TPOT-SH: a Faster Optimization Algorithm to Solve the AutoML Problem on Large Datasets

Laurent Parmentier
OVH
Lille, France
laurent.parmentier@corp.ovh.com

Olivier Nicol
OVH
Lille, France
olivier.nicol@corp.ovh.com

Laetitia Jourdan
University of Lille
CRIStAL, UMR 9189, CNRS
Lille, France
laetitia.jourdan@univ-lille.fr

Marie-Eleonore Kessaci
University of Lille
CRIStAL, UMR 9189, CNRS
Lille, France
mkessaci@univ-lille.fr

***Abstract*—Data are omnipresent nowadays and contain knowledge and patterns that machine learning (ML) algorithms can extract so as to take decisions or perform a task without explicit instructions. To achieve that, these algorithms learn a mathematical model using sample data. However, there are numerous ML algorithms, all learning different models of reality. Furthermore, the behavior of these algorithms can be altered by modifying some of their plethora of hyperparameters. Cleverly tuning these algorithms is costly but essential to reach decent performance. Yet it requires a lot of expertise and remains hard even for experts who tend to resort to exploration-only approaches like random search and grid search. The field of AutoML has consequently emerged in the quest for automatized machine learning processes that would be less expensive than brute force searches. In this paper we continue the research initiated on the Tree-based Pipeline Optimization Tool (TPOT), an AutoML based on Evolutionary Algorithms (EA). EAs are typically slow to converge which makes TPOT incapable of scaling to large datasets. As a consequence, we introduce TPOT-SH inspired from the concept of Successive Halving used in Multi-Armed Bandit problems. This solution allows TPOT to explore the search space faster and have much better performance on larger datasets.**

***Index Terms*—AutoML, TPOT, Evolutionary Algorithms, Successive Halving, Large Datasets**

I. Introduction

AutoML [2] is a recent and very challenging [1] field which tries to automatize the tedious task of finding efficient Machine Learning (ML) pipelines. It is a research problem that has been stated for the first time by Thornton et al. [7]. They introduce the CASH problem (1) and design a solution called Auto-Weka. CASH stands for Combined Algorithm Selection and Hyperparameter optimization. The problem consists in selecting the best model [20] and tuning its hyperparameters [18] to optimize one or more objectives related to the model (eg., accuracy, precision, recall). Optimizing the hyperparameters is a key element of the ML pipeline. It is motivated by the impact it has on the final performance of the ML pipeline [19].

Solving the AutoML problem is time consuming for three reasons. The first one is related to the training phase of the ML algorithm. Indeed, training ML algorithms on small datasets takes at least a few seconds and much more for bigger datasets which is increasingly common with big data. Secondly, ML algorithms can be stochastic and need several evaluations to have a fair idea of their performance on a dataset. The last reason is obviously the large choice of ML algorithms and their vaste amount of associated hyperparameters, which may lie in more or less large finite spaces or even in continuous spaces. Also, their contribution to the performance of the algorithm is generally not well studied, very different from one parameter to another and highly data-dependent. Thus, given the size of the search space and the time it takes to train a ML pipeline, it is computationally expensive to look over all the ML pipelines and find the best one for the dataset at hand.

Hence, AutoML solutions blossomed across research communities with the purpose of smartly finding ML pipelines with good performance. To name a few, Auto-Weka [7], Auto-Sklearn [8], Hyperopt-sklearn [10], TPOT [3], ATM [9], Recipe [14] and, Auto-Stacker [15] are AutoML solutions tackling at least the CASH problem.

However, most of these AutoML solutions address a more general problem than CASH. Indeed, a ML pipeline includes a preprocessing part (also called feature engineering) that consists in transforming the data in various ways to facilitate the learning of the ML pipeline *e.g.,* feature extraction, feature encoding, feature generation, dimensionality reduction.

All these facets are important and allow to enhance the performance of a ML pipeline. Similarly to the model selection and to the tuning of the hyperparameters, selecting the preprocessing methods remains a difficult task and requires expertise. Hence, solutions such as Deep Feature Synthesis [16] or ExploreKit [17] have emerged in an attempt to automatically select and configure the preprocessing methods. AutoML solutions such as TPOT [3] encompass an automatic preprocessing phase.

Despite the importance of this aspect, the problem has never been formally defined. Therefore, this paper introduces a new definition of the CASH problem in equation (2). This definition includes the preprocessing part and gets closer to the problem of automatically building an entire ML pipeline.

Another interesting part in machine learning is the Multi-Objective (MO) aspect. Indeed, optimizing a ML pipeline on a unique objective may be unadapted for many real world problems, *e.g.,* maximizing the probability of detection and minimizing the probability of false alarm in fraud detection. MO tends to improve machine learning algorithms [23] and

2375-0197/19/$31.00 ©2019 IEEE
DOI 10.1109/ICTAI.2019.00072

lets the user select various models according to his needs. Evolutionary Algorithms (EA) are well-suited for MO optimizations, however EAs are also well-known to evaluate a lot of candidates, *i.e.,* train ML pipelines. This characteristic leads to a slow optimization, because training a ML pipeline is time consuming as mentioned before. Among all the AutoML solutions, TPOT is the only one including a multi-objective mechanism.

In this paper we present TPOT-SH, a faster AutoML solution based on TPOT and Successive Halving (SH) that reduces the time that EAs consume to find ML pipelines on large datasets.

