Optimizing Data Acquisition to Enhance Machine Learning Performance

Tingting Wang
tingting.wang@student.rmit.edu.au
RMIT University

Shixun Huang
shixunh@uow.edu.au
The University of Wollongong

Zhifeng Bao*
zhifeng.bao@rmit.edu.au
RMIT University

J. Shane Culpepper
s.culpepper@uq.edu.au
The University of Queensland

Volkan Dedeoglu
volkan.dedeoglu@csiro.au
Data61, CSIRO

Reza Arablouei
reza.arablouei@csiro.au
Data61, CSIRO

ABSTRACT

In this paper, we study how to acquire labeled data points from a large data pool to enrich a training set for enhancing supervised machine learning (ML) performance. The state-of-the-art solution is the clustering-based training set selection (CTS) algorithm, which initially clusters the data points in a data pool and subsequently selects new data points from clusters. The efficiency of CTS is constrained by its frequent retraining of the target ML model, and the effectiveness is limited by the selection criteria, which represent the state of data points within each cluster and impose a restriction of selecting only one cluster in each iteration. To overcome these limitations, we propose a new algorithm, called CTS with incremental estimation of adaptive score (IAS). IAS employs online learning, enabling incremental model updates by using new data, and eliminating the need to fully retrain the target model, and hence improves the efficiency. To enhance the effectiveness of IAS, we introduce adaptive score estimation, which serves as novel selection criteria to identify clusters and select new data points by balancing trade-offs between exploitation and exploration during data acquisition. To further enhance the effectiveness of IAS, we introduce a new adaptive mini-batch selection method that, in each iteration, selects data points from multiple clusters rather than a single cluster, hence eliminating the potential bias due to using only one cluster. By integrating this method into the IAS algorithm, we propose a novel algorithm termed IAS with adaptive mini-batch selection (IAS-AMS). Experimental results highlight the superior effectiveness of IAS-AMS, with IAS also outperforming other competing algorithms. In terms of efficiency, IAS takes the lead, while the efficiency of IAS-AMS is on par with that of the existing CTS algorithm.

PVLDB Reference Format:
Tingting Wang, Shixun Huang, Zhifeng Bao, J. Shane Culpepper, Volkan Dedeoglu, and Reza Arablouei. Optimizing Data Acquisition to Enhance Machine Learning Performance. PVLDB, 17(6): 1310-1323, 2024.
doi:10.14778/3648160.3648172

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/rmitbggroup/da-ml.

1 INTRODUCTION

Machine learning (ML) has become standard practice in modern data analytics and is integral to extracting new insights from data and informing decision-making. The effectiveness and versatility of contemporary ML models have resulted in extensive adoption of these techniques in various application domains, such as image recognition [32], natural language processing [40]. The remarkable advances seen in ML rely heavily on the quality of training data [45]. Therefore, the challenge of data acquisition to enhance ML performance, which involves selecting high-quality training data from a data pool to improve the performance of an ML model, has recently attracted considerable attention [13, 15, 34, 60, 63].

Some recent studies assess the potential of data to enhance ML performance using indirect indicators (e.g., freshness [63], truthfulness [15], and novelty [34]), rather than directly quantifying the performance improvements produced by including the new data. Other studies [15, 34, 60] focus on data acquisition from a single distribution, which is rarely useful to content providers creating large data pools.

Our Scenario & Problem. With the goal of directly improving model performance for a target ML model, we address the broader scenario where a data pool contains data with a variety of distributions [13]. Conceptually, this can be characterized as a two-step process. The first step is to collect a large amount of labeled data from various sources and create a data pool using readily available tools. Several Web APIs (e.g., NYU Auctus REST API [2] and Google dataset search API [8]) can be used to aggregate datasets. By using techniques such as schema alignment [13], these datasets can be harmonized into a unified dataset, constituting the data pool. In the second step, a supervised ML model, which may have been trained on preliminary data, comes into focus. Our objective is to address the problem of data acquisition to enhance ML performance, referred to as DA-ML and defined as: Given a supervised ML model designed for a specific task, an initial training set, and a data pool, the goal is to select a subset of data points from the data pool to enrich the training set to maximize performance improvements of the model trained on the enriched training set.

State-of-the-art solution. A pioneering effort to address the DA-ML problem is presented in [13], which has introduced the state-of-the-art Clustering-based Training set Selection (CTS) algorithm. The framework of CTS consists of the two steps shown in Fig. 1. In

[*]The corresponding author

Proceedings of the VLDB Endowment, Vol. 17, No. 6 ISSN 2150-8097.
doi:10.14778/3648160.3648172

the first step, the data points in the data pool, which may follow different distributions, are grouped into clusters. In the second step, data points are iteratively selected from the clusters.

Limitations and challenges. While CTS has demonstrated merit in [13], there are three limitations: (L1) *Frequent retraining overhead*: It requires frequent retraining of the target ML model to evaluate the performance using the new training set in each iteration. This significantly degrades the efficiency of the algorithm. (L2) *Exploitation-exploration score marginality*: The use of an exploitation-exploration score to select a single cluster for sampling a mini-batch, helps prevent local optima. However, the improvements in effectiveness can be marginal as the score treats information from past iterations as equally significant, disregarding crucial factors such as recency and ignoring changes to the current cluster state. (L3) *Single-cluster sampling bias*: In each iteration, only one cluster is selected when sampling data points. This can reduce algorithm effectiveness since, if a cluster fails to improve effectiveness in an iteration, the consequences can cascade to subsequent iterations.

Our contributions. To resolve the above limitations, we develop novel model-agnostic solutions that excel in both efficiency and effectiveness. In summary, we make the following contributions:

- *The IAS algorithm.* We propose two key enhancements to the CTS algorithm: (i) *Online learning* to address limitation L1 using incremental updates of the model with new data, without requiring complete retraining of the model (Sec. 3.1). (ii) *Adaptive score estimation* to address limitation L2 by introducing a novel adaptive score, which combines adaptive exploitation and exploration scores to guide cluster selection (Sec. 3.2). Leveraging these two enhancements, we develop a new algorithm that demonstrates improvements in both effectiveness and efficiency, namely CTS with incremental estimation of adaptive score (IAS) (Sec. 3.3).
- *The IAS-AMS algorithm.* To address limitation L3, we propose an adaptive mini-batch selection technique, which involves sampling a mini-batch from each cluster based on the proportion of the cluster's adaptive score to the sum of all adaptive scores (Sec. 4.1.1). Hence, in each iteration, every cluster is explored, and the model performance improvements arise from the collective contribution from all of the clusters. Since adaptive score estimation considers the cluster contribution to both performance improvement and the "degree of exploration", we proceed to adjust the formulation of the adaptive score accordingly (Sec. 4.1.2 and 4.1.3). By integrating this technique with the enhancements introduced in the IAS algorithm, we propose another algorithm, called IAS with adaptive mini-batch selection (IAS-AMS), that further improves effectiveness (Sec. 4.2).
- *Evaluations.* We conduct extensive experiments using five real datasets and associated ML tasks. The results show that IAS exhibits superior efficiency, achieving speedups of up to an order of magnitude while surpassing state-of-the-art algorithms in effectiveness. Meanwhile, IAS-AMS excels in effectiveness, offering performance improvements of up to 60%, while maintaining comparable efficiency to state-of-the-art algorithms (Sec. 5).

2 PRELIMINARIES

We formally define the problem of data acquisition to enhance ML performance (DA-ML) in Sec. 2.1 and then introduce the state-of-the-art algorithms for solving the DA-ML problem in Sec. 2.2.

2.1 Problem Formulation

Training/validation/test sets. We describe a labeled dataset $d = \{(x_1, y_1), ..., (x_{|d|}, y_{|d|})\}$ as a set of (data point, label) pairs where x_i is a data point (e.g., a row in a tabular dataset or an image in an image dataset) and y_i is its label (e.g., a class label for a classification task and a dependent variable value for a regression task). A labeled dataset is typically split into three disjoint subsets: a training set d_{train}, a validation set d_{val}, and a test set d_{test}. The training and validation sets are utilized for model training, while the test set is reserved for assessing the performance of trained models.

Machine learning (ML) tasks. We consider a supervised ML task T that trains a model M_T to learn a mapping function $f : \mathbf{X} \rightarrow \mathbf{Y}$, where $\mathbf{X}$ denotes the feature space and $\mathbf{Y}$ denotes the label space. We use $M_T(d_{train})$ to denote the model M_T that is trained using d_{train}. We also use $M_T(d_{train}, d_{test})$ to denote the model M_T that is trained with d_{train} and evaluated with d_{test}.

Data pool. For a supervised ML task T, we define a data pool $P = \{(x_1, y_1), ..., (x_{|P|}, y_{|P|})\}$ as a set of (data point, label) pairs that are "relevant" to T. For tabular datasets, the relevance implies that these pairs share the same or highly overlapping schema with d_{train}. For image datasets, the relevance indicates that these pairs consist of images sharing the same labels as d_{train}. Typically, such pairs in P can be collected from various sources, such as data lakes, data markets, and open data portals. A dataset discovery process [13] can be utilized to generate a data pool, as illustrated in Fig. 1.

Definition 2.1 (Data Acquisition to Enhance ML Performance, DA-ML). Given a supervised ML model M_T for solving task T, a training set d_{train}, a test set d_{test}, a validation set d_{val}, and a data pool P, the DA-ML problem is to select a subset $S \subset P$ utilizing d_{train} and d_{val}, such that the performance of M_T, trained on the enriched training set $d_{train} \cup S$, is improved maximally, that is, $S = \arg\max_{S \subset P} M_T(d_{train} \cup S, d_{test}) - M_T(d_{train}, d_{test})$.

2.2 State-of-the-Art Algorithms

In this section, we introduce the Cluster-based Training set Selection (CTS) algorithm [13], a state-of-the-art solution for the DA-ML problem. The CTS algorithm contains two main steps, *data point clustering* and *iterative data point selection from clusters*, as depicted in Fig. 1 and discussed below.

Data point clustering. Since data points within the data pool P originate from multiple sources and may exhibit varying distributions, a conventional approach to representing the data points is through *data point clustering*, which involves grouping data points within each cluster based on similarity. Several clustering methods can be used, such as a multivariate Gaussian mixture model (GMM) [21], DBSCAN [20] or k-means [39]. The CTS algorithm uses GMM due to its robustness as shown in [13]. Hence, a set of clusters $C = \{C_1, ..., C_{|C|}\}$ can be created, and each cluster $C_i \in C$ encompasses a set of (data point, label) pairs that are similar.

Iterative data point selection from clusters. After the set of clusters is obtained, the CTS algorithm iteratively selects (data point, label) pairs from these clusters. This process involves selecting (data point, label) pairs from the data pool, to enrich the training set. This enrichment is achieved through a sequence of five operations, as shown in Fig. 1. One common selection strategy for this process,

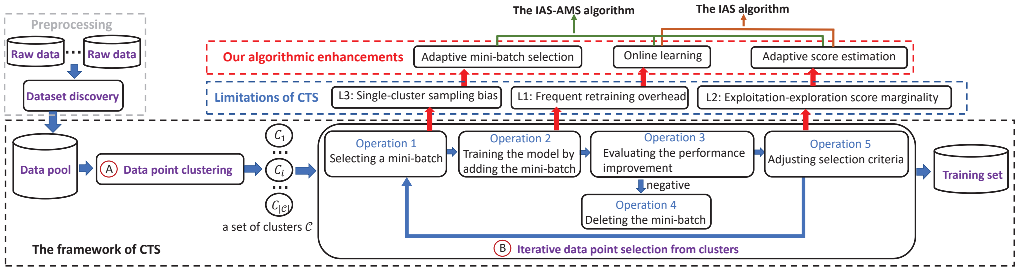

The IAS-AMS algorithm
The IAS algorithm
Preprocessing
Raw data
Raw data
Dataset discovery
Our algorithmic enhancements
Adaptive mini-batch selection
Online learning
Adaptive score estimation
Limitations of CTS
L3: Single-cluster sampling bias
L1: Frequent retraining overhead
L2: Exploitation-exploration score marginality
Data pool
A Data point clustering
C_1
C_i
$C_{|\mathcal{C}|}$
a set of clusters $\mathcal{C}$
Operation 1
Selecting a mini-batch
Operation 2
Training the model by adding the mini-batch
Operation 3
Evaluating the performance improvement
negative
Operation 4
Deleting the mini-batch
Operation 5
Adjusting selection criteria
Training set
The framework of CTS
B Iterative data point selection from clusters

Figure 1: The framework of the CTS algorithm (bottom) and an overview of our algorithmic enhancements upon CTS (top).

