Selective Data Acquisition in the Wild for Model Charging

Chengliang Chai
Tsinghua University
Beijing, China
ccl@mail.tsinghua.edu.cn

Jiabin Liu
Tsinghua University
Beijing, China
liujb19@mails.tsinghua.edu.cn

Nan Tang
QCRI
Doha, Qatar
ntang@hbku.edu.qa

Guoliang Li
Tsinghua University
Beijing, China
liguoliang@tsinghua.edu.cn

Yuyu Luo
Tsinghua University
Beijing, China
luoyy18@mails.tsinghua.edu.cn

ABSTRACT

The lack of sufficient labeled data is a key bottleneck for practitioners in many real-world supervised machine learning (ML) tasks. In this paper, we study a new problem, namely *selective data acquisition in the wild for model charging*: given a supervised ML task and data in the wild (*e.g.,* enterprise data warehouses, online data repositories, data markets, and so on), the problem is to select labeled data points from the data in the wild as additional train data that can help the ML task. It consists of two steps (Fig. 1). The first step is to discover relevant datasets (*e.g.,* tables with similar relational schema), which will result in a set of candidate datasets. Because these candidate datasets come from different sources and may follow different distributions, not all data points they contain can help. The second step is to select which data points from these candidate datasets should be used. We build an end-to-end solution. For step 1, we piggyback off-the-shelf data discovery tools. Technically, our focus is on step 2, for which we propose a solution framework called **`AutoData`**. It first clusters all data points from candidate datasets such that each cluster contains similar data points from different sources. It then iteratively picks which cluster to use, samples data points (*i.e.,* a mini-batch) from the picked cluster, evaluates the mini-batch, and then revises the search criteria by learning from the feedback (*i.e.,* reward) based on the evaluation. We propose a multi-armed bandit based solution and a Deep Q Networks-based reinforcement learning solution. Experiments using both relational and image datasets show the effectiveness of our solutions.

PVLDB Reference Format:
Chengliang Chai, Jiabin Liu, Nan Tang, Guoliang Li, and Yuyu Luo.
Selective Data Acquisition in the Wild for Model Charging. PVLDB, 15(7): 1466-1478, 2022.
doi:10.14778/3523210.3523223

1 INTRODUCTION

Data-centric ML. In many supervised ML projects, the main bottleneck is the lack of sufficient labeled train data (*a.k.a.* data-centric ML) [9, 28, 36, 37], not which ML models to use and how to optimize these models (*a.k.a.* model-centric ML), especially for ML practitioners.

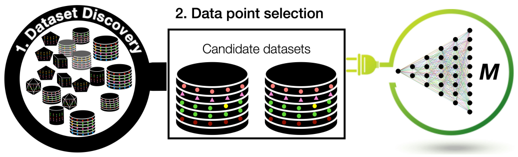


Figure 1: Selective data acquisition for model charging.

Example 1. [*Insufficient train data.*] *Consider Fig. 2(a) that shows a dataset* $T_{\texttt{train}}$*, which is used to train a regression model to predict the house* `Price` *at Kolkata, India, based on the features (*`City`*,* `Year`*,* `Area`*,* `Security`*). The test data* $T_{\texttt{test}}$ *is given in Fig. 2(b), whose* `Price` *values are to be predicted. The ground truth* `Price` *values are also provided for helping the discussion.*

Figure 3(a) shows both the learned model using only $T_{\texttt{train}}$ *(*i.e., *Line 1) and the ground truth model that we want to learn (*i.e., *Line 2). Because* $T_{\texttt{train}}$ *is small and does not contain sufficient data points (*e.g., *there are many houses with* `Area` > 1000 *or* < 700 *in* $T_{\texttt{test}}$*, but all houses in* $T_{\texttt{train}}$ *have* `Area` *in the range* $[700, 1000]$*), the model trained using* $T_{\texttt{train}}$ *(*i.e., *Line 1) is not good enough for* $T_{\texttt{test}}$*.*

Data acquisition. The process of getting more labeled data is known as data acquisition, which is categorized into two classes: human-in-the-loop and automatic data acquisition. Human-in-the-loop data acquisition includes weak supervision where users need to define rules (*e.g.,* Snorkel [43], data programming [42]), and crowd- and expert-sourcing. Automatic data acquisition uses automatic methods to obtain more train data.

Selective data acquisition in the wild for model charging. We ask whether it is possible to find useful data points from the data in the wild as shown in Fig. 1 so as to "charge" the model, where we know that different datasets may follow different data distributions, but meanwhile, we also hypothesize that some data points can help.

For doing so, the first step is to select candidate datasets. For tables, this is known as data discovery with many off-the-shelf

Proceedings of the VLDB Endowment, Vol. 15, No. 7 ISSN 2150-8097.
doi:10.14778/3523210.3523223

The first two authors contributed equally to this research. Guoliang Li is the corresponding author.

City	Year	Area	Security	Price
Kolkata	2009	710	No	3,200,000
Kolkata	2013	770	No	3,850,000
Kolkata	2007	935	No	2,524,000
Kolkata	2006	973	Yes	3,611,000

(a) $T_{\mathtt{train}}$: Train dataset (learn to predict "Price").

City	Year	Area	Security	Price	Ground Truth
Kolkata	2017	350	No	?	2,100,000
Kolkata	2019	465	Yes	?	4,365,000
Kolkata	2015	572	No	?	3,268,000
Kolkata	2012	655	Yes	?	2,599,000
Kolkata	2012	735	No	?	3,300,000
Kolkata	2017	881	Yes	?	4,698,000
Kolkata	2011	1123	Yes	?	3,324,000
Kolkata	2014	1210	Yes	?	5,000,000

(b) $T_{\mathtt{test}}$: Test dataset (predict the "Price" column).

Figure 2: Sample train and test datasets.

tools [11, 16, 34]. For images, there are many benchmarks, as well as Web APIs such as Google, Baidu or Azure image search.

Example 2. [*Using all data points in candidate datasets.*] *Fig. 3(b)(c) and (d) show three datasets that also contain house price information in different cities of India,* i.e., *D_1, D_2 and D_3 for Banglore, Mumbai, and Delhi, respectively. They have different schemata with $T_{\mathtt{train}}$ and $T_{\mathtt{test}}$ in Fig. 2, but they can be used as train data.*

*A straightforward solution is to add all these datasets to the train data (*i.e., *$T_{\mathtt{train}} := T_{\mathtt{train}} \cup D_1 \cup D_2 \cup D_3$) and train a model. By doing so, we can obtain a model as shown in Fig. 3(a) Line 3, which, unfortunately, deviates far from the ground truth model Line 2.*

Our question is that: given candidate datasets, whether selecting some data points will be better than using all?

Example 3. [*Selected data points.*] *If we can select "good" data points, such as $\{r_1\}$ from D_1, $\{s_1, s_2, s_3\}$ from D_2, and $\{t_1, t_4\}$ from D_3 as additional train data that are highlighted in Figs. 3(b–d), we can use them in Fig. 3(a) (*i.e., *those annotated by green frames) and train a model Line 4. Clearly, Line 4 is much closer, thus better than Line 1 and Line 3, to the ground truth model Line 2.*

Example 1 shows that more labeled data is needed. Example 2 tells us using all data points is not ideal. Example 3 shows it is more beneficial to select some data points.

Challenges. There are two essential challenges. First, candidate datasets may come from various data distributions that are different from the desired data distribution of the ML task, which is unknown. Second, many data points in these candidate sets are not *good w.r.t.* our ML task, which raises the challenge that how can we effectively select and measure which new data points should be added.

Contributions. Our contributions are summarized as follows.

(1) Selective data acquisition in the wild for model charging. We study the problem of automatic data acquisition for supervised ML in a new setting where the supervised ML task does not have enough train data, and it has access to the data in the wild (Section 2). Note that datasets in the wild are heterogeneous and not all of data points can help the task.

(2) A solution framework. We propose a solution framework (see Fig. 1) that consists of two steps: **dataset discovery** that selects candidate datasets and **data points selection** that selects data points from these candidate datasets. (Section 3)

(3)* `AutoData` *with multi-armed bandit. We introduce a classical multi-armed bandit based solution to handle the exploration-exploitation trade-off for **`AutoData`**. (Section 4)

(4)* `AutoData` *with Deep Q Network-based reinforcement learning (RL). Another effective model is to use Deep Q learning based RL, which learns a neural network to approximate the Q-table (*i.e.,* a simple but huge lookup table that calculates the maximum expected future rewards for action at each state) that decides which cluster to select based on the current status. (Section 5)

(5) Evaluation. We conduct extensive experiment to show that our methods can effectively select data points from different data sources and improve the ML performance by maximum 14.8% and 8.3% on relational and image datasets, respectively. (Section 6)

2 PRELIMINARY

Supervised machine learning. We consider *supervised ML* as training a model M to learn a function $f()$ that maps an input to an output based on example input-output pairs as $f : \mathbf{X} \rightarrow \mathbf{Y}$.

We use $M(A)$ to denote the model M that is trained with dataset A, and the notation $M(A, B)$ to denote the model M that is trained with A and is evaluated with dataset B.

Train/validation/test datasets. A set of labeled dataset T is typically split into three disjoint subsets as train/validation/test ($T_{\mathtt{train}}/T_{\mathtt{val}}/T_{\mathtt{test}}$). $T_{\mathtt{test}}$ is completely held-out during training.

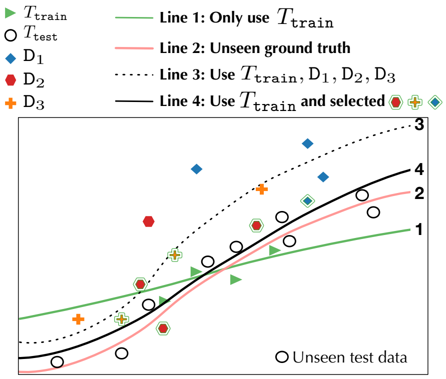
Data in the wild. We use the term *data in the wild* to generally refer to all datasets that one can have access to, including data lakes, data markets, online data repositories, enterprise data warehouses, and so on. More specifically, for supervised ML, we consider it as a set of datasets $\mathcal{D} = \{D_1, \ldots, D_m\}$, where D_i ($i \in [1, m]$) is a set of (data point, label) pairs.