The paper is organized as follows. Section II recalls preliminaries concerning the AutoML problem. Section III presents how TPOT and Successive Halving work. Section IV describes TPOT-SH. Section V details the experimental setup used to compare TPOT and TPOT-SH. Section VI presents results of TPOT versus TPOT-SH and offers a discussion that leads to some interesting perspectives. Section VII concludes the paper.

II. Preliminaries on AutoML

The AutoML problem is equivalent to solving the hyperparameter optimization problem in addition to the algorithm selection problem. It has been introduced by Thornton as the CASH problem in AutoWEKA [7]:

$$\mathrm{A}^{\star}_{\lambda^{\star}} \in \underset{\mathrm{A}^{(\mathrm{i})} \in \mathcal{A}, \lambda \in \Lambda^{(i)}}{\arg\min} \mathcal{L}(\mathrm{A}^{(\mathrm{i})}_{\lambda}(\mathcal{D}_t), \mathcal{D}_v) \tag{1}$$

Where:

- $\mathcal{A}$ is a set of n ML algorithms
- $\Lambda^{(1)}, ..., \Lambda^{(n)}$ are hyperparameter spaces respective to each ML algorithm $\mathrm{A}^{(1)}, ..., \mathrm{A}^{(n)}$
- $\mathcal{D}$ is a dataset split in two parts $\mathcal{D}_t$ and $\mathcal{D}_v$
- $\mathcal{L}(\mathrm{A}^{(\mathrm{i})}_{\lambda}(\mathcal{D}_t), \mathcal{D}_v)$ is a loss function for $\mathrm{A}^{(\mathrm{i})}$, using $\lambda \in \Lambda^{(i)}$, trained on $\mathcal{D}_t$ and evaluated on $\mathcal{D}_v$ for $i \in \{1, ..., n\}$

As mentioned previously, we introduce a new definition of the AutoML problem where the preprocessing part is included in the definition. We formally define Combined Algorithm Selection, Hyperparameter optimization And Preprocessing selection (CASHAP) as the following problem:

$$(\rho^{\star}, \mathrm{A}^{\star}_{\lambda^{\star}}) \in \underset{\rho \in \Phi, \mathrm{A}^{(\mathrm{i})} \in \mathcal{A}, \lambda \in \Lambda^{(i)}}{\arg\min} \mathcal{L}(\mathrm{A}^{(\mathrm{i})}_{\lambda}(\rho(\mathcal{D}_t)), \rho(\mathcal{D}_v)) \tag{2}$$

Where Φ is the space containing all different preprocessing methods, s.t. $\forall \rho_1, \rho_2 \in \Phi, \rho_1 \circ \rho_2 \in \Phi$. All other symbols are previously defined in the CASH problem.

In the rest of this article we consider the CASHAP problem, *i.e.,* a solution or a candidate is a ML pipeline represented as a combination of preprocessing method(s) and a single ML algorithm with its associated hyperparameters.

III. Presentation of TPOT and SH

In this section we briefly describe TPOT in order to have basics on how its authors create, select and represent the candidates within the evolutionary process. Then we introduce the basic concept of Successive Halving (SH) from which stems the solution we propose: TPOT-SH.

A. Tree-based Pipeline Optimization Tool (TPOT)

TPOT [3] is a tool tackling the AutoML problem. It is built on top of DEAP [11], a modular framework helping to build processes based on Evolutionary Algorithms (EA). The implementation uses genetic programming with a $(\mu + \lambda)$-ES strategy [12].

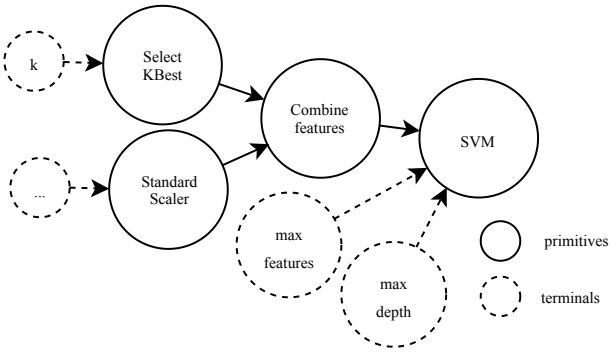

Fig. 1. Representation of an AutoML candidate in TPOT

In genetic programming, each solution or candidate is represented as a tree composed of nodes and leaves. Each internal node is called a primitive and each leaf a terminal. A primitive can be seen as a function, a terminal as an argument or a constant. Primitives and terminals are defined depending on the problem we solve. In the case of TPOT, primitives are the ML algorithms or the preprocessing methods and terminals are the hyperparameters. Note that the root node of the tree is necessarily a ML algorithm used as a final estimator and other nodes act as operators taking data as input and output the transformed data. Trees can be composed of different branches where two branches are merged through a special primitive introduced by TPOT. The merge combines the output from the previous nodes. An example of an AutoML candidate represented by a tree is given in Fig. 1.

First of all, the DEAP framework generates $\mathcal{P}$ candidates (also called individuals in evolutionary algorithms) that have the same probability to be composed of one or two primitives.

Once initial candidates are generated, they are evaluated based on a fitness function F. In TPOT, the fitness function is composed of two objectives. The first one is a loss function evaluating the final estimator (*e.g.,* maximizing the accuracy in classification) and, the second one consists in minimizing the pipeline's complexity (*i.e.,* number of nodes composing the tree). The second objective acts as a regularization method on the pipeline and allows the practitioner to have a better understanding of what the candidate is composed of.

The initial candidates that compose the generation 0 will be used as initial parent population for the first iteration of the loop process.

