Doing More with Less: Characterizing Dataset Downsampling for AutoML [Experiment, Analysis & Benchmark Papers]

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 datasets, the same procedure does not directly scale to larger datasets with 100,000s of observations, often producing fewer candidate pipelines and yielding lower performance. We empirically investigate downsampling as a way to mitigate this challenge. Our evaluation on 16 datasets (4 smaller datasets with less than 10,000 observations and 12 large datasets with 50,000 to over 1,000,000 observations) shows that for a widely-used AutoML search procedure downsampling increases the number of pipelines evaluated and improves the predictive performance of the final pipeline. More specifically, we find that for most of the larger datasets (11 out of 12), the sampling ratio yielding the best predictive performance is between 0.05 and 0.2. We also see that aggressive downsampling ratios can lead to up to 20x more pipelines being explored by the search procedure as compared to performing search with the full dataset. Interestingly, in a control experiment where we performed search for 60 minutes (instead of 5 minutes) on the full dataset, we find that resulting predictive performance still underperforms the pipeline resulting from a 5 minute search on the downsampled dataset. The release of an extensive reproducibility package, along with our empirical insights, open up the possibility to generate and investigate additional theoretical hypotheses in future analyses.

PVLDB Reference Format:
Fatjon Zogaj, José Pablo Cambronero, Martin C. Rinard, and Jürgen Cito. Doing More with Less: Characterizing Dataset Downsampling for AutoML [Experiment, Analysis & Benchmark Papers]. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

*Both authors contributed equally to the paper

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at http://vldb.org/pvldb/format_vol14.html.

1 INTRODUCTION

Building and tuning well performing machine learning systems is a difficult task that benefits from domain and specialized data science knowledge [8, 10, 27, 49]. 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 [11, 37]. 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 challenges of this task has motivated the use of automated systems which can generate pipelines from an extensive search space, validate them, and identify the best performing with the aim of requiring little-to-no human intervention. These systems have been broadly termed automated machine learning (AutoML) [23].

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 [22, 28]. These requirements further increase when a user would like to apply AutoML 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 [30, 35] and user reports [3–5].

Many typical 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 [36], 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 cache is used efficiently) [2]. While training a single model may not be onerous, in the context of AutoML we are interested in training (and evaluating) 1,000s to 10,000s of models, which becomes increasingly difficult with an increasing number of observations.

1.1 Impact of Downsampling in AutoML

We empirically investigate the use of a preprocessing step to improve the scalability of AutoML search to large 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.

To motivate the use of downsampling, we focus our study on the impact that downsampling has on a popular AutoML search procedure. We carry out this study by collecting a benchmark set of 16 datasets from the open data repository OpenML [43]. We focus our data collection on medium to large datasets, a subset of datasets that has traditionally been excluded from AutoML benchmarking suites [9] due to the computational burden. Medium to larger datasets are particularly important as they more closely reflect the increasingly large amounts of data in industry [33], and they are a challenge for existing AutoML systems. We use these 16 datasets (4 smaller datasets with less than 10,000 observations and 12 large datasets with 50,000 to over 1,000,000 observations) to explore the impact of varying downsampling ratios and execution time budgets, the two main algorithm hyperparameters in our proposed downsampling.

Our results show that, for genetic-programming-based AutoML, downsampling provided significant benefits for large datasets. In particular, we found that for 14 of our 16 datasets, the search procedure produced a *higher* scoring pipeline when we downsampled the training data. For 11 out of 12 of the larger datasets, the optimal downsampling ratio in our evaluation ranged between 0.05 and 0.2, a significant reduction in number of observations required. Our results also show that when executing the search with the dataset-optimal downsampling ratio, the search procedure evaluated up to 3 times more pipelines than when the procedure is carried out on the original dataset.

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 the 370,000 pipelines generated, which consist of over 720,000 individual operators. We found that 8 different API components constitute the top 5 most frequently occurring operators across all our sampling rations. Interestingly, the relative frequency of these operators varies substantially (by up to 8.2 percentage points) for lower sampling ratios, while these same components appear with a more similar relative frequency (differing by only 2.3 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, potentially reflecting the benefit of larger data over particular algorithm properties.

Carrying out our full set of experiments represented a significant computational cost, totaling approximately 2 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 AutoML to larger datasets.
- We collected a benchmark suite of 16 primarily medium to large datasets, and evaluated the impact of downsampling on a popular genetic-programming-based AutoML search. We focus our evaluation on the two main hyperparameters in our 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 presentation 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 Section 5. Section 6 provides an overview of the experimental dataset that we publish. In Section 7 we present possible threats to validity and mitigants. Section 8 presents existing literature and their relation to our contribution. Finally, Section 9 provides closing remarks on the contributions of this work.

