting classifier's training routine itself, and (2) the time complexity for gradient boosting classifiers. In particular, XGBoost's time complexity [17] scales linearly with the number of non-missing entries, and thus results in long training times for our large datasets.

We increased the time budget to 60 minutes to mitigate these failures and found the outcome to be the same. In contrast, when we employ downsampling, the AutoML search concludes successfully and produces high performing pipelines even in the case of a 5 minute search budget. As in previous experiments, we find that there are (varying) optimal downsampling rates across these datasets, which range between 0.01 and 0.1.

Separately, we carried out the same experiment but replaced the GP-based AutoML search with a random search without parallelization to mitigate the first failure factor outlined previously. In this experiment, we find that we can produce trained instances in all cases. When downsampling, we found that the search evaluated substantially more candidates and that performance was comparable to searching on the full dataset.

5.6 RQ6: Pipeline Search Space Impact

As detailed in Section 5.4, downsampling does not only impact pipeline performance, but also key properties such as pipeline length and the distribution of operators used. To investigate the

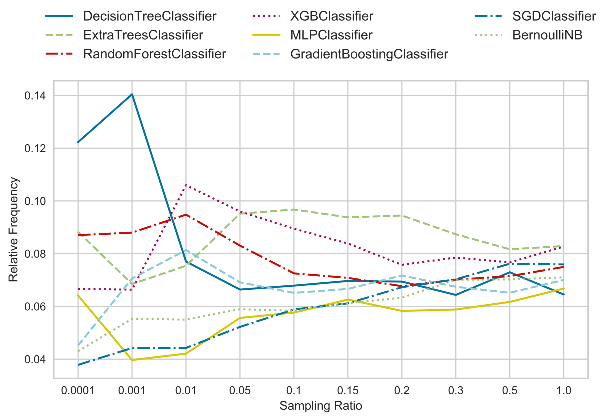


Figure 4: As we grow the dataset size, the relative frequencies of the top five operators in pipelines generated become increasingly similar.

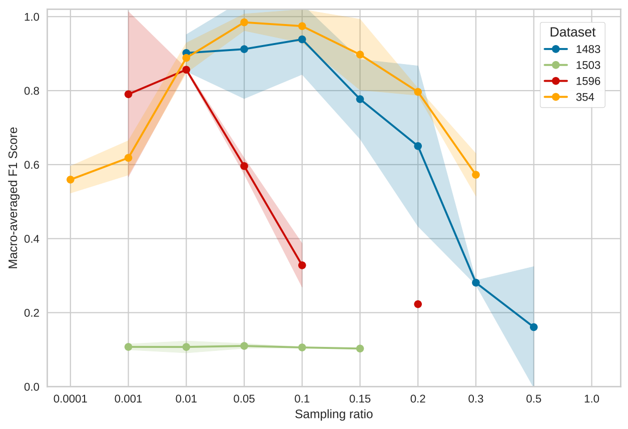


Figure 5: Restricting the subset of classifiers in TPOT's search space to gradient boosting classifiers. When using the full datasets, or a large portion of them, the search fails to produce a result given a 5 minute budget and a 60 minute budget. In contrast, the downsampled search concludes successfully even when given a 5 minute execution budget.

impact of the search space covered by downsampling compared to searching with the full dataset, we carried out the following experiment. First, we carried out the AutoML search with a 5 minute budget on a downsampled version of the training fold, collecting *all* pipeline candidates generated. We then ranked these candidates using the full training fold (excluding the portion of data used to carry out the candidate generation search), refit the best candidate, and evaluated the best candidate on the test fold. We compared the performance of this candidate versus the best candidate found when also using the downsampled data for candidate ranking.

We then carried out a second variant of this experiment, where pipeline candidate generation is done by running the search on the original (full) training fold with a 5 minute budget. The candidates produced are ranked based on a downsample of the fold (this downsample is excluded from the data used in candidate generation), and the best candidate is refit and evaluated on the test fold.

These experiments allow us to simulate a shared search space. In particular, when we generate candidates from a downsampled fold and rank using the full fold, we are simulating a setting where the full fold encountered the same set of pipeline candidates. And analogously for the second experiment variant. We emphasize that this is, by design, an artificial setup that allows us to isolate the impact of the search space.