The process iterates through G generations where μ candidates are selected from the parent population in order to create λ offspring candidates through variations (crossover or mutation). After the first iteration, the parent population is

composed of the previous parent population and of the newly created offspring candidates.

At the end of the process, TPOT returns the AutoML candidate with the lowest loss value on the Pareto front. The Pareto front is the set of non-dominated candidates, for which we cannot find any other candidate improving at least one objective.

B. Successive-Halving

Sequential Halving [4] also called Successive Halving [5] (SH) is a technique introduced in the Multi-Armed Bandit problem. The strategy consists in eliminating the worst half of the arms as well as increasing the number of time that surviving arms are pulled during the iterations of the process. This concept has been explored in PoSH Auto-Sklearn [6] an AutoML solution based on Bayesian optimization allowing this framework to scale on large datasets. SH has also been explored by Google Brain with its method called SHAC [13] to optimize the hyperparameters.

IV. TPOT-SH

TPOT is very slow when it comes to run on large datasets. Indeed, the training set size and the number of candidates to evaluate are substantial. Our proposal is to adapt the two main components of SH in the evolutionary process of TPOT. We replace the number of arms by the population size (3) and the number of times that arms are pulled by the sample size used by a candidate to be evaluated (4) that we call budget. The interest of such a method is to spend little time on as much candidates as possible, pruning bad ones and then spend more time on promising ones.

$$p_i = \mathcal{P}/2^i \tag{3}$$

$$b_i = b2^i \tag{4}$$

We denote p_i and b_i the population size and the budget at the i^{th} generation. We initialize $\mathcal{P}$ with the initial population size specified in the evolutionary algorithm and b as a certain percentage of the training set size.

In the classical use of SH, the reduction of the number of arms and the increase of the budget are applied at each step of the loop. However, applied on Evolutionary Algorithms (EA), the population size quickly decrease to one individual only and the budget is totally consumed (*i.e.,* budget is equal to the dataset size). Also EAs need several successive generations to find interesting region of the search space. Therefore we adapt the original formulas and end up with the following equations (8) and (9) which respectively reduce the population size and increase the budget after certain generations.

To find these equations we replace i by ia, where a is a coefficient that controls the point when the population is divided by two. So for the population size p_i, we have:

$$p_i = \mathcal{P}/2^{ia} \tag{5}$$

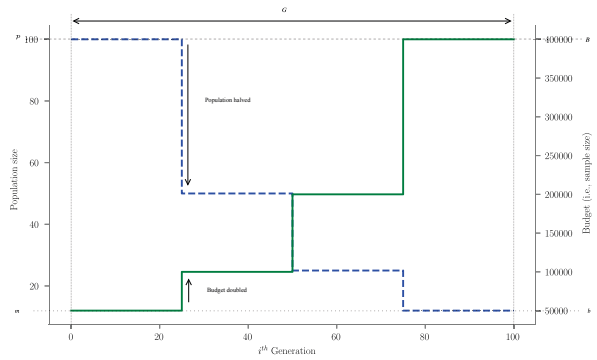

Fig. 2. Representation of $p_i^\star$ in blue dot line and $b_i^\star$ in green solid line. The horizontal axe represents the generations. The vertical left axe and the vertical right axe respectively represent the population size $p_i^\star$ and the budget $b_i^\star$ at i^{th} generation.

To find the coefficient, we need to be able to define the number of iterations needed such as there are m individuals left in the population:

$$\mathcal{P}/2^i = m \implies i = \log_2(\mathcal{P}/m) \text{ iterations} \tag{6}$$

Then we distribute the ratio on the G generations, it gives:

$$a = \log_2(\mathcal{P}/m)/G \tag{7}$$

We apply the floor function to the whole equation to keep a natural number that represents the population size. We also apply the *floor* to the exponent to ensure that the division of the population is performed by a factor 2. Finally, we insert "+1" to balance the equation such that the number of iterations between two divisions are equal. It yields:

$$p_i^\star = \lfloor \mathcal{P}/(2^{\lfloor i \lfloor (\log_2(\mathcal{P}/m)+1) \rfloor /(G+1) \rfloor}) \rfloor, \tag{8}$$

where m is the minimal number of individuals to keep in the parent population, G and $\mathcal{P}$ are respectively the number of generations and the initial population size specified in the evolutionary algorithm.

Concerning b_i, we applied the same process to replace the exponent.

$$b_i^\star = b2^{\lfloor i \lfloor (\log_2(B/b)+1) \rfloor /(G+1)) \rfloor}, \tag{9}$$

where b is the initial budget and B the maximum budget. Fig. 2 allows to have a better representation of the new defined functions $p_i^\star$ and $b_i^\star$.

The implementation of our tool uses TPOT v0.9.5[1] where we adapt the $(\mu + \lambda)$-ES in such a way that at generation i, $\mu = \lambda = p_i^\star$ and that the first objective of the fitness function F is evaluated on $b_i^\star$ samples.

Procedure of TPOT-SH is provided in Algorithm 1.