2 BACKGROUND

In this section, we present a self-contained overview of machine learning pipelines in the context of AutoML.

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 learning algorithms. Typically such a pipeline is implemented by composing operators in a domain specific library, such as Scikit-Learn [36], a popular Python machine learning library. To fully define the pipeline, a developer must choose values for hyperparameters (or opt to use defaults)

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

Figure 1: An example pipeline implemented using Scikit-Learn. The pipeline consists of a preprocessing step, which imputes values columnwise using the median value, and then uses a naive bayes classifier to learn classification labels. Note that both components have hyperparameters that the developer must set (`strategy`, `binarizer`, `alpha`, `fit_prior`). This pipeline was automatically generated using TPOT [34].

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 [44]. 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 expert developer to manually explore the space of pipelines.

2.2 AutoML

To formalize the AutoML problem, we follow the formulation presented in [11].

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. a label for classification, or a real value for regression). 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 $e \in \mathcal{E}$ be a scoring function that measures the predictive performance of a pipeline, such as cross-validated macro-averaged F1 score. Let $c \in C : \mathcal{P} \times \mathcal{D} \rightarrow \mathbb{R}$ be a cost function that evaluates the time it takes to construct, fit (i.e. learn any free parameters), and evaluate a pipeline p on the dataset d. 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

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

The pipeline search space $\mathcal{P}$ grows exponentially with the number of composable operators and their configurations. As a result, an AutoML search cannot evaluate all pipelines. In practice, AutoML systems often use heuristics (manually defined or learned) to efficiently navigate the space of possible pipelines [23, 48].

3 DOWNSAMPLING AUTOML

Large datasets pose a particular challenge for AutoML, as the the complexity of many key operators scales with the number of observations in a dataset. Table 1 presents example runtime complexities

Classifier	Time Complexity
Naive Bayes	$O(nm)$
SVM	$O(n^3)$
Decision Tree	$O(n \log(n) m)$
Random Forest	$O(n_{\text{trees}} n \log(n) m)$

Table 1: Runtime complexity of different classifiers where n is the number of observations and m is the number of features [1, 2, 14]. Note that runtime complexity can vary further depending on many factors, such as the underlying implementation, regularization penalties, the optimization algorithm (and its implementation) and any special cases it may have, etc.

for popular classification algorithms. These challenges are further exacerbated when we want to run an AutoML search with a low execution time budget.

We systematically evaluate one approach to alleviating this challenge: downsampling the dataset prior to executing the AutoML search. We formalize this approach by making a slight modification to Equation (1). Let $s \in \mathcal{S} : \mathcal{D} \times \mathbb{R} \rightarrow \mathcal{D}$ be a sampling function, which takes a dataset $d \in D$, a real-valued sampling ratio $r \in \mathbb{R}$, and returns a dataset $d' \in D$, such that $\frac{|d'|}{|d|} \approx r$, where $|x|$ counts the number of rows in x.

The downsampled AutoML problem is now:

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

The main search loop operates on the downsampled dataset to identify the most promising candidate pipeline. 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: A fitted pipeline $p_{\text{fitted}}$
  procedure DOWNSAMPLEANDSEARCH
      ▷ Sample down to ratio $r$
      $d'_{\text{train}} \leftarrow s(d_{\text{train}}, r)$
      ▷ Generate best pipeline, given the downsampled dataset
      $p \leftarrow$ SEARCH($d'_{\text{train}}$, $b$)
      ▷ Fit the best candidate on the full training set
      $p_{\text{fitted}} \leftarrow p(d_{\text{train}})$
      return $p_{\text{fitted}}$
```

In this formulation, the AutoML system is free to optimize the ratio r chosen for a given dataset and pipeline to maximize the predictive performance when evaluated on the *originally sized* training dataset. Introducing a downsampling ratio 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 performance (i.e. longer leads to better) still holds.

Does sampling impact the operator composition of pipelines? When using AutoML across different datasets, we often obtain pipelines with varying components or architectures, which reflect the need to tune pipelines to the dataset at hand. With the introduction of downsampling, we ask whether the components used in pipelines vary as a function of the downsampling ratio r.

4 METHODOLOGY

Given the empirical focus of our contribution, we dedicate this section to a detailed description of our experimental methodology.

4.1 Large 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[1] focus on small to medium sized datasets (up to 100,000 observations). To address the need for larger datasets, we collected a total of 16 classification datasets through OpenML [43]. Our criteria for selection was:

- have a varied number of features (i.e. columns)
- cover small, medium, *and large* numbers of observations
- have a varied number of target classes
- removed easily solved benchmarks
- remove datasets that require sophisticated preprocessing or cleaning, a standard criteria in AutoML benchmark collection
- cover various domains, to avoid domain-specific data skews

Table 2 presents an overview of the datasets we collected. The datasets range from small datasets (1,000s of rows) to large datasets (100,000s of rows). During our benchmark dataset collection task we noted that there is a scarcity of datasets with 1MM rows or more that satisfy the criteria outlined above.

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: $r \in (0.0001, 0.001, 0.01, 0.05, 0.15, 0.2, 0.3, 0.5, 1.0)$.

OpenML 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
1169	airlines	2	8	539383
1596	covertype	7	55	581012
42468	hls4ml-lhc-jets-hlf	5	17	830000
354	poker	2	11	1025010

Table 2: An overview of our benchmark suite of classification datasets showing the OpenML ID and abbreviated name as well as the number of target classes, features and instances. We specifically collected datasets to cover larger datasets, in addition to smaller and medium datasets as typically used in AutoML benchmarking.

We used k-fold cross-validation [32] to carry out our evaluation. When splitting the dataset to produce different folds, we performed a stratified splitting[2] 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. Note that our benchmark suite does not have datasets in the 10,000 – 50,000 observations range, as we focused on collecting larger datasets (see Table 2 for more details).

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 $r = 0.1$, 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 a disjoint 0.2 of the original dataset.

For completeness, we also evaluate and track the performance of the final candidate pipeline produced *without* fitting on the full training fold. Note that in both cases the overall structure and hyperparameters (e.g. max depth for a decision tree) of the pipeline are set based on the AutoML search procedure, which executed its search on the downsampled data, and the only distinction is

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

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

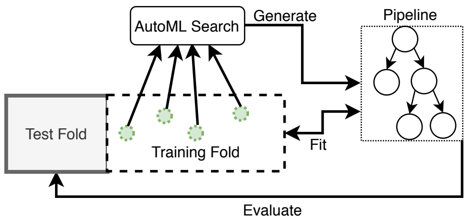

Figure 2: For a given iteration in our k-fold cross-validation procedure, we take the training fold (in white, dashed lines), and downsample to obtain a smaller set of observations (in green, dashed lines). The AutoML search is run using this downsampled set of observations. After a final pipeline has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

whether free parameters (such as model weights) are re-learned on the full training fold.

Implementation. We implemented our benchmarking methodology by building on the experimental framework used in [11]. We used the OpenML API to collect all datasets in our benchmark suite, Scikit-Learn's implementations of standard utilities such as k-fold cross validation splitting, F1-score, and yellowbrick's [7] learning curve visualization. AutoML systems are known [6, 51] to occasionally crash or freeze during benchmarking. To mitigate this issue, we relied on a mixed use of job queuing[3] and intermittent task monitoring, with associated job restarts when necessary for benchmarking completion.