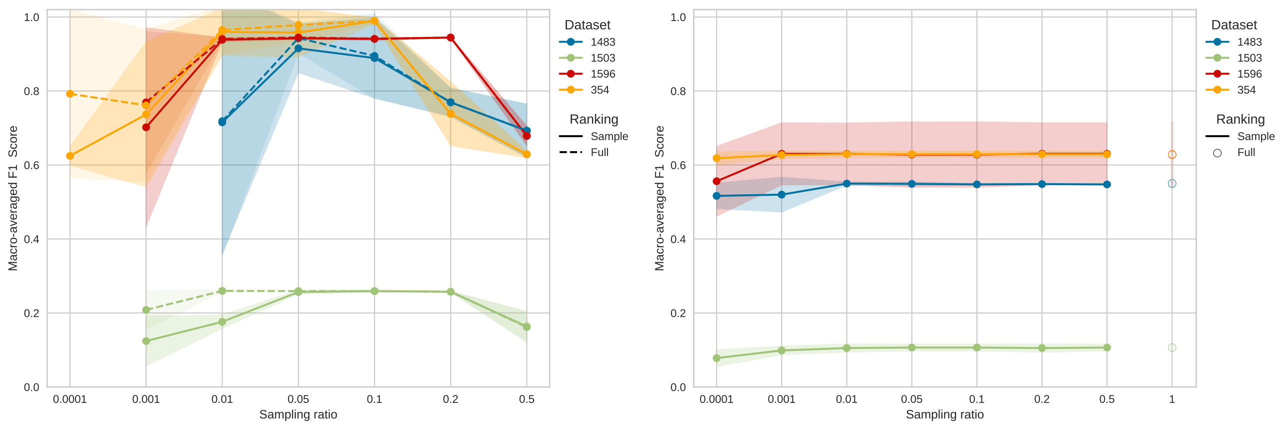
Figure 6 presents our results. Figure 6a shows the case where pipelines are generated using downsampled data and Figure 6b shows the case where pipelines are generated using the full dataset. In both cases, we find that the best pipeline produced by a re-ranking of pipelines with the downsampled or full dataset are nearly the same except for sampling ratio 0.0001. In contrast, there is a clear performance differential across the two scenarios for generating pipelines. This provides support to the hypothesis that the downsampled search is exploring a fundamentally different part of the search space. We re-ran these experiments using a 60 minute execution budget and obtained similar results.

5.7 Discussion

Our experiments explored various effects of downsampling in GP-based AutoML. Our results show that downsampling large datasets does not reduce predictive performance. In fact, for 15 of 16 datasets with over 10,000 rows, we found downsampling to actually improve performance over executing AutoML search on the original dataset. Next, we showed that downsampling allows the AutoML search procedure to explore more pipelines, up to 1.3 to 22.6 more pipelines for some datasets when using aggressive downsampling ratios, as compared to running the search on the original dataset. We found that predictive performance does increase slightly when we run a downsampled search for 60 minutes rather than a shorter execution time budget of 5 minutes. However, interestingly for 15 of the 16 datasets with over 10,000 observations we found that the downsampled 5 minute search produced a better performing pipeline than a 60 minute search on the full dataset. In fact, for the four datasets with the largest improvement in performance from downsampling, we found that a 240 minute search on the full dataset still produced lower performing pipelines than a 5 minute search on the dataset downsampled at its optimal rate. Even when we increase the time budget to 600 minutes we observe this same behavior for three out of these four datasets.

Our results show that when downsampling we produce pipelines that are (on average) slightly longer than those produced when using the full dataset. Furthermore, there is more variation in the type of predictor used at the end of the pipelines produced when searching with a smaller downsampling ratio.

When using the full dataset, our gradient-boosting focused search failed to produce a candidate given a 5 minute budget. Extending it to 60 minutes still resulted in failures. In contrast, using a downsampled search evaluated multiple candidates and concluded successfully, under both a 5 minute and 60 minute budget.



(a) Downsampled-generated pipelines **(b) Full-dataset-generated pipelines**

Figure 6: Our search space experiments show that ranking pipeline candidates using the downsampled dataset versus the full dataset produces similar performance, given the same set of candidate pipelines. Meanwhile, generating pipelines with downsampled searches appears to produce higher performing candidates overall. This provides support to the hypothesis that the downsampled search is exploring a fundamentally different part of the search space.