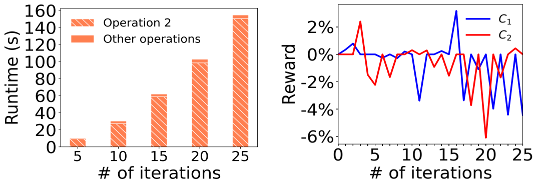
called *the CTS algorithm using a multi-armed bandit (MAB)*, solves a constrained multi-armed bandit problem where each cluster is treated as an arm from the MAB. Specifically, during each iteration, MAB selects a cluster (i.e., an arm) and samples a mini-batch from the chosen cluster (*Operation 1: selecting a mini-batch*). Then, it adds the mini-batch into the existing training set and retrains the model based on the new training set (*Operation 2: training the model by adding the mini-batch*). Next, the performance improvement attributed to the added mini-batch is evaluated (*Operation 3: evaluating the performance improvement*). If the observed performance improvement is negative, it removes the mini-batch from the new training set (*Operation 4: deleting the mini-batch*). Finally, it updates the reward of each cluster based on the performance improvement realized by the mini-batch and the relationship between clusters, followed by adjusting the selection criteria based on the assigned rewards (*Operation 5: adjusting selection criteria*). The selection criteria, as defined in the CTS algorithm, strikes a balance between the choices made for the clusters that yield high rewards on average from the initial iteration to the current iteration (i.e., exploitation) and the clusters that are rarely chosen (i.e., exploration). This is to prevent the emergence of local optima due to the emphasis on exploitation in the subsequent iteration. Such an iterative process terminates when a pre-specified stopping criterion is met.

Note that, there is an alternative solution to the CTS algorithm which uses reinforcement learning based on a deep Q-network (DQN) in [13]. We do not consider this option here as MAB is a more practical and efficient option, and is competitive in performance compared to DQN [13].

3 CTS WITH INCREMENTAL ESTIMATION OF ADAPTIVE SCORE

While the CTS algorithm with a multi-armed bandit (MAB) is a practical choice to address the DA-ML problem, it still exhibits certain limitations in both effectiveness and efficiency. Specifically, we discover the following limitations to two operations within the iterative processing (i.e., Operation 2 and Operation 5 in Fig. 1):

- *Operation 2 (training the model by adding the mini-batch)* consumes more than 90% of the total runtime in each iteration, as shown in Fig. 2(a). Hence, it has a significant impact on the algorithm's overall efficiency.
- *Operation 5 (adjusting selection criteria)* aims to find a balance between the exploitation score and the exploration score, but it faces two issues that impede effectiveness: (1) The exploitation score equally considers both the historical rewards from past iterations and the new reward from the current iteration for each cluster. However, the reward of a cluster often exhibits significant variance over time and may not follow any specific distribution, as illustrated in Fig. 2(b). Consequently, the current exploitation score, being less likely to accurately reflect the current cluster state, may not yield effective outcomes in practice. (2) The exploration score, while emphasizing clusters that are rarely chosen, relies solely on the exploration frequency and may not provide well-informed selection guidance in the subsequent iteration. In cases when two clusters have the same selection criteria due to identical exploration frequencies, the CTS algorithm fails to distinguish between them as potential candidates in the next iteration.



(a)

(b)

Figure 2: Evidence of limitations in CTS using the Crop dataset at Table 1: (a) runtime breakdown (the runtime of Operation 2 and the runtime of other operations) of CTS; (b) the impact of number of iterations based on rewards (only two clusters).

To address the above limitations, we propose a new algorithm – CTS with Incremental estimation of Adaptive Score (IAS). IAS follows a similar framework to CTS (see Fig. 1) but leverages two key strategies to enhance the efficiency or effectiveness of the two operations: *online learning* to optimize Operation 2 for improving efficiency (Sec. 3.1), and *adaptive score estimation* to optimize Operation 5 for improving effectiveness (Sec. 3.2). A complete IAS algorithm will be presented in Sec. 3.3.

3.1 Optimizing Operation 2 via Online Learning

The key to optimizing Operation 2 is to speed up model training by circumventing the necessity to retrain the ML model from scratch using the entire training set in each iteration. To achieve this, we use *online learning* [25], which is an ML approach designed to manage data arriving in a sequential order. With the arrival of each new data instance, online learning updates the underlying model based on the prediction made from the new data instance. Continuous updating improves the model's predictive accuracy for unseen data.

Online learning incrementally updates the ML model's learnable parameters, using the current state of the model along with the newly added training data. In this way, the necessity of full model retraining is eliminated. Next, we will delve deeper into the concept of online learning and explain how we use it to solve our problem.

Conventional online learning settings. Given a closed, bounded, and convex set $\mathcal{F} \in \mathbb{R}^n$ as input, the process is as follows: in each sequential round $t = 1, 2, ...\tau$, a point p_t is chosen from $\mathcal{F}$. A convex loss function f_t is taken into consideration, resulting in the loss $f_t(p_t)$. At the conclusion of τ rounds, the regret is computed as the difference between the cumulative loss incurred by point p_t and the minimum possible cumulative loss that could have been achieved by any other point from $\mathcal{F}$. That is, $\sum_{t=1}^{\tau} f_t(p_t) - \min_{p\in\mathcal{F}} \sum_{t=1}^{\tau} f_t(p)$. The goal of online learning is to minimize the cumulative loss after τ rounds (i.e., $\min \sum_{t=1}^{\tau} f_t(p_t)$) while ensuring a small regret.

Online learning in our work. We denote the feature vector of a data point x as $\mathbf{x}$. We denote the learnable parameters of an ML model M_T by $\mathbf{w}$, where $|\mathbf{w}|$ refers to the number of learnable parameters. We consider the following online learning framework: Given a mini-batch B, at each round $t = 1, ..., |B|$, we are tasked with making predictions for a training instance. This instance is characterized by a feature vector $\mathbf{x}_t$ corresponding to the data point $x_t \in B$ and its associated label y_t. Consequently, we update the learnable parameters of the model M_T as $\mathbf{w}_t$, and make predictions using $\hat{y}_t = \sigma(\mathbf{w}_t, \mathbf{x}_t)$, where $\hat{y}_t$ denotes the predicted value of y_t and $\sigma(\cdot)$ is the function yielding the model output. We use a regularized loss function $f_t(\mathbf{w}_t, \mathbf{x}_t, y_t) = \ell(\mathbf{w}_t, \mathbf{x}_t, y_t) + \Psi(\mathbf{w}_t)$ where $\ell(\cdot)$ is the loss function of M_T and $\Psi(\cdot)$ is a non-smooth regularization function. Our goal is to update the learnable parameters $\mathbf{w}_t$ at each round, such that the aggregated loss function after $|B|$ rounds is minimized, i.e., $\min \sum_{t=1}^{|B|} f_t(\mathbf{w}_t)$. The gradient of M_T at round t is denoted as $\mathbf{g}_t = \frac{\partial}{\partial \mathbf{w}} \ell(\mathbf{w})|_{\mathbf{w}=\mathbf{w}_t}$, and the i-th entry of $\mathbf{g}_t$ is denoted as $g_{t,i}$ where $1 \leq i \leq |\mathbf{w}|$. Additionally, the compressed sum $\mathbf{g}_{1:t} = \sum_{s=1}^{t} \mathbf{g}_s$ is used. The gradients are required to solve the optimization problem described above. For example, M_T is a logistic regression model. Hence, at round t, we have $\sigma(\mathbf{w}_t, \mathbf{x}_t) = 1/(1+\exp(-\mathbf{w}_t \cdot \mathbf{x}_t))$ and $\ell(\mathbf{w}_t, \mathbf{x}_t, y_t) = -y_t \log \hat{y}_t - (1 - y_t)\log(1 - \hat{y}_t)$ where $\hat{y}_t = \sigma(\mathbf{w}_t, \mathbf{x}_t)$. The gradient of M_T is also $\mathbf{g}_t = (\sigma(\mathbf{w}_t, \mathbf{x}_t) - y_t)\mathbf{x}_t$.

Many different online learning algorithms exist [42], with the primary difference being the regularization approach used. Here, we outline the state-of-the-art algorithm FTRL-Proximal [44], which we have used to implement online learning in this work. At round $t+1$, the FTRL-Proximal algorithm uses the following update equation

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}} \mathbf{g}_{1:t} \cdot \mathbf{w} + \frac{1}{2}\sum_{s=1}^{t} Q_s||\mathbf{w} - \mathbf{w}_s||_2^2 + L_1||\mathbf{w}||_1. \tag{1}$$

Here, the first term is an approximation of the sum of prior losses, i.e., $\sum_{s=1}^{t} \ell(\mathbf{w}_s)$, based on their respective gradients. The second term is a stabilizing regularization function that prevents regret from becoming too large. The matrix Q_s presents generalized learning rates, which can be chosen adaptively through techniques introduced by [44] (discussed later). The third term is a non-smooth regularization function $\Psi(\mathbf{w}) = L_1||\mathbf{w}||_1$. The L_1-norm regularization promotes solution sparsity, i.e., it encourages a solution $\mathbf{w}$ that contains many zero entries. This can be seen as a form of feature selection, as it maintains only the most relevant features in the model by retaining the corresponding non-zero values in the solution $\mathbf{w}$ [58].

Algorithm 1 Online Learning

```
Input: mini-batch $B$, learnable parameters $\mathbf{w}$, $\mathbf{z}$, $\mathbf{n}$, $\alpha$, $\beta$, $L_1$
Output: learnable parameters $\mathbf{w}$ and vectors $\mathbf{z}$ and $\mathbf{n}$
1: $z_{0,i} \leftarrow \mathbf{z}[i]$, $n_{0,i} \leftarrow \mathbf{n}[i]$ for $i$ = 1 to $|\mathbf{w}|$;
2: for $t$ = 1 to $|B|$ do
3:     receive the feature vector $\mathbf{x}_t$, representing $x_t \in B$, and its label $y_t$;
4:     for $i$ = 1 to $m$ do
5:         if $z_{t-1,i} \leq L_1$ then $w_{t,i} = 0$;
6:         else $w_{t,i} = -\frac{\alpha}{\beta+\sqrt{n_{t-1,i}}}(z_{t-1,i} - \text{sgn}(z_{t-1,i})L_1)$;
7:         $\mathbf{w}[i] \leftarrow w_{t,i}$;
8:     obtain the gradient $\mathbf{g}_t$ given the loss $\ell(\mathbf{w}, \mathbf{x}_t, y_t)$;
9:     for $i$ = 1 to $m$ do
10:        $z_{t,i} \leftarrow z_{t-1,i} + g_{t,i} + \frac{1}{\alpha}(\sqrt{n_{t-1,i} + g_{t,i}^2} - \sqrt{n_{t-1,i}})w_{t,i}$;
11:        $n_{t,i} \leftarrow n_{t-1,i} + g_{t,i}^2$, $\mathbf{z}[i] \leftarrow z_{t,i}$, $\mathbf{n}[i] \leftarrow n_{t,i}$;
   return $\mathbf{z}$, $\mathbf{n}$ and $\mathbf{w}$;
```

Implementing the FTRL-Proximal algorithm based on Eq. 1 is far from straightforward, as it must retain all of the prior coefficients, resulting in substantial storage overhead. To mitigate this, we streamline storage demands to a single number for each coefficient. Specifically, let $\sum_{s=1}^{t} Q_s = \frac{1}{\eta_t}$, where η_t is a non-increasing learning rate, e.g., one viable choice for η_t is $\eta_t = 1/\sqrt{t}$. This modification enables us to reformulate Eq. 1 as

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}} (\mathbf{g}_{1:t} - \sum_{s=1}^{t} Q_s \mathbf{w}_s) \cdot \mathbf{w} + \frac{1}{\eta_t}||\mathbf{w}||_2^2 + L_1||\mathbf{w}||_1. \tag{2}$$

In addition, instead of applying a single η_t to all coordinates, we use per-coordinate learning rates, since it has been shown to offer significant performance advantages [56]. Recall that $g_{t,i}$ is the i-th entry of the gradient vector $\mathbf{g}_t$. We calculate the per-coordinate learning rate $\eta_{t,i}$ via the formula $\eta_{t,i} = \alpha/(\beta + \sqrt{\sum_{s=1}^{t} g_{s,i}^2})$, which has been demonstrated to be near-optimal according to [43].