To carry out our experiments, we used a well-provisioned compute server. Our server provided four 14 core Intel Xeon E5-2697 v3 CPUs (2.60 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 1 to 2 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, after the pipeline has been refit on the full training set (see Figure 2). 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.

To organize our results, we revisit the key research questions posed in Section 3.

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

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
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
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

Table 3: F1 score averaged over test folds for the pipelines generated under different training fold 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 yields higher performance.

5.1 RQ1: How does sampling affect 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 3 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, in contrast to performing AutoML search on the full dataset size. Only two out of our 16 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, 11 out of 12 obtained their best performance when we downsampled the training folds to 0.05 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.

Table 4 focuses on the 12 large datasets with more than 50,000 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

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
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
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

Table 4: 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	0.0001	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
1169	**3055**	1869	919	422	367	288	307	206	228	190
1596		**827**	386	343	231	169	152	133	94	94
42468	**1299**	711	384	325	230	172	188	132	113	113
354	**1973**	965	429	325	285	243	240	202	164	95

Table 5: Average amount 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 20 times more pipelines generated when compared to the full dataset.

across downsampling ratios for all datasets but 23517, which performs best when we use the full dataset. The optimal ratio per dataset varies between 0.05 and 0.2 for 10 of the other 11 datasets.

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).

5.2 RQ2: How does sampling affect 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 and varied the downsampling ratio.

Table 5 shows the average (over cross-validation 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 10 to 20 times 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 largest dataset with over a 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 sampling ratio that yields the best predictive performance for dataset 354 ($r = 0.05$) results in almost 3 times as many pipelines when compared to searching on the original dataset.

5.3 RQ3: Does downsampling impact the traditional relationship between execution time budget and predictive performance?

As discussed earlier, it is traditionally the case that the longer the AutoML search procedure runs, the more likely it will be 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

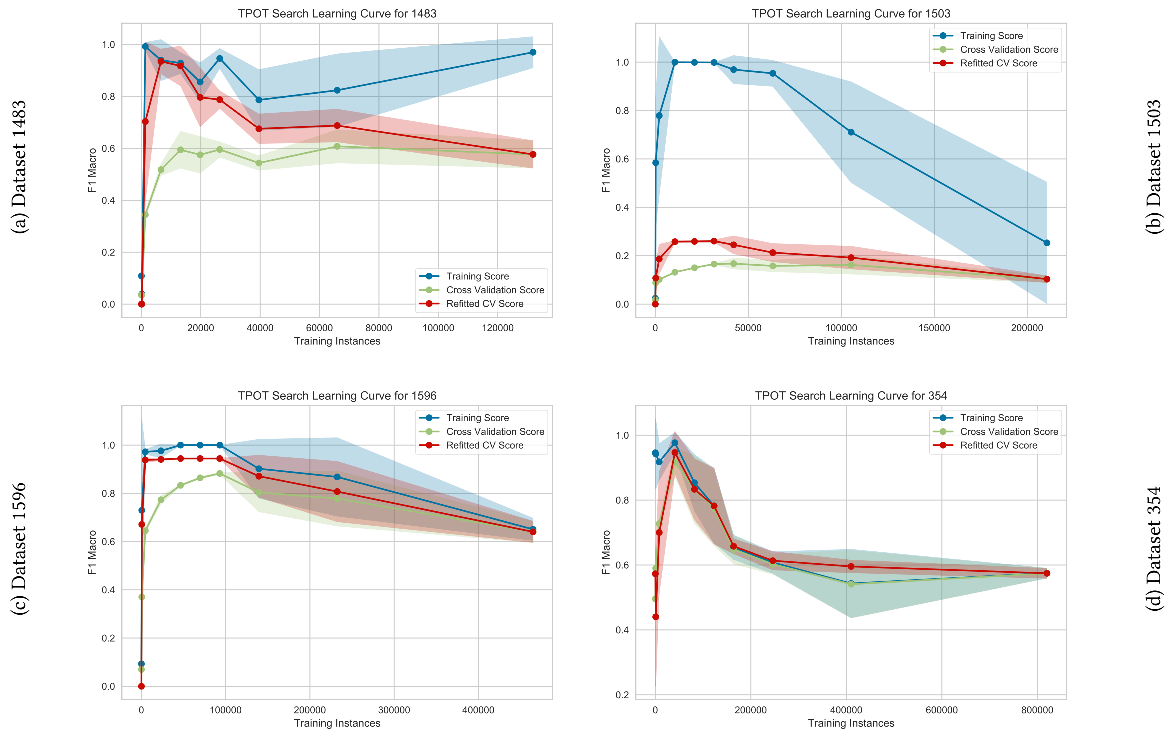

Figure 3: F1 score over different downsampling ratios for the datasets with the largest performance increase when using their optimal downsampling ratio. We show training F1 score (blue), test F1 score without refitting on the full training fold (green), and the final test F1 score when refitted on the 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.

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 6 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 (3 datasets: 41168, 1483, 354) these improvements are small. However, note that when we run the search procedure for 60 minutes, the number of pipelines explored does increase substantially (up to 10x).

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 with the optimal downsampling ratio in 11 of the 12 datasets (the exception being dataset 23517, which had an optimal sampling ratio of 1.0, i.e. the originally-sized dataset). More interestingly, we found that downsampling and running the search for 5 minutes resulted in better predictive performance than running the search for 60 minutes on the originally-sized data. We also note that the increase in number of pipelines explored when running for 60min is on average smaller, as expected.

Looking at the 5 minute result of dataset 1483 for example, we see that using just 0.05 of the data, we achieve a score of 0.935 after generating 386 pipelines. Comparing this to the 60 minute result using the full dataset, after generating 573 pipelines (almost 50% more) we achieve a significantly lower score of 0.610. We believe this result indicates 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.

5.4 RQ4: Does the distribution of API components change when downsampling?

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 the pipeline definitions change as a result of downsampling, in particular we investigate pipeline length and API usage. To do this, we analyzed over 370,000 pipelines with more than 720,000 operators, a result of searches carried out with 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 [37]).

	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
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
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