Our synthetic search space experiments, where we generate pipelines using a downsampled/full dataset and then rank candidates with the full/downsampled dataset provided a key insight. While the ranking of candidates was comparable with both the downsampled and the full dataset, candidates generated by the downsampled search had overall higher performance. This supports our hypothesis that downsampled searches are exploring a fundamentally different part of the pipeline search space.

Based on these results, we believe downsampling provides an non-intrusive and easy to implement option for scaling up GP-based AutoML to larger datasets – however, practitioners need to be aware of the fact that downsampling can fundamentally impact the search space explored and the resulting pipeline obtained.

From a practical point of view, downsampling between 0.01 and 0.2 of the original dataset appears to be a range that results in higher F1 scores, for our datasets. As such, practitioners may want to run GP-based AutoML searches with ratios in this range, as well as on the full dataset, and compare the resulting pipelines. In particular, the pipelines produced using the smaller downsampling rates may prove helpful in restricting the models and hyperparameter search spaces considered for future searches on the full dataset.

6 RELEASED EXPERIMENTAL DATASET

We have released our experimental results and configurations as a packaged dataset through Zenodo[6] [63].

An overview of this dataset can be found in Table 7. We include different performance scores (at training time, test time, with and without refitting on the full dataset). For further analysis, the dataset also contains the amount of explored pipelines, all evaluated pipelines, the Pareto front of fitted pipelines (produced by TPOT) and the final pipeline associated with each search.

[6]https://zenodo.org/record/4292739, DOI: 10.5281/zenodo.4292739

Table 7: Overview of the experimental dataset that we have released as part of our study. Here we detail the number of datasets, pipeline (PL) optimization times, as well as the number and average length of generated pipelines.

Datasets	20			
PL Optimization Time	5 & 60 minutes			
Runtime	8 weeks			
Analyzed Pipelines	480,000			
Analyzed Operators	920,000			
Operators per PL (μ, σ)	0.0001	(1.85, 0.30)	0.15	(1.74, 0.16)
	0.001	(2.07, 0.18)	0.2	(1.73, 0.15)
	0.01	(1.94, 0.16)	0.3	(1.72, 0.15)
	0.05	(1.82, 0.14)	0.5	(1.67, 0.14)
	0.1	(1.78, 0.16)	1.0	(1.60, 0.12)

The dataset released totals 45GB of compressed binary pipeline data. We also provide a CSV version of the files.[7] Our release also includes all code[8] necessary to rerun experiments, packaged in the form of an extendable experiment framework. This framework also includes utilities for querying and visualizing experimental results.

7 THREATS TO VALIDITY

Our experiments use genetic programming (GP), specifically the AutoML tool TPOT [43], as a search procedure. Other search techniques can be used for AutoML, including random search, multi-armed-bandit optimization, and Bayesian optimization, among others. We scope our observations to GP-based AutoML. GP represents

[7]Given the size of the CSVs files, we recommend the use of specialized editors, such as Ron's Editor, be used to view in spreadsheet form.
[8]https://github.com/ipa-lab/autoML-sampling-public

a common AutoML search procedure [23, 43, 51, 65], and it is notoriously hard to scale to large datasets [35, 43], thus downsampling can provide significant benefits.

Our evaluation of downsampling focuses on stratified, uniform random sampling, as such it does not employ techniques such as adaptive sampling [24, 38], which might improve performance further. Our choice of uniform random sampling is motivated by its simplicity and ubiquity: implementing uniform downsampling prior to an AutoML search is straightforward and provides observable benefits. Investigating the impact of adaptive downsampling on AutoML search remains an open question for future work.

We carried out our evaluation on 20 different datasets, which we collected to satisfy our "larger dataset" goal. The impact on performance may vary depending on the datasets. To mitigate this risk, we considered datasets from varied sources and domains.

In our results and discussion, we identified factors that may play a role in the performance improvement observed when downsampling. In particular, we highlighted the number of pipelines explored, pipeline length, and the choice of pipeline predictors as possible factors. Furthermore, it is possible that other aspects of GP-based searches can interact with downsampled datasets to improve performance. Investigating additional factors at play remains an open question, and our goal is to encourage such exploration by releasing our experimental data.