Candidate datasets. The *candidate datasets w.r.t.* a supervised ML task M, denoted by $\mathcal{D}^c$, is a subset of $\mathcal{D}$ that contains datasets "relevant" to M. For tabular data, the *relevance* typically means that these candidate datasets have the same or highly overlapping relational schema with $T_{\mathtt{train}}$. For image data, the *relevance* typically means that these candidate datasets contain images that have the same labels (*e.g.,* {cat, dog, bird, fish}) as $T_{\mathtt{train}}$.

Candidate data pool. The *candidate data pool* (or simply data pool), denoted by $\mathcal{P}$, is the union of all data points in the candidate datasets, *i.e.,* $\mathcal{P} = \bigcup_{(x,y) \in D_i, D_i \in \mathcal{D}^c} (x, y)$.

Selective data acquisition for model charging. Given a supervised ML task with a pre-specified model M, train/validation/test datasets ($T_{\mathtt{train}}/T_{\mathtt{val}}/T_{\mathtt{test}}$), and a candidate data pool $\mathcal{P}$, the problem is to select a subset $\mathcal{P}^* \subset \mathcal{P}$ using $T_{\mathtt{train}}$ and $T_{\mathtt{val}}$, such that it can obtain the most performance improvement of the supervised ML task on $T_{\mathtt{test}}$: $\mathcal{P}^* = \arg\max_{\mathcal{P}' \subset \mathcal{P}} M(T_{\mathtt{train}} \cup \mathcal{P}', T_{\mathtt{test}}) - M(T_{\mathtt{train}}, T_{\mathtt{test}})$.

Example 4. [*Selective data acquisition.*] *Given an ML task of training a regression model using $T_{\mathtt{train}}$ as shown in Fig. 2 and a data*



(a) Learned models using different train datasets

	City	Year	Area	Security	Swimming Pool	Garage	Price
r_1	Bangalore	2017	1210	Yes	Yes	Yes	5,700,000
r_2	Bangalore	2018	3340	Yes	Yes	No	30,000,000
r_3	Bangalore	2016	2502	Yes	Yes	Yes	20,000,000
r_4	Bangalore	2009	2293	Yes	Yes	No	9,630,000

(b) House price dataset from Bangalore (D_1).

	City	Year	Area	Security	Resale	Garden	Gym	Price
s_1	Mumbai	2019	425	Yes	Yes	No	No	5,500,000
s_2	Mumbai	2013	720	No	Yes	No	No	4,850,000
s_3	Mumbai	2018	1060	Yes	Yes	Yes	Yes	11,000,000
s_4	Mumbai	2016	1680	Yes	Yes	No	Yes	15,000,000

(c) House price dataset from Mumbai (D_2).

	City	Year	Area	24×7 Security	Intercom	Price
t_1	Delhi	2007	385	No	No	3,300,000
t_2	Delhi	2009	435	No	Yes	2,500,000
t_3	Delhi	2014	600	No	No	12,500,000
t_4	Delhi	2004	900	No	No	5,800,000

(d) House price dataset from Delhi (D_3).

Figure 3: Examples of models learned from different train datasets. (b) (c) and (d) are three candidate datasets D_1, D_2, and D_3, respectively. In (a), each line represents an ML model, where each point in the line represents a predicted value based on a set of features. Line 1 is a model learned only with $T_{\texttt{train}}$; Line 2 is the ideal but unknown ground truth *w.r.t.* $T_{\texttt{test}}$; Line 3 uses $T_{\texttt{train}}$ and all data points from candidate datasets (*i.e.,* $T_{\texttt{train}} \cup D_1 \cup D_2 \cup D_3$); Line 4 uses $T_{\texttt{train}}$ and selected data points.

pool $\mathcal{P}$ that consists of all data points of $\{D_1, D_2, D_3\}$ in Fig. 3, the problem is to select good data points, such as those data points with green frames as shown in Fig. 3(a), which will result in Line 4.

3 A SOLUTION FRAMEWORK

3.1 Dataset Discovery

As shown in Fig. 1, the first step, namely data discovery, is to discover datasets that are relevant to the given ML task. In this work, we mainly support relational and image datasets.

For tables, there have been many works on dataset discovery on data lakes [16, 34, 35, 38]. Most of them need to host a server locally to store and index datasets [16, 34, 35]. Instead, we adopt a more flexible and extensible strategy that uses Web APIs for this purpose. More specifically, by default, we use NYU Auctus REST API [38] that supports the search of "unionable tables" which have significant attribute overlap with the table $T_{\texttt{train}}$. Google and Kaggle only support keyword based dataset search, and wrappers for schema alignment are needed to select candidate datasets. Moreover, if the selected datasets do not have an aligned attribute with $T_{\texttt{train}}$, we use NULL values for this attribute.

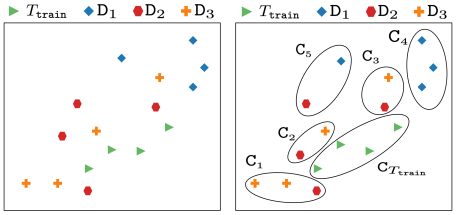
For image datasets, we can either search images in public benchmarks (*e.g.,* ImageNet) or use Web APIs (*e.g.,* Google, Azure, or Baidu) by specifying a label (*e.g.,* cat, dog, or bird).

3.2 Data Point Selection

3.2.1 ***Modeling Heterogeneous Datasets.*** Note that these candidate datasets come from different sources and may follow different data distributions. A common practice of modeling such heterogeneous datasets is data clustering, such that the data points in each cluster are similar (or "homogeneous").

Next, we will first show that keeping data points in their original dataset is not ideal, followed by discussing what will be a good clustering strategy.

Keeping in original datasets. A straightforward way is to keep each data point in its original dataset and acquire data from each dataset, which can be seen as clustering data points based on the datasets they come from.



(a) Keeping in original datasets

(b) Clustering based on GMM

Figure 4: From candidate datasets to clusters.

Example 5. [*Keeping in original datasets.*] *Consider the three datasets D_1, D_2, and D_3 in Fig. 3. If we keep data points by datasets, we will get three groups, each per dataset.*

Fig. 4(a) depicts that the feature spaces of different data points from different datasets may highly overlap. Evidently, deciding which dataset is helpful might be very hard, because only a small subset from each dataset might help. Therefore, we do not target at which datasets are helpful; instead, we aim at finding which data points are helpful, for which we need to model all data points in a more fine-grained manner than using datasets.

Naturally, we need clustering algorithms that can automatically partition all data points into a number of distinct groups that data points within groups are similar and across groups are dissimilar. We have evaluated several common clustering methods such as Multivariate Gaussian Mixture Model (GMM) [17], DBSCAN [15] and Mean-Shift [10] (see more details in Section 6.4). We empirically found that GMM is a robust choice and thus we use GMM by default. Note that practitioners or domain experts can decide which clustering methods to use for specific datasets and tasks.

Multivariate Gaussian Mixture Model (GMM). GMM is a probabilistic model for representing the presence of subpopulations within an overall population, *i.e.,* the data pool in our problem.

Similar data points can be clustered automatically as a subpopulation modeled by a Gaussian distribution, and multiple clusters (*i.e.,* subpopulations) constitute the GMM.

Formally, GMM takes as input $\mathcal{P}$ and uses Expectation-Maximization (EM) algorithm [14] to compute the Probability Distribution Function (PDF) of data in $\mathcal{P}$, *i.e.,* $p(x) = \sum_{i=1}^{g} \pi_i p_i(x; \mu_i, \Sigma_i)$, where x denotes each input object in $\mathcal{P}$. More specifically:

(1) g is the number of Gaussian distributions, the #-clusters;

(2) Each subpopulation $p_i(\mu_i, \Sigma_i)$ corresponds to a cluster, denoted by C_i $(i \in [1, g])$.

(3) $\forall x \in \mathcal{P}$, we compute the posterior probability that x belongs to each cluster (the probability can be obtained from the results of EM algorithm), and assign x to the cluster with the highest probability.

Fortunately, a good number g can be set automatically by minimizing Akaike Information Criterion (AIC) [1] of points in $\mathcal{P}$.

Example 6. [*Clustering by GMM.*] *Fig. 4(b) shows the five clusters* $\{C_1, C_2, C_3, C_4, C_5\}$ *that are computed by GMM over all data points in* $\{D_1, D_2, D_3\}$. *Each cluster may contain data points from different datasets,* e.g.,, C_1 *contains one data point from* D_2 *and two data points from* D_3.

As shown in Example 6, after we use GMM to generate clusters C_i $(i \in [1, 5])$, as well as their parameters to represent the data in $\mathcal{P}$, the data points in each cluster are relatively homogeneous. Thus, in the following of this paper, we will devise solutions based on the clusters computed by GMM.

`AutoData`: Iterative data point selection from clusters. `AutoData` takes train $T_{\texttt{train}}$, validation $T_{\texttt{val}}$, a candidate data pool $\mathcal{P}$ of data points that are modeled as clusters $C = \{C_1, \ldots, C_n\}$, and a pre-defined model M as input. It iteratively interacts with these clusters and modifies the train data with the following operations: selecting a mini-batch as new train data, re-training the model M by adding the new mini-batch, evaluating the benefit of this new mini-batch, deleting a mini-batch if it is not good, and updating the search criteria based on the feedback of model evaluation.

The above iterative process will terminate when some pre-specified stopping criteria is met (*e.g.,* at most 30 iterations, or at most 30 minutes, or the model has converged).