Therefore, the update process only needs to store two vectors. We denote these two vectors as $\mathbf{z}_t$ and $\mathbf{n}_t$. We compute the i-th entry of $\mathbf{n}_t$ as $n_{t,i} = \sum_{s=1}^{t} g_{s,i}^2$. Moreover, we have $\mathbf{z}_t = \mathbf{g}_{1:t} - \sum_{s=1}^{t} Q_s \mathbf{w}_s$ at the beginning of round $t+1$ and $\mathbf{z}_{t+1} = \mathbf{z}_t + \mathbf{g}_{t+1} + (\frac{1}{\eta_{t+1}} - \frac{1}{\eta_t})\mathbf{w}_{t+1} = \mathbf{z}_t + \mathbf{g}_{t+1} + \frac{1}{\alpha}(\sqrt{\sum_{s=1}^{t+1} g_{s,i}^2} - \sqrt{\sum_{s=1}^{t} g_{s,i}^2})\mathbf{w}_{t+1}$. Consequently, we compute the update $\mathbf{w}_{t+1}$ coordinate-wise via

$$w_{t+1,i} = \begin{cases} 0 & \text{if } z_{t,i} \leq L_1 \\ -\frac{\alpha}{\beta+\sqrt{n_{t,i}}}(z_{t,i} - \text{sgn}(z_{t,i})L_1) & \text{otherwise} \end{cases} \tag{3}$$

where $\text{sgn}(z_{t,i}) = 1$ if $z_{t,i} > 0$, $\text{sgn}(z_{t,i}) = 0$ if $z_{t,i} = 0$, and $\text{sgn}(z_{t,i}) = -1$ if $z_{t,i} < 0$.

Alg. 1 shows the pseudocode of our online learning algorithm. The input includes a mini-batch B, the learnable parameters $\mathbf{w}$, the stored vectors $\mathbf{z}$ and $\mathbf{n}$, and the hyperparameters of FTRL-Proximal, α, β, and L_1. At each round t, it receives a feature vector $\mathbf{x}_t$ representing a data point $x_t \in B$, along with the associated label y_t (Line 3). Then, Eq. 3 is used to update $\mathbf{w}$ (Lines 5-7) followed by updating $\mathbf{z}$ and $\mathbf{n}$ according to the respective equations (Lines 10-11).

Time complexity analysis. At k-iteration, when adding a mini-batch B into the current training set T_{train}^{k-1}, the time complexity of online learning is $O(|B||\mathbf{w}|)$. If the gradients $\mathbf{g}_t$ have at most a non-zero entries, only the corresponding non-zero entries are updated. Consequently, the time complexity is $O(|B|a)$, which can be significantly less than $O(|B||\mathbf{w}|)$ for sparse gradients. In contrast, the previous work [13] requires complete ML model retraining, yielding a time complexity of $O(|T_{train}^{k-1} \cup B||\mathbf{w}|)$.

3.2 Optimizing Operation 5 via Adaptive Score Estimation

Operation 5 enhances the cluster selection process in the subsequent iteration, guided by the refined selection criteria. In this section, we first outline the existing approach for Operation 5 and its drawbacks. Then, we introduce new selection criteria for Operation 5, namely *adaptive score*, which combines *adaptive exploitation score* and *adaptive exploration score*.

Th existing approach for Operation 5. Operation 5 is generally performed utilizing the upper confidence bound (UCB) scores [3] as the selection criteria. The UCB score of any cluster $C_i \in C$ at the first k iterations is calculated as

$$s_i^k = \bar{r}_i^k + \gamma\sqrt{2\ln n^k/(n_i^k+1)}. \tag{4}$$

Here, $\bar{r}_i^k = \frac{1}{n_i^k}\sum_{j=1}^{k} r_i^j$ is the average reward of C_i for the first k iterations, r_i^j is the reward of C_i at the j-th iteration. The reward is calculated based on the performance improvement achieved by including the mini-batch B_i from C_i, n_i^k is the total number of times that C_i has received non-zero rewards in the first k iterations, and n^k is the sum of n_i^k across all clusters up to iteration k (i.e., $n^k = \sum_{i=1}^{|C|} n_i^k$). The first term on the right-hand side of Eq. 4, $\bar{r}_i^k$, corresponds to the exploitation score, which directs attention to the cluster selected where data points have significantly contributed to improving model performance. The second term, $\gamma\sqrt{2\ln n^k/(n_i^k+1)}$, is the exploration score, which has more weight for clusters that have been infrequently chosen. The parameter γ adjusts the balance between the exploration and exploitation aspects of the selection process.

As discussed earlier, two challenges persist: (1) The exploitation score treats the rewards of all iterations equally, which may not accurately reflect the current cluster state. (2) The exploration score relies solely on the exploration frequency, which may not yield the best guidance when selecting clusters in subsequent iterations.

To tackle the above challenges, our new selection criteria calculation, referred to as *adaptive score estimation*, incorporates two key components. The first is *adaptive exploitation score estimation* (Sec. 3.2.1), which emphasizes recent rewards, granting them higher weight in the final score. This ensures that the current cluster state is more accurately reflected, contrary to the uniform treatment of rewards for all prior iterations. The second component is *adaptive exploration score estimation* (Sec. 3.2.2), which guides exploration efforts to enhance overall effectiveness.

3.2.1 *Adaptive exploitation score estimation.*

Adaptive estimation [4] has been widely used in streaming data scenarios to continuously monitor the mean of a sequence of observations. When given a sequence of observations o_1, ..., o_N and adaptive forgetting factors $\vec{\lambda} = (\lambda_1, ...\lambda_N)$, adaptive estimation estimates the mean of this sequence of observations after receiving a new observation o_{N+1}. This estimation is accomplished by computing an adaptive forgetting factor mean as $o_{N+1,\vec{\lambda}} = \frac{1}{w_{N+1,\vec{\lambda}}}\sum_{s=1}^{N+1}(\prod_{p=s}^{N}\lambda_p)o_s$. Here, $w_{N+1,\vec{\lambda}} = \sum_{s=1}^{N+1}(\prod_{p=s}^{N}\lambda_p)$, $\lambda_i \in [0,1]$, and $\prod_{p=N+1}^{N}\lambda_p = 1$.

Adaptive estimation is well-suited for our scenario, and we leverage it to develop the *adaptive exploitation score.* Suppose, at the k-th iteration, a cluster C_i is selected, resulting in a reward r_i^k. Using the adaptive forgetting factors associated with cluster C_i, denoted as $\vec{\lambda}_i = (\lambda_i^1, \lambda_i^2, ...\lambda_i^{k-1})$, we compute the adaptive exploitation score of C_i over the first k iterations as:

$$\bar{r}_i^k = \frac{1}{w_i^k}\sum_{s=1}^{k}(\prod_{p=s}^{k-1}\lambda_i^p)r_i^s \tag{5}$$

where $w_i^k = \sum_{s=1}^{k}(\prod_{p=s}^{k-1}\lambda_i^p)$ and $\prod_{p=k}^{k-1}\lambda_p = 1$.

Intuitively, early rewards are assigned diminishing weights by virtue of the higher powers of the adaptive forgetting factors. That is, recent rewards are given greater significance. As a result, we can rewrite Eq. 5 as

$$\bar{r}_i^k = \frac{m_i^k}{w_i^k}, m_i^k = \lambda_i^{k-1}m_i^{k-1} + r_i^k, w_i^k = \lambda_i^{k-1}w_i^{k-1} + 1 \tag{6}$$

where $m_i^0 = 0$ and $w_i^0 = 0$.

The sequence $\vec{\lambda}_i$ grows over time. The key consideration when using adaptive forgetting factors is how to update $\vec{\lambda}_i$ (i.e, add an adaptive forgetting factor λ_i^k to $\vec{\lambda}_i$ at k-th iteration). Similar to [4], we update $\vec{\lambda}_i$ using a stochastic gradient descent step [5]. To implement the stochastic gradient descent, we first need to formulate a cost function $L_{\vec{\lambda}_i}^k$ based on $\bar{r}_i^k$, and then update $\vec{\lambda}_i$ by stepping in a direction that minimizes $L_{\vec{\lambda}_i}^k$, i.e.,

$$\lambda_i^k = \lambda_i^{k-1} - \eta\frac{\partial}{\partial_{\vec{\lambda}_i}}L_{\vec{\lambda}_i}^k \tag{7}$$

where $\eta \ll 1$ is the step size. We adopt the cost function $L_{\vec{\lambda}_i}^k = (\bar{r}_i^{k-1} - r_i^k)^2$, a suitable choice for mean tracking [4], which can be perceived as one-step-ahead squared prediction error. We have

$$\tilde{m}_i^k = \lambda_i^{k-1}\tilde{m}_i^{k-1} + m_i^{k-1}, \tilde{w}_i^k = \lambda_i^{k-1}\tilde{w}_i^{k-1} + w_i^{k-1}$$

$$\frac{\partial}{\partial_{\vec{\lambda}_i}}L_{\vec{\lambda}_i}^k = 2(\bar{r}_i^{k-1} - r_i^k)\frac{\tilde{m}_i^{k-1} - \tilde{w}_i^{k-1}\bar{r}_i^{k-1}}{w_i^{k-1}}$$

where $\tilde{m}_i^1 = 0$ and $\tilde{w}_i^1 = 0$. Moreover, considering that the rewards of a cluster evolve, uncertainty surrounds unselected clusters in each iteration. Therefore, we refine the adaptive exploitation score of unselected clusters using a discount mechanism. This discount can be calculated using adaptive forgetting factors linked to prior selections. With this approach, we revise the adaptive exploitation score of any cluster $C_j \in C \setminus C_i$ for the first k iterations as:

$$\bar{r}_j^k = \frac{m_j^k}{w_j^k}, m_j^k = \frac{k - k_j^{lt}}{|C|}\lambda_j^{k_j^{lt}} m_j^{k_j^{lt}}, w_j^k = \frac{k - k_j^{lt}}{|C|}\lambda_j^{k_j^{lt}} w_j^{k_j^{lt}} \tag{8}$$

where k_j^{lt} is the last iteration during which C_j has been selected.