Table 6: Average F1 test score and number of pipelines explored (#PL), when running for 5 and 60 minutes. Datasets are sorted from smallest to largest number of observations. 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, with the exception of 3 datasets: 41168, 1483, 354, which display larger improvements. For 11 datasets the performance using downsampling and an execution time of 5min outperformed carrying out the search on the full dataset for 60 minutes.

We calculate the amount of operators per pipeline for different sampling ratios, taking into account all the pipelines evaluated during the search procedure to account for changes during evolutions. We average values within sampling ratios. We find that on average, smaller sampling ratios result in longer pipelines (i.e. containing more operators). For example, when we use a downsampling ratio of 0.0001 the average pipeline has 1.91 (0.29 sd) operators, while a full dataset results in an average pipeline with 1.62 (0.12 sd) operators. This result indicates that in addition to exploring more pipelines, downsampling allows the search procedure to consider more complex architectures.

To evaluate the frequency of operator components, we consider the top five most frequent predictor components in the pipelines produced when using each downsampling ratio. 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, random forests, k-nearest neighbors, extra trees classifier, multilayer perceptrons, gradient boosting classifier, extreme gradient boosting classifier, and a stochastic gradient descent classifier[4]. Interestingly, our results show that the relative frequency for these operators ranges more widely when we use lower downsampling ratios, while at higher sampling ratios, components appear with more similar frequency. For example, at a sampling ratio of 0.0001 the relative frequency ranges from 3.1–11.3%, while at a full dataset we have a relative frequency range of 6.1–8.4%. This change in frequency may suggest that the choice of predictor becomes increasingly less important as the dataset size grows.

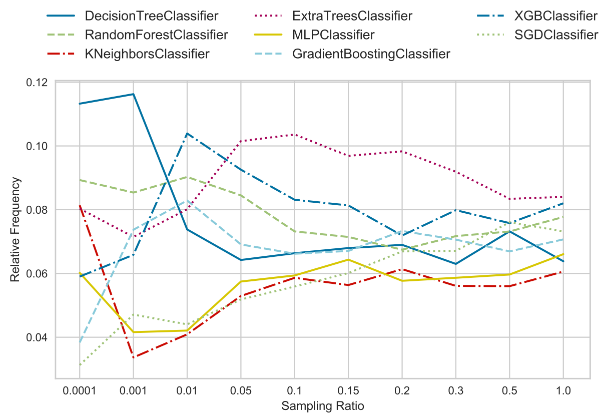

Figure 4: Frequency of the top 5 most frequently occurring operator components in the pipelines generated for each downsampling ratio. We take the union of such components and plot their relative frequency (count of component divided by number of pipelines) for each downsampling ratio. As we grow the dataset size, the relative frequencies of the top five operators become increasingly similar.

5.5 Discussion

Our experiments explored the impact of downsampling along multiple key dimensions. We first considered the impact of downsampling on predictive performance. Our results show that downsampling large datasets does not reduce predictive performance. In fact, for 14 of 16 datasets we found downsampling to actually improve performance over executing AutoML search on the original dataset. For 11 out of 12 larger datasets the downsampling ratio that yielded the best performance ranged between 0.05 and 0.2. For 3 out of 4 smaller datasets, the best performing sampling ratio was 0.5.

Next, we showed that downsampling allows the AutoML search procedure to explore more pipelines, up to 10 to 20 times more pipelines for some datasets when using aggressive downsampling ratios, as compared to running the search on the original dataset. Many key components in machine learning libraries have runtime complexity that grows as a (often quadratic) function of the number of observations in the data. Downsampling provides a straightforward approach to taming this complexity.

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

Datasets	1468, 12, 3, 1489, 40668, 41138, 41168, 23512 41150, 1483, 1503, 1169, 1596, 42468, 354			
PL Optimization Time	5 & 60 minutes			
Theoretical Runtime	1 − 2 weeks			
Actual Runtime	2 − 3 weeks			
Analyzed Pipelines	370.000			
Analyzed Operators	720.000			
Operators per PL (μ, σ)	0.0001	(1.91, 0.29)	0.15	(1.77, 0.16)
	0.001	(2.07, 0.19)	0.2	(1.76, 0.15)
	0.01	(1.95, 0.17)	0.3	(1.73, 0.17)
	0.05	(1.83, 0.15)	0.5	(1.69, 0.14)
	0.1	(1.80, 0.17)	1.0	(1.62, 0.12)

Table 7: Overview of the experimental dataset that we have released as part of our study. The theoretical runtime is calculated by the amount of time given per pipeline (PL) optimization. This runtime is further increased by e.g. TPOT exceeding the given budget or having to rerun experiments due to runtime exceptions or optimization failures.

We found that predictive performance does increase slightly when we run a downsampled search for 1 hour rather than a shorter execution time budget of 5 minutes. However, interestingly for 11 of the 12 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.

A particular challenge of applying genetic-programming-based AutoML to larger datasets lies in the evolutionary process itself: at relatively low execution time budgets GP not only explores fewer pipelines but is also unable to evolve the population past a few generations. For larger datasets, a 5 minute execution time budget may yield as few as a single generation. In contrast, the downsampled dataset enables multiple evolutionary steps, given the same execution time budget. While it may be tempting to attribute the success of downsampling exclusively to more evolutionary steps, we note that the performance differential remains when we increase the execution time budget to 60 minutes. In this setting, the 60 minute search procedure on the full dataset explored more pipelines and generations than the search on the downsampled dataset with a 5 minute budget, but in 11 of 12 large datasets the downsampled search still resulted in a pipeline with higher predictive performance. This phenomenon may suggest that evolution on subsamples of the data can provide additional benefits over evolution on the full dataset. Exploring the interaction of evolutionary generations, number of pipelines explored, and dataset size remains open for further investigation.