4 MULTI-ARMED BANDIT FOR `AUTODATA`

4.1 Bridging MAB and `AutoData`

Multi-armed bandit (MAB). MAB [53] is a simple yet powerful algorithmic framework that makes decisions over time under uncertainty. It considers N possible actions to choose from; pulling each possible action is also known as an *arm.* It models an agent that simultaneously attempts to acquire new knowledge (*i.e.,* exploration) and optimize their decisions based on existing knowledge (*i.e.,* exploitation). The agent attempts to balance these competing tasks to maximize their total values over the period of time considered.

Bridging MAB and `AutoData`. There is a natural connection between MAB and **`AutoData`**. More concretely, we have g clusters, where each cluster can be considered as an arm. During each iteration, we need to select a cluster (*i.e.,* pull an arm) and then sample a mini-batch from this selected cluster. We use stratified sampling [39] to sample from the mini-batch. It further divides the cluster into several subgroups before sampling based on the occurrence likelihood of data points, and simple random sampling can be applied within each stratum. Note that other sampling methods like uniform random sampling, systematic sampling and cluster sampling can also be used [39]. Then, a *reward* or a *penalty* needs to be calculated based on the selected mini-batch, depending on whether it increases or decreases the model performance. The method needs to balance the choices between selecting the clusters that currently achieve high rewards (*i.e.,* exploitation) and the clusters that are rarely selected (*i.e.,* exploration).

4.2 Solving `AutoData` with MAB

In order to apply MAB to our studied problem, we need to decide how to quantify the reward or penalty, and what is the strategy to balance exploration-exploitation. Let's start by defining the distance between clusters, which are used to compute reward/penalty scores.

Distance between clusters. Recall that each cluster corresponds to a Gaussian distribution (defined in Section 3.2), we use **Wasserstein distance** to compute the distance between any other two clusters C_i and C_j, which is defined as:

$$d(C_i, C_j) = W(p_i\,,\ p_j) = \left\|\mu_i - \mu_j\right\|_2^2 + \mathbf{tr}(\Sigma_i + \Sigma_j - 2(\Sigma_j^{\frac{1}{2}} \Sigma_i \Sigma_j^{\frac{1}{2}})^{\frac{1}{2}}) \quad (1)$$

where $p_i = (\mu_i, \Sigma_i)$ and $p_j = (\mu_j, \Sigma_j)$ are the Gaussian Distributions (*i.e.,* means and covariance matrices) of C_i and C_j, respectively; and **`tr`** is used to compute the *trace* (*i.e.,* the sum of the elements on the main diagonal) of the matrix. Wasserstein is a symmetric distance function, *i.e.,* $d(C_i, C_j) = d(C_j, C_i)$. Other symmetric distance functions can also be used. The distance $d(C_i, C_j)$ is normalized to $[0, 1]$, where 0 means C_i and C_j follow the same distribution, and 1 means that they are far away from each other.

$\mathcal{N}(C_i)$ denotes the **neighbors** of C_i, relative to a threshold τ as all clusters in C whose distances to C_i are less than τ, *i.e.,* $\forall C \in \mathcal{N}(C_i), d(C_i, C) < \tau$. Naturally, $C_i \in \mathcal{N}(C_i)$.

Reward/penalty score at one iteration. Intuitively, a **score** should be given to a cluster if it is selected, and a sampled mini-batch from it can change the model performance. More precisely, when it increases the performance, we call it **reward**, and when it decreases the performance, we call it **penalty**.

To this purpose, we first measure the performance difference between two models if a mini-batch B (with size b) is added to the currently train dataset $T_{\texttt{train}}$, $M(T_{\texttt{train}})$ and $M(T_{\texttt{train}} \cup B)$, both being evaluated using the validation dataset $T_{\texttt{val}}$: $\Delta = M(T_{\texttt{train}} \cup B, T_{\texttt{val}}) - M(T_{\texttt{train}}, T_{\texttt{val}})$. Let cluster C_i be the cluster where B is sampled from. If Δ is positive (resp. negative), we need to award (resp. penalize) C_i. Moreover, we empirically found that it is also beneficial to reward or penalize the neighbors $\mathcal{N}(C_i)$ of C_i. Concretely, if C_i is selected, we compute **score** r_j for C_j as:

- $r_j = \Delta \times (1 - d(C_i, C_j)/\tau)$, if $C_j \in \mathcal{N}(C_i)$
- $r_j = 0$, if $C_j \notin \mathcal{N}(C_i)$

That is, if $\Delta > 0$, all clusters in $\mathcal{N}(C_i)$ will be rewarded; otherwise, if $\Delta < 0$, all clusters in $\mathcal{N}(C_i)$ will be penalized. There is no reward/penalty for any cluster that is far from C_i (*i.e.,* $C_j \notin \mathcal{N}(C_i)$).

Example 7. [*Score.*] *Consider clusters in Fig. 4(b). Assume that* $\mathcal{N}(C_1) = \{C_1, C_2\}$, $\mathcal{N}(C_2) = \{C_1, C_2, C_5\}$, $\mathcal{N}(C_3) = \{C_3, C_4, C_5\}$, $\mathcal{N}(C_4) = \{C_3, C_4\}$, *and* $\mathcal{N}(C_5) = \{C_2, C_3, C_5\}$.

Let B be a mini-batch selected from C_1 and $\Delta = M(T_{\texttt{train}} \cup B) - M(T_{\texttt{train}}) = 0.1$. Since $\Delta > 0$, we need to reward all clusters that are neighbors of C_1, i.e., *$\{C_1, C_2\}$. Let $d(C_2, C_1) = 0.1$, $\tau = 0.5$, then,*

- *$r_1 = \Delta \times (1 - 0) = 0.1 \times 1 = 0.1$, and*
- *$r_2 = \Delta \times (1 - 0.1/0.5) = 0.1 \times (1 - 0.2) = 0.08$, and*
- *r_3, r_4, r_5 are 0, because $\{C_3, C_4, C_5\}$ are not in $\mathcal{N}(C_1)$.*

In the following, r_j^k denotes the score of C_j at iteration k.

Algorithm 1: A UCB-based MAB Algorithm

```
   Input: clusters $C$, $T_{\texttt{train}}$, $T_{\texttt{val}}$, $M$, $\tau$, total iterations $\mathbf{k}$
   Output: updated $T_{\texttt{train}}$.
 1 for each $C_i \in C$ do
 2     Compute $\mathcal{N}(C_i)$ based on $\tau$;
 3     $R_i^0 = 0, n_i^0 = 0, U_i^0 = 0, T_{\texttt{train}}^0 = T_{\texttt{train}}$;
 4 for $k$ from 1 to $\mathbf{k}$ do
 5     $C_i$ = Select the cluster with the largest UCB value;
 6     $B_i$ = A mini-batch sampled from $C_i$ ;
 7     $\Delta = M(T_{\texttt{train}}^{k-1} \cup B_i, T_{\texttt{val}}) - M(T_{\texttt{train}}^{k-1}, T_{\texttt{val}})$;
 8     if $\Delta > 0$ then
 9         $T_{\texttt{train}}^k = T_{\texttt{train}}^{k-1} \cup B_i$;
10     for each $C_j \in \mathcal{N}(C_i)$ do
11         $r_j^k = \Delta \cdot (1 - \frac{d(C_i,C_j)}{\tau})$;
12         Update $R_j^k$ and $U_j^k$ ;
13 return $T_{\texttt{train}}^{\mathbf{k}}$;
```

Aggregated reward/penalty scores for multiple iterations. Because the process for selective data acquisition is iterative with multiple iterations, we need to define the aggregated reward/penalty scores correspondingly.

Let R_i^k be the **aggregated score** of cluster C_i from iteration 1 to k. Let $R_i^k = \frac{1}{n_i^k} \sum_{j=1}^k r_i^j$, where r_i^j is score of cluster C_i at iteration j, and n_i^k is the total number of times that cluster C_i was assigned non-zeros scores from iterations 1 to k.

Next we use an example to illustrate the computation of n_i^k.

Example 8. [*Computing n_i^k.*] *Consider three iterations: a mini-batch from cluster C_1 is selected at iteration 1; a mini-batch from C_2 is selected at iteration 2; a mini-batch from C_4 is selected at iteration 3. All n_i^0 is initialized 0 at iteration 0.*

[*Iteration 1.*] *Because C_1 is selected and $\mathcal{N}(C_1) = \{C_1, C_2\}$ (see Example 7), $n_1^1 = 1$ and $n_2^1 = 1$.*

[*Iteration 2.*] *Because C_2 is selected and $\mathcal{N}(C_2) = \{C_1, C_2, C_5\}$, the numbers for C_1, C_2, and C_5 will be incremented by 1,* i.e., *$n_1^2 = 1+1 = 2$, $n_2^2 = 2$, and $n_5^2 = 1$.*

[*Iteration 3.*] *Because C_4 is selected and $\mathcal{N}(C_4) = \{C_3, C_4\}$, the numbers for C_3 and C_4 will be incremented by 1, resulting in $n_3^3 = 1$ and $n_4^3 = 1$. While the numbers for other clusters will not be affects,* i.e., *$n_1^3 = 2$, $n_2^3 = 2$, and $n_5^3 = 1$.*

A crude exploitation-only method is to select the cluster C_i with the largest R_i^k at iteration k. Clearly, this can easily result in local optimum, because it misses the exploration attempt for rarely selected but possibly helpful clusters.

