SubStrat: A Subset-Based Optimization Strategy for Faster AutoML

Teddy Lazebnik
University College London
t.lazebnik@ucl.ac.uk

Amit Somech
Bar-Ilan University
somecha@cs.biu.ac.il

Abraham Itzhak Weinberg
Bar-Ilan University
abraham-itzhak.weinberg@biu.ac.il

ABSTRACT

Automated machine learning (AutoML) frameworks have become important tools in the data scientist's arsenal, as they dramatically reduce the manual work devoted to the construction of ML pipelines. Such frameworks intelligently search among millions of possible ML pipelines - typically containing feature engineering, model selection, and hyper parameters tuning steps - and finally output an optimal pipeline in terms of predictive accuracy.

However, when the dataset is large, each individual configuration takes longer to execute, therefore the overall AutoML running times become increasingly high.

To this end, we present SubStrat, an AutoML optimization strategy that tackles the data size, rather than configuration space. It wraps existing AutoML tools, and instead of executing them directly on the entire dataset, SubStrat uses a genetic-based algorithm to find a small yet representative data *subset* that preserves a particular characteristic of the full data. It then employs the AutoML tool on the small subset, and finally, it refines the resulting pipeline by executing a restricted, much shorter, AutoML process on the large dataset. Our experimental results, performed on three popular AutoML frameworks, Auto-Sklearn, TPOT, and H2O show that SubStrat reduces their running times by 76.3% (on average), with only a 4.15% average decrease in the accuracy of the resulting ML pipeline.

PVLDB Reference Format:
Teddy Lazebnik, Amit Somech, and Abraham Itzhak Weinberg. SubStrat: A Subset-Based Optimization Strategy for Faster AutoML. PVLDB, 16(4): 772 - 780, 2022.
doi:10.14778/3574245.3574261

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/teddy4445/SubStrat.

1 INTRODUCTION

Automated machine learning (AutoML) frameworks [17, 25] are becoming increasingly popular, as they facilitate the time-consuming, difficult task of developing a machine learning model, allowing even non-expert users to build accurate and robust models for their datasets at hand. To automatically develop a model, AutoML frameworks compare millions of ML pipeline configurations, and finally output the optimal pipeline, which typically includes data pre-processing, feature engineering, model selection, and hyper-parameters optimization [20].

Clearly, a naive brute-force search scanning all pipeline configurations is often infeasible [49], therefore different AutoML frameworks employ a variety of optimizations and search heuristics, such as Bayesian optimization [24], meta-learning [28], reinforcement learning [21], and genetic algorithms [40], in order to reduce the search space and the number of expensive pipeline executions.

However, when the training data is large – each pipeline execution takes longer to run, which can add up to hours of search time, even when using state-of-the-art AutoML frameworks [20]. While cloud-based AutoML services may suggest using stronger hardware (e.g., larger RAM, more GPUs) when working with large datasets – this results in much higher costs to the user. To this end, we present SubStrat, a new strategy for reducing AutoML computation costs, tackling the *data size* rather than the *configuration search space*. In a nutshell, instead of employing an existing AutoML tool directly on the entire dataset, we first compute a special *data subset* that preserves some characteristics of the original one. We then employ the AutoML tool over the subset (which is significantly faster), and last, we refine the resulting model configuration by executing a limited, shorter AutoML process over the original dataset.

The main advantage of our system is the compatibility with state-of-the-art existing AutoML tools – allowing data scientists to continue using their favorite frameworks while significantly reducing computation times. **Our experiments show that our system, when applied to Auto-Sklearn [16], TPOT [41], and H2O [30], three of the most popular AutoML frameworks, successfully reduced computation times by an average of 76.3%, while retaining 95.85% of the best model accuracy.** The main contribution of this work is as follows:
(1) We present a subset-based optimization strategy for AutoML, aimed at reducing AutoML computation costs with a minimal decrease in model performance.
(2) We introduce the general notion of measure-preserving data subsets and formulate their generation as an optimization problem. We then devise a *dataset entropy* measure and provide an effective genetic algorithm that is able to efficiently generate such entropy-preserving subsets.
(3) We implemented SubStrat and performed an extensive experimental evaluation over 18 datasets from various domains and shapes, and compared our results to 10 different baselines

1.1 Problem & Solution Overview

In a typical AutoML scenario, a data scientist builds an ML model for predicting the value of some target feature y in dataset D. Rather than manually constructing the model, the data scientist employs an AutoML tool A which intelligently scans multitudes of ML pipelines (i.e., feature engineering, model selection, and hyper-parameters

Proceedings of the VLDB Endowment, Vol. 16, No. 4 ISSN 2150-8097.
doi:10.14778/3574245.3574261

optimizations) and outputs a configuration which achieves the highest predictive performance[1]. We denote the application of AutoML tool A over dataset D to predict the target y by $A(D, y) \rightarrow M^{\star}$, where $M^{\star}$ is the best configuration that A could find.

As mentioned above, the larger the dataset, the higher the computational cost of the AutoML, since each candidate-pipeline takes longer to execute. Let $Time(M^{\star})$ be the time it takes A to generate $M^{\star}$, with final model accuracy, denoted by $Acc(M^{\star})$.

The goal of SubStrat, our subset-based optimization strategy, is to utilize a data *subset* in order to reduce AutoML computation times, while retaining the output model performance. Namely, to generate a model configuration M_{sub} s.t. $Time(M_{sub}) << Time(M^{\star})$ but $Acc(M_{sub}) \approx Acc(M^{\star})$. Importantly, $Time(M_{sub})$ includes the time it takes to discover the data subset.

Abstractly, given a dataset D of size $N \times M$ and a target feature y, SubStrat works in three steps (See Figure 1 for illustration):

(1) Find a small data subset d, of size $n \times m$, s.t. $n << N$ and $m << M$.
(2) Employ the AutoML tool over d, i.e., $A(d, y) \rightarrow M'$.
(3) Fine-tune the intermediate pipeline configuration M', by employing a restricted, faster instance of A back on D to obtain the final configuration M_{sub}.

Although it is quite obvious that employing AutoML on a fraction of the data takes less time, finding an adequate subset in a timely fashion is challenging. For instance, one could easily take a random subset of the data, and employ AutoML over it. Unfortunately, as further discussed in Section 4.3, using such random subsets in our framework reduces the final model accuracy by more than 27% compared to the accuracy of $M^{\star}$.

While 27% accuracy loss in ML is unanimously considered too low, there is an ongoing discussion about the acceptability of model accuracy for different applications in light of other objectives such as interpretability, and training time (See, e.g., [19, 26, 51]). Following these discussions, in this work we assume that *a decrease of more than 5% in accuracy is largely unacceptable for AutoML.*

Solution & Paper Outline. We begin by reviewing related work (Section 2). We then describe the architecture and methods of SubStrat in Section 3: we first introduce the notion of measure-preserving data subsets, which are designed to capture qualities of the original data (Section 3.1). Next, since finding the optimal measure-preserving subset is computationally infeasible, we formulate an optimization problem (Section 3.2), and present a genetic-based algorithm to efficiently solve it (Section 3.3). Last, we discuss our method for fine-tuning the intermediate model configuration, adapting it to fit the full dataset (Section 3.4). Our experimental evaluation is brought in Section 4, and we conclude in Section 5.