Finally, our results indicate that when downsampling we produce pipelines that are (on average) slightly longer than those produced when using the full dataset. Furthermore, the predictor used by these pipelines seems to be more meaningful, with a larger variation in the relative frequency of predictor for pipelines produced using lower downsampling ratios.

Based on these results, we believe downsampling provides an non-intrusive and easy to implement option for scaling up AutoML to larger datasets.

6 RELEASED EXPERIMENTAL DATASET

We have released our experimental results as a packaged dataset through Zenodo[5]. The goal of releasing our raw experimental results is to enable other researchers to explore the impact of downsampling without the high computational cost associated with rerunning our experiments. We ran experiments using 5 and 60 minute execution time budgets on 16 datasets. Running these experiments from scratch takes up to 3 weeks of wall-clock time (as a result of job failures and restarts, a known challenge in AutoML [4, 6, 51]) on a well provisioned machine.

The dataset contains the configurations of all our experiments as well as general information such as dataset, sampling ratio, time per pipeline optimization etc. 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 the fitted pipelines (produced by TPOT) and the final pipeline associated with each search.

The dataset released totals 19GB of binary pipeline data. We also provide a CSV version of the files[6]. Our release also includes all code[7] necessary to rerun experiments, packaged in the form of an extendable experiment framework. This framework includes utilities for querying and visualizing experimental results as well.

7 THREATS TO VALIDITY

We now identify the main threats to the validity of our observations.

Our experiments use genetic programming (GP), specifically the AutoML tool TPOT [34], 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 constrain our observations here to the specific case of downsampling prior to GP. GP represents a common AutoML search procedure [19, 34, 39, 52], and it is notoriously hard to scale to large datasets [29, 34], 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 [21, 31], 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 procedure would be straightforward and provides observable benefits. Investigating the impact of adaptive downsampling on AutoML search procedures remains an open question for future work.

Our evaluation is focused on 16 datasets, which we collected to satisfy our "larger dataset" goal. The impact on performance may vary depending on the datasets used for evaluation. We believe that the difficulty associated with identifying suitably large datasets for evaluation further underscores the need for improving AutoML's scalability to increasingly large numbers of observations, and as such the performance evaluation we carry out provides initial grounds for further exploration.

[5]https://zenodo.org/record/4292740, DOI: 10.5281/zenodo.4292740

[6]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.

[7]https://github.com/ipa-lab/autoML-sampling-public

We identified possible factors resulting in the improved performance under downsampling. In particular, we highlighted the number of pipelines explored, their length, and the choice of predictors when downsampling as possible factors. Investigating additional factors underlying AutoML's performance remains an open question, and it is possible other aspects of a GP-based search can interact with downsampled datasets to improve performance. Our goal is to encourage additional exploration of such factors by releasing the data (including pipelines generated) produced by our experiments.

8 RELATED WORK

We discuss related work in three main areas: AutoML, heuristics for improving the runtime and scalability of search techniques, and benchmarking in the context of AutoML.

Interest in automated machine learning has steadily increased in recent years. Traditional techniques for hyperparameter optimization [24] such as grid-search and random search have given way to more sophisticated approaches such as Bayesian optimization [42], genetic programming [13], and meta-learning [17]. Beyond hyperparameter optimization, techniques are increasingly targeting the entire process of machine learning pipeline construction, from pipeline architecture to tuning. A wide range of AutoML systems have shown the potential of varying search techniques, including genetic programming (TPOT [34]), sequential-model-based optimization (Autosklearn [16]), ensemble-based random search (H2O [20]), program-analysis-based search (AL [12]), and multi-armed bandit optimization (MLBazaar [41]), among others. Alpine Meadow [40], OBOE [46], and TensorOBOE [47] are AutoML systems that aim to generate pipelines with short search budgets. In contrast to these systems, we are not proposing a new search technique or implementing a new AutoML system. Rather our contribution is centered on evaluating the impact of downsampling large datasets, a simple preprocessing step compatible with existing search techniques.

