MetaREVEAL: RL-based Meta-learning from Learning Curves *

Manh Hung Nguyen[1,2], Nathan Grinsztajn[3], Isabelle Guyon[2,4], and Lisheng Sun-Hosoya[4]

[1] CentraleSupélec, France, manh.nguyen@inria.fr
[2] LISN/Inria/CNRS, Université Paris-Saclay, France
[3] Inria, Univ. Lille, CNRS, France, nathan.grinsztajn@inria.fr
[4] ChaLearn, California, USA, {guyon,sun-hosoya}@chalearn.org

Abstract. This paper addresses a cornerstone of Automated Machine Learning: the problem of *rapidly* uncovering which machine learning algorithm performs best on a new dataset. Our approach leverages performances of such algorithms on datasets to which they have been previously exposed, i.e., implementing a form of *meta-learning*. More specifically, the problem is cast as a REVEAL Reinforcement Learning (RL) game: the meta-learning problem is wrapped into a RL environment in which an agent can start, pause, or resume training various machine learning algorithms to progressively "reveal" their learning curves. The learned policy is then applied to quickly uncover the best algorithm on a new dataset. While other similar approaches, such as Freeze-Thaw, were proposed in the past, using Bayesian optimization, our methodology is, to the best of our knowledge, the first that trains a RL agent to do this task on previous datasets. Using real and artificial data, we show that our new RL-based meta-learning paradigm outperforms Free-Thaw and other baseline methods, with respect to the Area under the Learning curve metric, a form of evaluation of *Any-time learning* (i.e., the capability of interrupting the algorithm at any time while obtaining good performance).

Keywords: Meta-Learning · Learning Curves · Reinforcement Learning.

1 Introduction and related work

Meta-learning in machine learning refers to learning from prior experience on other datasets than the current dataset of interest. There are many meta-learning settings, including learning from **Model Evaluations**, learning from **Task Properties**, and learning from **Prior Models** [31]. In this paper, we address a particular setting of meta-learning in which the goal is to **rapidly** find an algorithm that performs best on a new dataset. Since speed is of the essence, rather than fully training all algorithms, we interrupt (then eventually resume) training. Hence, we allow our meta-algorithm to switch between learning curves.

* The first author contributed most, the others are in alphabetical order of last name. Supported by ANR Chair of Artificial Intelligence HUMANIA ANR-19-CHIA-00222.

Our setting belongs to the family of meta-learning **Model Evaluations** methods, which make use of pre-defined performance measures, e.g. test accuracy and training time. One baseline approach is to select the algorithm performing best on previous datasets, e.g. according to average rank [1, 17]. Other prior art approaches include recommender systems for Meta-learning [7, 22, 23, 27, 29, 35], largely dominated by Collaborative Filtering methods (e.g. Matrix Factorization). In this line of work, ActivMetal [29] has inspired our approach. Our work is mostly in line with [28], casting the problem as a REVEAL game, a subclass of Markov Decision Processes.

Task Properties (meta-features) describe the characteristics of datasets. They may include statistical information, information-theoretic measures, or learned meta-features. In Meta-Regression, regression algorithms are used to predict the performances of algorithms based on the meta-features of the problems (and meta-features of the algorithms). One could estimate a classifier's performance by exploiting relationships between the dataset properties and the classifier's performance [3]. Kopf et al. explored deeper the choices of measurements for dataset characterization [11]. Another work made by Guerra et al. used Support Vector Machines to predict the Performance of Learning Algorithms [8]. In general, meta-regression highly depends on the quality of the meta-features used. Our present approach is not a *Task property* method, since it does not rely on such meta-features, although they could be added in the future.

Learning from **Prior Models** usually focuses on transfer learning and few-shot learning applied to deep learning models. While the former uses models trained on source tasks as starting points to develop models for a new target task, the latter aims at training a good model given very few training examples. Much progress has been made in these settings with some state-of-the-art methods, such as MAML [6], Reptile [24], MetaOptNet [12], and R2-D2 [4]. Our method does not leverage *prior models*, although this could be done in future work.

The setting considered in this paper is **active meta-learning**, where an agent actively requests to train and test algorithms to reveal their performance on a given dataset. We fuse three ideas: (1) that of "active meta-learning" exploited in ActivMetal [29], that of using Reinforcement Learning exploited in [28] by framing the meta-learning problem as a REVEAL game, and that of learning from partial learning curve information used in Freeze-Thaw, proposed for hyper-parameters optimization and model selection [30] (without any meta-learning).

Compared to previous approaches, we gain in speed and accuracy: Both ActivMetal and REVEAL are computationally demanding since they require fully training and evaluating models. Our new method using partially trained models (along the learning curve) is thus more effective. Furthermore, ActivMetal requires multiple computationally expensive matrix factorizations using the entire meta-dataset of past scores. Our method based on pre-trained policies does not require storing and using past scores on other datasets at utilization time. Finally, Freeze-Thaw, which inspired us to use learning curves, relies on heuristic policies derived from human expertise, not trainable agents performing meta-learning, which is the setting considered in this paper. Other learning-curve

based methods [13–15] rely on pairwise comparisons of algorithms, which would not scale well with the number of algorithms and involve"hard-coded" policies (no meta-learning). Our principal contributions are:

1. We introduce **meta-learning environments** using learning curve information with two reward functions specifically designed for Fixed-time learning and Any-time learning. These two types of learning are described in Section 3.1 and Section 3.2.
2. We implement and evaluate various **RL agents and baseline methods** on a meta-dataset from the AutoDL challenge [19] and a novel artificial meta-dataset. We experimentally show that RL agents can "meta-learn" the underlying structure of training meta-datasets to later solve similar learning tasks more efficiently.
3. We propose a **Switching Frequency (SF) metric** to quantify how often an agent pauses running an algorithm and switches to running another one during an episode. This metric is related to the trade-offs between exploitation and exploration. We study the correlation between this metric and the cumulative reward achieved by the agents.

2 Mathematical statement of the problem

2.1 Meta-learning as algorithm recommendation

Meta-learning is learning to learn. In this paper, we consider the algorithm recommendation setting of meta-learning: The goal is to find, from a set of algorithms, the algorithm performing best on a new dataset, given the experience of these algorithms on previous datasets. This experience can be embedded in a meta-dataset.