2 RELATED WORK

We survey related works in the field of AutoML as well as other works which aim to reduce datasets' size in different contexts.

Automated Machine Learning (AutoML). Existing AutoML can be roughly divided into two main categories: search-space optimizations and meta-learning solutions. Search space optimizations employ intelligent search strategies and heuristics to perform the configuration selection more efficiently on ad-hoc datasets. Example methods used are Bayesian optimization [14, 24], directed search [48, 53] and genetic programming [41]. Meta-learning solutions for AutoML [12, 21] take a different approach in order to produce an optimal pipeline configuration – by training, in advance, an ML model on a large corpus of datasets, then predicting the optimal configuration given the dataset and task at hand. This solution, while significantly faster, is more resource-intensive and assumes the user has a suitable collection of datasets to train on [48]. In particular, we note the works in [14, 15], describing the popular Auto-Sklearn system, which combines the two approaches and uses meta-learning with search optimizations to obtain further speedup.

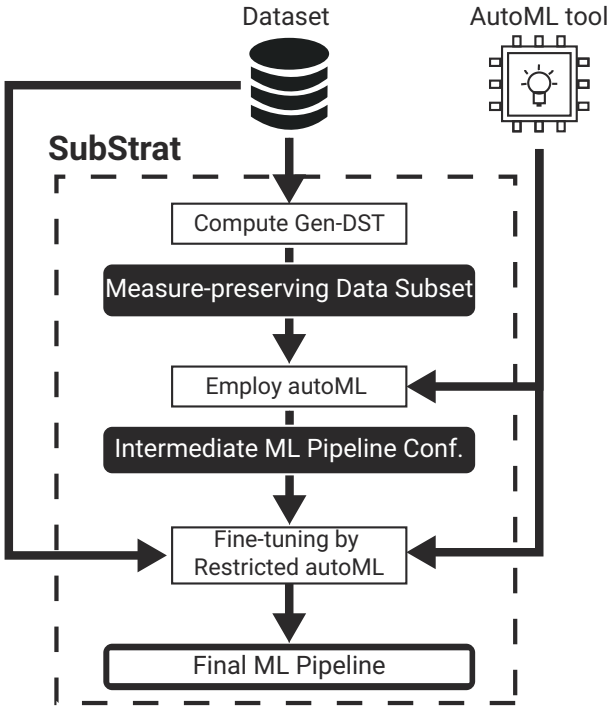

Figure 1: SubStrat Workflow

AutoML has also recently attracted the attention of the database community. First, industry works from Oracle [54] and Amazon [34] describe the challenges and solutions in deploying AutoML systems in their proprietary cloud environments. Similarly, Ease-ML [32] tackles the problem of executing multiple AutoML processes by different users, on the same server. SystemDS [5] and VolcanoML [33] take a different direction and suggest declarative languages and abstract building blocks for AutoML and data science components, facilitating the composition of DB-like execution plans.

Other works in our community focus on AutoML as a meta-learning task: Auto-Model[47] infers the model and parameters by mining research papers, Assassin [38] does so by efficiently mining previous experience, and KGpip [22] builds ML pipelines using a meta-learning model based on graph neural networks.

Differently from these works, which all suggest different, end-to-end AutoML tools, the goal of SubStrat is to improve the running

[1]Note that other AutoML objectives can be used, such as finding the most compact configuration [20], which is easier to deploy in a production environment.

time of *existing* AutoML tools. This is by running the majority of computation on a significantly smaller data subset, discovered by our genetic-based algorithm (See Section 3.3).

Coresets for Deep Learning. Several recent papers, e.g., [27, 36, 37], focus on finding data *coresets* to improve the training of deep learning models. Namely, given a set of training instances X, the goal is to find a subset $S \subset X$ which minimizes or maximizes some objective function. The latter can be, for example, training loss [36], denoising [37], and training robustness [27].

SubStrat differs from these methods in two key aspects: (1) since it is designed for *tabular* data, it jointly selects rows (samples) *and* columns. (2) The coresets methods mentioned above assume a particular network architecture, and take its weights as input. In contrast, SubStrat is particularly designed for AutoML, where the ML pipeline steps, including the predictive model and its hyper-parameters, are yet unknown.

Additional Data Reduction Methods. Reducing the dataset size is considered in previous work, where numerous methods are suggested for selecting either rows or columns (features).

Feature selection [7, 52] is a prominent step in many ML pipelines, where the goal is to reduce the number of input variables considered by the model. This is done in order to reduce training times as well as the complexity of the model. There is a plethora of research works (See [7] for a survey), roughly categorized as Filter-based techniques, that yield the Top-k features in terms of a given metric (e.g., Chi-Square, ANOVA, and Information-Gain) [52]; as well as Embedded and Wrapper methods, which directly utilize the ML models to determine the important features [8]. Selecting dataset rows is also widely considered in previous research, for either general-purpose methods that produce a norm-preserving sub-matrix [9] or for specific tasks such as search-results diversification [13] and faster generation of data visualizations [43]. The latter use dedicated, task-dependent utility definitions.

SubStrat is different from these works as it generates data subsets by selecting both rows *and* columns, hence solving a different, more complex optimization problem. We show in our experimental evaluation, that data subsets composed by separately applying feature-selection and row-sampling methods yield inferior results to the ones generated by SubStrat.

3 SOLUTION ARCHITECTURE

We next describe the components of SubStrat in more detail.

3.1 Measure-Preserving Data Subsets

As mentioned above, our goal is to find a subset of the original dataset which preserves a particular characteristic of the data.

Let D be a dataset of N rows and M columns. Denote its row and column indices by $R = 1, 2, \ldots, N$ and $C = 1, 2, \ldots, M$, respectively. Intuitively, a data subset (referred to as DST, for short) of a full dataset D is simply a subset of the rows of D, projected over a subset of the columns.

Definition 3.1 (Data Subset (DST)). Given a dataset D with row-indices R and column-indices C, a DST of size $n \times m$ is defined as follows. Let $[R]^n$ be the set of all n-subsets of R, i.e., $[R]^n = \{R' | R' \subseteq R \ \wedge \ |R| = n\}$, and $[C]^m$ be the set of all m-subsets of C. Then, given $r \in [R]^n$ and $c \in [C]^m$, the DST is defined by $D[r, c]$, i.e., the rows in D indicated in r, projected over the columns indicated in c. We also denote a dataset by d, when possible.

	Age	Gender	Flight distance	Delay [minutes]	Satisfied (target)
R_1	25	1	460	18	1
R_2	62	1	460	0	0
R_3	25	0	460	40	1
R_4	41	0	460	0	1
R_5	27	1	460	0	1
R_6	41	1	1061	0	0
R_7	20	0	1061	0	0
R_8	25	0	1061	51	0
R_9	13	0	1061	0	1
R_{10}	52	1	1061	0	1

Figure 2: An example dataset with two 5X3 subsets marked in green and red. d_{green} is a measure-preserving subset (w.r.t. the dataset-entropy measure), while d_{red} is not.