3.2.2 *Adaptive exploration score estimation.*

To optimize the exploration score in Eq. 4, we pay special attention to the exploration component from two perspectives, aiming to maximize benefits from the selected clusters:

- Instead of only keeping track of the count n_i^k, we maintain a list v_i that records the iterations when each cluster is explored. For example, if cluster C_i is explored in the first and fifth iterations and the current iteration number is six, we have $v_i = [1, 0, 0, 0, 1, 0]$.

Hence, in the k-th iteration, the adaptive exploration score for any selected cluster C_i is computed as:

$$\gamma\sqrt{2\ln(k)/\left(\sum_{s=1}^{k}\frac{k-s+1}{k}v_i^s+1\right)}. \tag{9}$$

This approach reduces the weight of older information, ensuring that the decision to explore a cluster is influenced by more recent information, even in scenarios where certain clusters have been explored the same number of times.

• To further enhance effectiveness, we offer additional exploration opportunities to clusters that have different distributions from the cluster currently selected in the iteration. This permits data points from diverse distributions to be included during ML model training. As such, at the k-th iteration, the adaptive exploration score of an unselected cluster $C_j \in C \setminus C_i$ can be calculated as:

$$\left(\gamma+\frac{d(C_i,C_j)}{\max(d(C_i,C_j))}\right)\sqrt{2\ln(k)/\left(\sum_{s=1}^{k}\frac{k-s+1}{k}v_j^s+1\right)}. \tag{10}$$

where the distance $d(C_i, C_j)$ between C_i and C_j is used to control the likelihood of selecting the cluster in the next iteration and calculated by Wasserstein distance [48].

Combining the adaptive exploitation score with the adaptive exploration score, we are ready to introduce our final *adaptive score* as new selection criteria. Specifically, at the k-th iteration, the adaptive score s_i^k of any selected cluster C_i is calculated by combining Equations 6 and 9:

$$s_i^k=\bar{r}_i^k+\gamma\sqrt{2ln(k)/(\sum_{s=1}^{k}\frac{k-s+1}{k}v_i^s+1)}. \tag{11}$$

For any unselected cluster $C_j \in C \setminus C_i$, combining Equations 8 and 10 yields the adaptive score s_j^k of C_j as

$$s_j^k=\bar{r}_j^k+\left(\gamma+\frac{d(C_i,C_j)}{\max(d(C_i,C_j))}\right)\sqrt{2\ln(k)/\left(\sum_{s=1}^{k}\frac{k-s+1}{k}v_j^s+1\right)}. \tag{12}$$

Example 3.1 (Our adaptive scores (Eq. 11 and Eq. 12) versus the UCB scores (Eq. 4)). Consider a data pool containing three clusters, $C = \{C_1, C_2, C_3\}$. Initialize $\bar{r}_i^0$ and s_i^0 to 0 for all clusters and set $\gamma = 0.05$. At the first iteration, C_1 is selected, resulting in a 22% improvement in model performance.

Using existing UCB scores. At the second iteration, C_1 is chosen, leading to a 1% decrease in model performance. At the third iteration, C_1 is selected, leading to a 1% reduction in model performance. At the fourth iteration, C_1 is picked, leading to a 1% decrease in model performance. At the fifth iteration, C_1 is selected, leading to a 1% decrease in model performance. C_1 is chosen at the first five iterations even if it does not improve the model performance after the first iteration. At the sixth iteration, we have $s_1^5 = 0.79$, $s_2^5 = 0.089$, and $s_3^5 = 0.089$. In the CTS algorithms, C_2 is chosen randomly, yielding no improvement in model performance, while selecting C_3 can improve model performance.

Using our adaptive scores. Let $\eta = 0.1$ in Eq. 7. At the second iteration, C_1 is selected, leading to a model performance decrease of 1%. At the third iteration, C_1 is chosen, leading to a 1% decrease in model performance. At the fourth iteration, $s_1^3 = 0.108$, $s_2^3 = 0.111$, and $s_3^3 = 0.175$. Hence, C_3 is picked, leading to a model performance increase of 1%. Our adaptive score estimation enables a better exploration of all clusters. Given that the exploration score in Eq. 10 accounts for the distributions across various clusters, it provides a more informed basis for cluster selection. Thus, we can select a cluster that can deliver an earlier enhancement in model performance.

Algorithm 2 The IAS algorithm

```
Input: the clusters $C$, the training set $d_{train}$, the ML model $M_T$, the validation set $d_{val}$, the size of a mini-batch $l$, the iterations $K$, $\alpha$, $\beta$, $L_1$, $\eta$, $\gamma$
Output: the updated training set and the trained model
1: obtain the gradients $\mathbf{g}$ and learnable parameters $\mathbf{w}$ from $M(d_{train})$;
2: $\mathbf{z}[i] \leftarrow \mathbf{g}[i] + \mathbf{g}[i]\mathbf{w}[i]/\alpha$, $\mathbf{n}[i] \leftarrow \sum_{s=1}^{t}\mathbf{g}[i]^2$ for $1 \le i \le |\mathbf{w}|$;
3: $\bar{r}_i^0 \leftarrow 0$, $s_i^0 \leftarrow 0$ for each $C_i \in C$, $\lambda_i^0 \leftarrow 1$, $p \leftarrow M_T(d_{train}, d_{val})$;
4: for $k$ from 1 to $K$ do
5:     if k == 1 then $C_i \leftarrow \arg\max_{C_i \in C} d(C_i, d_{train})$;
6:     else select the cluster $C_i \in C$ with the largest adaptive score;
7:     sample a mini-batch $B_i$ of size $l$ from $C_i$;
8:     $\mathbf{w}, \mathbf{z}, \mathbf{n} \leftarrow$ OnlineLearning$(B_i, \mathbf{w}, \mathbf{z}, \mathbf{n}, \alpha, \beta, L_1)$
9:     $\Delta \leftarrow M_T^{\mathbf{w}}(B_i, d_{val}) - p$, $p \leftarrow M_T^{\mathbf{w}}(B_i, d_{val})$;
10:    if $\Delta > 0$ then $d_{train}^k \leftarrow d_{train}^{k-1} \cup B_i$;
11:    $r_i^k \leftarrow \Delta$ and update $\bar{r}_i^k$ with $r_i^k$ and $\vec{\lambda}_i$ using Eq. 6;
12:    $s_i^k$ using Eq. 11 and update $\vec{\lambda}_i$ with $\eta$ using Eq. 7;
13:    for $C_j \in C \setminus C_i$ do
14:        update $\bar{r}_j^k$ using Eq. 8 and $s_j^k$ using Eq. 12;
   return $d_{train}^K$, $M_{\mathbf{w}}$
```

3.3 The IAS Algorithm

With all of the key components of online learning and adaptive score estimation introduced in Sec. 3.1 and Sec. 3.2, we now present the complete procedure for the proposed IAS algorithm in Alg. 2. First, the ML model M_T trains on the initial training set d_{train}, obtaining the gradients $\mathbf{g}$ and learnable parameters $\mathbf{w}$ (Line 1). Then, vectors $\mathbf{z}$ and $\mathbf{n}$ are stored for the online learning stage (Lines 2). In the first iteration, the cluster that exhibits the most different distribution compared to d_{train} is selected where the distance $d(C_i, d_{train})$ between C_i and d_{train} is calculated by Wasserstein distance [48] (Line 5), to enable the extraction of diverse information from d_{train} to enhance model training. In subsequent iterations, the cluster with the largest adaptive score is selected (Line 6) and a mini-batch B_i is sampled from the selected cluster C_i (Line 8). Lines 8-9 provide online-learning-based updates to M_T using mini-batch B_i (Alg. 1), and the performance improvement Δ of M_T attributed to the mini-batch B_i is computed. Here, $M_T^{\mathbf{w}}(B_i, d_{val})$ represents M_T updated via $\mathbf{w}$ with the inclusion of mini-batch B_i and evaluated by d_{val}. Then, the training set is updated to d_{train}^k if $\Delta > 0$ (Line 10). Finally, the adaptive score of the selected cluster is updated in Lines 11-12, and the adaptive scores of any unselected clusters are updated in Lines 13-14. Assuming that each gradient $\mathbf{g}$ has at most a non-zero entries, the total time complexity of IAS is $O((la + |C|)K)$.

4 IAS WITH ADAPTIVE MINI-BATCH SELECTION

Our IAS algorithm enhances Operations 2 and 5, leading to improvements in both efficiency and effectiveness. However, as mentioned in Sec. 1, another limitation exists when selecting a cluster to sample

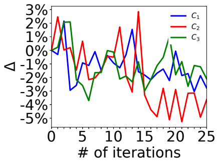


Figure 3: Performance improvement Δ obtained by IAS using the Crop dataset (see Table 1) starting from different clusters.

a mini-batch (i.e. Operation 1). With regard to Operation 1, the IAS algorithm selects the cluster with the least similar distribution to d_{train} in the first iteration, while in subsequent iterations, it selects the cluster with the highest adaptive score. Our experiments show that this approach has the potential to reduce the overall effectiveness of the IAS algorithm. That is, the effectiveness improvement in each iteration depends on the single cluster used, which may not always produce positive performance gains for that iteration. This causes a cascading effect in subsequent cluster selection iterations.

To illustrate this point, consider a data pool containing three clusters, $C = \{C_1, C_2, C_3\}$. Fig. 3 shows the performance improvement for each iteration when different clusters are selected during the first iteration. We can see that selecting C_1 in the first iteration fails to produce a positive performance improvement. Conversely, selecting C_2 or C_3 in the first iteration does show improvements. By selecting C_2 in the first iteration, the maximum performance gain can be achieved early. However, Alg. 2 does not make the best choice since C_1 is selected in the first iteration. Thus, we aim to address this limitation in the IAS algorithm.

4.1 Optimizing Operation 1 through Adaptive Mini-batch Selection

In this section, we first discuss how to optimize Operation 1 described in Sec. 4.1.1, to mitigate the impact of single cluster selection on the effectiveness of the algorithm. Then, we introduce how to assign the reward of each cluster after optimizing Operation 1 in Sec. 4.1.2 and how to calculate the adaptive score of each cluster after optimizing Operation 1 in Sec. 4.1.3. Finally, in Sec. 4.2, we present a novel algorithm that combines the enhancement on Operation 1 and the enhancements offered by the IAS algorithm. We call this algorithm IAS with Adaptive Mini-batch Selection (IAS-AMS). Fig. 1 shows the IAS-AMS algorithm in more detail.

4.1.1 Adaptive mini-batch selection to optimize Operation 1. The key to optimizing Operation 1 is to reduce the impact of single cluster selection on the algorithm's effectiveness. To achieve this, instead of sampling a mini-batch from a single cluster in each iteration, we optimize Operation 1 using *adaptive mini-batch selection*, which samples (data point, label) pairs from every cluster. The selection process is based on the proportion of the adaptive score for each cluster w.r.t. the sum of all adaptive scores. These proportions guide the composition of the mini-batch in each iteration. Specifically, let l be the mini-batch size. The adaptive mini-batch selection samples a set of (data point, label) pairs from each C_i, denoted by B_i. The cardinality of B_i is $|B_i| = (s_i^{k-1} / \sum_{j=1}^{|C|} s_j^{k-1}) \times l$. The selected mini-batch is then $B = \cup_{C_i \in C} B_i$, which serves as the input in subsequent operations.

The above enhancement approach for Operation 1 affects the reward assignments as well as the adaptive score estimation for the clusters. The performance improvement is due to the chosen mini-batch that stemmed from a single cluster in each iteration in the IAS algorithm.In contrast, our adaptive mini-batch selection improves performance with mini-batch selection in each iteration, which uses all of the clusters. Next, we will discuss reward assignments after optimizing Operation 1 (Sec. 4.1.2), as well as adaptive score estimation after optimizing Operation 1 (Sec. 4.1.3).