[1]https://github.com/EpistasisLab/tpot/tree/507b45db01e

Algorithm 1 TPOT-SH

Input: population size, $\mathcal{P}$; minimum of individuals, m; initial budget, b; maximum budget, B; number of generations, G; training set, $\mathcal{D}_t$

Output: best candidate from the Pareto front

$b_0 \leftarrow b$
$p_0 \leftarrow \mathcal{P}$
$\alpha^{(0)} \leftarrow$ generate($\mathcal{P}$)
$\mathcal{D}_0 \leftarrow D' : D' \subseteq \mathcal{D}_t, |D'| = b_0$
$\mathfrak{P}_p^{(1)} \leftarrow \{(\alpha_k^{(0)}, \text{evaluate}(\alpha_k^{(0)}, \mathcal{D}_0)), k = 1, ..., \mathcal{P}\}$
for $i := 1$ **to** G **do**
  $b_i \leftarrow b2^{\lfloor b \lfloor (\log_2(B/b)+1) \rfloor / (G+1)) \rfloor}$
  $\mathcal{D}'' \leftarrow D' : D' \subseteq \mathcal{D}_t, |\mathcal{D}' \cup \mathcal{D}_{i-1}| = b_i, D' \cap D_{k-1} = \emptyset$
  $\mathcal{D}_i \leftarrow \mathcal{D}_{i-1} \cup D''$
  $\alpha^{(i)} \leftarrow$ clone($\mathfrak{P}_p^{(i)}$, p_{i-1})
  $\alpha^{(i)} \leftarrow$ variation($\alpha^{(i)}$)
  $\mathfrak{P}_o^{(i)} \leftarrow \{(\alpha_k^{(i)}, \text{evaluate}(\alpha_k^{(i)}, \mathcal{D}_i)), k = 1, ..., p_{i-1}\}$
  $p_i \leftarrow \lfloor \mathcal{P} / (2^{\lfloor i \lfloor (\log_2(\mathcal{P}/m)+1) \rfloor / (G+1) \rfloor}) \rfloor$
  $\mathfrak{P}_p^{(i+1)} \leftarrow$ select ($\mathfrak{P}_p^{(i)} \cup \mathfrak{P}_o^{(i)}$, p_i)
end for
return $\alpha^\star \in \text{opt.}F$

We denote $\alpha^{(i)}$ the list of non-evaluated candidates, $\mathfrak{P}_p^{(i)}$ the parent population, $\mathfrak{P}_o^{(i)}$ the offspring population and $\mathcal{D}_i$ the i^{th} subset of the training set $\mathcal{D}_t$. The procedures are the following (each one being instantiated according to the experiments presented in Section VI):

- **generate**(p): generates a list of p candidates. In our case each candidate is a tree whose nodes are picked up from primitives and terminals from the leaves.
- **evaluate**(c, d): evaluates candidate c on dataset d. In the experiments, a 5-fold cross-validation is used on the dataset d. The objective value assigned to a candidate is the performance computed on the validation set.
- **clone**(p, λ): clones λ candidates by selecting them randomly from the population p.
- **variation**(p): applies on each individual of the population p a variation operator, *i.e.,* a mutation or a crossover, according to the rates defined in Tab. I.
- **select**(p, μ): selects μ candidates from the population p. TPOT used the efficient selection from NSGA-II algorithm. In the $(\mu + \lambda)$-ES, the population p is composed of the parent population $\mathfrak{P}_p^{(i)}$ and the offspring $\mathfrak{P}_o^{(g)}$.

V. Experimental Setup

In this section, we describe our experimental protocol that allows us to compare the optimization performance of TPOT-SH versus TPOT.

Evolutionary algorithms are stochastic and have to be run several times in order to measure their performance. In our experiments, we run TPOT and TPOT-SH 30 times each. We keep the original settings from the article of TPOT [3] as described in Tab. I. For large datasets, we reduce the number of generations and for TPOT, add a supplementary termination criterion based on time. Indeed, TPOT would use too much time to provide results on large datasets. By observing early convergence around 25 generations during the experiments on small datasets, we fix the number of generations to 25 for the large ones with TPOT-SH. The termination criterion is fixed to the time needed by TPOT-SH to stop naturally (*i.e.,* after the 25 generations).

Both TPOT and TPOT-SH are run on 8 datasets (see Tab. III). The small datasets have been picked from the experiments in the article of TPOT [3] in an attempt to validate our experimental setup. We additionally provide 4 large datasets (composed of hundred thousand instances) from an AutoML benchmark [21].

Contrarily to TPOT protocol [3] which presents the results from the performance of the best candidate (candidate maximizing accuracy in the Pareto front at the 100$^{\text{th}}$ generation, *i.e.,* the last one), we measure the performance of each individual in the population at each generation. This measure allows us to know how the optimization process converges.

To evaluate our experiments, we use a sample of 75% of the original dataset as training data and test the candidates on the remaining 25%. The split is shuffled in a stratified way with a different seed per run, and are identical for both algorithms with the aim to have comparable results (same training and test set for each paired run).

By design, at generation i, the number of samples seen by TPOT-SH depends on the budget $b_i^\star$, which is a percentage of instances taken from the 75% of the training set. This budget serves to perform a cross-validation on candidates. In the case of TPOT, the whole training set serves along the generations to perform the cross-validation of the candidates. Hence TPOT-SH is evaluated on fewer instances than TPOT. It makes the results from the cross-validation less accurate. To solve this problem, we use an identical test set (25% remaining) between TPOT and TPOT-SH that fairly evaluate and compare both algorithms.

In order to compare the performance of TPOT and TPOT-SH, we compute the average elapsed time between generations and, for all generation, the average performance of the ML pipelines.

TPOT-SH has its own parameters that are fixed as described in Tab. II. We naturally set the maximum budget to 100% in order to make the whole training set available in the latest iterations of the optimization process.