Definition 1. *(Meta-dataset). A meta-dataset of m algorithms on n datasets can be expressed as a performance matrix P with a size of $(m \times n)$, where column j (for $j = 1, ..., n$) corresponds to algorithm A_j, row i (for $i = 1, ..., m$) corresponds to dataset D_i, and $P(i, j)$ is the performance score of A_j tested on D_i.*

Definition 2. *(1D Meta-Learning Problem). Given a meta-dataset P with a size of $((m-1) \times n)$, a new dataset D_m, and the partial performance information I_m of algorithms on this new dataset D_m (which is progressively revealed at a given cost), the meta-learning problem is to find the best algorithm A_{j^*} for D_m such that:*

$$j^* = \underset{j=1,...,n}{\operatorname{argmax}} P(m, j) \tag{1}$$

From *Definition 1*, we concentrate on **zero-level** meta-learning, as defined in [18]. Meta-learning algorithms are categorized in 3 families, related to the taxonomy of [31] into *Model Evaluation*, *Task Properties*, and *Prior Models*, but based on the *level of information used*:

- **Zero-level meta-learning,** or black-box meta-learning: Only past performances of *Model Evaluations* (e.g., accuracy score on datasets).
- **First-level meta-learning,** or gray-box meta-learning: Performance scores, dataset meta-features (i.e. *Task Properties*) and/or algorithm hyper-parameters.
- **Second-level meta-learning,** or white-box meta-learning: First and second level information is complemented by full knowledge of the datasets and inner functioning of the algorithms (related to the notion of *Prior Models*).

From *Definition 2*, we concentrate on **1D meta-learning**. Meta-learning was divided into 1D meta-learning and 2D meta-learning in [28]. In 1D meta-learning, a search for the best algorithms for a single dataset at a time is performed. In 2D meta-learning, good matches of algorithm-dataset pairs $\{D_i, A_j\}$ are seeked over the 2D score matrix (initialized with many missing values).

2.2 REVEAL games

In this section, we relate meta-learning problems to REVEAL games, which has been previously discussed in [28]. Meta-learning problems, in the recommendation setting introduced in the previous section, can be cast as REVEAL games, a particular class of Markov Decision Processes (MDP), amenable to Reinforcement Learning [28]. Since we will be using this framework, we first briefly recall what REVEAL games are.

In a REVEAL game, the agent's action can only influence the amount of information it can gain, not the underlying data generative process, i.e., the agent's actions have no influence over the course of the "world". Consequently, a good operational test of whether a MDP is a REVEAL game is to find out whether it is possible to pre-compute all states and rewards *a priori*, before the start of a game episode. A simple metaphor for a REVEAL game is a "game board" covered with "cards". Each card is associated with some information. When the game starts, all cards are placed face down, such that the information is hidden from game players or agents. The goal of an agent is to move around the board and reveal the card's information to maximize rewards received in an episode. Examples of REVEAL games include Battleship [32], Mouse in a maze [2], Minesweeper [33], Pacman [34] without ghosts, etc. One example of a game that is not a REVEAL game is the Pacman but with ghosts because the agent's moves affect the motions of the ghosts.

Meta-learning problems can be viewed as REVEAL games where a new dataset corresponds to a new board. An action of the agent on the board is a choice of pair $\{training\ algorithm, dataset\}$ yielding a reward based on the performance achieved by the chosen algorithm on the chosen dataset. Figure 1 shows an overview of how a meta-learning problem is related to a REVEAL game.

As an additional twist, "cards" in REVEAL games can be partially or progressively revealed. This metaphor portrays well the case in which learning machines are progressively trained, and revealing a card step-by-step corresponds to obtaining the next performance of the algorithm after training one more epoch.

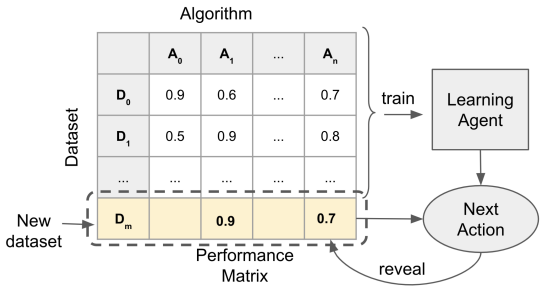


Fig. 1: Meta-learning as a REVEAL game where agents progressively reveal the performance of algorithms on a given new dataset and try to find the best algorithm for that dataset as fast as possible.

Concerning the meta-datasets, this corresponds to adding one more dimension (time or number of epochs) in the meta-dataset performance matrix to store an entire **learning curve** as opposed to a single final score, as explained further in the next section.

3 MetaREVEAL

In this section, we introduce RL-based meta-learning from **learning curves**, which are an essential ingredient for time management in the search for the best performing algorithm. Indeed, training all algorithms fully (to the point of reaching asymptotic training performance) is wasteful, considering that the least promising algorithms can be abandoned early on. Given a limited time budget, it is therefore preferable to probe first the performance of algorithms by training them only a few epochs, then eventually train more certain algorithms, perhaps switching back and forth as more of the learning curve is revealed. The goal of our RL agent is to uncover an optimal strategy, monitoring exploration and exploitation.

We investigate two settings, which have implications in the time management and the exploration-exploitation tradeoff: **Fixed-time Learning** and **Any-time Learning**. In the Fixed-time Learning setting, an overall time budget is given, and the goal of the agent is to find the best algorithm before the time is out. The agent can therefore explore freely within this time budget, without the need to find a good solution early on. In contrast, in the Any-time Learning setting, the agent can be stopped and judged for its performance at any time. There is therefore pressure on the agent that it finds a good solution early on and keeps improving it incrementally. Figure 2 shows a concrete example of two algorithms competing to show the difference between Fixed-time learning and Any-time learning settings. We introduce meta-learning environments designed for each setting.