4.1.2 Reward assignments after optimizing Operation 1. When it comes to assigning rewards to individual clusters after optimizing Operation 1 using adaptive mini-batch selection, a natural question arises: Can we accurately measure the relative contributions for each cluster to the performance improvement from the selected mini-batch? Our answer is affirmative. To achieve this objective, we use the Shapley value (SV) [50], which is one of the most reliable methods for equitable contribution assessment. It enables us to quantify unique contributions from each cluster, and subsequently, assign these contributions as the reward for each cluster.

In cooperative game theory [6], within a set of players denoted as $\mathcal{N} = \{z_1, ..., z_{|\mathcal{N}|}\}$, let $\mathcal{S} \subseteq \mathcal{N}$ represent a coalition of players and u be a utility function. The Shapley value allocated to a player z signifies the marginal utility contribution ascribed to z, averaged across all player coalitions, i.e.,

$$\phi(z) = \frac{1}{|\mathcal{N}|} \sum_{\mathcal{S} \subseteq \mathcal{N} \setminus z} \frac{1}{\binom{|\mathcal{N}|-1}{|\mathcal{S}|}} (u(S \cup z) - u(S)).$$

Accordingly, in our problem, the Shapley value assigned to a cluster $C_i \in C$ at the k-th iteration can be used as the reward attributable to C_i in iteration k, expressed as:

$$r_i^k = \frac{1}{|C|} \sum_{\mathcal{S} \subseteq C \setminus C_i} \frac{1}{\binom{|C|-1}{|\mathcal{S}|}} (\Delta(\mathcal{S} \cup C_i) - \Delta(\mathcal{S})). \qquad (13)$$

Here, $\Delta(S)$ represents the performance improvement offered by a mini-batch $B = \bigcup_{C_j \in \mathcal{S}} B_j$ where B_j is a set of (data point, label) pairs sampled from C_j.

Generally, calculating the exact Shapley values for clusters involves enumerating every possible subset of clusters, resulting in a time complexity of $O(2^{|C|})$. Carrying out such computation with exponential time complexity is intractable, particularly with a large number of clusters. There are many existing studies on accelerating Shapley value computations using various techniques [9, 11, 23, 29]. However, since we require a model-agnostic approach, sampling-based methods are better-suited for our problem. Therefore, we use a state-of-the-art method [62] to approximate the Shapley values. This method estimates approximate Shapley values using stratified sampling, which can be solved with a time complexity $O(|C|^2)$. The accuracy of the approximation is influenced by the size of the sample used to estimate marginal contributions [41]. Specifically, according to Hoeffding's inequality [52], given the range of a utility function R, an error bound ϵ, and a confidence level $1 - \delta$, if the sample size of marginal contributions for a player z fulfills the condition $m \geq \frac{2R^2 log2/\delta}{\epsilon^2}$, then $P(|\bar{\phi}(z) - \phi(z)| \geq \epsilon) \leq \delta$. Here, $\bar{\phi}(z)$ denotes the approximate Shapley value for player z, and $\phi(z)$ is the exact Shapley value for player z.

4.1.3 Adaptive score estimation after optimizing Operation 1. After assigning rewards for each cluster using the Shapley value approximation, another critical question arises: How should we calculate the adaptive score for each cluster after optimizing Operation 1 (adaptive mini-batch selection)? Clearly, with adaptive mini-batch selection, every cluster is explored in each iteration. In this context, calculating the adaptive exploration score in the adaptive score (i.e., the second term in Eq. 11) is equivalent to calculating the exploration score in UCB (i.e., the second term in Eq. 4). However, it is important to not overlook the fact that each cluster contributes varying amounts of information, quantified by the number of (data point, label) pairs it contains, in each iteration. To address this concern, we modify the second term in Eq. 11 to account for this variability. The essence of this modification lies in considering the proportion of sampled (data point, label) pairs within a cluster relative to the total number of (data point, label) pairs in the cluster, as a measure of how extensively the cluster is explored. Therefore, the adaptive score of cluster $C_i \in C$ at the k-th iteration is computed as:

$$s_i^k = \bar{r}_i^k + \gamma \sqrt{2 \ln \left(\sum_{i=1}^{|C|} |B_i|/|C_i| + 1 \right) / (|B_i|/|C_i| + 1)} \tag{14}$$

where B_i is the set of (data point, label) pairs sampled from cluster C_i at k-th iteration via adaptive mini-batch selection.

Example 4.1 (Our adaptive mini-batch selection versus single-cluster sampling). Consider three clusters as shown in Fig. 3 and let each mini-batch contain 30 (data point, label) pairs. Using adaptive mini-batch selection, in the first iteration, 10 pairs are sampled from each cluster. This results in a performance improvement compared to IAS's sampling from C_1, which leads to performance degradation. The calculated scores are $s_1^1 = 0.041$, $s_2^1 = 0.071$, and $s_3^1 = 0.061$. In the second iteration, 8 pairs are sampled from C_1, 12 pairs from C_2, and 10 pairs from C_3. This continues to improve performance while exploring more information from each cluster.

4.2 The IAS-AMS Algorithm

Combining the enhancement on Operation 1 and the enhancements from the IAS algorithm, we get a new IAS-AMS algorithm that further improves the effectiveness. We present the pseudocode of the proposed IAS-AMS algorithm in Alg. 3. In this algorithm, the ML model M_T is first trained using an initial training set d_{train} and the associated learnable parameters $\mathbf{w}$ and gradients $\mathbf{g}$ are obtained (Line 1). Then, two vectors $\mathbf{z}$ and $\mathbf{n}$ are stored for online learning (Lines 2). At the k-th iteration, adaptive mini-batch selection obtains a mini-batch B by sampling (data point, label) pairs from each cluster according to the proportional adaptive score of each cluster (Lines 6-7). In Lines 8-9, the ML model M_T is updated via online learning using a mini-batch B (Alg. 1), and the performance improvement Δ of M_T provided by the mini-batch B is computed. Here $M_T^{\mathbf{w}}(B, d_{val})$ denotes M_T that is updated using learnable parameters $\mathbf{w}$ and the mini-batch B, and evaluated with d_{val}. Next, the training set is updated to d_{train}^k, if $\Delta > 0$ (Line 10). Finally, the reward for each cluster is assigned using Eq. 13 and the adaptive score for each cluster is updated using Eq. 14 (Lines 12-13).

Algorithm 3 The IAS-AMS algorithm

```
Input: the clusters $C$, the training set $d_{train}$, the ML model $M_T$, the validation set $d_{val}$, the size of a mini-batch $l$, the iterations $K$, $\alpha$, $\beta$, $L_1$, $\eta$, $\gamma$
Output: the updated training set and the trained model
 1: obtain the gradients $\mathbf{g}$ and learnable parameters $\mathbf{w}$ from $M(d_{train})$;
 2: $\mathbf{z}[i] \leftarrow \mathbf{g}[i] + \mathbf{g}[i]\mathbf{w}[i]/\alpha$, $\mathbf{n}[i] \leftarrow \sum_{s=1}^{t} \mathbf{g}[i]^2$ for $1 \le i \le |\mathbf{w}|$;
 3: $\bar{r}_i^0 \leftarrow 0$, $s_i^0 \leftarrow 0$ for each $C_i \in C$, $\lambda_i^0 \leftarrow 1$, $p \leftarrow M_T(d_{train}, d_{val})$;
 4: for $k$ from 1 to $K$ do
 5:     $B \leftarrow \emptyset$
 6:     for $C_i \in C$ do // Adaptive mini-batch selection
 7:         sample a mini-batch $B_i$ with size $\frac{s_i^{k-1} l}{\sum_{j=1}^{|C|} s_j^{k-1}}$ from $C_i$, $B \leftarrow B \cup B_i$;
 8:     $\mathbf{w}, \mathbf{z}, \mathbf{n} \leftarrow$ OnlineLearning$(B, \mathbf{w}, \mathbf{z}, \mathbf{n}, \alpha, \beta, L_1)$;
 9:     $\Delta \leftarrow M_T^{\mathbf{w}}(B, d_{val}) - p$, $p \leftarrow M_T^{\mathbf{w}}(B, d_{val})$;
10:     if $\Delta > 0$ then $d_{train}^k \leftarrow d_{train}^{k-1} \cup B$;
11:     for $C_i \in C$ do // Adaptive score estimation
12:         compute $r_i^k$ using Eq. 13 and update $\lambda_i^{k-1}$ with $\eta$ using Eq. 7;
13:         update $\bar{r}_i^k$ with $\lambda_i^{k-1}$ using Eq. 6 and $s_i^k$ using Eq. 14;
    return $d_{train}^K$, $M_{\mathbf{w}}$
```

Table 1: Statistical Properties of the Datasets

Dataset	Dataset size	# attributes	Task type	# classes
HR	19,159	12	Classification	2
Crop	325,835	175	Classification	7
CIFAR10	60,000	1,024	Classification	10
House	357,583	18	Regression	N/A
Traffic	87,840	121	Regression	N/A

Time complexity analysis. Since each gradient contains at most a non-zero entries, the per-iteration time to update the ML model using online learning and a mini-batch B is $O(la)$. Computing the reward r_i^k (i.e., Shapley value) for each cluster C_i requires $O(|C|^2 la)$ and the adaptive score estimation requires $O(|C|^3 la)$ time. Thus, the total time complexity of IAS-AMS is $O(|C|^3 laK)$. Notably, while that exceeds IAS, which is $O((la + |C|)K)$, the primary advantage of IAS is its superior effectiveness, as demonstrated in Sec. 5.2.

5 EXPERIMENTS

We conduct extensive experiments to verify: (1) the effectiveness of our algorithms compared to state-of-the-art competitors (Sec. 5.2); (2) the efficiency of our algorithms compared to state-of-the-art competitors (Sec. 5.3); (3) the impact parameter settings and the choice of clustering methods have on performance (Sec. 5.4); (4) benefits from the proposed enhancements of our algorithms (Sec. 5.5).

5.1 Experimental setup

Datasets and tasks. We use five datasets, whose statistical properties are shown in Table 1: (1) HR [13] is a dataset for the classification task "predicting whether an employee will change their job". It contains information provided by Finance, Sales, International, Purchasing, Marketing, and Technology departments. (2) Crop [34] is a dataset for the classification task "predicting which crop type a cropland is used for". It includes temporal, spectral, textural, and polarimetric information. (3) House [22] is a dataset with the regression task "predicting the price of a house". It includes information on the house profile, transportation availability, and nearby amenities. (4) Traffic is a dataset for the regression task "predicting the travel flow of a region at a future time interval". We use the methods in [61] to construct this dataset. Specifically, the Chengdu Road

Table 2: Parameter settings

Parameter	Value
# of iterations K	1, 2, 3, ..., **20**, 21, 22, ... 25
Size ratio ρ	5%, 10%, 15%, **20%**, 25%
Sampling rate s	**1%**, 3%, 5%, 7%, 9%
# of clusters	**10**

Network (CRN) is sourced from OpenStreetMap [46], and divided into 156 non-overlapping regions with each region spanning 2km × 2km. Then, region-based traffic flows and travel demands within a time interval of 5 mins are generated using Didi Chuxing's orders and associated trajectories in Chengdu from 10/01/2016 to 11/30/2016 [19]. After excluding rarely used regions with traffic flows of fewer than 100, 40 regions remain. Each row records a time interval, including travel demand, inflow, and outflow data for each region. (5) CIFAR10 [31] is an image dataset for the classification task "predicting the class of an image".