There is a vast literature on techniques to improve the runtime of iterative (search) routines in the context of machine learning. For example, early stopping [38] is commonly used to reduce the risk of overfitting by terminating the optimization procedure (often gradient descent) after a number of predefined steps or an increase in an external validation metric. Successive halving [25] has been successfully used to prune the number of candidate pipelines evaluated in an AutoML search [15], by repeatedly removing half of the candidate configurations as the execution budget is used. Rather than start each search with an empty set of candidates, warm-starting [26] initializes the set of candidate pipelines with pre-existing configurations that have performed well. Often warm-starting is paired with meta-learning [45], where the system predicts what candidate pipelines may perform well, given dataset characteristics and online performance feedback. 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 [23, 50]. TPOT-SH [35] introduced the use of successive-halving to improve the scalability of genetic-programming-based AutoML to larger datasets. As part of their successive halving, TPOT-SH starts with a downsample of the initial dataset and iteratively grows it. Our contributions in this paper are complementary to this body of work, as we focus on extensively evaluating the impact of downsampling across a larger number of datasets and providing an analysis of performance, runtimes, and pipeline composition.

Gijsbers et al [18] identified shortcomings of existing benchmarking work and developed a benchmarking suite (including code and datasets) to evaluate state of the art AutoML systems. Zöller and Huber [51] provide an extensive evaluation of state of the art 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. In contrast to these lines of work, we focus on the impact of downsampling large datasets prior to AutoML search, rather than evaluating AutoML systems in general.

9 CONCLUSION

We present an extensive evaluation of the impact of downsampling in machine learning (AutoML). 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 16 datasets and analyze predictive performance, runtime performance, and the types of pipelines generated. We find that downsampling allows the AutoML search to validate more candidate pipelines, produces pipelines with better predictive performance, and these pipelines tend to be slightly longer on average and with more variance in the predictor components used. 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. https://scikit-learn.org/stable/modules/tree.html#complexity

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

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

[4] [n.d.]. Possible speed up at large data sets Issue #87 EpistasisLab/tpot. https://github.com/EpistasisLab/tpot/issues/87

[5] [n.d.]. Speed and time budgets Issue #57 automl/auto-sklearn. 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] Benjamin Bengfort, Rebecca Bilbro, Nathan Danielsen, Larry Gray, Kristen McIntyre , Prema Roman, Zijie Poh, et al. 2018. *Yellowbrick.* https://doi.org/10.5281/zenodo.1206264

[8] 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

[9] 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

[10] 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

[11] José Cambronero, Jürgen Cito, and Martin Rinard. 2020. AMS: Generating AutoML Search Spaces from Weak Specifications. In *ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) (ESEC/FSE).*

[12] José P. Cambronero and Martin C. Rinard. 2019. AL: Autogenerating Supervised Learning Programs. *Proc. ACM Program. Lang.* 3, OOPSLA, Article 175 (Oct. 2019), 28 pages. https://doi.org/10.1145/3360601

[13] Peng-Wei Chen, Jung-Ying Wang, and Hahn-Ming Lee. 2004. Model selection of SVMs using GA approach, Vol. 3. 2035 – 2040 vol.3. https://doi.org/10.1109/IJCNN.2004.1380929

[14] Pedro Domingos and Michael Pazzani. 1996. Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier. In *Machine Learning*. Morgan Kaufmann, 105–112.

[15] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. 2018. Practical automated machine learning for the automl challenge 2018. In *International Workshop on Automatic Machine Learning at ICML*. 1189–1232.

[16] 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

[17] Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. 2014. Using Meta-Learning to Initialize Bayesian Optimization of Hyperparameters. In *Proceedings of the 2014 International Conference on Meta-Learning and Algorithm Selection - Volume 1201* (Prague, Czech Republic) *(MLAS'14)*. CEUR-WS.org, Aachen, DEU.

[18] Pieter Gijsbers, Erin LeDell, Janek Thomas, SÃĺbastien Poirier, Bernd Bischl, and Joaquin Vanschoren. 2019. An Open Source AutoML Benchmark. *arXiv preprint arXiv:1907.00909 [cs.LG]* (2019). https://arxiv.org/abs/1907.00909 Accepted at AutoML Workshop at ICML 2019.