Last, since the target column is crucial for the AutoML process, our framework automatically inserts it into every DST.

Example 3.2. Consider the 10X5 dataset in Figure 2, taken from the *flight service review* dataset in our experiments (See Section 4.1). The green and red cells represent two different 5X3 data subsets: $d_{green} = D[(1, 2, 3, 6, 8)(1, 4, 5)]$ and $d_{red} = D[(4, 5, 7, 9, 10), (2, 3, 5)]$. Note that both contain the target column (the right-most column in Figure 2).

As will be shown in Section 4, simply using a random DST (in which the row and column subsets are chosen uniformly at random) in our solution induces a substantial decrease in the accuracy of the AutoML process. Our goal is therefore to find a more *representative* DST, that preserves some characteristic of the original dataset. Let $F : \mathbb{D} \rightarrow \mathbb{R}$ be a *dataset measure* which takes a dataset as input and evaluates a characteristic of it by a real number.

We define a measure-preserving DST as follows.

Definition 3.3 (Measure-Preserving DST). Given a dataset D, a DST $d = D[r, c]$, and a dataset-measure $F : \mathbb{D} \rightarrow \mathbb{R}$, we call a DST d *measure-preserving* if $F(d) \approx F(D)$.

While any measure that evaluates a characteristic of the data may be applicable, in this work we use a *dataset entropy* function, which assesses the "amount of information" conveyed in the data. In our context, we define dataset entropy as follows.

Definition 3.4 (Dataset Entropy). Given dataset D of size $N \times M$, Let D_{ij} be the value in row i and column j.

$$H(D) = \frac{\sum_{j=1}^{M} \left(\sum_{i=1}^{N} P_j(D_{ij}) \cdot Log_2 P(D_{ij}) \right)}{M}$$

Where $P_j(D_{ij})$ is a probability function corresponding to the frequency of the value in D_{ij} w.r.t. Column j. For $D_{ij} = v$:

$$P_j(v) = \frac{\sum_{k=1}^{N} I[D_{kj} = v]}{N}$$

Example 3.5. Consider again the dataset and two subsets depicted in Figure 2. Calculating the dataset entropy we obtain:

$$H(D) = \frac{2.65 + 1 + 1 + 1.4 + 0.97}{5} = 1.395$$

We indeed observe that D contains two columns with high entropy ('Age' and 'Delay'). These columns are also selected in the green DST, which obtains the score:

$$H(d_{green}) = \frac{1.37 + 1.92 + 0.97}{3} = 1.42$$

d_{green} is the 5X3 DST which obtains the closest dataset-entropy score to D. However, the red DST, which contains low-entropy columns, obtains a lower score of $H(d_{red}) = 0.89$. Hence, d_{green} is considered a measure-preserving DST, whereas d_{red} is not.

Suitability of entropy & alternative dataset measures. Entropy-based measures such as KL-divergence, cross-entropy and information gain are widely used to characterize data (e.g., in data profiling [1] and meta-learning [6]). Such measures determine the "closeness" of two data distributions (which is widely used for variational inference [3], regression and classification loss [18], and more).

In our context, the dataset-entropy measure has three main advantages: (1) It is a non-parametric measure, having no prior assumption on the data distributions. This allows SubStrat to support a wide range of datasets. (2) Our suggested measure focuses on the data distributions rather than the values themselves (as is common for distance metrics such as Euclidean distance or Manhattan distance). This is more suitable to our setting, where the size of the compared arrays greatly differ. (3) A low entropy difference between distributions P and Q implies that a model based on P can very well predict Q. This property is well suited for the downstream task in our setting, which is choosing an ML pipeline for the original dataset, based on computations performed on the subset.

Last, note that while dataset-entropy worked well in our experiments (see Section 4.5), our optimization algorithm, as described below, is generic and can take other possible dataset measures as input (e.g., p-norm, mean-correlation, and coefficient of variation).

3.2 DST as an Optimization Problem

Ideally, we would like to find the best-preserving DST for a dataset D. Namely, the best DST of size $n \times m$ can be found by:

$$\underset{r \in [R]^n, c \in [C]^m}{\operatorname{argmin}} \; |F(D[r,c]) - F(D)|$$

If n and m are small, then finding the best-preserving DST can be done in $O(N^n \cdot M^m)$ time, by a brute-force search that traverses through all possible DST of size $n \times m$. Clearly, this becomes infeasible for large datasets or when a larger DST is needed.

We therefore define an optimization problem, which is to minimize the difference between the DST and the original dataset, i.e.,

$$\mathcal{L}(r,c) = |F(D[r,c]) - F(D)|$$

Note that while numerous methods and algorithms can be used to minimize $\mathcal{L}(r,c)$ (See Section 4.2), we must use an approach that also obtains short convergence times. Otherwise, the optimization process will take too long, hence diminishing the efficacy of our overall solution in reducing AutoML running times.

Algorithm 1 Gen-DST

Input: dataset (D), dataset-measure (F), DST size (n, m)
Output: data subset (d)
$P_0 \Leftarrow$ generate (ϕ) candidates in random
$best\_dst \Leftarrow argmax_{G \in P_0} F(G, D)$
for generation $i \in [1, \ldots, \psi]$ **do**
  $P_i \Leftarrow Mutation\_Operator(P_i, \xi, p_{rc})$
  $P_i \Leftarrow Crossover\_Operator(P_i, p_m)$
  $P_{i+1} \Leftarrow Selection\_Operator(P_i, \alpha)$
  if $\max_{G \in P_{i+1}} F(G, D) > F(best\_dst, D)$ **then**
    $best\_dst \Leftarrow argmax_{G \in P_{i+1}} F(G, D)$
  end if
end for
return $d := D[best\_dst(r),\ best\_dst(c)]$

3.3 A Genetic-Based Algorithm for Finding DST

Our framework employs a Genetic Algorithm (GA), a well-known and commonly-used meta-heuristic search method, based on the biological theory of evolution [23]. Briefly, GA simulates *evolution* through a natural selection process: First, a population of ϕ candidate-solutions, each comprising a set of properties, referred to as *genes* is selected at random. The algorithm then iteratively mutates and alters the genes in order to create "better" solutions w.r.t. a *fitness function*, which corresponds to the optimization objective. In particular, at each *generation* (i.e., iteration), the GA typically performs several stochastic operators [4, 10]: (1) a *mutation* operator which induces random noise into the genes of a candidate-solution, (2) a *cross-over* operator which combines the genes of two candidate-solutions, and (3) a *selection* operator which refines the population of the next generation, by keeping fitter candidate-solutions with higher probability than less-fitting solutions. Finally, after a number of generations (ψ - chosen according to predefined stopping criteria), the fittest candidate-solution is selected as the output of the GA algorithm.

We next describe Gen-DST, our genetic-based algorithm for finding measure-preserving DSTs. Importantly, Gen-DST jointly selects rows *and* columns, and therefore uses dedicated genetic representation and operators, adapted from the standard ones as appear in [4, 10, 23]. The adaptations made in Gen-DST ensure a balanced mutation and combination of candidate solutions, w.r.t. both rows and columns, as described below.