Upper Confidence Bounds (UCB) based solution. A popular strategy to combine "exploration" with "exploitation" is to use UCB-based solutions [2]. UCB estimates the score if a cluster is selected and we acquire data points from the cluster. At a high level, the score is computed by the summation of an exploration and an exploitation score. The less frequently a cluster is picked, the higher the exploration score. The higher reward the cluster has obtained, the higher the exploitation score. Hence, a combination of the two scores used by UCB-based method can well handle the exploration-exploitation trade-off. Specifically, for each C_i, at iteration k:

$$UCB(C_i, k) = U_i^k = R_i^k + \alpha\sqrt{2 \ln n^k / (n_i^k + 1)} \tag{2}$$

where α is a pre-defined parameter and n^k is the sum of n_i^k for all clusters at iteration k (*i.e.,* $n^k = \sum_{i \in 1,g} n_i^k$ and g is the total number of clusters). In Eq. 2, the former part (R_i^k) refers to the *exploitation*, which means that we will focus more on the cluster in which some data points have resulted in much performance improvement. The latter part ($\sqrt{2 \ln n^k / (n_i^k + 1)}$) refers to the *exploration*, *i.e.,* focusing more on the clusters that are rarely picked.

Consider Example 8: at iteration 1, $n^1 = \sum_{i \in [1,5]} n_i^1 = 1 + 1 + 0 + 0 + 0 = 2$; at iteration 2, $n^2 = \sum_{i \in [1,5]} n_i^2 = 2 + 2 + 0 + 0 + 1 = 5$; and at iteration 3, $n^3 = \sum_{i \in [1,5]} n_i^3 = 2 + 2 + 1 + 1 + 1 = 7$.

At iteration k, we select cluster C_i with the maximum UCB value, *i.e.,* $U_i^k \geq U_j^k$ for any $j \in [1, g]$.

Algorithm. Algorithm 1 shows the overall process of the UCB-based solution. It first initializes the algorithm (lines 1-3). At each iteration (lines 4–12), it selects the cluster with the largest UCB value (line 5), and samples a mini-batch from the selected cluster (line 6). It then evaluates the selected mini-batch by re-training the model (line 7). It will add the mini-batch to the train dataset only if it can cause a reward (lines 8–9). It then updates the reward or penalty for neighbors of the selected cluster (line 11), as well as the aggregated reward/penalty scores and UCB values (line 12). Finally, it returns the updated train dataset (line 13).

Let's better illustrate Algorithm 1 using an example.

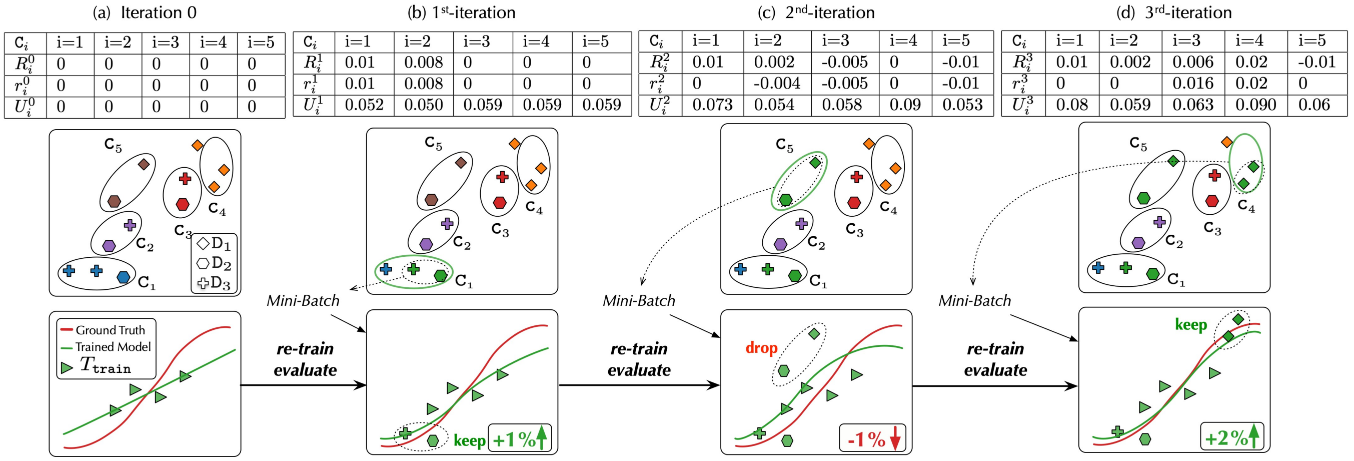
Example 9. [*UCB for* **`AutoData`**.] *Fig. 5 is 3 iterations.*

[*Initialization: Fig. 5(a).*] *There are five clusters. It initializes R_i^0, r_i^0 and U_i^0 as 0 for all clusters. Let $\tau = 0.5$, the threshold for deciding whether two clusters are neighbors, and $\alpha = 0.05$, the parameter in Equation 2 for computing UCB values. Each mini-batch contains two data points.*

[*Iteration 1: Fig. 5(b).*] *At this point, all clusters have the same UCB values, so we randomly select one cluster, say C_1. After sampling a mini-batch from C_1 and re-training the model, we observe that the model performance improves by 1%. Hence, $\Delta = 0.01$ and we will add this mini-batch to $T_{\texttt{train}}^0$ as $T_{\texttt{train}}^1$.*

Suppose $\mathcal{N}(C_1) = \{C_1, C_2\}$ and $d(C_2, C_1) = 0.1$, then:

- *$r_1^1 = \Delta = 0.01$, $r_1^2 = \Delta \times (1 - 0.1/0.5) = 0.008$;*
- *$R_1^1 = R_0^1 + r_1^1 = 0.01$, $R_1^2 = R_0^2 + r_1^2 = 0.008$;*
- *$n_1^1 = n_2^1 = 1$, $n^1 = n_1^1 + n_2^1 + 1 = 2$;*
- *$U_1^1 = R_1^1 + \alpha\sqrt{(2 \times \ln n^1)/(n_1^1 + 1)} = 0.052$, $U_2^1 = R_2^1 + \alpha\sqrt{(2 \times \ln n^1)/(n_2^1 + 1)} = 0.050$, $U_3^1, U_4^1, U_5^1 = 0.059$.*



(a) Iteration 0

C_i	i=1	i=2	i=3	i=4	i=5
R_i^0	0	0	0	0	0
r_i^0	0	0	0	0	0
U_i^0	0	0	0	0	0

(b) 1st-iteration

C_i	i=1	i=2	i=3	i=4	i=5
R_i^1	0.01	0.008	0	0	0
r_i^1	0.01	0.008	0	0	0
U_i^1	0.052	0.050	0.059	0.059	0.059

(c) 2nd-iteration

C_i	i=1	i=2	i=3	i=4	i=5
R_i^2	0.01	0.002	-0.005	0	-0.01
r_i^2	0	-0.004	-0.005	0	-0.01
U_i^2	0.073	0.054	0.058	0.09	0.053

(d) 3rd-iteration

C_i	i=1	i=2	i=3	i=4	i=5
R_i^3	0.01	0.002	0.006	0.02	-0.01
r_i^3	0	0	0.016	0.02	0
U_i^3	0.08	0.059	0.063	0.090	0.06

Figure 5: A running example of UCB-based MAB algorithm.

[*Iteration 2.*] *In this iteration, it randomly selects one from those with the largest UCB values (*i.e., $\{C_3, C_4, C_5\}$ *because their UCB values are 0.059 after the 1st iteration), say* C_5 *is picked. After re-training and evaluation, we observe a model performance decrease of 1%. We will not add this mini-batch to the train data,* i.e., $T^2_{\texttt{train}} = T^1_{\texttt{train}}$.

Suppose $\mathcal{N}(C_5) = \{C_2, C_3, C_5\}$, $d(C_5, C_2) = 0.3$ *and* $d(C_5, C_3) = 0.25$. *All values are updated similar to above.*

[*Iteration 3.*] *In this iteration, it will pick* C_4*, the one with the largest UCB value after the 2nd iteration. After re-training and evaluation, we observe a model performance increase of 2%. It will add this mini-batch as* $T^3_{\texttt{train}}$ *and update all values.*

Example 9 shows how the UCB-based MAB solution works for **AutoData**. Moreover, it illustrates the exploitation-exploration trade-off. For example, even if at iteration 1, it observes a performance increase using a mini-batch sampled from C_1, it still *explores* other clusters at the next iteration.

Remark. When there are several clusters having the same UCB values, what is described above is to randomly pick one. Another intuitive option is to pick the one whose distance is the closest to $T_{\texttt{train}}$, which is a more conservative choice.

5 DQN-BASED RL FOR AUTODATA

The above MAB-based solution leverages a heuristic search method to retrieve data based on the human-provided criteria with fixed parameters, which is not generalizable enough and hard to reach the optimal solution. DQN-based RL is a more powerful and robust framework in such a dynamic setting.

5.1 Bridging RL and AutoData

Reinforcement learning (RL) is an ML paradigm where an **agent** learns from the feedback by trial-and-error interactions with the **environment**. A typical example is the AlphaGo [48] where the **agent** considers the current **state** (*i.e.,* the board situation in the Go game), and then predicts the next optimal action (*i.e.,* where the next piece goes on the board) because the action can bring about the highest **reward**, *i.e.,* the largest probability to win the game.

Bridging RL and AutoData. As discussed in Section 3.2, selective data acquisition needs to (1) update the search criteria, and (2) get feedback from evaluating the new mini-batch. Hence, there is a natural connection between RL and **AutoData** for the problem of selective data acquisition. The train data $T_{\texttt{train}}$ at the current iteration is considered as the **state**. The **action** represents whether to select and keep/drop a mini-batch of new train data, decided by an **agent**. The **environment** will re-train the model so as to evaluate the benefit of keeping/dropping a mini-batch, based on which a **reward** will be calculated.

5.2 Solving AutoData with RL

State s^k represents the situation of the train data $T^k_{\texttt{train}}$ at the k-th training iteration in each episode. Different from the MAB solution, we take the data distribution of $T^k_{\texttt{train}}$ into account and encode it as the state, for more fine-grained optimization.