Data Partitioning. Recall Sec. 2.1, the input includes the initial training set d_{train}, the test set d_{test}, the validation set d_{val}, and the data pool P. Here, we employ the technique from [34] to split the datasets. Specifically, for HR, Crop, and House, we sample 1% of the (data point, label) pairs for d_{train} to replicate scenarios where limited training data is available. In addition, we sample 10% for d_{test} and another 10% for d_{val}. We designate the remaining pairs as P. For Traffic, which is used to predict traffic flows, we divide all tuples into P, d_{train}, d_{val}, d_{test} by partitioning the time intervals with the respective proportions of 79%, 1%, 10%, and 10%. CIFAR10 has a training set of 50,000 images and a test set of 10,000 images. Thus, we sample 1% for d_{train} and 99% for P using the training set for model learning. We sample 50% for d_{val} and 50% for d_{test} using the test dataset for performance evaluation.

Parameter settings. In Table 2, we summarize the key parameters (default settings in bold). Following the state-of-the-art [13], we use GMM as the default clustering method. After evaluating the GMM clustering results using all datasets and AIC scoring [1], we fix the number of clusters, $|C|$, to 10. We define the parameter ρ as the size ratio and consider the size of a mini-batch to be $l = \rho \times |d_{train}|$. We set $\rho = 20\%$ by default. We define the parameter s as the sampling rate for extracting the initial training set from the complete dataset. As previously mentioned, the default of s is set to 1%. In Eq. 3, α and β are parameters associated with the learning rate, and L_1 is a regularization parameter. Typically, $\beta = 1$ is sufficient, as suggested by [43]. Using cross-validation and a grid search, we determine near-optimal values for $\alpha = 0.1$ and $L_1 = 0.1$. In Eq. 7, the step size $\eta \ll 1$ has a minimal impact on learning, as discussed by [43]. Here, we set $\eta = 0.1$. In Eq. 11 and Eq. 12, γ adjusts the balance between the exploitation and exploration aspects of the selection process. Here, we set $\gamma = 0.05$, which mainly focuses on the exploitation.

Model selection. We assess the performance of various models using the initial training set and select the most effective one. We aim to show that our enrichment algorithms are useful even when combined with an already best-performing model. Specifically, for classification tasks with HR and Crop, we opt for a multilayer perceptron (MLP) model and an XGBoost model, respectively, due to their superior performance compared to other common classification models such as logistic regression and random forest. For the regression task with House, we choose a support vector regression (SVR) model and an XGBoost model as they exhibit superior performance compared to common regression models such as linear regression and random forest. For the regression task with Traffic, both a deep sequence learning model, LSTM [35], and a deep meta-learning model, ST-MetaNet [47], are used due to their established competence in the previous work [61]. For the classification task with CIFAR10, we employ ResNet50 [24] and VGG16 [54], recognized as well-known deep learning models for image classification.

Evaluation. To evaluate the performance of the algorithms considered, we use the following performance measures:

- *Root mean squared error (RMSE)* for regression tasks [57].
- *Area under curve (AUC)* for classification tasks [13], AUC is the area under the receiver operator characteristic (ROC) curve, and quantifies a model's capability to distinguish between classes.
- *Runtime.* The average running time over five independent runs.

Compared methods.

- ALL – The simple method that adds all (data point, label) pairs from the data pool to the training set.
- RANDOM – The method that randomly selects a mini-batch at each iteration.
- NN (*nearest neighbors*) – The method that, at each iteration, selects a mini-batch that contains the data points closest to those in the training set.
- MAB [13] – The CTS algorithm using a multi-armed bandit process to select a mini-batch at each iteration.
- DQN [13] – The CTS algorithm using reinforcement learning based on deep Q-network to select a mini-batch at each iteration.
- IAS – The proposed IAS algorithm (Alg. 2).
- IAS-AMS – The proposed IAS-AMS algorithm (Alg. 3).

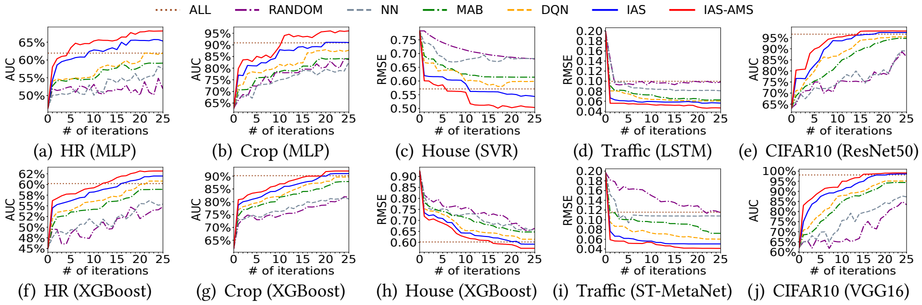
All methods except ALL, are iterative and, in each iteration, select data using a fixed mini-batch size. Similar to MAB and DQN, our algorithms, IAS and IAS-AMS, also model the data pool P as a set of clusters. Hence, we use the same clustering method and the same number of clusters in all four algorithms. We tune the parameters of MAB and DQN that are not used by our algorithms to optimize the performance and to ensure a fair and unbiased comparison.

Moreover, XGBoost being a tree-based model, faces limitations in updating parameters associated with the structure of its constituent trees through online learning for Operation 2 optimization. Thus, for Operation 2, our algorithms IAS and IAS-AMS maintain the optimal tree structure of the XGBoost model trained on the initial training set. They solely update the weights of its leaf nodes using online learning in each iteration, as these weights are computed from the gradients of training instances.

Implementation. Experiments are run on a server running Red Hat Enterprise Linux with an Intel® Xeon® CPU@2.60GHz, 512GB RAM, and two Nvidia Tesla P100 GPUs, each with 16GB of memory. We implement all algorithms in Python and code is available at [18].

5.2 Effectiveness Study

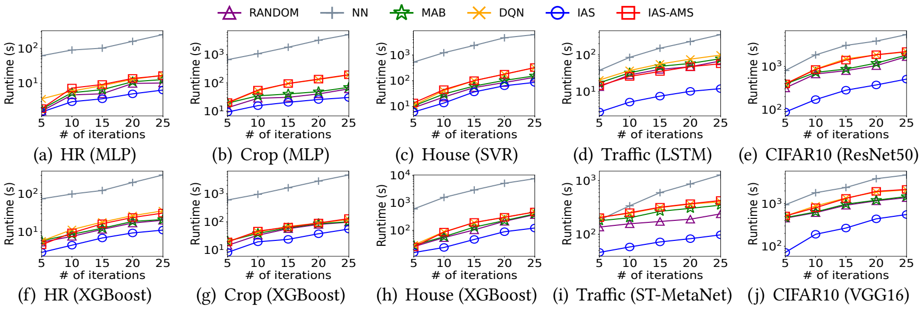
The effectiveness of the proposed algorithms, IAS and IAS-AMS, is compared to that of their competitors in Fig. 4(a)-Fig. 4(j). The analysis of results reveals compelling insights: (1) All examined algorithms exhibit performance improvement with increasing iterations, signaling the efficacy of data acquisition. (2) Among the baselines, RANDOM and NN perform least optimally. This is because



(a) HR (MLP) (b) Crop (MLP) (c) House (SVR) (d) Traffic (LSTM) (e) CIFAR10 (ResNet50)

(f) HR (XGBoost) (g) Crop (XGBoost) (h) House (XGBoost) (i) Traffic (ST-MetaNet) (j) CIFAR10 (VGG16)

Figure 4: The effectiveness of algorithms w.r.t the number of iterations.



(a) HR (MLP) (b) Crop (MLP) (c) House (SVR) (d) Traffic (LSTM) (e) CIFAR10 (ResNet50)

(f) HR (XGBoost) (g) Crop (XGBoost) (h) House (XGBoost) (i) Traffic (ST-MetaNet) (j) CIFAR10 (VGG16)

Figure 5: The efficiency of algorithms w.r.t the number of iterations.

RANDOM selects an arbitrary mini-batch in each iteration without any alignment with the ML task. NN is biased towards selecting data points closest to d_{train} in each iteration. This restricts its capacity to effectively model the relevant data distribution essential for the ML task. Moreover, RANDOM and NN struggle to converge compared to other algorithms that tend to be stable after around 25 iterations. Their lack of consideration for the ML task's unique characteristics introduces uncertainty regarding their impact on the overall task performance. (3) MAB and DQN are superior to RANDOM and NN, thanks to their ability to improve performance directly during data acquisition. The results also confirm the efficacy of clustering in representing data points within a data pool as well as the merit of employing an exploration-exploitation strategy to acquire new data iteratively. (4) Our algorithms, IAS and IAS-AMS, exhibit superior effectiveness, especially underscoring the significance of adaptive score estimation. (5) IAS-AMS outperforms IAS, validating the efficacy of IAS-AMS and its adaptive mini-batch selection in diversifying the added data per iteration. (6) With the Traffic dataset, ALL displays the poorest performance, potentially due to the inclusion of noisy data points or outliers that negatively impact model performance. This highlights the importance of selecting a pertinent subset of data points for training ML models. (7) The noticeable initial drop in RMSE or rapid rise in AUC within the initial iterations, followed by marginal changes, underscores the crucial role of data point selection and indicates that selective data choices enable more robust model training with minimal data.

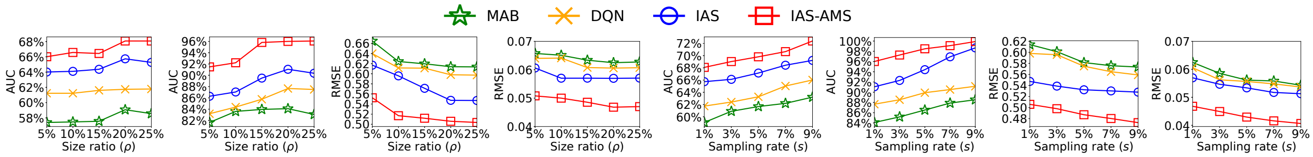
5.3 Efficiency Study

We present the runtime for the considered algorithms in Figs. 5(a)-5(j). We observe that: (1) As the number of iterations increases, the runtime for all algorithms also increases. (2) NN is the slowest algorithm because it requires too many distance computations. Note that it could be accelerated by utilizing indexing techniques [27], which we have not tested here. (3) Our algorithm IAS is the fastest, consistently outperforming all other algorithms. This highlights the value of incorporating online learning to efficiently optimize Operation 2. Online learning provides exceptional performance gains, particularly with computationally intensive DL models. IAS achieves a speed-up that is an order of magnitude better than the other algorithms on the Traffic dataset and the CIFAR10 dataset. (4) While IAS-AMS is slightly less efficient than MAB, it is still competitive with DQN. This is due to IAS-AMS having to calculate Shapley values for each cluster in every iteration, this equates to additional computational overhead compared to MAB. However, the use of online learning in IAS-AMS expedites model training, and using an efficient Shapley value approximation method [62] removes the exponential growth expected when computing exact Shapley values. Consequently, IAS-AMS remains a viable and practical choice.

5.4 Sensitivity Analysis

Here, we evaluate how the effectiveness of the proposed algorithms is impacted by varying the size of the mini-batches, the size of the initial training set, and the choice of the clustering algorithm.

The impact of mini-batch size. We show the performance evaluation results, w.r.t. mini-batch sizes in Fig. 6. Observe that: (1) As the mini-batch size ratio ρ increases, the performance generally improves for all algorithms since a greater number of training data points within each iteration is available. Conversely, a smaller mini-batch size requires more iterations to achieve acceptable performance, leading to an increase in the overall runtime. (2) The ratio $\rho = 20\%$ is the best choice in our experiments as it delivers the most favorable trade-off between effectiveness and efficiency. (3) The performance of MAB and IAS [Figs. 6(a) and 6(b)] can degrade as ρ is increased as the inclusion of noisy data points to the training set increases. In contrast, IAS-AMS is not influenced by noisy data points, which highlights the robustness of the selection process.