Genetic representation of candidate-DSTs. The genetic representation of a candidate-DST, denoted G, comprises of $n + m$ chromosomes: n row-chromosomes, that correspond to n row indices of dataset D, and m column-chromosomes, that correspond to m column-indices. More formally, $G := (r, c),\ r \in [R]^n, c \in [C]^m$, where R and C denote the row and column indices of dataset D.

Fitness Function. The fitness function $f(G)$ is simply the negative loss of the DST-candidate $G = (r, c)$, Namely,

$$f(G) := -\mathcal{L}(r,c) = -|F(D[r,c]) - F(D)|$$

Gen-DST Workflow & Operators. Gen-DST, as depicted in Algorithm 1, works as follows. First, an initial population P of candidate DSTs is randomly generated, s.t. each candidate-DST $G(r, c)$ contains the target column y, i.e. $t \subset c$. Then, for each generation i, we perform (1) mutation, (2) cross-over and (3) selection, in order to generate the next-generation population P_{i+1}:

(1) Mutation. The mutation operator is stochastically employed, for each candidate-solution $G = (r, c)$ in the population P_i with probability ξ. First, we randomly decide if to mutate rows or columns w.r.t. probability p_{rc}, which we define to be $N/(N+M)$ (for choosing rows). If, for example, a row-mutation is decided upon, we randomly replace one of the row-indices in r. Namely, we mutate G and form G' s.t.

$$G(r', c),\ r' \in [R]^n\ \wedge\ |r \cap r'| = n - 1$$

A similar process is performed for column mutations, only that the target column y cannot be mutated.

(2) Cross-Over. Cross-over is employed for two candidate-DSTs $G_a = (r_a, c_a)$ and $G_b = (r_b, c_b)$ in population P_i with the goal of creating two next-generation DSTs, G_{ab} and G_{ba}. We begin by selecting whether to cross rows or columns (similar to the mutation operator), with probability p_{rc}. Then, assuming (w.l.o.g.) that columns cross-ever is selected, we randomly choose a split-size $1 < s < m$, and use it to split both c_a and c_b, each to two random subsets - one of size s and one of size $m - s$, i.e., $c_a = c_a^s \cup c_a^{m-s}$ and $c_b = c_b^s \cup c_b^{m-s}$. The cross-over then unifies complementing subsets from a and b, creating c_{ab} and c_{ba}:

$$c_{ab} = c_a^s \cup c_b^{m-s},\ c_{ba} = c_b^s \cup c_a^{m-s}$$

Finally, the next-generation DSTs are set as $G_{ab} = (r, c_{ab})$ and $G_{ba} = (r, c_{ba})$[2]. The cross-over operation is performed over the entire population P_i: P_i is first split into disjointed pairs of candidate-DSTs, then the cross-over is performed on each such pair.

(3) Selection. Last, after employing mutation and cross-over, we employ the selection operator which forms the next-generation population P_{i+1}. We use the *royalty tournament* operator [4], which selects the best $\alpha \cdot \phi$ candidate-DSTs from P_i according to the fitness function $f(G)$. The rest of the $\phi(1-\alpha)$ DSTs are sampled (with repetitions) according to their fitness score, i.e., with probability:

$$p_{select}(G) = \frac{f(G)}{\sum_{G' \in P_i} f(G')}$$

Last, the stopping criterion of Gen-DST is either reaching a predefined limit on the generations number, or a convergence criterion that stops the execution when the fittest DST of population P_{i+1} is not significantly better than the fittest solution in P_i. In this case, we return the DST that obtained the highest fitness score, over all previous generations.

3.4 Fine-Tuning the Intermediate Configuration

Gen-DST generates a DST d, which is then given as input to the Auto-ML tool A, instead of the full dataset D, which in turn output an intermediate ML pipeline configuration M'.

[2]In case the size of c_{ab} or c_{ba} is smaller than m, we insert the required amount of columns at random, while also making sure the target column y is contained in both.

The final step performed by SubStrat is to fine-tune the intermediate configuration M' by a restricted execution of A on the full dataset D. The restriction of the process is twofold: (1) We restrict the configuration search space by forcing it to use the same ML model discovered in M'. (2) We further restrict the process by time, using a stopping condition on the predictive accuracy of the current best pipeline. In our implementation, if the derivative of the obtained accuracy is less than 0.02 for three consecutive steps, the fine-tuning process is terminated.

As shown in our experimental results in Section 4.3, this step slightly increases the running times of SubStrat, but boosts the relative accuracy (compared to the full AutoML) by about 6%.

4 EXPERIMENTS

We conducted a thorough experimental study with the goal of examining the effectiveness of SubStrat in reducing the running times of existing AutoML tools while retaining the accuracy of their output ML pipelines.

4.1 Setup & Methodology

Experimental Framework & Methodology. Given an input dataset and a target feature, we first directly employ an AutoML tool and obtain its output ML pipeline configuration.

We record both the running time and the accuracy of the resulting model, which serve as our primary baseline, denoted Full-AutoML. We then examine whether our subset-based strategy can indeed reduce AutoML running times, and still generate ML pipelines as accurately as Full-AutoML. To generate the data subsets, we used Gen-DST as well as 10 other baselines (see below). For each instance, we compute the relative running time (including the generation of the subset) and accuracy w.r.t Full-AutoML. We report the following metrics: *time-reduction*, which indicates how much time was saved:

$$Time\text{-}Reduction = 1 - \frac{Time(M_{sub})}{Time(M^{\star})}$$

We also report the *relative accuracy*, indicating the proportion of accuracy of Full-AutoML that was successfully retained:

$$Relative\text{-}Accuracy = \frac{Acc(M_{sub})}{Acc(M^{\star})}$$

Datasets. We used 18 public datasets from Kaggle [50], UCI Machine Learning Repository [46], and OpenML [42]. The datasets, as depicted in Table 1, are of different shapes that can be categorized as follows: (1) *standard*, containing several thousand rows and a few dozen columns, as most datasets in the popular OpenML-C18 benchmark [2]; (2) *Long*, containing more than 1M rows and a dozen columns; (3) *HighDim* are high dimensional dataset, with several hundred columns; (4) *HighDim-Wide* are particularly wide datasets that contain up to 11K columns, and have a columns-to-rows ratio of at least 70%. Links to the full datasets can be found in our code repository [45].