Computational environment. We conduct the experiments on OVH Public Cloud with different clusters depending on our requirements. For small datasets, we use two C2-120 virtual machines (VM) composed of 32 cores of 3.1Ghz and 120GB RAM each. The first VM is used to run TPOT and the second one for TPOT-SH. This way we ensure fair computation environments for both algorithms. The large datasets require more memory, thus we use multiple R2-240 VMs with 240GB RAM and 16 cores of 2.3Ghz. For example, each run of TPOT for the airlines dataset took up to 20GB of memory. With only 12 runs, the machine was out of memory. In order to prevent this problem, we set up multiple VMs and fairly balance runs

on clusters by keeping fair computation environments for each algorithm.

TABLE I
SHARED SETTINGS OF TPOT AND TPOT-SH

Parameter	Value
Population size $\mathcal{P}$	100
Generations G	25* or 100
Per-individual mutation rate	90%
Per-individual crossover rate	10%
TPOT Pareto selection	100 individuals according to NSGA-II
Mutation	Point, insert, shrink 1/3 chance of each
Crossover	OnePoint
Candidate evaluation	5-fold cross-validation
Maximum evaluation time per candidate	5 minutes
Number of jobs	1

* only for large datasets.

TABLE II
OWN SETTINGS OF TPOT-SH

Parameter	Value
Initial budget b	30% of samples from training set
Maximal budget B	100% of samples from training set
Minimum individuals m	10

TABLE III
CLASSIFICATION DATASETS

Dataset	# Inst.	# Attr.	# Class.	Majority class
wine-quality-red	1599	12	10	43%
car-evaluation	1728	6	4	70%
spambase	4601	57	2	60%
wine-quality-white	4898	12	10	45%
miniboone	130064	51	2	72%
kddcup99	494020	41	23	57%
airlines	539383	8	2	55%
covertype	581012	54	7	49%

denote the cardinality.

VI. RESULTS

In this section we describe the results of our experiments. Firstly, we compare the performance of TPOT-SH and TPOT on small datasets, and then on the large datasets.

A. Results for small datasets

Fig. 3 presents the results on the small datasets. TPOT-SH does not perform well on these small datasets. Indeed, the performance of TPOT-SH is always below TPOT over the 100 generations.

The results are explained by the small number of instances in each dataset at the beginning of the run of TPOT-SH that leads to overfitting. However, as mentioned in our motivation, TPOT-SH has not been designed for small datasets.

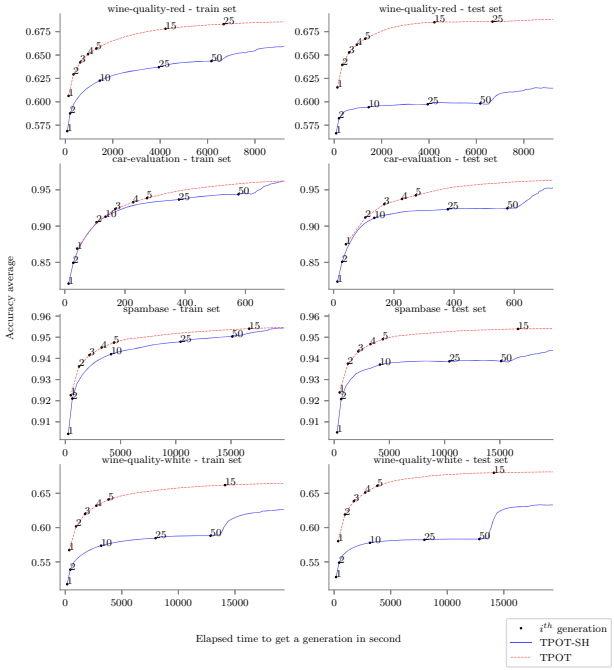

Fig. 3. Results for small datasets. Panes on left side are performance from cross-validation on training set, panes on right side are performance from test set. Each horizontal axe represents the elapsed time in seconds and each vertical axe represents the accuracy. TPOT is represented as a red dot line and TPOT-SH as a blue solid line. Each black point represents when a generation is completed, i.e. all individuals of the population are evaluated.

TABLE IV
ACCURACY ON TEST SET AT DIFFERENT TIME WHERE T1 AND T2 RESPECTIVELY REPRESENT THE TIME THAT TPOT-SH AND TPOT OBTAINED THE FIRST GENERATION AND T3 IS THE TIME THAT TPOT-SH OBTAINED THE LATEST GENERATION.

Dataset	T1		T2		T3	
	TPOT	TPOT-SH	TPOT	TPOT-SH	TPOT	TPOT-SH
wine-quality-red	72 seconds		180 seconds		2.22 hours	
	N.A.	56.6%	61%	58.26%	68.47%	62.1%
car-evaluation	72 seconds		212 seconds		1.73 hours	
	N.A.	82.79%	87.45%	85.92%	97%	95.15%
spambase	360 seconds		810 seconds		7.54 hours	
	N.A.	90.74%	92.39%	92.1%	95.4%	94.6%
wine-quality-white	165 seconds		432 seconds		5.22 hours	
	N.A.	52.75%	57.9%	55.12%	68.2%	63.2%
miniboone	2.61 hours		6.17 hours		46 hours	
	N.A.	89.4%	83.5%	91.8%	93.3%	94%
kddcup99	6.57 hours		18.6 hours		72 hours	
	N.A.	98%	96.2%	99.9%	99.9%	99.9%
airlines	2.32 hours		8.8 hours		53.5 hours	
	N.A.	63.8%	61.4%	65.6%	66.2%	66.6%
covertype	7.5 hours		12.15 hours		101.5 hours	
	N.A.	71.2%	61.8%	75.2%	79.45%	96.2%

N.A. for Not Available.