(a) HR (MLP) (b) Crop (MLP) (c) House (SVR) (d) Traffic (LSTM)

Figure 6: The impact of mini-batch size on effectiveness.

(a) HR (MLP) (b) Crop (MLP) (c) House (SVR) (d) Traffic (LSTM)

Figure 7: The impact of initial training set size on effectiveness.

The impact of initial training set size. Fig. 7 shows the performance evaluation results for different sizes of the initial training set. We observe that: (1) Increasing the sampling rate s improves the performance of all algorithms. This is because a larger initial training set provides richer data for superior model training from the outset. (2) Our algorithms, IAS-AMS and IAS, continue to surpass two baselines even with the enhanced initial model performance. This indicates their adeptness in selecting data points effectively, further boosting the model performance.

The impact of clustering methods. We tested four classic clustering methods, GMM [21], DBSCAN [20], k-means [39] and Mean-Shift [16], to cluster the data points. We determine the number of clusters for GMM using the AIC score [1]. We use the technique from [51] to select two parameters in DBSCAN, the radius *eps* and the density threshold *minPts*. We use the Silhouette score [49] to determine the number of clusters in k-means. We use bandwidth estimation [55] to set the parameter *bandwidth* of MeanShift.

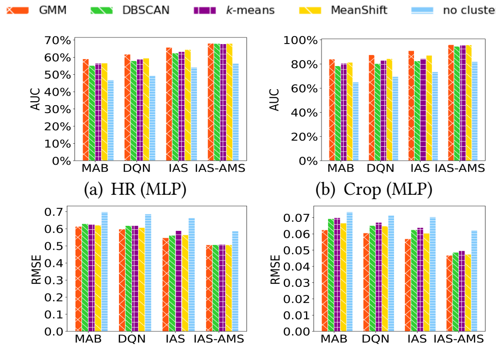
Fig. 8 shows the performance for each of the algorithms (including the no clustering case) with IAS and IAS-AMS, compared to the baselines MAB and DQN using several datasets. Observe that: (1) The choice of clustering algorithm affects the performance of the algorithm used. GMM is the most effective. For instance, on the HR dataset, the AUC for IAS is 65.7% using GMM, surpassing the performance with DBSCAN (62.4%), k-means (63.2%) or MeanShift (64.4%). (2) Regardless of the clustering algorithm used, IAS-AMS is the overall winner. This can be attributed to the sampling for the mini-batch on all clusters rather than from a single cluster; (3) The algorithms operating without data point clustering perform worse compared to any clustering-based counterpart. This is strong evidence that the advantages of data point clustering in data acquisition will enhance the performance of downstream ML tasks.

5.5 Ablation Study

Now, we perform an ablation study of our algorithms IAS and IAS-AMS. Recall that IAS introduces two new enhancements – online learning in Operation 2 to improve efficiency, and adaptive score estimation in Operation 5 to improve effectiveness. Thus, we wish to examine the following two variants of IAS in an ablation study:

- *IS*: This variant of the IAS algorithm excludes adaptive score estimation. Instead, it uses the UCB score from Eq. 4 in Operation 5 but includes online learning in Operation 2.
- *AS*: This variant of the IAS algorithm removes online learning in Operation 2 and retrains the model from scratch in each iteration while retaining adaptive score estimation in Operation 5.

Our IAS-AMS algorithm also added three other enhancements, two of which are shared with IAS, and the third which is in Operation 1 – adaptive mini-batch selection. Therefore, we also consider three other variants of IAS-AMS in the ablation study:



(a) HR (MLP) (b) Crop (MLP)

(c) House (SVR) (d) Traffic (LSTM)

Figure 8: The impact of clustering methods on effectiveness.

Table 3: The effectiveness (AUC for Crop and RMSE for House) of the proposed algorithms and their variants for two datasets and different iteration numbers.

Dataset	Algorithm	# of iterations				
		5	10	15	20	25
Crop	IAS	80.9%	87.9%	89.2%	91.1%	91.1%
	IS	-2.6%	-3.7%	-3%	-2%	-1.3%
	AS	+2.4%	+2.7%	+2.7%	+1.7%	+2.1%
	IAS-AMS	85.1%	89.6%	93.4%	95.8%	96%
	IS-AMS	-3.1%	-2.9%	-1.5%	-2.7%	-2.6%
	AS-AMS	+1.9%	+3.4%	+1.8%	+1.6%	+1.8%
	S-AMS	-1.5%	-1.2%	-0.7%	-1.2%	-1%
House	IAS	0.615	0.59	0.561	0.552	0.544
	IS	+0.025	+0.027	+0.029	+0.03	+0.03
	AS	-0.02	-0.035	-0.021	-0.022	-0.019
	IAS-AMS	0.586	0.559	0.513	0.503	0.504
	IS-AMS	+0.029	+0.028	+0.044	+0.028	+0.027
	AS-AMS	-0.024	-0.022	-0.016	-0.022	-0.024
	S-AMS	+0.017	+0.013	+0.02	+0.015	+0.01

- *IS-AMS*: This variant of the IAS-AMS algorithm removes adaptive score estimation in Operation 5 but keeps online learning in Operation 2, and adaptive mini-batch selection in Operation 1.
- *AS-AMS*: This variant of the IAS-AMS algorithm removes online learning in Operation 2 and keeps adaptive score estimation in Operation 5 and adaptive mini-batch section in Operation 1.
- *S-AMS*: This variant of the IAS-AMS algorithm removes both online learning and adaptive score estimation and only retains adaptive mini-batch section in Operation 1.

Table 3 presents the effectiveness of these variants using two datasets. To facilitate the comparison, we provide the value differences between the variant and its corresponding algorithm that includes all components. Observe that: (1) IS and IS-AMS are less effective than IAS and IAS-AMS, respectively. This is because, in online learning, the ML model is updated with each new data point added. Such updates can be influenced by noisy or outlier data

Table 4: The runtime (in seconds) of the proposed algorithms and their variants for two datasets and different iteration numbers.

Dataset	Algorithm†	# of iterations				
		5	10	15	20	25
Crop	IAS	9.4	15.7	20.6	25.9	30.6
	AS	+47.2	+78.7	+87.2	+109.9	+166.5
	IAS-AMS	20.2	55.5	94.9	135.9	194.3
	AS-AMS	+55.7	+96.2	+167.9	+302.1	+441.2
House	IAS	6.1	13.6	36.6	63.8	86.3
	AS	+9.8	+24.4	+49.1	+115.6	+514.5
	IAS-AMS	13.7	44.9	100.7	179.4	325.6
	AS-AMS	+76.3	+205.5	+330.3	+514.5	+1002.1

†We compare IAS and IAS-AMS solely with AS and AS-AMS, respectively, as an efficiency enhancement using only online learning.

points, causing the ML model to overfit individual observations, and hence noise, which in turn results in higher variance. In contrast, retraining the ML model using the entire training set helps to reduce the impact of noise and provides a more robust result. (2) AS and AS-AMS are more effective compared to IAS and IAS-AMS, respectively. This demonstrates that adaptive score estimation excels in selecting data points that lead to the greatest performance improvements in an ML model. Thus, while there may be a trade-off in model performance due to online learning, IAS and IAS-AMS still outperform the baselines, as illustrated in Fig. 4. (3) IAS-AMS and its variants consistently outperform IAS, even when including only adaptive mini-batch selection (S-AMS). This highlights the advantages introduced in the changes to Operation 1 using adaptive mini-batch selection in terms of overall effectiveness.

Table 4 shows the runtime for our algorithms and each variant, as the number of iterations is varied. Observe that: (1) AS and AS-AMS are less efficient than IAS and IAS-AMS, respectively. This is attributed to the addition of online learning in IAS and IAS-AMS to improve Operation 2. In contrast, AS and AS-AMS retrain the model in each iteration using the entire training set, incurring a high computational cost. (2) Despite IAS-AMS being slower than IAS, it is still competitive with the baselines, as shown in Fig. 5. It also has the highest overall effectiveness in Fig. 4. This trade-off between effectiveness and efficiency positions IAS-AMS as the best option in new data acquisition strategies.

6 RELATED WORK

Data acquisition is the process of extracting labeled data that are the most useful to enhance the performance of a supervised model from a pool of available data. Existing work uses data acquisition to improve a model's performance [13, 15, 30, 34, 37, 60, 63]. Li et al. [34] introduce an interactive process where consumers provide queries to obtain data for enhancing the accuracy of a specific model, with data providers possessing the data to make them available for training purposes. Zhang et al. [63] present an information update system that ensures that new datasets can be easily obtained by a data market for the purpose of improving model performance. Chai et al. [13] study the same problem as ours and propose a state-of-the-art CTS algorithm using a two-step framework to iteratively select data points from a set of clusters. However, the CTS algorithm has several practical limitations that limit its effectiveness and efficiency, as discussed in Sec. 1. To remove the limitations in existing CTS-based algorithms, we propose new algorithms which enhance CTS in several ways, including the use of online learning, adaptive score estimation, and adaptive mini-batch selection.

Feature augmentation refers to the process of enhancing an ML model by introducing new features during training. This approach has recently garnered significant research attention [17, 28, 33, 36, 38, 53, 64, 66]. For example, Kumar et al. [33] design easy-to-understand decision rules to augment features when joins can be avoided safely without affecting ML accuracy significantly. Chepurko et al. [17] propose an end-to-end approach to automatically augment features to improve the performance of a targeted model. Liu et al. [36] use reinforcement learning to augment features with an exploration-exploitation strategy. However, these approaches select features (columns in structured datasets) rather than data points (rows in structured datasets), which is the focus of this work.

Coreset selection refers to the process of identifying a subset (a.k.a. coreset) in a training set such that the ML model being trained on this subset performs on par with the ML model trained on the entire training set. There is a great deal of previous work addressing this problem [7, 10, 12, 14, 26, 57, 59, 65]. For instance, Huang et al. [26] propose a sequential method to select and update the coreset during training and can reduce the complexity when using gradient descent algorithms. Wang et al. [57] propose an efficient coreset selection approach that avoids materializing the improved table by performing the gradient computations using partial feature similarity between tuples. Chai et al. [12] consider coreset selection over incomplete data by modeling the unknown complete data as the combinations of possible data repairs. Despite apparent similarities, coreset selection is a different problem from the one explored here as it is relevant in scenarios where ample labeled data is available for training, and one of the key goals is to improve ML training efficiency by reducing the size of the training set. In contrast, we explore scenarios where limited labeled data is available for training, and the aim is to improve the existing model performance using new data points from multiple sources to enrich the original training set.

7 CONCLUSION

In this paper, we study the problem of data acquisition for enhancing ML performance. The state-of-the-art solution for this problem is the CTS algorithm, which relies on five operations to select data points from relevant clusters iteratively. However, CTS is not efficient and proves ineffective in certain cases. Specifically, Operations 1, 2, and 5 (Fig. 1) can be bottlenecks. Therefore, we propose the IAS algorithm, which improves the effectiveness and efficiency by redefining Operation 2 using online learning and redefining Operation 5 using adaptive score estimation. Additionally, we propose the IAS-AMS algorithm, which further improves the effectiveness of IAS by adding adaptive mini-batch selection to Operation 1. Comprehensive experiments using real datasets verify that IAS outperforms all related baselines in both effectiveness and efficiency, and IAS-AMS is the most effective overall.

ACKNOWLEDGMENTS

This work is supported in part by ARC DP240101211 and DP220101434.

REFERENCES

[1] Ken Aho, DeWayne Derryberry, and Teri Peterson. 2014. Model selection for ecologists: the worldviews of AIC and BIC. *Ecology* 95, 3 (2014), 631–636.

[2] NYU Auctus. 2024. https://auctus.vida-nyu.org/

[3] Peter Auer. 2000. Using Upper Confidence Bounds for Online Learning. In *FOCS*. 270–279.