Auto-ML methods. We evaluated SubStrat using Auto-Sklearn, TPOT, and H2O, three highly-popular AutoML tools. The tools have a substantially different underlying technology: **(1) Auto-Sklearn[14, 15]**, an industry-standard tool that uses Bayesian optimization methods together with meta-learning. It works on top of the Python Scikit-Learn library [44], and generates an ML pipeline

Table 1: Dataset descriptions and properties

Symbol	Domain	#Rows	#Cols	#Cells
Standard-1	Heart disease	79K	7	0.55M
Standard-2	Flight service review	130K	23	2.98M
Standard-3	Signal processing	10K	5	70K
Standard-4	Air quality	57K	7	0.40M
Standard-5	Bike demand	17K	9	150K
Standard-6	Car insurance	10K	18	180K
Standard-7	Lead generation form	7K	15	100K
Standard-8	Mushroom classif.	8K	23	180K
Long-1	Criteo Click Predict.	2M	12	24M
Long-2	Poker matches	1M	12	12M
HighDim-1	KDD 98	82K	478	39.19M
HighDim-2	Myocardial infarction	1.7K	123	210K
HighDim-3	KDD Cup 2009	50K	231	11.55M
HighDim-4	Philippine	6K	309	1.80M
HighDim-5	Isolet	8K	614	4.78M
HighDim-W-1	AP Breast Colon	630	11K	6.88M
HighDim-W-2	Micro-mass	571	1301	0.74M
HighDim-W-3	Gisette	7K	5K	35M

configuration comprising of feature prepossessing, model selection, and hyper-parameters optimization. **(2) Tree-Based Pipeline Optimization Tool (TPOT) [39]**, a tool that also utilizes Scikit-Learn but uses a genetic programming approach to explore the configuration search space. **(3) H2O [30]** uses a narrower configuration search space but focuses on *stacked ensembles* of models.

Implementation & Hardware. SubStrat and the rest of the baseline algorithms were implemented in Python 3. Our source code is fully available in [45]. We ran the experiments on an Ubuntu Server with an Intel Core i7-9700K CPU and 64GB RAM.

4.2 Baseline Methods

We implemented 10 different baselines in 6 different categories (A-F). To clarify the scope of comparison, recall again that AutoML methods require the raw data as input, therefore any approach that *alters* the data (e.g., PCA, embedding) is inapplicable. Also, we only compare SubStrat to other methods for reducing the data size rather than the configuration space, as the latter is performed by the chosen AutoML tool. The baselines in categories A-F were therefore devised to answer the following questions:

i. Can a trivial, random DST perform well enough? (Category A)
ii. Can we use different, existing optimizations for finding measure-preserving DSTs? (Categories A-C)
iii. Can we generate effective DSTs using existing techniques for row sampling and column selection? (Categories D-E)
iv. Can SubStrat obtain good performance without the fine-tuning phase? (Category F)

A. Monte-Carlo Search. We began with a simple random search technique, which given a predefined time/iteration budget B, randomly generates DSTs, calculates their measure-preserving loss (as defined in Section 3.2), and at the end of the time limit (or max iteration) returns the DST that obtained the minimal loss. We use three instances with different budgets: (1) *MC-100*, which examines 100 DSTs, (2) *MC-100K*, designed to have approximately the same running- times as Gen-DST, allowing it to compare about 100K DSTs. Last, to demonstrate the optimization challenge of finding DSTs, we also examined (3) *MC-24H*, which stops after 24 hours. While the latter cannot improve the running time of AutoML, we examine its performance only in terms of relative accuracy.

B. Multi-Arm Bandit. Additionally, we implemented a more sophisticated *Multi-Arm Bandit (MAB)* baseline, which also attempts to find a measure-preserving DST. MAB is a well-known search framework that balances exploration and exploitation within the search space[31]. We implemented the MAB baseline by formulating two types of arms: row-arms and column-arms. At each round, the model needs to choose n rows and m columns, and balance the exploration/exploitation of its choices using an ϵ-greedy policy.

C. Greedy Selection. Another possible optimization is to use a greedy selection process. Since the loss is dependent on both the rows and the columns, we used two instances of the algorithm: (1) *(*Greedy-Seq) which first selects n rows and then m columns. The n rows are found in a greedy manner, s.t. at each step we add to the DST d a new row from D which locally diminishes the local loss of d (while using all columns in D). In the second step we choose m columns in a similar manner, only that the loss is computed w.r.t. the rows already found in the row-selection phase. We also implemented (2) *Greedy-Mult* which attempts to greedily select both a row and a column at each step.

D. Clustering-Based Approach. This method does not attempt to find measure-preserving DSTs, yet tries to select representative rows and columns using clustering. The *KM Baseline* first clusters the rows in D into n clusters, by employing K-means clustering [35]. Then, to choose n representative rows, we pick the ones that are the closest to each of the n cluster centroids. To select m columns, we do the same process by applying K-Means on the column vectors.

E. Information-Gain (Feature Selection). Information-gain (IG) is a commonly used technique for feature selection [29]. Similarly to our dataset-entropy measure, it is also based on entropy calculations, where the goal is to select m columns that have the highest IG, w.r.t. the target feature y. Intuitively, these are the columns that provide the most "information" about y. As IG can only be used for feature selection, we implemented two different baselines here: (1) *IG-Rand* which selects columns using *IG* and chooses the rows at random, and (2) *IG-KM*, which uses IG for column selection, and the KM baseline to choose the rows.

F. SubStrat Without Fine-Tune. Last, we examine the importance of the fine-tuning phase, by using a limited version of SubStrat, denoted SubStrat-NF. This version outputs the intermediate configuration M' – resulted by applying the AutoML tool only on the DST generated by Gen-DST, without employing fine tuning on the full dataset.

Baselines Default Configurations. For each dataset shape category (as depicted in Table 1) we performed a grid search, optimizing on the harmonic mean of time reduction and relative accuracy. The DST size grid used for all baselines is $\{sqrt, ln\} \cup \{0.05i \cdot n\}_{i=1}^{19}$ for the rows and columns (replacing n with m). We further varied the following parameters in SubStrat: ψ (num. of generations), ranged in (30,40,45); ϕ (population size) ranged in (200,250,300,350). For the baselines, MAB has an additional hyper-parameter (other than the DST size) of ϵ (varied from 0.001 to 0.05, in intervals of 0.005).

Table 2: Mean Time Reduction and Relative Accuracy (Rel. Acc.) Scores

Algorithm	AutoSklearn		TPOT		H2O	
	Time Reduction	Rel. Acc.	Time Reduction	Rel. Acc.	Time Reduction	Rel. Acc.
SubStrat	76.4 ± 10.38%	96.11 ± 2.11%	75.82 ± 8.68%	96.29 ± 1.94%	76.64 ± 8.69%	95.14 ± 1.30%
IG-KM	80.24 ± 8.68%	92.36 ± 2.56%	77.55 ± 7.46%	92.66 ± 2.01%	79.28 ± 7.57%	92.86 ± 2.57%
MAB	78.78 ± 7.3%	93.08 ± 2.96%	78.16 ± 6.42%	93.24 ± 2.18%	78.12 ± 7.08%	92.84 ± 2.62%
SubStrat-NF	87.24 ± 7.62%	89.81 ± 1.74%	86.13 ± 9.68%	89.69 ± 2.79%	85.92 ± 8.12%	90.19 ± 3.44%
IG-Rand	77.04 ± 7.84%	91.56 ± 2.36%	78.03 ± 6.38%	90.72 ± 2.08%	75.37 ± 4.89%	89.37 ± 2.77%
KM	82.73 ± 5.23%	82.09 ± 3.19%	76.23 ± 5.21%	82.23 ± 2.26%	76.44 ± 6.49%	82.99 ± 2.48%
MC-100K	84.5 ± 0.31%	68.46 ± 5.42%	84.93 ± 0.38%	69.85 ± 4.87%	84.02 ± 0.33%	70.01 ± 5.05%
MC-100	98.06 ± 0.03%	38.18 ± 7.21%	97.24 ± 0.05%	40.53 ± 8.18%	97.57 ± 0.05%	40.8 ± 7.72%

For SubStrat, the default DST size configurations are as follows: *Standard*: $(\sqrt{N}, 0.25M)$; *Long*: $(0.05N, 0.9M)$; *HighDim*: $(0.05N, 0.1M)$; and *HighDim-Wide*: $(0.9N, 0.05M)$. See [45] for the full configuration.