8 RELATED WORK

A wide range of AutoML systems have been developed based on different search techniques. Hyperopt [10], one of the earliest hyperparameter optimization toolkits, allows users to write structured search spaces and optimizes these using a combination of random search and Bayesian optimization. TPOT [43], the tool that we use in our experiments, is based on genetic programming. TPOT-SH [45] introduced the use of successive-halving to improve the scalability of genetic-programming-based AutoML to larger datasets. Autosklearn [20] applied sequential model-based algorithm configuration [26], a generalization of traditional Bayesian optimization (BO), and meta-learning to automatically generate scikit-learn pipelines. H2O AutoML [36] combines randomly generated pipelines with ensembling. AL [16] uses dynamic program analysis on existing programs to learn a pipeline completion model that can be used to generate pipelines for new datasets. Autobazaar [53] and Alpine Meadow [52] use a combination of multi-armed bandits and Bayesian optimization to produce pipelines. OBOE [58] and TensorOBOE [59] are AutoML systems that use matrix and tensor completion techniques, in combination with active learning, to generate pipelines under limited execution time budgets.

Compared to these systems, we are not proposing a new search technique or implementing a new AutoML system. Our contribution is centered on extensively evaluating the impact of downsampling.

Bergstra and Bengio [9] introduced the use of random search for efficient hyper-parameter optimization in neural networks. Snoek et al [54] showed that BO could also be efficiently applied to hyperparameter tuning. Sequential model-based algorithm configuration [26] generalized BO to tackle core challenges in AutoML such as sets of hyperparameters with numeric and non-numeric domains. Karnin et al [28] showed that successive halving, in the context of multi-armed bandits, can be an effective exploration strategy. Li et al [37] introduce Hyperband, which formulates hyperparameter optimization as infinite-armed bandit problem, where configurations are randomly sampled and resource allocation is dynamic. Meta-learning has also been successfully applied to improve the efficiency of model and hyperparameter searches [47, 48, 52]. Early stopping of model training has a long history in the machine learning research community [7, 40, 50]. Downsampling the training data for the search procedure, as we evaluate in this paper, has previously been deployed in the context of classical machine learning [62]. Existing work [13, 21] has also shown that random projections – which can be efficiently computed and reduce overall learning cost – can be used as an effective form of dimensionality reduction for supervised learning.

Relatedly, there is a long line of work on improving the efficiency of machine learning from a systems perspective. For example, Elgohary et al [19] introduce the use of database compression techniques in linear algebra. Weld [44] optimizes data pipelines across independent libraries, speeding up data intensive computations. Kunft et al [33] developed Lara, a language and accompanying intermediate representation for machine learning pipelines which can be used to optimize their training. HELIX [57] can cache executions and the intermediate state of machine learning pipelines across iterations to improve the efficiency of pipeline development. Cerebero [41] can accelerate model selection within the context of neural network architectures by exploiting parallelism.

In contrast, we are not proposing a new search technique or system, but rather our contribution is a detailed empirical study of the impact of downsampling on a popular GP-based AutoML tool.

Gijsbers et al [22] identified shortcomings of existing benchmarking work and developed a benchmarking suite (including code and datasets) to evaluate AutoML systems. Zöller and Huber [64] provide an extensive evaluation of AutoML systems and baseline implementations of various of their underlying search techniques, including Bayesian optimization, multi-armed bandit search, random search, and grid search. Balaji and Allen [6] evaluate a set of AutoML systems on both regression and classification tasks. Milutinovic et al [39] present a unified framework for benchmarking different AutoML systems – simplifying the use of shared operators and standardizing pipeline definitions – along with their benchmarking results on a collection of existing AutoML systems. In contrast to these studies, we focus on the impact of downsampling large datasets prior to GP-based AutoML search, rather than evaluating AutoML systems in general.

9 CONCLUSION

We present an extensive evaluation of the impact of downsampling on genetic programming-based automated machine learning for classification tasks. Large datasets pose a scalability challenge for AutoML systems, which often require long optimization times for even moderately sized datasets. We evaluate varying downsampling ratios for 20 classification datasets and analyze predictive performance, runtime performance, and pipeline characteristics. We release the raw data collected during our experiments and the experimental framework used to carry them out to facilitate future research into the role of downsampling in AutoML.