Recall that g is the number of clusters, *i.e.,* $g = |C|$. We can map each data point $O \in T^k_{\texttt{train}}$ to a g-dimensional vector. Moreover, let p^i_O be the probability that O will be in C_i, and thus each data point corresponds to a probability vector with length g. In this way, the s^k can be represented as a $|T^k_{\texttt{train}}| \times g$ matrix. It has two limitations. (1) At every iteration the size of $T^k_{\texttt{train}}$ changes, so does the dimension of s^k, which is difficult for a neural network to train. (2) The matrix can be large, so the model training is hard to converge. To address the limitations, we need a representation of the state that has a fixed and relatively small size.

(1) [Initialization.] Initialize G_i as an empty set $\emptyset$ ($i \in [1, g]$), which denotes the set of train data in $T^k_{\texttt{train}}$ that should belong to C_i.

(2) [Partition train data.] For each data point $O \in T^k_{\texttt{train}}$, we compute the most likely cluster that it should belong to (*e.g.,* cluster C_j), and then add O to G_j as $G_j = G_j \cup \{O\}$. After doing this for all data points in $T^k_{\texttt{train}}$, $G_1, \ldots, G_g$ are partitions of $T^k_{\texttt{train}}$.

(3) [Representation of state.] We compute the mean (μ_{G_i}) and covariance (Σ_{G_i}) of each G_i, and use the triple ($\mu_{G_i}, \Sigma_{G_i}, |G_i|$) to represent

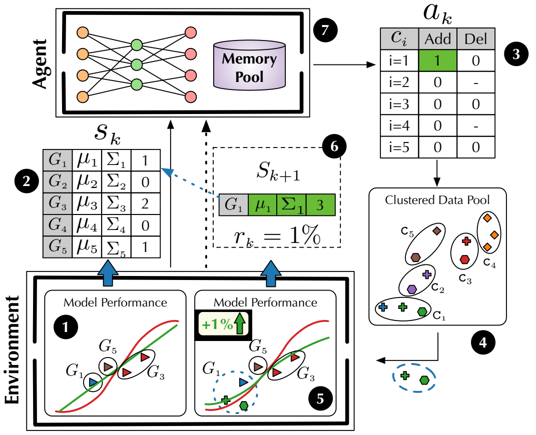


Figure 6: A running example of DQN-based RL solution.

G_i. Suppose that each data point has m attributes, and thus the dimension of Σ_{G_i} (μ_{G_i}) is $m^2(m)$. Hence, the representation of state s^k has size $O(m^2) \times g$. Dimensionality reduction techniques [52] can be utilized if the matrix is too large.

Example 10. [*State.*] *As shown in Fig. 6, to represent the state, we map data objects in* $T^k_{\texttt{train}}$ *(with four objects) to the data pool* $\mathcal{P}$. *And we can see that there are one data point in* G_1, *one in* G_5, *and two in* G_3 *(❶). Therefore, we compute the means and covariance matrices for these three sets of data. For* G_2 *and* G_4, *since there is no data here, we fill the vectors and covariance matrices with zeroes. Then we obtain the state encoding* s^k *of current* $T^k_{\texttt{train}}$ *(❷).*

Action a^k. For each cluster, we have two choices, either to sample a new mini-batch from it, or to delete a mini-batch of data points selected from this cluster in previous iterations. There are g clusters, so there are totally $2 \times g$ possible actions. Note that, if all data points of a cluster have been added, we will mask out the action acquiring from the cluster.

Example 11. [*Action.*] *Given the state* s^k, *the agent uses the Q-network to select an action to be executed next. At a high level, we have two options for each* C_i, i.e., *add a mini-batch of data from* C_i *to* $T_{\texttt{train}}$ *or delete a mini-batch of data points that are previously selected from* C_i. *Thus, the action space is* $2 \times g$, i.e., *10 in the example. But note that not every action is valid,* e.g., *there is no data belonging to* C_2 *and* C_4, *so the actions "delete* C_2*" and "delete* C_4*" are masked. Suppose that the agent chooses the action "add* C_1*" and sends this action to the environment (❸).*

Reward r^k is calculated through the performance difference of the ML model between the $(k-1)$-th iteration and k-th iteration, *i.e.,* $M(T^k_{\texttt{train}}, T_{\texttt{val}}) - M(T^{k-1}_{\texttt{train}}, T_{\texttt{val}})$.

Example 12. [*Reward.*] *Continue with Example 11, a mini-batch is sampled from* C_1 *(❹) and added to* $T^k_{\texttt{train}}$. *Then the model is retrained and obtain* 1% *improvement (❺), which is regarded as the reward* r^k.

State update. After an action is applied, $T_{\texttt{train}}$ will change, and thus the state will be updated. Note that we acquire data points from a single cluster (*e.g.,* C_i) or delete from G_i, so in each iteration, we just need to update a small part of state representation, *i.e.,* the triple of G_i.

Example 13. [*State update.*] *Following Example 12, the state will be updated after data points from* C_1 *are acquired,* i.e., *changing the mean, covariance matrix and the number of data points of* G_1. *Then new state* s^{k+1} *and reward* r^k *(❻, for updating the Q-network) will be sent to the agent for the* $(i+1)$*-th iteration (❼).*

Environment encodes the train dataset as state by capturing the data distribution. Besides, given an action, it adds (resp. deletes) a mini-batch of data to (resp. from) the train dataset and re-trains the model. Furthermore, environment evaluates the reward and sends it as a reinforcement signal to agent.

Agent takes into account the situation of the current train dataset (*i.e.,* current state), and chooses an action according to the reward estimation of the expected long-term reward. After receiving the reward of each iteration, the agent uses the memory pool to manage these experiences and conducts the learning process.

Vanilla Q-learning. The typical RL framework is Q-learning [50], which is based on Q-function. Q-function can be viewed as a *state-action* value function of a **policy** π, namely Q-value and denoted by $Q^\pi(s, a)$. It measures the expected long-term rewards obtained from state s by taking action a first and following policy π thereafter. We aim at selecting the optimal action that leads to the largest long-term reward by Bellman optimality equation, *i.e.,* $Q^*(s, a) = \mathbb{E}[r + \gamma \max'_a Q^*(s', a')]$, where r is the immediate reward, γ is the discount factor and $Q^*(s', a')$ denotes the following optimal Q-value from the next state s'. A vanilla Q-learning approach uses a Q-table [20] to store the Q-value of each *state-action* pair.

Algorithm 2: A DQN-based RL Training Algorithm

```
Input: $C$, $T_{\texttt{train}}$, $T_{\texttt{val}}$, $M$, # of training iterations t/ episode
Output: a trained RL model
for each episode during training do
    for $k$ from 1 to t do
        state $s^k$ = is computed using ($T_{\texttt{train}}$, $C$) ;
        action $a^k$ = $\max_a Q(s^k, a; \theta)$;
                // delete/add a mini-batch
        Apply $a^k$ to update $T^k_{\texttt{train}}$ ;
        Re-train and compute a reward $r^k$, which will be used to
          update $Q$-network;
return the RL model
```

Deep Q-Network (DQN). Using a Q-table [20] is impractical for our problem, because our states (*i.e.,* training data distribution in each iteration) are impossible to enumerate. Therefore, we adopt DQN [30, 31] that trains a function approximator, such as a neural network with parameters θ, to estimate the Q-values, i.e., $Q(s, a; \theta) \approx Q^*(s, a)$. It can automatically attempt to learn the optimal combinatorial space of different mini-batches from multiple clusters, leading to a data acquisition strategy with high performance improvement.

Algorithm. Algorithm 2 illustrates how to train a selective data acquisition model using RL, which consists of multiple training episodes. In each episode we have **t** iterations, where different actions are tried and the feedback is used to train the model. Specifically, in each iteration, the algorithm first computes the state representation s^k of the train data (line 3). Then, based on the trained Q-network, it selects the optimal action a^k with the largest long-term estimated reward (line 4). The action is to either add a mini-batch to $T^k_{\mathtt{train}}$ or delete a mini-batch from $T^k_{\mathtt{train}}$. Afterwards, it applies the action (line 6), re-trains the ML model and computes a reward that will be used to update the Q-network (line 7).

Discussion about DQN training. In our framework, we use an off-policy [32] strategy to learn the target policy $a = \max_a Q(s, a; \theta)$ while a different behavior policy (*i.e.,* $\epsilon-$greedy) runs for collecting data in the environment. It is called *off-policy* when behavior policy $\neq$ target policy. It selects the greedy action with probability $1 - \epsilon$ and a random action with probability ϵ to ensure good coverage of the state-action space. Since DQN is off-policy, we use experience memory replay for Q-Network training. In order to break the temporal correlation between actions and avoid forgetting the rare but valuable experiences, we use the memory replay technique [30] to store the experiences observed by the agent, and allow the agent to reuse these experiences later.

DQN inference. During inference, we also have a total acquisition number of iterations **k** as input. Then at each iteration, we repeat the lines 3-7 in algorithm 2 except updating the network in line 7.

Note that, once the model is trained, it can be used for selecting data points for the same supervised ML task but the dataset is not necessarily the same. Consider Fig. 3 for example, the model learned by using the data in Fig. 2 (*i.e.,* city Kolkata) can be used for acquiring data for house price prediction of other cities (*e.g.,* Bangalore, Mumbai, and Delhi).

Take-away of MAB and DQN based methods. The MAB-based solution is easy to implement and efficient. The DQN-based solution can achieve higher performance improvement but spends long time on training. Therefore, by default, we choose DQN-based solution because it can achieve higher performance improvement and effectiveness is typically the main concern. However, if efficiency is the main concern, the MAB-based solution is a good alternative.

6 EXPERIMENT

Datasets. We used 5 real-world datasets (`HR`, `House`, `Image-6`, `Image-10`, `Credit`) to evaluate our proposed framework. Table 1 shows the statistics. `HR`, `House` and `Credit` are tabular data. `Image-6` and `Image-10` contain images, as shown in the "Type" column. The train/validation/test split of the three datasets is also shown in the column "$|T_{\mathtt{train}}|/|T_{\mathtt{val}}|/|T_{\mathtt{test}}|$".