4.3 Overall Baseline Comparison Results

Table 2 depicts the time reduction and relative accuracy scores obtain by each baseline approach (for both, higher is better). Each baseline was executed 5 times on each dataset, and the table lists the mean and std. value across all 18 datasets and executions. Baselines that did not reduce the AutoML times (i.e., their execution took longer than Full-AutoML) are omitted from the table.

First, see that *the only method obtaining a higher relative accuracy than 95% is SubStrat, with mean scores of 96.11% for Auto-Sklearn 96.29% for TPOT, and 95.14% for H2O, while reducing their original running times by 76.4%, 75.82%, and 76.64%, respectively.*

As mentioned above, the DST size was selected a priori, w.r.t. the input dataset shape category. We further calculated the loss in performance compared to the best configuration (fitted individually for each dataset). If fitted individually, SubStrat obtains slightly better performance, 1.88 points more in time reduction and 0.11 points better in relative accuracy. For example, in AutoSKlearn it obtains an average of 78.39% in time reduction and 96.17% in relative accuracy, compared to the default configuration which obtains 76.4% and 96.11% (as depicted in Table 2). Similar differences were observed for the other baselines.

We next analyze the rest of the baselines' results w.r.t. the categories and questions, as detailed in Section 4.2.

(i) The simple Monte Carlo baseline MC-100, was naturally the fastest, yet obtained poor accuracy results (~70%). The slower instance MC-100K, which chooses the best out of 100K random DSTs obtained slightly better accuracy (~80%). *This shows that a trivial random sampling approach is highly ineffective, as the ML pipelines result in poor accuracy scores.* (ii) All baselines in Categories A-C (i.e., MC, MAB, and Greedy baselines) attempt to generate entropy-preserving DSTs. As expected, the more sophisticated MAB surpasses the simple MC and greedy approaches with relative-accuracy scores of 93.08%, 93.24%, and 92.84% for AutoSklearn, TPOT, and H2O (MC and greedy scores are omitted, as they took longer than 24 hours to run). *However, SubStrat significantly outperforms all these alternative optimizations, proving the efficiency of our genetic-based Gen-DST algorithm.* Moreover, the only alternative optimization that achieved relative accuracy higher than 95% was MC-24H, which was able, after 24 hours of execution to obtain a relative accuracy of 95.95% (which is still slightly lower than SubStrat).

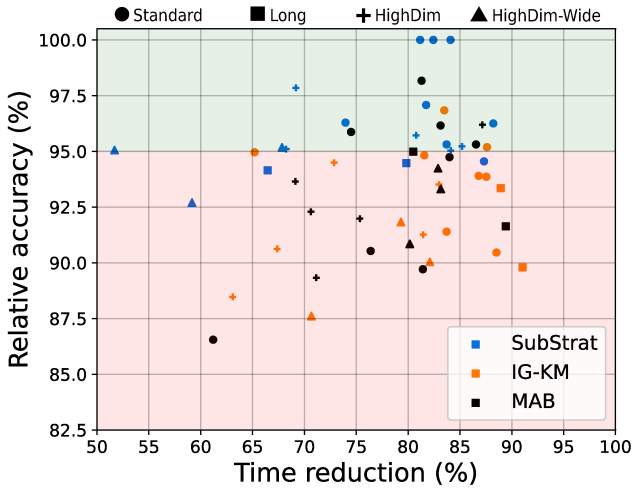

Figure 3: Per-Dataset Performance

(iii) Next, we inspect the performance baselines that combine existing techniques for row sampling and column selections (Categories D and E). The best results in these categories were obtained by IG-KM, combining IG feature selection and clustering-based row selection. While its time-reduction scores are slightly better (80.24%, 77.55%, and 79.28%) see that its relative-accuracy score are inferior to SubStrat, and are similar to the ones obtained by MAB. Several reasons may explain the sub-optimal results of IG-KM. First, it sequentially chooses rows and columns, compared to SubStrat which selects them jointly. Second, the DST generated by SubStrat is unbiased, representing the entire dataset rather than the portion of columns that are (individually) correlated with the target attribute (as done in IG). For the task of AutoML optimization, which also includes feature-selection steps, it appears that the unbiased column selection is more suitable.

(iv) Last, we validate the effectiveness of the fine-tuning phase. See that SubStrat-NF – which does not perform the fine-tuning on the full dataset – is indeed faster than SubStrat, but its relative accuracy scores drop by 6.29, 6.6, and 4.95 points, for AutoSklearn, TPOT, and H2O, resp. *This proves the importance of fine-tuning the configuration on the full dataset.* Note that similar reductions

Table 3: Performance Break-down for Each Dataset Shape Category

Dataset type	SubStrat		IG-KM		MAB	
	Time Reduction	Rel. Acc.	Time Reduction	Rel. Acc.	Time Reduction	Rel. Acc.
Standard	82.83 ± 4.38%	97.44 ± 2.25%	83.05 ± 7.62%	93.93 ± 2.08%	78.56 ± 8.04%	93.38 ± 3.98%
Long	73.15 ± 9.46%	94.31 ± 0.23%	89.99 ± 1.48%	91.57 ± 2.52%	84.96 ± 6.31%	93.32 ± 2.37%
HighDim	77.46 ± 8.25%	95.79 ± 1.18%	73.55 ± 8.66%	91.67 ± 2.39%	74.68 ± 7.35%	92.69 ± 2.51%
HighDim-Wide	59.63 ± 8.19%	94.31 ± 1.4%	77.36 ± 5.96%	89.83 ± 2.12%	82.07 ± 1.66%	92.8 ± 1.75%

were observed for the rest of the baselines when removing the fine-tuning phase (results omitted for space constraints).

Last, to shed more light on the performance for each dataset, we illustrate the individual relative-accuracy and time-reduction scores in Figure 3 for SubStrat and the top-two performing baselines: IG-KM and MAB. Each point in the plot depicts the performance of a baseline for a given dataset when using Auto-Sklearn (similar results were obtained for TPOT and H2O, thus omitted). The marker shape denotes the dataset shape, as depicted in Table 1. The light-red zone highlights all instances with less than 95% accuracy. See that SubStrat, while having some variation in time-reduction, surpasses the 95% accuracy bar in 14 out of 18 datasets. This is significantly better than the other baselines, compared to IG-KM which achieves 2/18, and MAB with 5/18.

4.4 Scalability Analysis

We next discuss the performance of SubStrat and the top-two performing baselines, IK-KM and MAB, when operated on the different dataset shape categories. For lack of space, we focus here on the results for AutoSklearn and report that similar trends were observed for TPOT and H2O as well.