REFERENCES

[1] [n.d.]. 1.10. Decision Trees scikit-learn 0.23.2 documentation. Retrieved 2020-10-12 from https://scikit-learn.org/stable/modules/tree.html#complexity

[2] [n.d.]. 1.4. Support Vector Machines scikit-learn 0.23.2 documentation. Retrieved 2020-10-12 from https://scikit-learn.org/stable/modules/svm.html#complexity

[3] [n.d.]. Manual AutoSklearn 0.10.0 documentation. Retrieved 2020-10-12 from https://automl.github.io/auto-sklearn/master/manual.html#time-and-memory-limits

[4] 2016. Possible speed up at large data sets Issue #87 EpistasisLab/tpot. Retrieved 2020-10-12 from https://github.com/EpistasisLab/tpot/issues/87

[5] 2016. Speed and time budgets Issue #57 automl/auto-sklearn. Retrieved 2020-10-12 from https://github.com/automl/auto-sklearn/issues/57

[6] Adithya Balaji and Alexander Allen. 2018. Benchmarking Automatic Machine Learning Frameworks. *CoRR* abs/1808.06492 (2018). arXiv:1808.06492 http://arxiv.org/abs/1808.06492

[7] Andrew R. Barron, Albert Cohen, Wolfgang Dahmen, and Ronald A. DeVore. 2008. Approximation and learning by greedy algorithms. *The Annals of Statistics* 36, 1 (2008), 64 – 94. https://doi.org/10.1214/009053607000000631

[8] Benjamin Bengfort and Rebecca Bilbro. 2019. *Yellowbrick: Visualizing the Scikit-Learn Model Selection Process*. https://doi.org/10.21105/joss.01075

[9] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *J. Mach. Learn. Res.* 13 (2012), 281–305. http://dl.acm.org/citation.cfm?id=2188395

[10] James Bergstra, Daniel Yamins, and David D. Cox. 2013. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013 (JMLR Workshop and Conference Proceedings)*, Vol. 28. JMLR.org, 115–123. http://proceedings.mlr.press/v28/bergstra13.html

[11] Daniel Berrar, Philippe Lopes, and Werner Dubitzky. 2019. Incorporating domain knowledge in machine learning for soccer outcome prediction. *Mach. Learn.* 108, 1 (2019), 97–126. https://doi.org/10.1007/s10994-018-5747-8

[12] Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Frank Hutter, Michel Lang, Rafael Gomes Mantovani, Jan N. van Rijn, and Joaquin Vanschoren. 2017. OpenML Benchmarking Suites and the OpenML100. *CoRR* abs/1708.03731 (2017). arXiv:1708.03731 http://arxiv.org/abs/1708.03731

[13] Avrim Blum. 2005. Random Projection, Margins, Kernels, and Feature-Selection. In *Subspace, Latent Structure and Feature Selection, Statistical and Optimization, Perspectives Workshop, SLSFS 2005, Bohinj, Slovenia, February 23-25, 2005, Revised Selected Papers (Lecture Notes in Computer Science)*, Craig Saunders, Marko Grobelnik, Steve R. Gunn, and John Shawe-Taylor (Eds.), Vol. 3940. Springer, 52–68. https://doi.org/10.1007/11752790_3

[14] Aniket Bochare, Aryya Gangopadhyay, Yelena Yesha, Anupam Joshi, Yaacov Yesha, Mary Brady, Michael A. Grasso, and Naphtali Rishe. 2014. Integrating domain knowledge in supervised machine learning to assess the risk of breast cancer. *Int. J. Medical Eng. Informatics* 6, 2 (2014), 87–99. https://doi.org/10.1504/IJMEI.2014.060245

[15] José Pablo Cambronero, Jürgen Cito, and Martin C. Rinard. 2020. AMS: generating AutoML search spaces from weak specifications. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, 763–774. https://doi.org/10.1145/3368089.3409700

[16] José Pablo Cambronero and Martin C. Rinard. 2019. AL: autogenerating supervised learning programs. *Proc. ACM Program. Lang.* 3, OOPSLA (2019), 175:1–175:28. https://doi.org/10.1145/3360601

[17] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[18] DARPA. 2021. Public D3M datasets. Website. Retrieved April 21, 2021 from https://datasets.datadrivendiscovery.org/d3m/datasets