[4] Dean A. Bodenham and Niall M. Adams. 2017. Continuous monitoring for changepoints in data streams using adaptive estimation. *Stat. Comput.* 27, 5 (2017), 1257–1270.

[5] Vivek S Borkar. 2009. *Stochastic approximation: a dynamical systems viewpoint.* Vol. 48. Springer.

[6] Rodica Branzei, Dinko Dimitrov, and Stef Tijs. 2008. *Models in cooperative game theory.* Vol. 556. Springer Science & Business Media.

[7] Vladimir Braverman, Dan Feldman, and Harry Lang. 2016. New Frameworks for Offline and Streaming Coreset Constructions. *CoRR* abs/1612.00889 (2016). http://arxiv.org/abs/1612.00889

[8] Dan Brickley, Matthew Burgess, and Natasha F. Noy. 2019. Google Dataset Search: Building a search engine for datasets in an open Web ecosystem. In *WWW*. 1365–1375.

[9] Mark Alexander Burgess and Archie C. Chapman. 2021. Approximating the Shapley Value Using Stratified Empirical Bernstein Sampling. In *IJCAI*. 73–81.

[10] Trevor Campbell and Tamara Broderick. 2018. Bayesian Coreset Construction via Greedy Iterative Geodesic Ascent. In *ICML*. 697–705.

[11] Javier Castro, Daniel Gómez, and Juan Tejada. 2009. Polynomial calculation of the Shapley value based on sampling. *Comput. Oper. Res.* 36, 5 (2009), 1726–1730.

[12] Chengliang Chai, Jiabin Liu, Nan Tang, Ju Fan, Dongjing Miao, Jiayi Wang, Yuyu Luo, and Guoliang Li. 2023. GoodCore: Data-effective and Data-efficient Machine Learning through Coreset Selection over Incomplete Data. *Proc. ACM Manag. Data* 1, 2 (2023), 157:1–157:27.

[13] Chengliang Chai, Jiabin Liu, Nan Tang, Guoliang Li, and Yuyu Luo. 2022. Selective Data Acquisition in the Wild for Model Charging. *Proc. VLDB Endow.* 15, 7 (2022), 1466–1478.

[14] Chengliang Chai, Jiayi Wang, Nan Tang, Ye Yuan, Jiabin Liu, Yuhao Deng, and Guoren Wang. 2023. Efficient Coreset Selection with Cluster-based Methods. In *KDD*. 167–178.

[15] Yiling Chen, Nicole Immorlica, Brendan Lucier, Vasilis Syrgkanis, and Juba Ziani. 2018. Optimal Data Acquisition for Statistical Estimation. In *EC*. 27–44.

[16] Yizong Cheng. 1995. Mean Shift, Mode Seeking, and Clustering. *IEEE Trans. Pattern Anal. Mach. Intell.* 17, 8 (1995), 790–799.

[17] Nadiia Chepurko, Ryan Marcus, Emanuel Zgraggen, Raul Castro Fernandez, Tim Kraska, and David R. Karger. 2020. ARDA: Automatic Relational Data Augmentation for Machine Learning. *Proc. VLDB Endow.* 13, 9 (2020), 1373–1387.

[18] Source Code. 2024. https://github.com/rmitbggroup/da-ml.

[19] GAIA Open Data. 2016. https://outreach.didichuxing.com/research/opendata/.

[20] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *KDD*. 226–231.

[21] Mário A. T. Figueiredo and Anil K. Jain. 2002. Unsupervised Learning of Finite Mixture Models. *IEEE Trans. Pattern Anal. Mach. Intell.* 24, 3 (2002), 381–396.

[22] Guangliang Gao, Zhifeng Bao, Jie Cao, A. Kai Qin, and Timos Sellis. 2022. Location-Centered House Price Prediction: A Multi-Task Learning Approach. *ACM Trans. Intell. Syst. Technol.* 13, 2 (2022), 32:1–32:25.

[23] Amirata Ghorbani and James Y. Zou. 2019. Data Shapley: Equitable Valuation of Data for Machine Learning. In *ICML*. 2242–2251.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR*. 770–778.

[25] Steven C. H. Hoi, Doyen Sahoo, Jing Lu, and Peilin Zhao. 2021. Online learning: A comprehensive survey. *Neurocomputing* 459 (2021), 249–289.

[26] Jiawei Huang, Ruomin Huang, Wenjie Liu, Nikolaos M. Freris, and Hu Ding. 2021. A Novel Sequential Coreset Method for Gradient Descent Algorithms. In *ICML*. 4412–4422.

[27] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 1 (2011), 117–128.

[28] Bin-Bin Jia and Min-Ling Zhang. 2019. Multi-Dimensional Classification via kNN Feature Augmentation. In *AAAI*. 3975–3982.

[29] Ruoxi Jia, David Dao, Boxin Wang, Frances Ann Hubis, Nezihe Merve Gürel, Bo Li, Ce Zhang, Costas J. Spanos, and Dawn Song. 2019. Efficient Task-Specific Data Valuation for Nearest Neighbor Algorithms. *Proc. VLDB Endow.* 12, 11 (2019), 1610–1623.

[30] Yuqing Kong, Grant Schoenebeck, Biaoshuai Tao, and Fang-Yi Yu. 2020. Information Elicitation Mechanisms for Statistical Estimation. In *AAAI*. 2095–2102.

[31] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*. 1106–1114.

[33] Arun Kumar, Jeffrey F. Naughton, Jignesh M. Patel, and Xiaojin Zhu. 2016. To Join or Not to Join?: Thinking Twice about Joins before Feature Selection. In *SIGMOD*. 19–34.

[34] Yifan Li, Xiaohui Yu, and Nick Koudas. 2021. Data Acquisition for Improving Machine Learning Models. *Proc. VLDB Endow.* 14, 10 (2021), 1832–1844.

[35] Binbing Liao, Jingqing Zhang, Chao Wu, Douglas McIlwraith, Tong Chen, Shengwen Yang, Yike Guo, and Fei Wu. 2018. Deep sequence learning with auxiliary information for traffic prediction. In *KDD*. 537–546.

[36] Jiabin Liu, Chengliang Chai, Yuyu Luo, Yin Lou, Jianhua Feng, and Nan Tang. 2022. Feature Augmentation with Reinforcement Learning. In *ICDE*. 3360–3372.

[37] Jiabin Liu, Fu Zhu, Chengliang Chai, Yuyu Luo, and Nan Tang. 2021. Automatic Data Acquisition for Deep Learning. *Proc. VLDB Endow.* 14, 12 (2021), 2739–2742.

[38] Zichang Liu, Zhiqiang Tang, Xingjian Shi, Aston Zhang, Mu Li, Anshumali Shrivastava, and Andrew Gordon Wilson. 2023. Learning Multimodal Data Augmentation in Feature Space. In *ICLR*.

[39] Stuart P. Lloyd. 1982. Least squares quantization in PCM. *IEEE Trans. Inf. Theory* 28, 2 (1982), 129–136.

[40] Yuyu Luo, Nan Tang, Guoliang Li, Chengliang Chai, Wenbo Li, and Xuedi Qin. 2021. Synthesizing Natural Language to Visualization (NL2VIS) Benchmarks from NL2SQL Benchmarks. In *SIGMOD*. 1235–1247.

[41] Sasan Maleki. 2015. *Addressing the computational issues of the Shapley value with applications in the smart grid.* Ph.D. Dissertation. University of Southampton.

[42] H. Brendan McMahan. 2011. Follow-the-Regularized-Leader and Mirror Descent: Equivalence Theorems and L1 Regularization. In *AISTATS*, Vol. 15. 525–533.

[43] H. Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, and Jeremy Kubica. 2013. Ad click prediction: a view from the trenches. In *KDD*. 1222–1230.

[44] H. Brendan McMahan and Matthew J. Streeter. 2010. Adaptive Bound Optimization for Online Convex Optimization. In *COLT*. 244–256.

[45] Fatemeh Nargesian, Abolfazl Asudeh, and H. V. Jagadish. 2022. Responsible Data Integration: Next-generation Challenges. In *SIGMOD*. 2458–2464.

[46] OpenStreetMap. 2021. https://www.openstreetmap.org/.

[47] Zheyi Pan, Yuxuan Liang, Weifeng Wang, Yong Yu, Yu Zheng, and Junbo Zhang. 2019. Urban traffic prediction from spatio-temporal data using deep meta learning. In *KDD*. 1720–1730.

[48] Victor M Panaretos and Yoav Zemel. 2019. Statistical aspects of Wasserstein distances. *Annual review of statistics and its application* 6 (2019), 405–431.

[49] Peter J Rousseeuw. 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics* 20 (1987), 53–65.

[50] Benedek Rozemberczki, Lauren Watson, Péter Bayer, Hao-Tsung Yang, Oliver Kiss, Sebastian Nilsson, and Rik Sarkar. 2022. The Shapley Value in Machine Learning. In *IJCAI*. 5572–5579.

[51] Erich Schubert, Jörg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. 2017. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. *ACM Trans. Database Syst.* 42, 3 (2017), 19:1–19:21.

[52] Robert J Serfling. 1974. Probability inequalities for the sum in sampling without replacement. *The Annals of Statistics* (1974), 39–48.

[53] Ruoqi Shen, Sébastien Bubeck, and Suriya Gunasekar. 2022. Data Augmentation as Feature Manipulation. In *ICML*. 19773–19808.

[54] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *ICLR*.

[55] Sklearn. 2023. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.estimate_bandwidth.html

[56] Matthew J. Streeter and H. Brendan McMahan. 2010. Less Regret via Online Conditioning. *CoRR* abs/1002.4862 (2010).

[57] Jiayi Wang, Chengliang Chai, Nan Tang, Jiabin Liu, and Guoliang Li. 2022. Coresets over Multiple Tables for Feature-rich and Data-efficient Machine Learning. *Proc. VLDB Endow* 16, 1 (2022), 64–76.

[58] Lin Xiao. 2009. Dual Averaging Method for Regularized Stochastic Learning and Online Optimization. In *NIPS*. 2116–2124.

[59] Yu Yang, Hao Kang, and Baharan Mirzasoleiman. 2023. Towards Sustainable Learning: Coresets for Data-efficient Deep Learning. In *ICML*. 39314–39330.

[60] Jinsung Yoon, Sercan Ömer Arik, and Tomas Pfister. 2020. Data Valuation using Reinforcement Learning. In *ICML*, Vol. 119. 10842–10851.

[61] Haitao Yuan, Guoliang Li, Zhifeng Bao, and Ling Feng. 2021. An Effective Joint Prediction Model for Travel Demands and Traffic Flows. In *ICDE*. 348–359.

[62] Jiayao Zhang, Qiheng Sun, Jinfei Liu, Li Xiong, Jian Pei, and Kui Ren. 2023. Efficient Sampling Approaches to Shapley Value Approximation. *Proc. ACM Manag. Data* 1, 1 (2023), 48:1–48:24.

[63] Meng Zhang, Ahmed Arafa, Ermin Wei, and Randall Berry. 2021. Optimal and Quantized Mechanism Design for Fresh Data Acquisition. *IEEE J. Sel. Areas Commun.* 39, 5 (2021), 1226–1239.

[64] Yifei Zhang, Hao Zhu, Zixing Song, Piotr Koniusz, and Irwin King. 2022. COSTA: Covariance-Preserving Feature Augmentation for Graph Contrastive Learning. In *KDD*. 2524–2534.

[65] Haizhong Zheng, Rui Liu, Fan Lai, and Atul Prakash. 2023. Coverage-centric Coreset Selection for High Pruning Rates. In *ICLR*.

[66] Tianfei Zhou and Ender Konukoglu. 2023. FedFA: Federated Feature Augmentation. In *ICLR*.