Table 3 depicts the average time reduction and relative accuracy for each dataset shape category. First, see that for the *standard* datasets, while all baselines obtain about 80% time reduction, SubStrat obtains a much higher relative accuracy, 3.51 and 4.12 points higher than IG-KM and MAB. As for *long* datasets, SubStrat still outperforms the baselines in relative accuracy, while IG-KM and MAB provide better time-reduction scores. Next, regarding *HighDim* datasets, see that SubStrat outperforms the baselines in *both* time reduction and relative accuracy. However, see that the biggest performance difference for SubStrat occurs for the *HighDim-Wide* datasets (which contain 5K-11K columns) where it obtains a time reduction of 59.63%. The increase in computation for SubStrat is due to the fact that in each generation, it takes Gen-DST longer to compute the dataset-entropy measure (as the latter is computed for each column, for each DST-candidate). However, see that SubStrat still outperforms the baselines w.r.t. relative accuracy, with a 4.5 and 1.5 points difference from IG-KM and MAB.

The latter results show that while taking more time, *SubStrat is still able to produce satisfying results for these difficult, high-dimensional datasets.* We expect that time reduction can be further optimized in future work by employing solutions for caching and approximations for entropy calculations [11].

4.5 Effectiveness of Dataset Entropy

Last, we examine the usefulness of the dataset-entropy measure (Definition 3.4). To do so, we performed a step-by-step execution of Gen-DST on the first dataset from each category. Namely, we stopped the execution of Gen-DST after each generation i and continued the workflow of SubStrat (i.e, running the AutoML tool on d_i, then fine-tuning). We then logged the dataset entropy and the relative accuracy for each d_i, as appears in Figure 4 (dataset entropy is scaled, reported as $H(d_i)/H(D)$). For all categories, there is a strong positive correlation between the dataset entropy of d_i and the corresponding relative accuracy score (Pearson correlation, on all 18 datasets, is 0.917). *This result demonstrates the effectiveness of the dataset-entropy measure in finding useful DSTs for AutoML.*

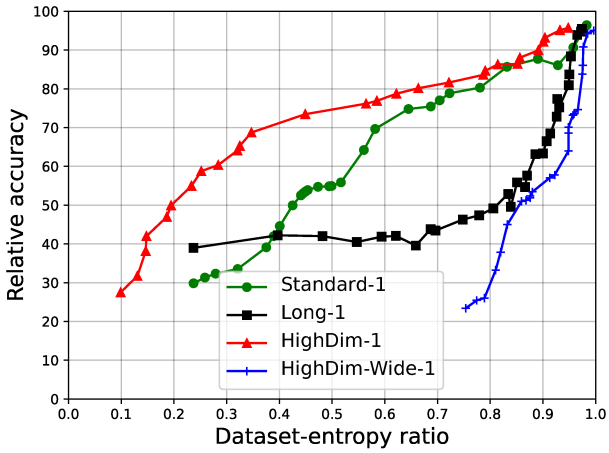

Figure 4: Dataset-entropy of DSTs and their Relative Acc.

5 CONCLUSION & FUTURE WORK

We present SubStrat, a subset-based optimization strategy for AutoML. To the best of our knowledge, SubStrat is the first work that suggest externally reducing the data size rather than the configuration search space, for a given input AutoML tool. This allows data scientists to keep using their favorite AutoML libraries, while significantly reducing their running times and computational costs.

Our experimental results show the inadequacy of simple solutions such as applying AutoML on a random sample, or employing techniques for row sampling and feature selection. In fact, we show that SubStrat, which obtained 95.85% average relative accuracy, while reducing 76.3% of the computational costs, was the only approach that was able to exceed 95% relative accuracy.

In future work, we intend to investigate the application of more sophisticated dataset measures, as well as additional fine-tuning strategies. Furthermore, reducing entropy calculation costs and applying meta-learning for inferring the DST size is also a promising direction for future research.

REFERENCES

[1] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. 2015. Profiling relational data: a survey. *The VLDB Journal* 24, 4 (2015), 557–581.

[2] Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Frank Hutter, Michel Lang, Rafael G Mantovani, Jan N van Rijn, and Joaquin Vanschoren. 2017. Openml benchmarking suites. *arXiv preprint arXiv:1708.03731* (2017).

[3] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. 2017. Variational inference: A review for statisticians. *Journal of the American statistical Association* 112, 518 (2017), 859–877.

[4] Z. W. Bo, L. Z. Hua, and Z. G. Yu. 2006. Optimization of process route by genetic algorithms. *Robotics and Computer-Integrated Manufacturing* 22 (2006), 180–188.

[5] Matthias Boehm, Iulian Antonov, Sebastian Baunsgaard, Mark Dokter, Robert Ginthör, Kevin Innerebner, Florijan Klezin, Stefanie Lindstaedt, Arnab Phani, Benjamin Rath, et al. 2020. SystemDS: A declarative machine learning system for the end-to-end data science lifecycle. *The Conference on Innovative Data Systems Research (CIDR)*.

[6] Ciro Castiello, Giovanna Castellano, and Anna Maria Fanelli. 2005. Meta-data: Characterization of input features for meta-learning. In *International Conference on Modeling Decisions for Artificial Intelligence*. Springer, 457–468.

[7] Girish Chandrashekar and Ferat Sahin. 2014. A survey on feature selection methods. *Computers & Electrical Engineering* 40, 1 (2014), 16–28.

[8] Andrzej Cichocki. 2014. Era of big data processing: A new approach via tensor networks and tensor decompositions. *arXiv preprint arXiv:1403.2048* (2014).

[9] Michael B Cohen and Richard Peng. 2015. Lp row sampling by lewis weights. In *Proceedings of the 47th annual ACM symposium on Theory of computing*. 183–192.

[10] L. Davis. 1985. Applying adaptive algorithms to epistatic domains. *Proceedings of the international joint conference on artificial intelligence* (1985), 162–164.

[11] Alfonso Delgado-Bonal and Alexander Marshak. 2019. Approximate entropy and sample entropy: A comprehensive tutorial. *Entropy* 21, 6 (2019), 541.

[12] Iddo Drori, Yamuna Krishnamurthy, Remi Rampin, Raoni de Paula Lourenco, Jorge Piazentin Ono, Kyunghyun Cho, Claudio Silva, and Juliana Freire. 2021. AlphaD3M: Machine learning pipeline synthesis. *arXiv* (2021).

[13] Marina Drosou and Evaggelia Pitoura. 2010. Search result diversification. *ACM SIGMOD Record* 39, 1 (2010), 41–47.

[14] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. 2020. Auto-sklearn 2.0: Hands-free automl via meta-learning. *arXiv preprint arXiv:2007.04074* (2020).

[15] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and robust automated machine learning. *Advances in neural information processing systems* 28 (2015).

[16] M. Feurer, A. Klevin, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter. 2019. *Auto-sklearn: Efficient and Robust Automated Machine Learning*.

[17] Pieter Gijsbers, Erin LeDell, Janek Thomas, Sébastien Poirier, Bernd Bischl, and Joaquin Vanschoren. 2019. An open source AutoML benchmark. *arXiv preprint arXiv:1907.00909* (2019).