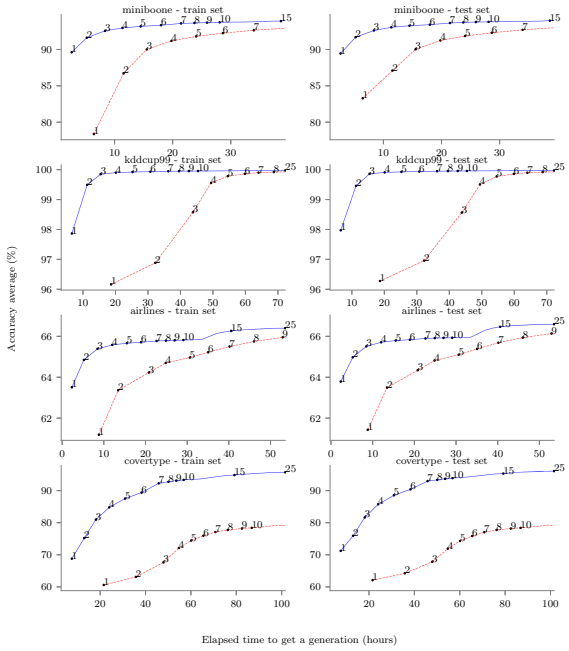

Fig. 4. Results for large datasets. Panes on left side are performance from cross-validation on training set, panes on right side are performance from test set. Each horizontal axe reprents the elapsed time in hours and each vertical axe represents the accuracy. TPOT is represented as a red dot line and TPOT-SH as a blue solid line. Each black point represents when a generation is completed, i.e. all individuals of the population are evaluated.

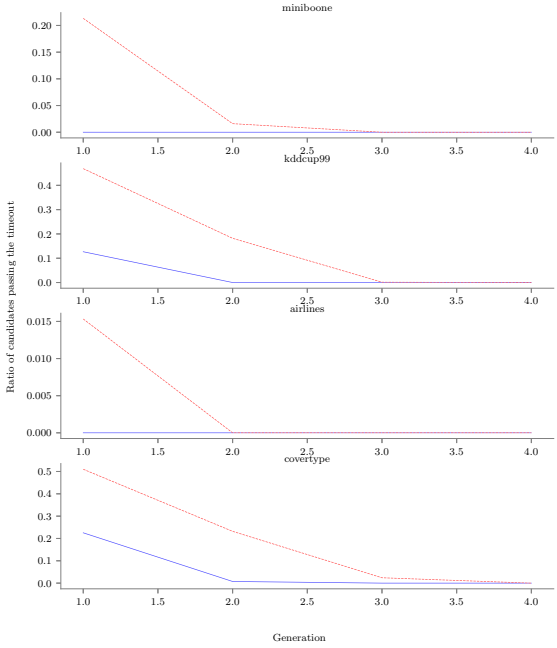

Fig. 5. Unevaluated candidates on large datasets. TPOT is represented as a red dot line and TPOT-SH as a blue solid line.

We can notice that TPOT-SH's convergence line is irregular with some bumps. It is explained by the population size which is divided at some generations. It implies that the performance of the population is not homogeneous. Indeed, when the population size decreases between two successive generations, the average is computed on less and better candidates.

As expected, we observe that TPOT-SH obtained the first generation before TPOT, since the size of the training set, controlled by SH, is very small. This benefit is insignificant for small datasets but becomes more significant for large datasets.

B. Results for large datasets

Fig. 4 presents the computed results for large datasets. We clearly observe that TPOT-SH performs better than TPOT on the training set as well as on the test set. TPOT-SH finds in average better candidates and faster. In order to have a better understanding, we detail the results for miniboone dataset and provide an overview of the results for all datasets in Tab. IV.

At the first generation of miniboone dataset, TPOT-SH obtains an accuracy of 89.6% on the training set and 89.4% on the test set in 2.61 hours. TPOT obtains the first generation in 6.17 hours with an accuracy of 78.3% on the training set and 83.2% on the test set. Thus, we divide the time of getting first generation by two, and at the same time the accuracy is increased by 6% on the test set. It would take a total of 14 hours from the initial run, *i.e.,* five times longer than TPOT to reach a similar level of performance as the one TPOT-SH reaches. After a few generations, the population fastly converges and we clearly observe that TPOT stays below TPOT-SH. After 46 hours of run, TPOT converges to a performance of 93.3% that is already achieved by TPOT-SH 29 hours ealier.

At first reading, these results can confuse machine learning practitioners, indeed, TPOT has more knowledge, *i.e.,* 75% of the training set, compared to TPOT-SH during the first generations (*e.g.,* 30% from the 75% of the training set). Hence TPOT should give better performance, however, TPOT has a timeout of 5 minutes by default to evaluate a candidate. If the time is elapsed, the candidate is discarded. This explains why good candidates that take too much time to be evaluated are not included in the computation. To verify this hypothesis, we draw the ratio of non-evaluated candidates per generation in Fig. 5. We can observe that timeout is reached for almost half of the candidates at the first generation in TPOT (see for example covertype and kddcup99 datasets, the largest datasets in terms of instances $\times$ features), but then, the ratio decreases to zero after two or three generations. Even if it is less significant, this phenomena also happens with TPOT-SH. We did not include results after generation four because it is just a flat line with no candidates having a timeout. Concerning small datasets they never reached the timeout regardless of

the generation. One manner to solve this issue is to increase the timeout, that would certainly be a benefit for TPOT on early generations as well as for TPOT-SH. Nonetheless it would also be a disadvantage in term of time to get the first generation. Hence, increasing the timeout does not solve the problem of accelerating the optimization process. Moreover, the number of non evaluated candidates is not significant for the airlines dataset that exhibits a similar behavior to the other large datasets. By this fact, we can be quite confident that timeout procedure does not bias our results. In consequence, having a smaller subset of the training set that represents the dataset well enough to train ML algorithm leads to faster and better performance.