For each dataset, we conducted an end-to-end experiment. For tabular data (`HR` and `House`), we used the NYU Auctus API [38] to search using $T_{\mathtt{train}}$. Specifically, we used the API `Datamart.search_with_data(supplied_data)` to search unionable datasets, where `supplied_data` is replaced with $T_{\mathtt{train}}$. Then we used `DatamartSearchResult.download` to download these datasets that have high overlapping attributes with $T_{\mathtt{train}}$, where the schema alignment has been done. As discussed in Section 3, if the searched datasets do not have an aligned attribute with $T_{\mathtt{train}}$, we use NULL values for this attribute. In Table 1, "# Sel. Src" denotes the number of retrieved tables that could be well integrated with the train data. Note that other popular dataset search toolkits (*e.g.,* Aurum [16], Google dataset search [18]) can also be incorporated into our framework. They can find relevant datasets given a keyword-based but not dataset-based query, thus is not directly suitable for unionable dataset search. In practice, some wrappers *w.r.t.* some schema alignment techniques [35] need to be developed in order to incorporate these toolkits. We used all the data points in selected sources as the pool. The total number of points in the pool is given in the "$|\mathcal{P}|$" column. Details are as below.

Table 1: Statistics of datasets.

Dataset	Type	$\lvert T_{\mathtt{train}}\rvert/\lvert T_{\mathtt{val}}\rvert/\lvert T_{\mathtt{test}}\rvert$	# Sel. Src	$\lvert\mathcal{P}\rvert$
`HR`	Tabular	1200/400/400	5	21287
`House`	Tabular	1620/425/425	5	32963
`Image-6`	Image	2300/700/700	3	62000
`Image-10`	Image	5400/2100/2100	5	87400
`Credit`	Tabular	11200/4000/4000	4	126300

(i) `HR` is a dataset with **a classification task** of *"predicting whether an employee would change the job"*. # of train, validation and test points are 1200, 400 and 400 from Finance Dept. respectively. Five other departments (*i.e.,* Sales Dept., International Dept., Purchasing Dept., Marketing Dept. and Technology Dept.) can supplement the train data because they have similar schema with the train data, and thereby retrieved by the NYU Auctus APIs. We have 12 attributes after alignment and 21287 data points in the pool.

(ii) `House` is a dataset of "*predicting house price*" (**a regression task**) in India, and # of train/validation/test points was 1620/425/425 from Mumbai. We found five tables from different cities (*i.e.,* Bangalore, Chennai, Delhi, Kolkata and Hyderabad) that could be integrated with train/test points. We have 39 attributes and 32963 data points.

(iii) `Image-6` is to **classify a collection of images**, and totally, there are 6 categories {Dog, Tiger, Cat, Lion, Rabbit, Pig}. Unlike tabular data, image data does not need schema alignment. We crawled 2300/700/700 train/valiadation/test points from `Google.com`. For external sources, we used images from many sources. We used 62000 pictures from Imagenet [21], Caltech-256, Bing [4] as the pool. For public datasets (*e.g.,* Imagenet and Caltech), we only used the images that fall into the six categories. For web-based APIs (*e.g.,* Bing image search [4]), we only retrieved the images using the above categories within the limit of free API calls.

(iv) `Image-10` is a dataset for image classification task, which has 10 categories, *i.e.,* {binoculars, cake, calculator, eyeglasses, knife, mushroom, octopus, rainbow, snake, spoon}. The sizes of train/validation/test points are 5400, 2100 and 2100, respectively, which are crawled from `Google.com`. There are 87400 images in $\mathcal{P}$, which are from Baidu, Imagenet [21], Bing [4] and Caltech-256.

(v) `Credit` is a classification dataset, whose task is to predict "whether the loan will be deferred based on a person's economic situation". The sizes of train, validation and test dataset are 11200, 4000 and 4000, respectively. We have 11 attributes after alignment and 126,300 data points in the pool.

Evaluation Metrics. We used two different metrics, *Area Under Curve (*AUC*)* for classification (HR, Image-6, Image-10, Credit) tasks, and *Mean Squared Error (*MSE*)* for regression (House) task.

(I) AUC indicates a model's capability to distinguish between classes, where the AUC value is the area under the Receiver Operator Characteristic (ROC) curve.

(II) MSE is used to evaluate the performance of the regression model. It takes the distances (these distances are the "errors") from the normalized points of ground truth (y) to the regression line ($y_{predict}$) and squaring them, *i.e.,* $MSE = \frac{1}{n}\sum_{i=1}^{n}(y - y_{predict})^2$. The lower the MSE, the better the prediction.

Solutions. We compared **AutoData** with a variety of typical algorithms. Almost all methods (except ALL) select data iteratively, and in each iteration, the same size of mini-batch of data points is acquired. Note that for our methods, we could either add or remove one mini-batch in one iteration.

(1) ALL simply adds all data points from $\mathcal{P}$ to $T_{\mathtt{train}}$.

(2) Random randomly selects a mini-batch per iteration.

(3) Uncertainty-based active learning (U-AL) is an active learning based method, which selects data points with the largest uncertainty measured by the trained model iteratively.

(4) Error-based active learning (E-AL) is another active learning strategy, which considers how much error would incur if a data point is acquired and added for retraining. Since typical active learning methods cannot know label in advance, it only compute the expected error induced by each data point [44]. But in our scenario (data points in $\mathcal{P}$ had labels), we could exactly know whether current model predicted correctly or not, so we just added a mini-batch of points that were incorrectly predicted to $T_{\mathtt{train}}$ in each iteration.

(5) Outlier-based active learning (O-AL) considers the active learning scenario that the fetched data points (with high uncertainty) maybe outliers for the train data [24]. Therefore, it first removes the data points that have rather different distribution from *train* and then conducts the active learning.

(6) Coreset-based active learning [47] (CS-AL) iteratively selects a subset of unlabeled data points (*i.e.,* the coreset) to label. In our scenario, we select the coreset from $\mathcal{P}$ using their method without considering the label.

(7) Similarity search (SS) searches points of $\mathcal{P}$ that are the closest to $T_{\mathtt{train}}$, adds them to $T_{\mathtt{train}}$ and retrain iteratively. The distance between a data point o in $\mathcal{P}$ and $T_{\mathtt{train}}$ is measured by the average Euclidean distance between o and all points in $T_{\mathtt{train}}$.

(8) RL-based data valuation [54] (RL-DV) uses an RL model to learn a valuation score of each data point. The higher the score, the more benefit the data point has for the ML task, and thus the data point should be selected.

(9) Exploitation-only (EO) first fetches a mini-batch of data points from each cluster in the pool, and then keeps sampling from the cluster that has brought the most performance improvement, without the change of exploration.

(10) AutoData-MAB. This method is the algorithm using Upper Confidence Bounds based MAB solution (Section 4).

(11) AutoData-DQN is the DQN-based RL solution (Section 5). To train the RL model (Algorithm 2), we set a maximum number of episodes of training, say 600. Each episode consisted of a series of actions, the corresponding states and rewards. The training process of RL model consumed around 2.5 hours and 3.8 hours on HR and House, respectively. On Image-6 dataset, we used the pre-trained model (*i.e.,* Resnet50) for the image classification task, and the training process totally consumed about 6.3 hours.

Note that when comparing **AutoData**-DQN with other solutions in Section 6.1, the model has been trained and we compared the inference results (**DQN inference** in Section 5). For training process, we will provide more details in Section 6.2.

Hyper-parameter setting. We set τ in the **AutoData**-MAB as 0.5. For **AutoData**-DQN, we used 4 fully connected layers, and we set $\mathbf{t} = 20$. In addition, $\epsilon = 0.95$ and we use exponential decay policy to decrease ϵ. We used XGBoost and XGBoost Regression as downstream models for the two tabular dataset HR and House.

6.1 Efficacy of Selective Data Acquisition

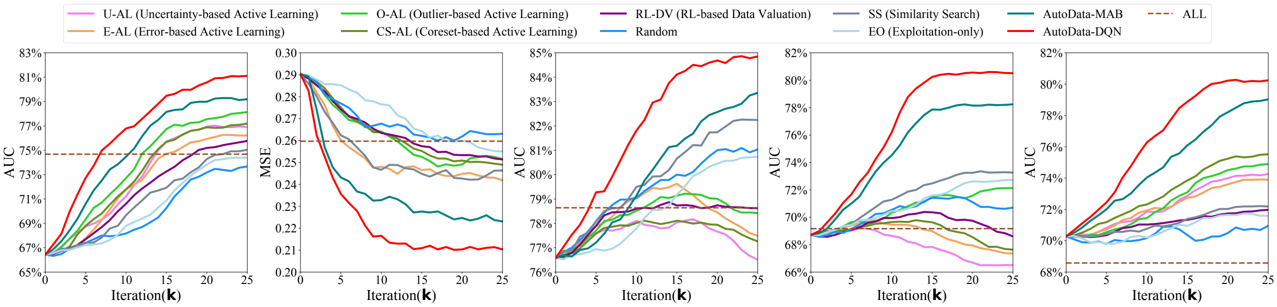
Can selective data acquisition from external sources help supervised ML? We quantitatively evaluated the above ten solutions and the result is shown in Fig. 7. The x-axis denotes the number of iterations for selective data acquisition. The y-axis denotes the evaluation metric: for Fig. 7 (a) and (c), AUC is used (the higher the better); for Fig. 7 (b), MSE is used (the lower the better).

Effectiveness on HR (AUC). The result is shown in Fig. 7 (a), which tells us the followings.

On this dataset, all methods can improve the model performance through data acquisition, *i.e.,* along with the increasing number of iterations, the AUC values of all methods increase. At the beginning, the AUC begins to increase rapidly, and then gradually slows down.