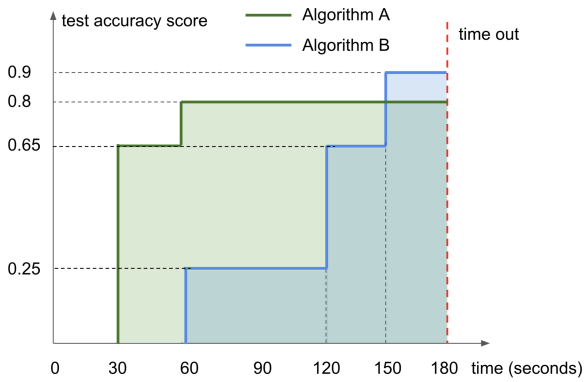


Fig. 2: **Fixed-time learning versus Any-time learning**. In *Fixed-time learning*, within a time budget $\mathcal{T} = 180$ seconds, algorithm B obtained a higher final test accuracy score (0.9) than algorithm A (0.8), making algorithm B the winner in this setting. However, in *Any-time learning*, we use the Area under the Learning Curve (ALC) metric to compare. Thus, algorithm A clearly outperformed algorithm B in this setting. If both algorithms were to stop at any point in time, algorithm A would most likely achieve better performance than algorithm B, indicating that algorithm A possesses a greater capacity for any-time learning.

3.1 Fixed-time Learning

In the Fixed-time Learning scenario, an agent is given a total time budget $\mathcal{T}$ to be spent on training any algorithm in the algorithm set A. The agent's goal is to find the best algorithm for a given dataset within the time budget. $\mathcal{T}$ may be varied to have the agents exposed to different conditions (e.g., $\mathcal{T}$ is drawn uniformly from a pre-defined set of time budgets).

Definition 3. *(State). A state s_t is a matrix of dimensions $2 \times n$, which consists of two channels represented by two vectors with the same length of n. The first vector T stores the amount of time that has been spent so far for each algorithm and the second vector V represents the current test score of each algorithm (current value on the learning curve) in the current episode:*

$$T = [t_j] \ for \ j = 1, ..., n \tag{2}$$

$$V = [v_j] \ for \ j = 1, ..., n \tag{3}$$

where n is the number of algorithms. At the beginning of an episode, all values of t_j are initialized to 0 and v_j to -1, to indicate that performances of algorithms A_j have not been revealed yet.

Definition 4. *(Action). An action is to start/continue training an algorithm in a fixed amount of time Δt (pre-defined by the environment creator, e.g., $\Delta t = 10$*

seconds) and then make predictions on the test data to receive the next test score.[5] *For simplicity, we define an action by the corresponding algorithm index:*

$$a_t = j, \tag{4}$$

where j is the index of the algorithm A_j which is going to be trained and tested next. Once the action a_t is done, t_j and v_j in the state are updated to form the next state.

Definition 5. *(Fixed-time Learning Reward Function). A shaping reward function based on performance improvement, which gives rewards more frequently to the agent and lets the agent knows that it is getting better and getting closer to the best algorithm:*

$$r(t) = V^*(t) - V^*(t - \Delta t) \ , \tag{5}$$

where $V^*(t)$ and $V^*(t - \Delta t)$ are the best algorithm performances found in this step and the previous step respectively:

$$V^*(t) = \max_{k \leq t} V(k) \ , \tag{6}$$

$$V^*(0) = 0 \ , \tag{7}$$

Definition 6. *(Termination condition). An episode ends when $\mathcal{T}$ is exhausted.*

At the end of the episode, the **cumulative reward** is equal to $V^*(\mathcal{T}) - V^*(0) = V^*(\mathcal{T})$, **the score of the best algorithm found within the time budget $\mathcal{T}$**. Our agent therefore implements a meta-algorithm whose (meta-)learning curve is given by $V^*(t)$, but it is judged only by its end result.

3.2 Any-time Learning

In the Any-time Learning setting, we want to encourage the agent to obtain a meta-learning curve, which is steep at the beginning, i.e., to uncover **good algorithms as fast as possible**. In this way, even if the agent is stopped early, we will get as good performance as possible, thus obtaining Any-time Learning capabilities.

States, actions, time budgets, and termination conditions are defined similarly as in Fixed-time learning. We designed a specific reward function for this type of learning:

[5] Agents' action are based on the test performance V_j, which is assumed to be accurate and a good approximation of the generalization error (i.e. we assume large test sets and very small error bars). In this work, we focus on meta-learning, hence, the problem of possibly "overfitting/underfitting the test set" is not discussed in this paper and left for future works.

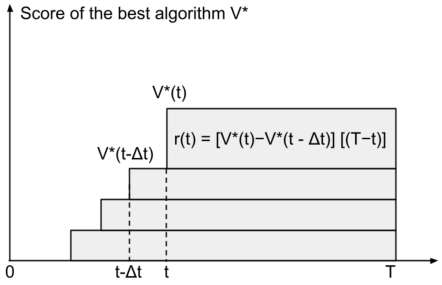


Fig. 3: Computation of the ALC.

Definition 7. *(Any-time Learning Reward Function). This function puts more emphasis on performance improvement at the beginning of an episode. The reward is defined by:*

$$r(t) = [V^*(t) - V^*(t - \Delta t)]\ [(\mathcal{T} - t)] \tag{8}$$

The weight $[(T - t)]$ *is the only difference compared to the reward function in Fixed-time Learning. If we scale the x-axis logarithmically then the reward function becomes:*

$$r(t) = [V^*(t) - V^*(t - \Delta t)]\ [(1 - \tilde{t})] \tag{9}$$

with

$$\tilde{t} = \frac{\log(1 + t/t_0)}{\log(1 + \mathcal{T}/t_0)} \tag{10}$$

The larger t_0 is, the more important the beginning of the learning curve is. If $\mathcal{T} \gg t_0$, then $\tilde{t} \to 0$ and the reward function becomes equivalent to that of Fixed-time Learning (Equation 5). In our experiments, t_0 is set to 50 (seconds).

At the end of the episode, the **cumulative reward** will be the **Area under the Learning Curve** (ALC) within the time budget $\mathcal{T}$. The computation of the cumulative reward can be carried out by integrating the learning curve using horizontal rectangles, in the style of Lebesgue integrals (Figure 3). The ALC metric was used in the AutoDL challenge with the same purpose of emphasizing Any-time Learning [20, 21].

4 Experiments and Results

In this section, we first describe how the meta-datasets used in our experiments are obtained. Then, we discuss the experimental results and findings from running the implemented RL agents and baselines on the meta-datasets. The code for reproducing the experiments is available on our Github repository [6].