[19] Ahmed Elgohary, Matthias Boehm, Peter J. Haas, Frederick R. Reiss, and Berthold Reinwald. 2016. Compressed Linear Algebra for Large-Scale Machine Learning. *Proc. VLDB Endow.* 9, 12 (2016), 960–971. https://doi.org/10.14778/2994509.2994515

[20] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and Robust Automated Machine Learning. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett (Eds.). 2962–2970. http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning

[21] Dmitriy Fradkin and David Madigan. 2003. Experiments with random projections for machine learning. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*, Lise Getoor, Ted E. Senator, Pedro M. Domingos, and Christos Faloutsos (Eds.). ACM, 517–522. https://doi.org/10.1145/956750.956812

[22] Pieter Gijsbers, Erin LeDell, Janek Thomas, Sébastien Poirier, Bernd Bischl, and Joaquin Vanschoren. 2019. An Open Source AutoML Benchmark. *CoRR* abs/1907.00909 (2019). arXiv:1907.00909 http://arxiv.org/abs/1907.00909

[23] Pieter Gijsbers and Joaquin Vanschoren. 2019. GAMA: Genetic Automated Machine learning Assistant. *J. Open Source Softw.* 4, 33 (2019), 1132. https://doi.org/10.21105/joss.01132

[24] Haibo He, Yang Bai, Edwardo A. Garcia, and Shutao Li. 2008. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In *Proceedings of the International Joint Conference on Neural Networks, IJCNN 2008, part of the IEEE World Congress on Computational Intelligence, WCCI 2008, Hong Kong, China, June 1-6, 2008*. IEEE, 1322–1328. https://doi.org/10.1109/IJCNN.2008.4633969

[25] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2019. AutoML: A Survey of the State-of-the-Art. *CoRR* abs/1908.00709 (2019). arXiv:1908.00709 http://arxiv.org/abs/1908.00709

[26] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. In *Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers (Lecture Notes in Computer Science)*, Carlos A. Coello Coello (Ed.), Vol. 6683. Springer, 507–523. https://doi.org/10.1007/978-3-642-25566-3_40

[27] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren (Eds.). 2019. *Automated Machine Learning - Methods, Systems, Challenges*. Springer. https://doi.org/10.1007/978-3-030-05318-5

[28] Zohar Karnin, Tomer Koren, and Oren Somekh. 2013. Almost Optimal Exploration in Multi-Armed Bandits *(Proceedings of Machine Learning Research)*, Sanjoy Dasgupta and David McAllester (Eds.), Vol. 28. PMLR, Atlanta, Georgia, USA, 1238–1246. http://proceedings.mlr.press/v28/karnin13.html

[29] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. 2018. Data Scientists in Software Teams: State of the Art and Challenges. *IEEE Trans. Software Eng.* 44, 11 (2018), 1024–1038. https://doi.org/10.1109/TSE.2017.2754374

[30] Ron Kohavi. 1995. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*. Morgan Kaufmann, 1137–1145. http://ijcai.org/Proceedings/95-2/Papers/016.pdf

[31] John R. Koza. 1993. *Genetic programming - on the programming of computers by means of natural selection*. MIT Press.

[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (2017), 84–90. https://doi.org/10.1145/3065386

[33] Andreas Kunft, Asterios Katsifodimos, Sebastian Schelter, Sebastian Breß, Tilmann Rabl, and Volker Markl. 2019. An Intermediate Representation for Optimizing Machine Learning Pipelines. *Proc. VLDB Endow.* 12, 11 (2019), 1553–1567. https://doi.org/10.14778/3342263.3342633

[34] Criteo Labs. 2014. Kaggle Display Advertising Challenge Dataset. Website. Retrieved April 21, 2021 from https://labs.criteo.com/2014/02/kaggle-display-advertising-challenge-dataset/

[35] Trang T. Le, Weixuan Fu, and Jason H. Moore. 2020. Scaling tree-based automated machine learning to biomedical big data with a feature set selector. *Bioinform.* 36, 1 (2020), 250–256. https://doi.org/10.1093/bioinformatics/btz470

[36] Erin LeDell and Sebastien Poirier. 2020. H2O AutoML: Scalable Automatic Machine Learning. *7th ICML Workshop on Automated Machine Learning (AutoML)* (July 2020). https://www.automl.org/wp-content/uploads/2020/07/AutoML_2020_paper_61.pdf