[18] Elliott Gordon-Rodriguez, Gabriel Loaiza-Ganem, Geoff Pleiss, and John Patrick Cunningham. 2020. Uses and abuses of the cross-entropy loss: Case studies in modern deep learning. (2020).

[19] Suyog Gupta, Wei Zhang, and Fei Wang. 2016. Model accuracy and runtime tradeoff in distributed deep learning: A systematic study. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 171–180.

[20] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. AutoML: A Survey of the State-of-the-Art. *Knowledge-Based Systems* 212 (2021), 106622.

[21] Yuval Heffetz, Roman Vainshtein, Gilad Katz, and Lior Rokach. 2020. Deepline: Automl tool for pipelines generation using deep reinforcement learning and hierarchical actions filtering. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2103–2113.

[22] Mossad Helali, Essam Mansour, Ibrahim Abdelaziz, Julian Dolby, and Kavitha Srinivas. 2022. A Scalable AutoML Approach Based on Graph Neural Networks. *Proc. VLDB Endow.* 15, 11 (jul 2022), 2428–2436. https://doi.org/10.14778/3551793.3551804

[23] J. H. Holland. 1992. Genetic Algorithms. *Scientific American* 267, 1 (1992), 66–73.

[24] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*. Springer, 507–523.

[25] Shubhra Kanti Karmaker, Md Mahadi Hassan, Micah J Smith, Lei Xu, Chengxiang Zhai, and Kalyan Veeramachaneni. 2021. AutoML to Date and Beyond: Challenges and Opportunities. *ACM Computing Surveys (CSUR)* 54, 8 (2021), 1–36.

[26] Matthew Kay, Shwetak N Patel, and Julie A Kientz. 2015. How good is 85%? A survey tool to connect classifier evaluation to acceptability of accuracy. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*. 347–356.

[27] Krishnateja Killamsetty, Durga Sivasubramanian, Ganesh Ramakrishnan, and Rishabh Iyer. 2021. Glister: Generalization based data subset selection for efficient and robust learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 8110–8118.

[28] Jaehong Kim, Sangyeul Lee, Sungwan Kim, Moonsu Cha, Jung Kwon Lee, Youngduck Choi, Yongseok Choi, Dong-Yeon Cho, and Jiwon Kim. 2018. Auto-meta: Automated gradient based meta learner search. *arXiv preprint arXiv:1806.06927* (2018).

[29] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. 2004. Estimating mutual information. *Physical review E* 69, 6 (2004), 066138.

[30] Erin LeDell and Sebastien Poirier. 2020. H2o automl: Scalable automatic machine learning. In *Proceedings of the AutoML Workshop at ICML*, Vol. 2020.

[31] Mian Li, Shapour Azarm, and Vikrant Aute. 2005. A multi-objective genetic algorithm for robust design optimization. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*. 771–778.

[32] Tian Li, Jie Zhong, Ji Liu, Wentao Wu, and Ce Zhang. 2018. Ease. ml: Towards multi-tenant resource sharing for machine learning workloads. *Proceedings of the VLDB Endowment* 11, 5 (2018), 607–620.

[33] Yang Li, Yu Shen, Wentao Zhang, Ce Zhang, and Bin Cui. 2022. VolcanoML: speeding up end-to-end AutoML via scalable search space decomposition. *The VLDB Journal* (2022), 1–25.

[34] Edo Liberty, Zohar Karnin, Bing Xiang, Laurence Rouesnel, Baris Coskun, Ramesh Nallapati, Julio Delgado, Amir Sadoughi, Yury Astashonok, Piali Das, et al. 2020. Elastic machine learning algorithms in amazon sagemaker. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 731–737.

[35] Aristidis Likas, Nikos Vlassis, and Jakob J Verbeek. 2003. The global k-means clustering algorithm. *Pattern recognition* 36, 2 (2003), 451–461.

[36] Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. 2020. Coresets for data-efficient training of machine learning models. In *International Conference on Machine Learning*. PMLR, 6950–6960.

[37] Baharan Mirzasoleiman, Kaidi Cao, and Jure Leskovec. 2020. Coresets for robust training of deep neural networks against noisy labels. *Advances in Neural Information Processing Systems* 33 (2020), 11465–11477.

[38] Tianyu Mu, Hongzhi Wang, Shenghe Zheng, Shaoqing Zhang, Cheng Liang, and Haoyun Tang. 2021. Assassin: an automatic classification system based on algorithm selection. *Proceedings of the VLDB Endowment* 14, 12 (2021), 2751–2754.

[39] 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*. 485–492.

[40] Randal S Olson and Jason H Moore. 2016. TPOT: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*. PMLR, 66–74.

[41] R. S. Olson and J. H. Moore. 2016. TPOT: A Tree-based Pipeline Optimization Tool for Automating Machine Learning. In *JMLR: Workshop and Conference Proceedings*, Vol. 64. 66–74.

[42] OpenML. 2022. https://www.openml.org/.

[43] Y. Park, M. Cafarella, and B. Mozafari. 2016. Visualization-aware sampling for very large databases. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. 755–766. https://doi.org/10.1109/ICDE.2016.7498287

[44] 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 Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[45] SubStrat Github Repository. 2022. https://github.com/teddy4445/SubStrat.

[46] UCI Machine Learning Repository. 2022. https://archive.ics.uci.edu/.

[47] Chunnan Wang, Hongzhi Wang, Tianyu Mu, Jianzhong Li, and Hong Gao. 2020. Auto-model: utilizing research papers and HPO techniques to deal with the cash problem. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1906–1909.

[48] Chi Wang, Qingyun Wu, Markus Weimer, and Erkang Zhu. 2021. FLAML: a fast and lightweight AutoML Library. *Proceedings of Machine Learning and Systems* 3 (2021), 434–447.

[49] Jonathan Waring, Charlotta Lindvall, and Renato Umeton. 2020. Automated machine learning: Review of the state-of-the-art and opportunities for healthcare. *Artificial Intelligence in Medicine* 104 (2020), 101822.

[50] Kaggle Website. 2022. https://github.com/teddy4445/SubStrat.

[51] Abraham Itzhak Weinberg and Mark Last. 2019. Selecting a representative decision tree from an ensemble of decision-tree models for fast big data classification. *Journal of Big Data* 6, 1 (2019), 1–17.

[52] Duch Wlodzislaw, Tadeusz Wieczorek, Jacek Biesiada, and Marcin Blachnik. 2004. Comparison of feature ranking methods based on information entropy, Vol. 2. 1415 – 1419 vol.2. https://doi.org/10.1109/IJCNN.2004.1380157

[53] Qingyun Wu, Chi Wang, and Silu Huang. 2021. Frugal optimization for cost-related hyperparameters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 10347–10354.

[54] Anatoly Yakovlev, Hesam Fathi Moghadam, Ali Moharrer, Jingxiao Cai, Nikan Chavoshi, Venkatanathan Varadarajan, Sandeep R Agrawal, Sam Idicula, Tomas Karnagel, Sanjay Jinturkar, et al. 2020. Oracle automl: a fast and predictive automl pipeline. *PVLDB* 13, 12 (2020), 3166–3180.