[6] https://github.com/hungnm2008/metaREVEAL.git

4.1 Meta-datasets

We use learning curves collected from the AutoDL Challenge [20] to build our first meta-dataset. However, since this meta-dataset is quite small and not complete, we generate artificial learning curves using parameterized sigmoid functions. Both of them will be discussed in detail below.

Learning curves from the AutoDL challenge [20]. This meta-dataset is made by the predictions of 13 Automated Deep Learning algorithms on 66 datasets in the AutoDL Challenge [20]. These algorithms include top 9 algorithms and 4 baselines competed in the challenge. The fact that we use a meta-dataset from the AutoDL challenge might cause a misunderstanding that our method is comparable to the methods competed in the challenge. However, we are doing one level up, meta-learning from past performances of these AutoDL methods. The score used in the challenge is the Area under the Learning Curve (ALC) computed using the Normalized Area Under ROC Curve (NAUC) scores gathered during the learning process. The NAUC score is obtained by making predictions on the test set at any timestamp during 20 minutes. One difficulty is that each algorithm in the meta-dataset made predictions at different timestamps while our agents do it regularly every Δt seconds. Thus, some data points on the learning curves at desired timestamps are not available to the agents. In this case, the learning curve's most recent value (data point) will be returned. The learning curves obtained from the AutoDL challenge are not monotonic. During the competition, some algorithms' performances decrease after some time of training.

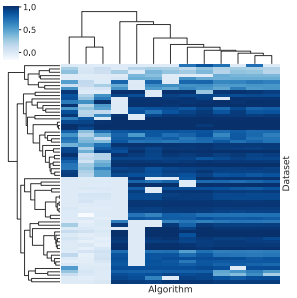


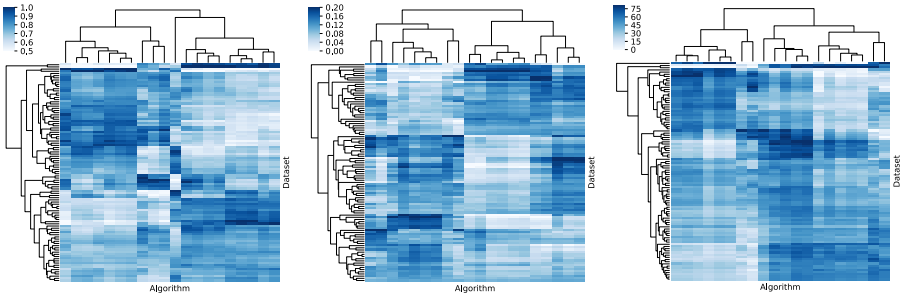
Fig. 4: (AutoDL meta-dataset) Hierarchically-clustered heatmap showing *nauc_mean* score of algorithms on datasets in the AutoDL meta-dataset. The figure demonstrates that there is some structure in the data, which can potentially be exploited by the learning agents. The 'blocks' indicate that some algorithms are more suitable for solving some dataset tasks. This is some transferable knowledge that the agent can learn.

Artificial Learning Curves. We have created an artificial meta-dataset that contains learning curves of 20 algorithms on 100 datasets. The purpose of creating

these curves is to have a meta-dataset with a larger size, no missing data, and containing underlying structure indicating some groups of algorithms are good for some groups of datasets. We assume these learning curves have the S-shape-like sigmoid curves, hence, they are monotonically increasing by definition. Each learning curve of algorithm A_j on dataset D_i is a sigmoid function defined by three parameters a, b and c as follows:

$$lc_j^i = \frac{a}{1 + e^{-b*(x-c)}} \tag{11}$$

These parameterized functions allow us to experiment with various learning curves, by adjusting their asymptotic performance (specified by a), increasing rate (specified by b), and "warm-up" time (specified by c). Values of each parameter a, b, and c are shown in matrices in Figure 7. Each matrix was constructed from a matrix factorization, which means it was obtained as a product of three matrices $U\Sigma V$ where U and V are random orthogonal matrices and Σ is a diagonal matrix of "singular values". The values are then scaled to desired range for each parameter.



(a) Parameter a (a) Parameter b (a) Parameter c

Fig. 7: (Artificial meta-dataset) Hierarchically-clustered heatmaps showing values of the three parameters used to build the artificial learning curves. Blocks appear, revealing that some groups of algorithms have correlated parameter values on groups of datasets (e.g. learning curve asymptotic value, controlled by the parameter a). The learning agents are expected to learn such properties and output an effective search strategy.

4.2 Reinforcement Learning Agents

In Reinforcement Learning, the goal of an agent is to find a policy that maximizes expected (discounted) rewards. Reinforcement Learning methods can be categorized into *value-based*, *policy-based*, and *hybrid* methods. Value-based methods learn a value function that is used to evaluate a state or a state-action pair. Then the policy is derived directly from the value function. In contrast, policy-based

methods explicitly learn a representation of a policy and keep updating it during learning. Many hybrid approaches learn both value function and a policy simultaneously gain great success in RL. Actor-Critic is a well-known architecture used in these hybrid approaches, where the "Critic" computes estimated values and the "Actor" updates the policy according to the values provided by the Critic. We have chosen a diverse group of RL agents due to their characteristics and their strategies to be evaluated in our experiments:

Double Deep Q Networks (DDQN) [10]: value-based, off-policy, ε-exploration strategy.

Soft Actor-Critic (SAC) [9]: hybrid (actor-critic architecture), off-policy, entropy-based exploration strategy.

Proximal Policy Optimization (PPO) [26]: hybrid (actor-critic architecture), on-policy, entropy-based exploration strategy.

4.3 Baselines

We compare the performance of RL agents with established baseline methods, which allow us to select an algorithm that should perform well on a novel dataset.

Freeze-Thaw Bayesian Optimization. [30]. This method aims at efficiently searching for good model hyper-parameters. It maintains a set of "frozen" models that are partially trained and takes advantage of the partial information to decide which ones to "thaw" and resume training. This avoids spending too much time on bad models, and only promising models should be exploited more. Freeze-Thaw requires hyper-parameters to be able to search for good models. However, we are working on Zero-level meta-learning, hyper-parameters are not considered in choosing an algorithm (model). We made some changes to make the Freeze-Thaw method able to run in our experiments. The performance matrix has been arranged so that similar algorithms are placed together. Then we use the algorithm index as a "hyper-parameter" that describes and represents the locality of the algorithm in the searching space.