Note that dataset airlines has an irregularity. This abrupt change is not visible in other large datasets because the performance of the candidates is homogeneous. When the performance of the candidates is not homogenous, our selection process that decreases the population size impacts the average performance. Indeed, we start with a population of 100 candidates which falls to 25 individuals at generation 13. This is significant when there is variance in the performance of the population.

Another remark concerns the performance on the training set versus the test set. The difference is almost always negligible. This confirms that despite the very little amount of data used by TPOT-SH during the first iterations, there is no overfitting thanks to the stratified cross-validation scheme. However we insist that keeping the test set as a measure is a good thing to ensure that this is the case (see the small dataset experiments).

As we can see, our algorithm is far ahead of TPOT on large datasets. The first generation is always obtained much faster. This can be of paramount importance for applications requiring results in a limited amount of time. Moreover, we obtain better results. The strength of our solution mainly resides on the budget, which on large datasets considers subsets that represent the dataset well enough.

C. Discussion

Here we discuss our results and provide some hints for future work on short term to further improve TPOT-SH.

First of all, we expected better results for small datasets. One way to improve the current results is to setup a higher initial budget in such a way that the subset will be representative enough. However, when a dataset is very small, it is complicated to find such a subset. It would be interesting to study some statistical methods to know what percentage of the training set size would represent the dataset well enough. This percentage would serve as initial budget in equation (9). Another solution to consider is to change the principle of doubling the budget by multiplying it with another factor than two in equation (4) and this could also be done for the population size in equation (3). Changing the factor would not only be an improvement for the performance on small datasets but also for the performance on larger ones. Studying different factors is another lead to follow. We did not explore

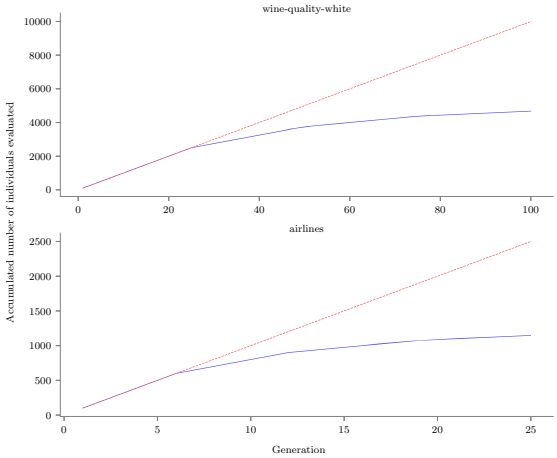

Fig. 6. Total of accumulated evaluations per generation. TPOT is represented as a red dot line and TPOT-SH as a blue solid line.

this aspect because we initially planned to respect the concept of successive halving.

In complement to the results on the optimization convergence in section VI, we extracted the total number of evaluated candidates per generation for two datasets in Fig. 6, a small one and a large one. In total, TPOT evaluates 10100 candidates, *i.e.,* 100 pipelines times 100 generations plus 100 candidates from generation 0 for the small dataset and, 2600 for the large one. In the case of TPOT-SH, it evaluates 4763 candidates for the small dataset and 1235 candidates for the large dataset. TPOT-SH evaluates approximately two times fewer candidates compared to TPOT. This behavior depends on the parameter m, representing the minimum number of individuals in equation (8). These curves are interesting because they show that TPOT-SH evaluates the same number of candidates as TPOT during the first generations and obtains faster as well as better results as seen in section VI. In consequence, we deduce that budget in equation (9) is the main component influencing the performance at the beginning. Also, these curves express how TPOT-SH does not explore the search space as much as TPOT. It means that TPOT-SH can pontentially be improved by starting with more candidates at the initialization of the population size in such a way that the total number of evaluations is fairly equal to TPOT evaluations. This gain of candidates should give more diversity and more choice during the successive selections and thus a better overall optimization performance. As a side effect, starting with more candidates would increase the time needed to obtain results due to more candidates to evaluate. However this would only happen at the beginning of the optimization process because the population size decreases later. Moreover this side effect can be reduced by having a low initial budget. Since the focus of our work was the speed of the optimization process, we did not investigate on higher population sizes.

Studying different population sizes with different budgets at the initialization of TPOT-SH would be one more leading point for enhancements.

Another perspective of amelioration indirectly related with our solution, but where we questioned ourselves during the experiments, is how to specify the number of generations. We followed the parameters from the article of TPOT [3], however there is no explanations of why they are using 100 generations. This issue could be simply solved by using their early stopping feature available on Github. Similarly, there is no explanations of why they are using 100 individuals per generation. These parameters had been chosen arbitrary. It would be interesting to study different population sizes to see how the optimization process performs. As seen on results from small datasets, 100 generations and 100 individuals are enough to converge, but it will not necessarily be the case far all dataset types, *e.g.,* very large ones. Please note that studying the initial population size from TPOT is different than it is in TPOT-SH. Indeed, while TPOT keeps it constant, TPOT-SH varies the population size along the generations.