Among the baselines, Random, SS and EO do not perform well (73.8%, 75.2% and 74.4% respectively after 25 iterations). The reason is that Random is simple that does not consider any factors influencing the ML task. For SS, it only retrieves similar data with $T_{\mathtt{train}}$ without considering the performance, and loses the opportunities to explore the pool for approaching the true distribution of the ML task. For EO, although a certain cluster performs the best under the state of initial train set, it is far from enough to use the cluster to represent the data distribution of underlying ML task, which is likely to be more complicated than the distribution of a single cluster. Besides, ALL (74.6%) does not perform well either. This verifies that simply acquiring all available data points is likely to involve many "outliers" that are not beneficial to the performance, and thus selective data acquisition is necessary.

Active learning based methods outperform SS because they consider the uncertainty or prediction errors of data points. For example, U-AL (E-AL) achieves an AUC of 76.8% (76.2%). E-AL is worse than U-AL because the pool has many heterogeneous points that are not beneficial for the model, and the model can not predict them accurately. Therefore, these points are acquired, which results in low performance. CS-AL only achieves 77.4% AUC because it aims to select the coreset that is the representation of the training data, rather than improving the model performance. Hence, the coreset selection does not consider how the model performs on the validation or test set. Naturally, it cannot perform well when there is not

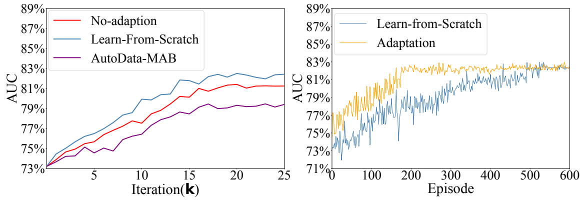


(a) HR (b) House (c) Image-6 (d) Image-10 (e) Credit

Figure 7: Effectiveness of selective data acquisition.

Table 2: Efficiency (seconds).

Method	HR	House	Image-6	Image-10	Credit
Random	4.27	5.28	10.69	19.87	23.27
U-AL	11.49	-	34.77	54.73	61.76
E-AL	10.61	23.28	32.52	49.87	58.81
O-AL	13.29	26.52	40.28	75.28	93.73
CS-AL	12.79	25.37	37.14	58.33	73.24
RL-DV	45.38	93.62	114.56	136.85	133.19
SS	36.24	70.23	106.25	113.64	127.85
AutoData-MAB	4.76	10.64	15.35	21.92	26.45
AutoData-DQN	5.72	12.78	17.47	26.67	29.87



(a) No-adaption (b) Adaptation

Figure 8: AutoData-DQN: No-adaptation vs. Adaptation (HR)

enough training data and the data distributions of training and test data are different. It also shows that O-AL (78.2%) performs better than U-AL because O-AL first removes the outliers (distribute much differently from points in $T_{\mathtt{train}}$) in the pool.

Reinforcement learning-based data valuation (RL-DV) achieves only 75.9% AUC because it mainly focuses on retrieving data points from a homogeneous dataset, but selecting data points from our heterogeneous pool is likely to introduce many noise and leads to instable training. Another possible reason is that in each training iteration, RL-DV just randomly samples a batch of data points in the pool, and thus the points between iterations are irrelevant. But during training, we select data points considering the historical experience, including incremental data acquisition and deletion.

AutoData-MAB and **AutoData**-DQN outperform baselines, with AUC 79.3% and 81.2% respectively. The reasons are two-fold. First, our methods directly consider the performance improvement or degradation to guide the acquisition. Second, we carefully model the relationships among data points in the pool and use an exploration-exploitation strategy, based on which we acquire data iteratively. The DQN-based method **AutoData**-DQN is better than **AutoData**-MAB because (a) it considers the intrinsic characteristics of train data points by encoding them as state, which can be regraded as a key feature, and (b) it uses the more powerful DQN model to update the search policy, which is more flexible.

Effectiveness on House (MSE). The result is shown in Fig. 7 (b). Different from the AUC value in Fig. 7 (a) that is the higher the better, for MSE in Fig. 7 (b), it is the lower the better. Hence, it shows that these methods had similar trends that different methods can improve the model performance along with the increasing number of iterations. Note that we did not plot U-AL because it is not practical to measure the uncertainty for the regression task. For example, **AutoData**-DQN achieved a MSE of 0.209, which outperformed E-AL (0.241), O-AL (0.250), SS (0.246), EO (0.255), CS-AL (0.248), RL-DV (0.240) and **AutoData**-MAB (0.226) due to its high generalization and learning ability. E-AL outperforms O-AL because some beneficial points were wrongly removed in O-AL. **AutoData**-MAB outperforms other baselines because it selectively acquires data based on the performance feedback.

Effectiveness on Image-6 (AUC). The result is shown in Fig. 7 (c). It shows **AutoData**-DQN still performed the best, with 85.1% AUC. The difference is active learning methods perform worse than SS. The reason is that for the image dataset, images in the pool is much more heterogeneous than the above two tabular datasets, so active learning methods are likely to acquire many unhelpful objects. Even worse, if they acquire more and more points, the performance would degrade because they contain more noisy train points.

Effectiveness on Image-10 (AUC). As shown in Fig 7, we can see that **AutoData**-MAB and **AutoData**-DQN significantly outperforms other baselines, achieving AUC of 78.3% and 80.8%, respectively.

Effectiveness on Credit (AUC). As shown in Fig. 7, we find the **AutoData** (79.2% of MAB and 80.4% of DQN) significantly outperforms the active learning-based methods, RL-DV, SS and Random.

Efficiency. Table 2 shows the efficiency of different methods on 3 datasets for 15 iterations (*i.e.,* **k** = 15). We can see that Random is the fastest because it does not need complex computation. AL-based methods are much slower because they need to train the downstream model and iterate the points in the pool for prediction, so as to choose the acquired points. SS is the slowest since it needs many high dimensional vector computations, although one can use the index [22] to accelerate it. RL-DV is also slow because it needs to iterate through the points in the pool, compute the valuation

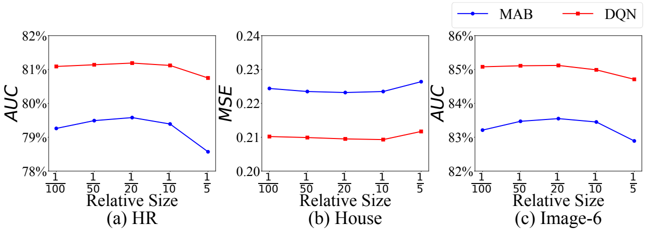


Figure 9: Hyper-parameter tuning for mini-batch sizes.

scores, and select the ones with high scores. Moreover, the training process of RL-DV is rather slow (more than 30 hours for Image-6 dataset) because it needs to train a number of sampled batches of data points, in order to obtain an accurate valuation score for each one. Since the pool is always large, it is always impractical to conduct this method for data acquisition in the wild. For our methods, **AutoData**-MAB performs fast because it does not need to iterate the pool, and also does not have complex computations.

Summary. **AutoData**-MAB and **AutoData**-DQN outperform the other competitors (Fig. 7), and both are efficient for selecting data points (Table 2). However, **AutoData**-DQN may need extra training time for a new supervised ML task (see Section 6 **Solutions**–10), which is automatic and thus practically feasible.

6.2 AutoData–DQN: No-adaption vs. Adaption

Can we apply a trained **AutoData**-DQN *model for a new dataset from the same domain?*

No-adaption. Consider the model that was trained, validated and tested for Finance Dept of the HR dataset as mentioned in Section 6.1. Next, we wanted to perform the same ML task, but on a different dataset, for which we used a HR Dept table with similar number of train/validation/test points as Finance Dept. Fig. 8(a) provides the result, where the x-axis denotes the number of iterations and the y-axis denotes the metric AUC. It shows that even by directly applying a trained model for a dataset from the same domain, it (*i.e.,* the "no-adaption" line) still outperforms **AutoData**-MAB. The "no-adaption" line performs slightly worse than we train the model for the new dataset, *i.e.,* the "learn-from-scratch" line, as expected.

Adaption. Instead of learning from scratch, next we examine whether we can adapt the model trained from a different dataset but in the same domain. Fig. 8(b) shows the result, where the x-axis denotes the number of iterations for the training process of **AutoData**-DQN and the y-axis denotes the metric AUC. It shows that if we adapt a pre-trained model (*i.e.,* the "adaptation" line), it will take much less episodes (*i.e.,* less than 200) to converge, compared with the "learn-from-scratch" approach (*i.e.,* above 500).

6.3 Sensitivity Analysis

Varying the size of mini-batch. Typically, we can regard the size of mini-batch as a hyper-parameter tuning problem. We evaluate the relative sizes η ($\frac{1}{100}$, $\frac{1}{50}$, $\frac{1}{20}$, $\frac{1}{10}$, $\frac{1}{5}$ respectively) of the mini-batches and training set. We use the training and validation set during the evaluation process. Experimental results are shown in Fig. 9, where the x-axis is η and y-axis denotes the metric (AUC and MSE). We can see that for **AutoData**-DQN, when η is small, it does not have much impact on performance, but more iterations are needed to acquire data from pool, which is time-consuming. When η is too large, the performance decreases because some noisy data points are incorporated. In addition, we find that **AutoData**-MAB is more sensitive to η than **AutoData**-DQN. The reason is that **AutoData**-DQN can remove more noisy points using the deletion action. Concretely, we can see that on the three datasets, $\eta = \frac{1}{20}$ is the best choice considering both the effectiveness and efficiency. For example, on HR, when $\eta = \frac{1}{20}$, **AutoData**-MAB and **AutoData**-DQN can achieve the AUC of 79.5% and 81.2%, respectively.

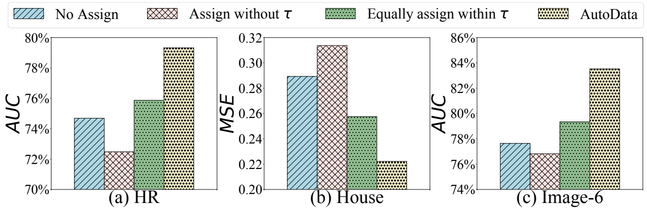


Figure 10: Different reward assignment methods.