Average Rank. Inspired by these works [1, 5, 16, 17], we build a global ranking of algorithms across training datasets. This is done in the training phase by running all algorithms for all training datasets and taking the average of their ranks to form the final ranking. The global average rank for each algorithm A_j is obtained by:

$$global\_rank(A_j) = \frac{\sum_{i=1}^{D_{train}} rank_j^i}{D_{train}} \tag{12}$$

where D is the number of training datasets, and $rank_j^i$ is the rank of algorithm A_j on the dataset D_i. Given a new test dataset, only the algorithm with the highest global rank is selected to run with the entire time budget $\mathcal{T}$. This baseline is very time-consuming in practice since it needs to try all algorithms on all datasets in training.

Best on Samples. This baseline is adapted from [25] by using a fixed amount of time $t_{sampling}$ instead of a fixed number of samples. At the beginning of each episode, it trains each algorithm with the same amount of time $t_{sampling}$ and then

selects the one that performed best within $t_{sampling}$ to run with the remaining time budget. In our experiments, we set $t_{sampling} = \Delta t$.

Random. This baseline performs a random search over the algorithm space. Each action is to randomly choose an algorithm for training and testing within Δt. This baseline has a **very large variance**. When we report results, we first average results over 5 trials of the random search method, therefore reducing its variance, and report average performance. One needs to bear in mind though that this is just for comparison purposes and in *not a realistic setting* (because in practice one would not average over several runs, this is impossible because once the performances of algorithms are revealed, one cannot take them back).

4.4 Setup and Evaluation Metrics

We train the agents in two learning scenarios: **Fixed-time Learning** and **Any-time Learning** using two meta-datasets: the **AutoDL meta-dataset** and the **Artificial meta-dataset**. Since these meta-datasets are quite small, we use k-Fold Cross-Validation with $k = 4$ to train and test the agents.[7]

To compare the agents, we use two metrics: **Average Cumulative Reward** and **Average Switching Frequency** (defined in Definition 8). The means of cumulative reward and switching frequency are calculated for each test fold. The final average cumulative reward, average switching frequency, and their corresponding standard deviations are computed over all folds.

Definition 8. *(Switching Frequency). We proposed a Switching Frequency (SF) metric for evaluating how frequently an agent switches between algorithms. In an episode, the SF value of an* $agent_k$ *is defined as:*

$$SF(agent_k) = \frac{\sum_{t=1}^{\mathcal{T}} \mathbb{1}_{a_t \neq a_{t-\Delta t}}}{\mathcal{T}/\Delta t} \tag{13}$$

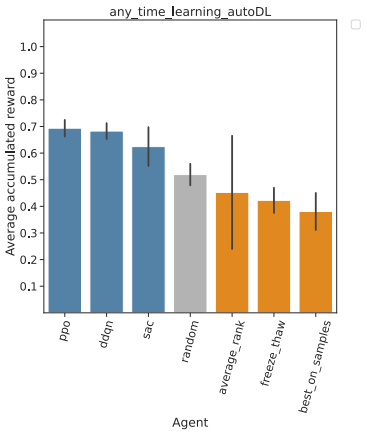
with $\mathcal{T}$ *is the total time budget,* Δt *is the amount of time spent for an algorithm in one step.*

4.5 Results

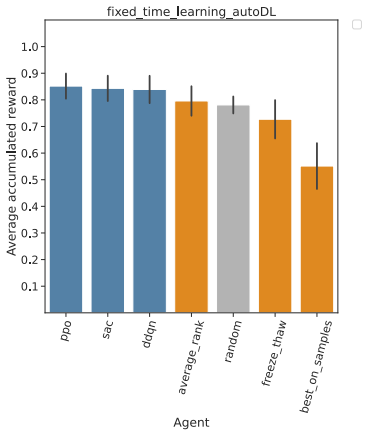
We discuss our experimental results in two learning scenarios and focus on two points: (i) the average cumulative reward and (ii) the correlation between average cumulative reward and average switching frequency.

Any-time learning, (Figure 8a, 8c, 10a, 9c). The results indicate that a good strategy to be successful in Any-time learning is to bet at the beginning on algorithms that performed well on past datasets and stick to them to climb the learning curve fast, then start exploring. This is illustrated by the *ppo* agent, which obtained the highest cumulative reward, followed by other RL agents.

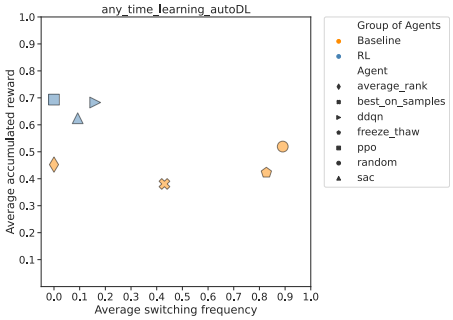
[7] This violates the assumption that we have large test sets made earlier and is a limitation of this mode of evaluation.



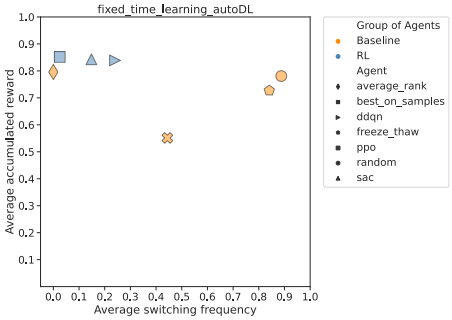
(a) Cumulative reward in *Any-time learning*



(b) Cumulative reward in *Fixed-time learning*

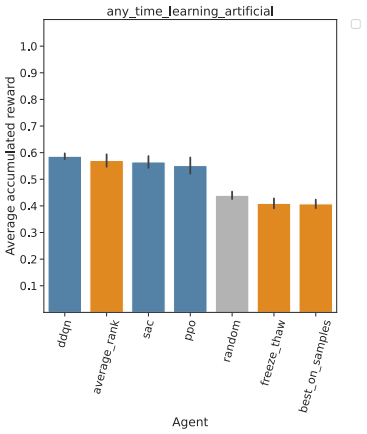


(c) Correlation between Switching frequency and Cumulative reward in *Any-time learning*