[37] Lisha Li, Kevin G. Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *J. Mach. Learn. Res.* 18 (2017), 185:1–185:52. http://jmlr.org/papers/v18/16-558.html

[38] Mario Lucic, Mesrob I. Ohannessian, Amin Karbasi, and Andreas Krause. 2015. Tradeoffs for Space, Time, Data and Risk in Unsupervised Learning. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2015, San Diego, California, USA, May 9-12, 2015 (JMLR Workshop and Conference Proceedings)*, Guy Lebanon and S. V. N. Vishwanathan (Eds.), Vol. 38. JMLR.org. http://proceedings.mlr.press/v38/lucic15.html

[39] Mitar Milutinovic, Brandon Schoenfeld, Diego Martinez-Garcia, Saswati Ray, Sujen Shah, and David Yan. 2020. On evaluation of automl systems. In *Proceedings of the ICML Workshop on Automatic Machine Learning*.

[40] Nelson Morgan and Hervé Bourlard. 1990. Generalization and Parameter Estimation in Feedforward Nets: Some Experiments. (1990), 630–637.

[41] Supun Nakandala, Yuhao Zhang, and Arun Kumar. 2020. Cerebro: A Data System for Optimized Deep Learning Model Selection. *Proc. VLDB Endow.* 13, 11 (2020), 2159–2173. http://www.vldb.org/pvldb/vol13/p2159-nakandala.pdf

[42] Fatemeh Nargesian, Erkang Zhu, Renée J. Miller, Ken Q. Pu, and Patricia C. Arocena. 2019. Data Lake Management: Challenges and Opportunities. *Proc. VLDB Endow.* 12, 12 (2019), 1986–1989. https://doi.org/10.14778/3352063.3352116

[43] Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. 2016. Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data

Science. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, Denver, CO, USA, July 20 - 24, 2016*, Tobias Friedrich, Frank Neumann, and Andrew M. Sutton (Eds.). ACM, 485–492. https://doi.org/10.1145/2908812.2908918

[44] Shoumik Palkar, James J Thomas, Anil Shanbhag, Deepak Narayanan, Holger Pirk, Malte Schwarzkopf, Saman Amarasinghe, Matei Zaharia, and Stanford InfoLab. 2017. Weld: A common runtime for high performance data analytics. In *Conference on Innovative Data Systems Research (CIDR)*. 45.

[45] Laurent Parmentier, Olivier Nicol, Laetitia Jourdan, and Marie-Eléonore Kessaci. 2019. TPOT-SH: A Faster Optimization Algorithm to Solve the AutoML Problem on Large Datasets. In *31st IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2019, Portland, OR, USA, November 4-6, 2019*. IEEE, 471–478. https://doi.org/10.1109/ICTAI.2019.00072

[46] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* 12 (2011), 2825–2830. http://dl.acm.org/citation.cfm?id=2078195

[47] Valerio Perrone, Rodolphe Jenatton, Matthias W. Seeger, and Cédric Archambeau. 2018. Scalable Hyperparameter Transfer Learning. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (Eds.). 6846–6856. https://proceedings.neurips.cc/paper/2018/hash/14c879f3f5d8ed93a09f6090d77c2cc3-Abstract.html

[48] Valerio Perrone and Huibin Shen. 2019. Learning search spaces for Bayesian optimization: Another view of hyperparameter transfer learning. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 12751–12761. https://proceedings.neurips.cc/paper/2019/hash/6ea3f1874b188558fafbab78e8c3a968-Abstract.html

[49] Fotis Psallidas, Yiwen Zhu, Bojan Karlas, Matteo Interlandi, Avrilia Floratou, Konstantinos Karanasos, Wentao Wu, Ce Zhang, Subru Krishnan, Carlo Curino, and Markus Weimer. 2019. Data Science through the looking glass and what we found there. *CoRR* abs/1912.09536 (2019). arXiv:1912.09536 http://arxiv.org/abs/1912.09536

[50] Garvesh Raskutti, Martin J. Wainwright, and Bin Yu. 2014. Early stopping and non-parametric regression: an optimal data-dependent stopping rule. *J. Mach. Learn. Res.* 15, 1 (2014), 335–366. http://dl.acm.org/citation.cfm?id=2627446

[51] Esteban Real, Chen Liang, David R So, and Quoc V Le. 2020. AutoML-Zero: Evolving Machine Learning Algorithms From Scratch. *arXiv preprint arXiv:2003.03384* (2020).