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

[20] H2O.ai. 2017. *H2O AutoML*. http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html H2O version 3.30.0.1.

[21] Haibo He, Yang Bai, Edwardo Garcia, and Shutao Li. 2008. ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning. *Proceedings of the International Joint Conference on Neural Networks*, 1322 – 1328. https://doi.org/10.1109/IJCNN.2008.4633969

[22] 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

[23] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren (Eds.). 2018. *Automated Machine Learning: Methods, Systems, Challenges*. Springer. In press, available at http://automl.org/book.

[24] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. 2019. *Hyperparameter Optimization*. Springer International Publishing, Cham. https://doi.org/10.1007/978-3-030-05318-5_1

[25] 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

[26] Jungtaek Kim, Saehoon Kim, and S. Choi. 2017. Learning to Warm-Start Bayesian Hyperparameter Optimization. *arXiv: Machine Learning* (2017).

[27] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. 2017. Data scientists in software teams: State of the art and challenges. *IEEE Transactions on Software Engineering* 44, 11 (2017), 1024–1038.

[28] 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

[29] Trang Le, Weixuan Fu, and Jason Moore. 2019. Scaling tree-based automated machine learning to biomedical big data with a feature set selector. *Bioinformatics (Oxford, England)* 36 (06 2019). https://doi.org/10.1093/bioinformatics/btz470

[30] 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. *Bioinformatics* 36, 1 (2020), 250–256.

[31] 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

[32] F. Mosteller and J. W. Tukey. 1968. Data analysis, including statistics. In *Handbook of Social Psychology, Vol. 2*, G. Lindzey and E. Aronson (Eds.). Addison-Wesley.

[33] 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 (Aug. 2019), 4. https://doi.org/10.14778/3352063.3352116

[34] 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 Genetic and Evolutionary Computation Conference 2016* (Denver, Colorado, USA) *(GECCO '16)*. ACM, New York, NY, USA, 485–492. https://doi.org/10.1145/2908812.2908918

[35] Laurent Parmentier, Olivier Nicol, Laetitia Jourdan, and Marie-Eleonore Kessaci. 2019. TPOT-SH: a Faster Optimization Algorithm to Solve the AutoML Problem on Large Datasets. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 471–478.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in P ython. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[37] Fotis Psallidas, Yiwen Zhu, Bojan Karlas, Matteo Interlandi, Avrilia Floratou, Konstantinos Karanasos, Wentao Wu, Ce Zhang, Subru Krishnan, Carlo Curino, et al. 2019. Data Science through the looking glass and what we found there. *arXiv preprint arXiv:1912.09536* (2019).

[38] Garvesh Raskutti, Martin J Wainwright, and Bin Yu. 2014. Early stopping and non-parametric regression: an optimal data-dependent stopping rule. *The Journal of Machine Learning Research* 15, 1 (2014), 335–366.

[39] 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).

[40] 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* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 18. https://doi.org/10.1145/3299869.3319863

[41] 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 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. ACM, New York, NY, USA, 16. https://doi.org/10.1145/3318464.3386146

[42] 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

[43] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. 2013. OpenML: Networked Science in Machine Learning. *SIGKDD Explorations* 15, 2 (2013), 49–60. https://doi.org/10.1145/2641190.2641198

[44] 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*. 1–3.

[45] Ricardo Vilalta and Youssef Drissi. 2002. A perspective view and survey of meta-learning. *Artificial intelligence review* 18, 2 (2002), 77–95.

[46] 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 and Data Mining* (Anchorage, AK, USA) *(KDD '19)*. Association for Computing Machinery, New York, NY, USA, 11. https://doi.org/10.1145/3292500.3330909

[47] Chengrun Yang, Jicong Fan, Ziyang Wu, and Madeleine Udell. 2020. *AutoML Pipeline Selection: Efficiently Navigating the Combinatorial Space*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3394486.3403197

[48] 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

[49] Ting Yu, Simeon Simoff, and Tony Jan. 2010. VQSVM: A case study for incorporating prior domain knowledge into inductive machine learning. *Neurocomputing* 73, 13 (2010), 2614 – 2623. https://doi.org/10.1016/j.neucom.2010.05.007 Pattern Recognition in Bioinformatics Advances in Neural Control.

[50] Xueqiang Zeng and Gang Luo. 2017. Progressive Sampling-Based Bayesian Optimization for Efficient and Automatic Machine Learning Model Selection. *Health Information Science and Systems* 5 (12 2017). https://doi.org/10.1007/s13755-017-0023-z

[51] Marc-Andre Zöller and Marco Huber. 2019. Benchmark and Survey of Automated Machine Learning Frameworks. *arXiv: Learning* (2019). arXiv:1904.12054 [cs.LG]

[52] 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 2015 Annual Conference on Genetic and Evolutionary Computation* (Madrid, Spain) *(GECCO '15)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/2739480.2754694