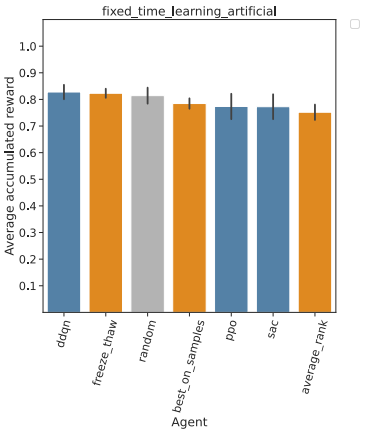


(d) Correlation between Switching frequency and Cumulative reward in *Fixed-time learning*

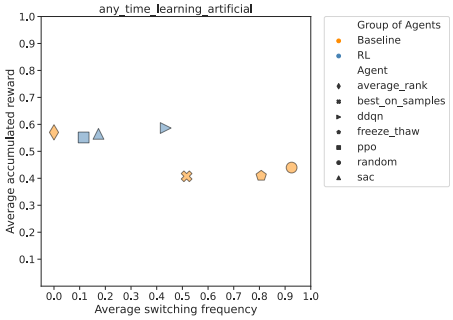
Fig. 8: Experimental results on the **AutoDL meta-dataset**. Time budget $\mathcal{T}$ is drawn uniformly from [200, 300, 400, 500] (seconds) and Δt is set to 10 (seconds). From the bar plots, it can be seen that RL agents (in blue color) outperformed baselines in both learning settings, more significantly in the *Any-time learning*. These RL agents tend to have very low switching frequency, as shown in the scatter plots. In *Any-time learning*, the *average_rank* agent has a high error bar because the algorithm chosen by the agent does not consistently perform well at the beginning of an episode. To stress that the "random" agent is an average over 5 random runs, we highlight its bar in gray. Its total variability is higher than suggests the 4-fold error bar represented in sub-figures (a) and (b).



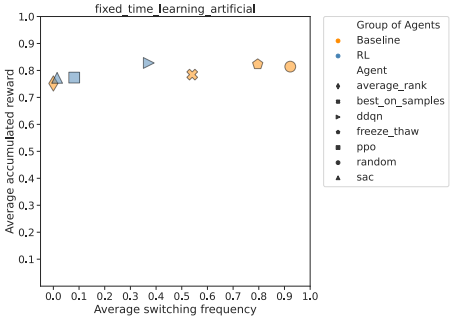
(a) Cumulative reward in *Any-time learning*



(b) Cumulative reward in *Fixed-time learning*



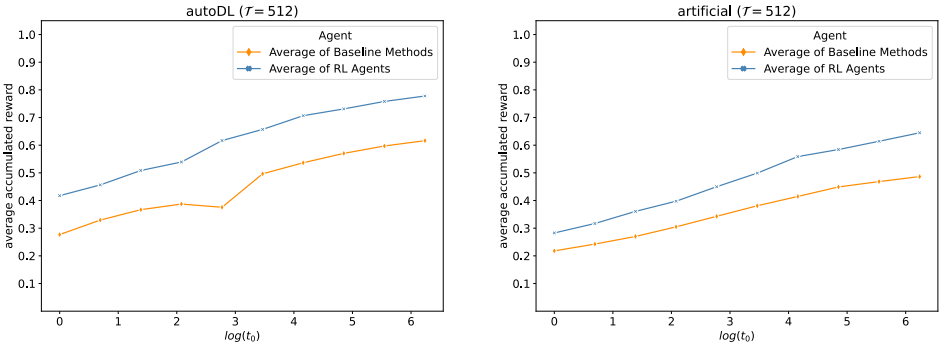
(c) Correlation between Switching frequency and Cumulative reward in *Any-time learning*



(d) Correlation between Switching frequency and Cumulative reward in *Fixed-time learning*

Fig. 9: Experimental results on the **Artificial meta-dataset**. Time budget $\mathcal{T}$ is drawn uniformly from [500, 700, 900, 1100] (seconds) and Δt is set to 20 (seconds). In *Any-time learning*, RL agents and *average_rank* agent achieved similar cumulative rewards and higher than the rest of the baselines. These agents do not switch algorithms frequently as the others, as shown in the scatter plots. In *Fixed-time learning*, within the given time budget $\mathcal{T}$, all agents performed almost the same.

They are among the algorithms with the lowest switching frequency. Their low switching frequency can explain their success at the beginning of the learning curve, as they favor more exploitation than exploration. In contrast, the policy of *best_on_samples* and *freeze_thaw* forces agents to try each algorithm at least once at the beginning (train and test the algorithm in Δt first seconds). Thus, if they manage to find the best algorithm, this should happen only near the end of the episode, which makes it less valuable in the Any-time learning setting. This explains why they performed worst in the Any-time learning setting in both meta-datasets. We vary the value of t_0 to investigate its influence on agents' performances. More precisely, the value of t_0 is drawn from the set: $[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]$, while the time budget $\mathcal{T}$ is set to 512. The results of this experiment are shown in Figure 10.



(a) AutoDL meta-dataset (b) Artificial meta-dataset

Fig. 10: Tuning hyperparameter t_0 in Any-time learning. We compare the average accumulated reward of RL agents (in blue) and baseline methods (in orange). The x-axis shows the value of t_0 on a log scale. In Any-time learning, changing t_0 leads to changing the reward function. Thus, the purpose of these figures is not to show that agents achieve higher rewards when t_0 increases. The key finding is that the performance difference between RL agents and baseline methods gets *larger* as t_0 increases, indicating that RL agents can learn better when we emphasize more on the any-time learning capability (with a high value of t_0). The difference is more obvious in the Artificial meta-dataset, which can be explained by the chosen time budget $\mathcal{T}$. In the AutoDL meta-dataset, the time budget $\mathcal{T}$ of 512 is large enough for the baseline methods to maintain the difference with the RL agents when t_0 increases, which is not the case in the Artificial meta-dataset.