[52] Zeyuan Shang, Emanuel Zgraggen, Benedetto Buratti, Ferdinand Kossmann, Philipp Eichmann, Yeounoh Chung, Carsten Binnig, Eli Upfal, and Tim Kraska. 2019. Democratizing Data Science through Interactive Curation of ML Pipelines. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 1171–1188. https://doi.org/10.1145/3299869.3319863

[53] Micah J. Smith, Carles Sala, James Max Kanter, and Kalyan Veeramachaneni. 2020. The Machine Learning Bazaar: Harnessing the ML Ecosystem for Effective System Development. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 785–800. https://doi.org/10.1145/3318464.3386146

[54] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, L é on Bottou, and Kilian Q. Weinberger (Eds.). 2960–2968. http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms

[55] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luís Torgo. 2013. OpenML: networked science in machine learning. *SIGKDD Explor.* 15, 2 (2013), 49–60. https://doi.org/10.1145/2641190.2641198 https://openml.github.io/OpenML/benchmark.

[56] Manasi Vartak, Harihar Subramanyam, Wei-En Lee, Srinidhi Viswanathan, Saadiyah Husnoo, Samuel Madden, and Matei Zaharia. 2016. ModelDB: a system for machine learning model management. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics, HILDA@SIGMOD 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Carsten Binnig, Alan D. Fekete, and Arnab Nandi (Eds.). ACM, 14. https://doi.org/10.1145/2939502.2939516

[57] Doris Xin, Stephen Macke, Litian Ma, Jialin Liu, Shuchen Song, and Aditya G. Parameswaran. 2018. Helix: Holistic Optimization for Accelerating Iterative Machine Learning. *Proc. VLDB Endow.* 12, 4 (2018), 446–460. https://doi.org/10.14778/3297753.3297763

[58] Chengrun Yang, Yuji Akimoto, Dae Won Kim, and Madeleine Udell. 2019. OBOE: Collaborative Filtering for AutoML Model Selection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis (Eds.). ACM, 1173–1183. https://doi.org/10.1145/3292500.3330909

[59] Chengrun Yang, Jicong Fan, Ziyang Wu, and Madeleine Udell. 2020. AutoML Pipeline Selection: Efficiently Navigating the Combinatorial Space. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash (Eds.). ACM, 1446–1456. https://doi.org/10.1145/3394486.3403197

[60] Quanming Yao, Mengshuo Wang, Hugo Jair Escalante, Isabelle Guyon, Yi-Qi Hu, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, and Yang Yu. 2018. Taking Human out of Learning Applications: A Survey on Automated Machine Learning. *CoRR* abs/1810.13306 (2018). arXiv:1810.13306 http://arxiv.org/abs/1810.13306

[61] Ting Yu, Simeon J. Simoff, and Tony Jan. 2010. VQSVM: A case study for incorporating prior domain knowledge into inductive machine learning. *Neurocomputing* 73, 13-15 (2010), 2614–2623. https://doi.org/10.1016/j.neucom.2010.05.007

[62] Xueqiang Zeng and Gang Luo. 2017. Progressive sampling-based Bayesian optimization for efficient and automatic machine learning model selection. *Health Inf. Sci. Syst.* 5, 1 (2017), 2. https://doi.org/10.1007/s13755-017-0023-z

[63] Fatjon Zogaj, José Cambronero, Martin Rinard, and Jürgen Cito. 2020. *Released Experimental Dataset for Sampled Automated Machine Learning.* https://doi.org/10.5281/zenodo.4292739

[64] Marc-André Zöller and Marco F. Huber. 2021. Benchmark and Survey of Automated Machine Learning Frameworks. *J. Artif. Intell. Res.* 70 (2021), 409–472. https://doi.org/10.1613/jair.1.11854

[65] Jason Zutty, Daniel Long, Heyward Adams, Gisele Bennett, and Christina Baxter. 2015. Multiple Objective Vector-Based Genetic Programming Using Human-Derived Primitives. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015*, Sara Silva and Anna Isabel Esparcia-Alcázar (Eds.). ACM, 1127–1134. https://doi.org/10.1145/2739480.2754694