Lastly, an interesting aspect we did not explore is to try our solution on bigger datasets with more than millions of samples. We are confident that our method would perform pretty well on such datasets thanks to the notion of maximum budget in equation (9). Indeed, this aspect prevents the optimization process from taking too much time. TPOT's maximum evaluation time would not be equivalent to this aspect, because when this timeout is reached, the candidate is simply discarded, which is not the case with the maximum budget. So the maximum budget not only reduces the optimization time but also permits complex candidates to be evaluated and kept along the runs.

To summarize, TPOT-SH could be enhanced by trying different factors in equations (4) and (3). Also exploring different population sizes, initial budgets and maximum budgets, should give even better results and handle larger datasets in reasonable time.

VII. Conclusion and future works

To conclude, in this paper we propose a solution permitting evolutionary algorithms to solve the AutoML problem faster on large datasets with better results. The implementation of the solution is pretty simple and does not increase the complexity of the algorithm. Also, we did not insist on this point but we notice that our solution requires two times fewer memory space at least, along the whole optimization process. This can be considerable for infrastructures with hardware constraints.

In the future we plan to adapt TPOT-SH to handle time series problems. And, we also plan to take advantage of TPOT-SH to widely explore the search space in an attempt to have a diversity of candidates which would be a benefit for a Multi-objective AutoML.

References

[1] I. Guyon et al., Design of the 2015 ChaLearn AutoML challenge, 2015, pp. 18.

[2] F. Hutter, Automatic Machine Learning:Methods, Systems, Challenges. Springer, 2018.

[3] R. S. Olson, N. Bartley, R. J. Urbanowicz, and J. H. Moore, Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science, Proceedings of GECCO 2016, Mar. 2016.

[4] Z. Karnin, T. Koren, and O. Somekh, Almost Optimal Exploration in Multi-armed Bandits, in Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28, Atlanta, GA, USA, 2013, pp. III1238III1246.

[5] K. Jamieson and A. Talwalkar, Non-stochastic Best Arm Identification and Hyperparameter Optimization, arXiv:1502.07943 [cs, stat], Feb. 2015.

[6] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter, Practical Automated Machine Learning for the AutoML Challenge 2018, in ICML 2018 AutoML Workshop, 2018.

[7] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms, in Proc. of KDD-2013, 2013, pp. 847855.

[8] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, Efficient and Robust Automated Machine Learning, in Advances in Neural Information Processing Systems 28, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 29622970.

[9] T. Swearingen, W. Drevo, B. Cyphers, A. Cuesta-Infante, A. Ross, and K. Veeramachaneni, ATM: A distributed, collaborative, scalable system for automated machine learning, in 2017 IEEE International Conference on Big Data (Big Data), 2017, pp. 151162.

[10] B. Komer, J. Bergstra, and C. Eliasmith, Hyperopt-Sklearn: Automatic Hyperparameter Configuration for Scikit-Learn, in Proceedings of the 13th Python in Science Conference, 2014, pp. 3339.

[11] F.-A. Fortin, F.-M. D. Rainville, M.-A. Gardner, M. Parizeau, and C. Gagn, DEAP: Evolutionary Algorithms Made Easy, Journal of Machine Learning Research, vol. 13, pp. 21712175, Jul. 2012.

[12] H.-G. Beyer and H.-P. Schwefel, Evolution strategies A comprehensive introduction, Natural Computing, vol. 1, no. 1, pp. 352, Mar. 2002.

[13] M. Kumar, G. E. Dahl, V. Vasudevan, and M. Norouzi, Parallel Architecture and Hyperparameter Search via Successive Halving and Classification, arXiv:1805.10255 [cs], May 2018.

[14] A. G. C. de S, W. J. G. S. Pinto, L. O. V. B. Oliveira, and G. L. Pappa, RECIPE: A Grammar-Based Framework for Automatically Evolving Classification Pipelines, in Genetic Programming, vol. 10196, J. McDermott, M. Castelli, L. Sekanina, E. Haasdijk, and P. Garca-Snchez, Eds. Cham: Springer International Publishing, 2017, pp. 246261.

[15] B. Chen, H. Wu, and W. Mo, Autostacker: A Compositional Evolutionary Learning System, GECCO 2018, 2018.

[16] J. M. Kanter and K. Veeramachaneni, Deep feature synthesis: Towards automating data science endeavors, in 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA), 2015, pp. 110.

[17] G. Katz, E. C. R. Shin, and D. Song, ExploreKit: Automatic Feature Generation and Selection, in 2016 IEEE 16th International Conference on Data Mining (ICDM), 2016, pp. 979984.

[18] Y. Bengio, Gradient-Based Optimization of Hyperparameters, Neural Computation, vol. 12, no. 8, pp. 18891900, Aug. 2000.

[19] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kgl, Algorithms for Hyper-parameter Optimization, in Proceedings of the 24th International Conference on Neural Information Processing Systems, USA, 2011, pp. 25462554.

[20] J. R. Rice, The Algorithm Selection Problem, vol. 15, M. Rubinoff and M. C. Yovits, Eds. Elsevier, 1976, pp. 65118.

[21] P. Gijsbers, An Open Source AutoML Benchmark, 6th ICML Workshop on Automated Machine Learning (2019), 2019.

[22] H. Rakotoarison, M. Schoenauer, and M. Sebag, Automated Machine Learning with Monte-Carlo Tree Search (Extended Version), arXiv:1906.00170 [cs, stat], Jun. 2019.

[23] Y. Jin, Ed., Multi-Objective Machine Learning, vol. 16. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006.

[24] G. Katz, E. C. R. Shin, and D. Song, ExploreKit: Automatic Feature Generation and Selection, in 2016 IEEE 16th International Conference on Data Mining (ICDM), 2016, pp. 979984.