Fixed-time learning, (Figure 8b, 8d 10b, 9d). In both meta-datasets, the winner is a RL agent. In the AutoDL meta-datasets, RL agents achieved higher cumulative rewards than the baselines. However, in the Artificial meta-dataset, there was no significant difference between all agents. Within the given time budget $\mathcal{T}$, all agents managed to find a good algorithm at the end. This emphasizes

the fact that learned policies to manage time budget are mostly beneficial in the Any-time learning setting, where monitoring the exploration-exploitation tradeoff is critical.

Comparison between datasets. The AutoDL meta-dataset has a clear block structure in the vertical (dataset) direction, which means there is some algorithm ranking transferable across datasets in the same group. The fact that RL agents outperform others in both any-time and fixed-time learning indicates that the RL agents successfully meta-learn those rankings, which let them finds the best algorithms for similar datasets with less exploration than other agents that cannot meta-learn (best on samples, freeze-thaw, random or average-rank that uses the same ranking for all datasets), this make RL agents shine even more in any-time learning. The structure of the artificial dataset is more subtle and harder to learn, as it appears. More work needs to be done to fully elucidate this.

5 Conclusion

Meta-learning can be viewed as a sequential decision-making problem where an agent selects and trains algorithms progressively for a given dataset. The goal is to find the algorithm performing best within a fixed amount of time (Fixed-time learning) or at any time (Any-time learning). We have proposed learning environments that allow RL agents to learn policies (as opposed to hard-coding them) using past experiences on similar datasets (meta-learning). Trained agents operate by training algorithms step by step, thus revealing their learning curves. By doing so, they create a meta-learning curve from the performance of the best algorithm revealed so far.

Both knowledge from past dataset experience (captured in the learned agent policy), and current information on the dataset at hand (embedded in the current state) are used by agents to make decisions. By leveraging partial learning curve information, an agent may stop training algorithms that are not promising and concentrate hardware resources on an algorithm that has more potential to be the best-performing one on the given dataset, which would save a huge amount of time. In both Any-time and Fixed-time learning, the RL agents successfully acquired two important skills: (1) Meta-learning, which allows trained RL agents to identify good algorithms with less exploration for new datasets thanks to the previous training, this is more prominent in Any-time learning; (2) Exploration-exploitation trade-off, which explains the different policies they derive in Fixed-time and Any-time settings. In Any-time learning, RL agents obtained a higher cumulative reward (Area under Learning Curve) than the baselines. In contrast, in Fixed-time Learning, all methods obtain a similar cumulative reward (best final score). From a RL perspective, this outlines that the Any-time learning problem offers more possibilities to learn clever policies monitoring the exploration-exploitation trade-off. When the number of algorithms increases, MetaREVEAL with RL agents would show more advantages over the baselines in terms of computational time (e.g. the average_rank agent needs to try all algorithms on the training

datasets). In addition, if we have numerous sets of hyperparameters of the same model, we can adapt MetaREVEAL to work with continuous action spaces, which would be more efficient in searching for the optimal set of hyperparameters.

Future work includes performing more experiments on the artificial data, varying its parameter settings, to elucidate relationships between data structure and policy learning. Work is also under way to apply our method to other real-world meta-datasets. Systematic experiments must be performed to vary values for the parameters of our meta-learning RL environments: $\mathcal{T}$ and t_0. Last but not least, it would be interesting to do some theoretical research and propose RL methods more dedicated to the meta-learning REVEAL game setting and investigate the computational complexity of such methods. We would also like to extend this work to the First-level meta-learning, Second-level meta-learning, and 2D meta-learning problems.

A Appendix A - Full Experimental Results

Table 1: **Average Cumulative Reward** achieved by each agent in each setting. Since we are using k-fold cross-validation (k=4), we compute the mean of cumulative reward in each test fold. The final average cumulative reward and standard deviation (represented by the error bar) are computed by taking the average across the test folds (4 test folds in total). Bold numbers indicate the winners in each setting. RL agents performed better than the baselines in all settings, but more remarkably in *Any-time learning*.

		Any-time learning (**acc_reward** = $ALC(\mathcal{T})$)		**Fixed-time learning** (**acc_reward** = $V^*(\mathcal{T})$)	
		AutoDL	**Artificial**	**AutoDL**	**Artificial**
RL agents	**ddqn**	0.68 ± 0.03	$\mathbf{0.59 \pm 0.01}$	0.84 ± 0.06	$\mathbf{0.83 \pm 0.03}$
	sac	0.62 ± 0.08	0.56 ± 0.03	0.84 ± 0.06	0.77 ± 0.05
	ppo	$\mathbf{0.69 \pm 0.04}$	0.55 ± 0.04	$\mathbf{0.85 \pm 0.05}$	0.77 ± 0.06
Baselines	**freeze-thaw**	0.42 ± 0.05	0.41 ± 0.02	0.73 ± 0.08	0.82 ± 0.02
	average_rank	0.45 ± 0.25	0.57 ± 0.03	0.80 ± 0.06	0.75 ± 0.03
	best_on_samples	0.38 ± 0.08	0.41 ± 0.02	0.55 ± 0.10	0.78 ± 0.02
	random	0.52 ± 0.05	0.44 ± 0.02	0.78 ± 0.04	0.81 ± 0.03

Acknowledgements

We would like to thank Adrien Pavao and Michael Vaccaro for supplying us with the AutoDL meta-dataset.

References

1. Abdulrahman, S., Brazdil, P., van Rijn, J., Vanschoren, J.: Speeding up algorithm selection using average ranking and active testing by introducing runtime. Machine Learning **107** (01 2018)
2. Alexander Smith: Mouse in a maze, https://videogamehistorian.wordpress.com/tag/mouse-in-a-maze/, [Online; accessed 06-July-2021]
3. Bensusan, H., Kalousis, A.: Estimating the predictive accuracy of a classifier. In: De Raedt, L., Flach, P. (eds.) Machine Learning: ECML 2001. pp. 25–36. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
4. Bertinetto, L., Henriques, J.F., Torr, P., Vedaldi, A.: Meta-learning with differentiable closed-form solvers. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=HyxnZh0ct7
5. Brazdil, P.B., Soares, C.: A comparison of ranking methods for classification algorithm selection. In: López de Mántaras, R., Plaza, E. (eds.) Machine Learning: ECML 2000. pp. 63–75. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)
6. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: Proceedings of the 34th International Conference on Machine Learning - Volume 70. p. 1126–1135. ICML'17, JMLR.org (2017)
7. Fusi, N., Sheth, R., Elibol, M.: Probabilistic matrix factorization for automated machine learning. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 31. Curran Associates, Inc. (2018), https://proceedings.neurips.cc/paper/2018/file/b59a51a3c0bf9c5228fde841714f523a-Paper.pdf
8. Guerra, S.B., Prudêncio, R.B.C., Ludermir, T.B.: Predicting the performance of learning algorithms using support vector machines as meta-regressors. In: Kůrková, V., Neruda, R., Koutník, J. (eds.) Artificial Neural Networks - ICANN 2008. pp. 523–532. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
9. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 1861–1870. PMLR (10–15 Jul 2018), http://proceedings.mlr.press/v80/haarnoja18b.html
10. Hasselt, H.v., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. p. 2094–2100. AAAI'16, AAAI Press (2016)
11. Kopf, C., Taylor, C.: Meta-analysis: From data characterisation for meta-learning to meta-regression (2000)
12. Lee, K., Maji, S., Ravichandran, A., Soatto, S.: Meta-learning with differentiable convex optimization. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) pp. 10649–10657 (2019)
13. Leite, R., Brazdil, P.: Predicting relative performance of classifiers from samples. In: Proceedings of the 22nd International Conference on Machine Learning. p. 497–503. ICML '05, Association for Computing Machinery, New York, NY, USA (2005), https://optdoi.org/10.1145/1102351.1102414
14. Leite, R., Brazdil, P.: An iterative process for building learning curves and predicting relative performance of classifiers. In: Proceedings of the Aritficial Intelligence 13th Portuguese Conference on Progress in Artificial Intelligence. p. 87–98. EPIA'07, Springer-Verlag, Berlin, Heidelberg (2007)

15. Leite, R., Brazdil, P.: Active testing strategy to predict the best classification algorithm via sampling and metalearning. p. 309–314. IOS Press, NLD (2010)
16. Leite, R., Brazdil, P., Vanschoren, J.: Selecting classification algorithms with active testing. vol. 7376, pp. 117–131 (07 2012)
17. Lin, S.: Rank aggregation methods. Wiley Interdisciplinary Reviews: Computational Statistics **2**, 555 – 570 (09 2010)
18. Liu, Z., Guyon, I.: Asymptotic Analysis of Meta-learning as a Recommendation Problem. In: Meta-learning Workshop @ AAAI 2021. Virtual, Canada (Feb 2021)
19. Liu, Z., Pavao, A., Xu, Z., Escalera, S., Ferreira, F., Guyon, I., Hong, S., Hutter, F., Ji, R., Junior, J.C., Li, G., Lindauer, M., Zhipeng, L., Madadi, M., Nierhoff, T., Niu, K., Pan, C., Stoll, D., Treger, S., Jin, W., Wang, P., Wu, C., Xiong, Y., Zela, A., Zhang, Y.: Winning solutions and post-challenge analyses of the chalearn autodl challenge 2019. IEEE Transactions on Pattern Analysis and Machine Intelligence pp. 1–1 (2021)
20. Liu, Z., Pavao, A., Xu, Z., Escalera, S., Ferreira, F., Guyon, I., Hong, S., Hutter, F., Ji, R., Nierhoff, T., Niu, K., Pan, C., Stoll, D., Treguer, S., Wang, J., Wang, P., Wu, C., Xiong, Y.: Winning solutions and post-challenge analyses of the ChaLearn AutoDL challenge 2019. IEEE Transactions on Pattern Analysis and Machine Intelligence p. 17 (2020)
21. Liu, Z., Xu, Z., Rajaa, S., Madadi, M., Junior, J.C.S.J., Escalera, S., Pavao, A., Treguer, S., Tu, W.W., Guyon, I.: Towards automated deep learning: Analysis of the autodl challenge series 2019. In: Escalante, H.J., Hadsell, R. (eds.) Proceedings of the NeurIPS 2019 Competition and Demonstration Track. Proceedings of Machine Learning Research, vol. 123, pp. 242–252. PMLR (08–14 Dec 2020), http://proceedings.mlr.press/v123/liu20a.html
22. Misir, M., Sebag, M.: Alors: An algorithm recommender system. Artif. Intell. **244**, 291–314 (2017)
23. Misir, M., Sebag, M.: Algorithm selection as a collaborative filtering problem (12 2013)
24. Nichol, A., Schulman, J.: Reptile: a scalable metalearning algorithm (03 2018)
25. Petrak, J.: Fast subsampling performance estimates for classification algorithm selection. In: Proceedings of the ECML-00 Workshop on Meta-Learning: Building Automatic Advice Strategies for Model Selection and Method Combination. pp. 3–14 (2000)
26. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms (07 2017)
27. Stern, D., Samulowitz, H., Herbrich, R., Graepel, T., Pulina, L., Tacchella, A.: Collaborative expert portfolio management. vol. 1 (12 2010)
28. Sun-Hosoya, L.: Meta-Learning as a Markov Decision Process. Theses, Université Paris Saclay (COmUE) (Dec 2019), https://hal.archives-ouvertes.fr/tel-02422144
29. Sun-Hosoya, L., Guyon, I., Sebag, M.: Activmetal: Algorithm recommendation with active meta learning. In: IAL@PKDD/ECML (2018)
30. Swersky, K., Snoek, J., Adams, R.: Freeze-thaw bayesian optimization (06 2014)
31. Vanschoren, J.: Meta-learning: A survey. ArXiv **abs/1810.03548** (2018)
32. Wikipedia contributors: Battleship (game), https://en.wikipedia.org/wiki/Battleship_(game), [Online; accessed 06-July-2021]
33. Wikipedia contributors: Minesweeper (video game), https://en.wikipedia.org/wiki/Minesweeper_(video_game), [Online; accessed 06-July-2021]
34. Wikipedia contributors: Pac-man, https://en.wikipedia.org/wiki/Pac-Man, [Online; accessed 02-July-2021]

35. Yang, C., Akimoto, Y., Kim, D.W., Udell, M.: Oboe: Collaborative filtering for automl model selection. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. p. 1173–1183. KDD '19, Association for Computing Machinery, New York, NY, USA (2019), https://optdoi.org/10.1145/3292500.3330909