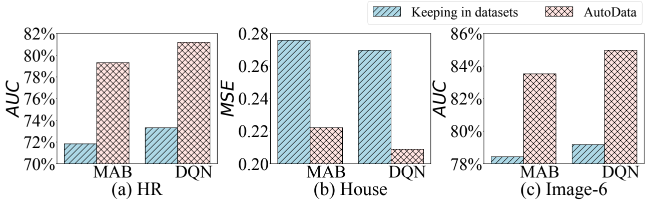


Figure 11: Different clustering strategies

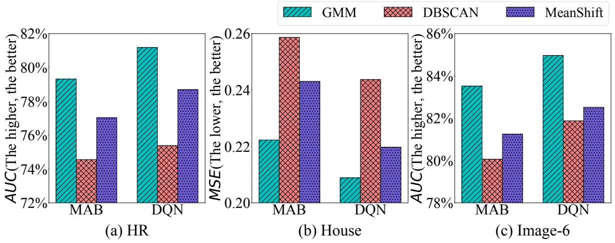


Figure 12: Performance improvement of clustering algorithms (k = 20).

Varying reward computation methods. We have added 3 different choices of different reward computation methods. (1) No-assign does not assign the score to any other neighbours when a cluster C_i is assigned with a score, *i.e.,* $r_i = \Delta$ and all of $r_j = 0$ when $j \neq i$. (2) Assign-without-τ does not have the distance threshold τ. Once a cluster C_i is assigned with a score, we distribute it to all clusters based on the their distances with C_i, *i.e.,* $r_i = \Delta$ and $r_j = \Delta \times (1 - d(C_i, C_j)/D_{max})$, $C_j \in C$, where D_{max} is the largest distance among pairs of clusters in C. (3) Assign-equally-within-τ equally distributes the score to neighbours with distances within τ, once a cluster C_i is assigned with a score,*i.e.,* $r_j = r_i = \Delta$, if $C_j \in \mathcal{N}(C_i)$. Otherwise, $r_j = 0$ when $C_j \notin \mathcal{N}(C_i)$.

As shown in Fig 10 (the y-axis denotes the metric (AUC and MSE)), on three datasets, **AutoData** achieves the best performance. For example, on HR dataset, the AUC of **AutoData** is 79.3%, which is better than that of Assign-equally-within-τ (75.9%) because the latter one does not distinguish the neighbour clusters of C_i, which leads to inappropriate reward assignment. **AutoData** also outperforms No-assign (74.7%) because it totally ignores the relationships between other clusters with C_i. For Assign-without-τ, **AutoData** performs better because the baseline inappropriately distributes the reward to clusters that are far away from C_i, but they almost have no relationship with C_i.

6.4 Comparison of Clustering Methods

Clustering *vs*. **No-clustering.** We first verify whether clustering is necessary. We design a baseline that does not cluster over the $\mathcal{P}$, but just regards each dataset as a "cluster". As shown in Fig. 11 (the x-axis denotes **AutoData**-MAB and **AutoData**-DQN and the y-axis denotes the metric AUC and MSE), this method performs worse than us a lot. The reason is that data points in the wild are heterogeneous, and thus not all of them can help. Hence, we cannot distinguish helpful data points without clustering.

Comparing different clustering algorithms. We evaluate the performance of different typical clustering methods, including GMM [17], DBSCAN [15] and MeanShift [10], for clustering the data points in $\mathcal{P}$. Recap that in Section 3.2, our RL-based solution needs the statistics of each cluster (*i.e.,* means and covariance matrices) to capture the relationship between the clusters via the distance between them. Using GMM, the Gaussian distribution of each category can be naturally obtained when clustering. But for the other two algorithms, we need to calculate corresponding mean and covariance matrix for each cluster separately after clustering.

Settings. The clustering result is affected by the parameters of the clustering algorithms and we can use different methods to select proper parameters. For GMM, we can use AIC score [1] to determine the appropriate number of components(g). For DBSCAN, there are 2 key parameters: (1) *eps* (the radius of a neighborhood *w.r.t.* some data points) (2) *minPts* (a data point is considered as a core point if at least *minPts* data points are within *eps* of it). They can be set using the method in [46]. Mean-Shift is a centroid-based method that updates the centroids to be the mean of the points within a given region. The size of the region is controlled by *bandwidth*, which can be set by the bandwidth estimation [49].

Effect of different clustering methods. The result is shown in Figure 12, where the x-axis denotes **AutoData**-MAB and **AutoData**-DQN, the y-axis denotes the metric AUC and MSE. We can see that using different methods can affect the performance of **AutoData**-MAB and **AutoData**-DQN, and GMM can obtain the best performance. For example, on Image-6, **AutoData**-DQN has the AUC 85.1% with GMM clustering, better than DBSCAN (81.9%) and Mean-Shift (82.5%). This indicates that GMM is a more robust clustering method, mainly because it can well handle heterogeneous data from different sources, while DBSCAN and Mean-Shift cannot perform well on dataset with large variances in densities.

7 RELATED WORK

Active learning. Active learning [12] interactively selects unlabeled data points just from T and queries users to label them. Different criteria can be used to select these points, such as the most informative ones [3, 27] or the most confusing ones [45]. We have tested 4 active learning methods and shown that our methods were better (see Section 6.1 and Fig. 7).

Coresets. Coresets [29, 55] study the problem that given a full training labeled data, how to select a small subset from it such that training on the small set performs on par with the full training set. The goal is to make the training efficient, rather than considering whether the model performs better on the ML task. Thus, they study a very different problem from us (with different goals), and the solutions are naturally different.

Data acquisition for model fairness. There have been works [33, 51] that use data acquisition to improve model fairness. For the fairness of ML models, *what is a fair model* is pre-defined based on groups, *e.g.,* the desired target distribution is given. Different from them, our problem is to train a good ML model that can generalize without knowing the target distribution.

Data acquisition for model performance. Closer to our work is [25], which tries to search labeled points by querying data markets. However, it assumes that the datasets have the same true data distribution of the ML task at hand, which is hard to guarantee. Moreover, it assumes that buyers can iteratively issue SQL queries by varying predicates and purchase slices of datasets, which are not yet widely supported by data markets. Another close work [54] is data valuation using RL, which assigns a valuation score for each point. The higher the score, the more likely the points will be added to the train set. However, it focuses on selecting points from homogeneous datasets, while not from heterogeneous ones.

Other database techniques *w.r.t.* ML. Data preparation [6] can be utilized to improve the effectiveness of ML model, including data discovery [26], data cleaning [5, 19], data labeling [7, 8, 13, 23] and data exploration [40, 41].

8 CONCLUSION

We study the problem of selective data acquisition in the wild for model charging, which makes an attempt to verify our hypothesis that whether we can boost the model performance by selecting data points from data that is publicly available. The main challenge is that data in wild is messy and many of them are noises that cannot help a downstream ML task. We propose an end-to-end framework, and two solutions, namely MAB-based **AutoData** and DQN-based **AutoData**. We have experimentally verified that our proposed methods can help select data points.

ACKNOWLEDGMENTS

This work is supported by NSF of China (62102215, 61925205, 62072261), Huawei, TAL education, China National Postdoctoral Program for Innovative Talents (BX2021155), China Postdoctoral Science Foundation (2021M691784), Shuimu Tsinghua Scholar and Zhejiang Lab's International Talent Fund for Young Professionals.

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

[3] Kedar Bellare and Suresh Iyengar et al. 2013. Active Sampling for Entity Matching with Guarantees. *ACM Trans. Knowl. Discov. Data* 7, 3 (2013), 12:1–12:24.

[4] Bing API. 2022. https://docs.microsoft.com/en-us/. Accessed: 2022-03-14.

[5] Chengliang Chai, Lei Cao, Guoliang Li, Jian Li, Yuyu Luo, and Samuel Madden. 2020. Human-in-the-loop Outlier Detection. In *SIGMOD Conference 2020*. 19–33.

[6] Chengliang Chai, Guoliang Li, Ju Fan, and Yuyu Luo. 2021. CrowdChart: Crowd-sourced Data Extraction From Visualization Charts. *TKDE* 33, 11 (2021), 3537–3549.

[7] Chengliang Chai, Guoliang Li, Jian Li, Dong Deng, and Jianhua Feng. 2016. Cost-Effective Crowdsourced Entity Resolution: A Partial-Order Approach. In *SIGMOD Conference 2016*. ACM, 969–984.

[8] Chengliang Chai, Guoliang Li, Jian Li, Dong Deng, and Jianhua Feng. 2018. A partial-order-based framework for cost-effective crowdsourced entity resolution. *VLDB J.* 27, 6 (2018), 745–770.

[9] Chengliang Chai, Jiayi Wang, Yuyu Luo, Zeping Niu, and Guoliang Li. 2022. Data Management for Machine Learning: A Survey. *TKDE* (2022), 1–1.

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

[11] Nadiia Chepurko and Ryan Marcus et al. 2020. ARDA: Automatic Relational Data Augmentation for Machine Learning. *PVLDB* 13, 9 (2020), 1373–1387.

[12] Victor Christen, Peter Christen, and Erhard Rahm. 2019. Informativeness-Based Active Learning for Entity Resolution. In *ECML PKDD*, Vol. 1168. 125–141.

[13] Lizhen Cui, Jing Chen, Wei He, Hui Li, Wei Guo, and Zhiyuan Su. 2021. Achieving Approximate Global Optimization of Truth Inference for Crowdsourcing Microtasks. *Data Science and Engineering* 6, 3 (2021), 294–309.

[14] Arthur P et al. Dempster. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society* 39, 1 (1977), 1–22.

[15] Martin Ester and Hans-Peter Kriegel et al. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *KDD-96*. 226–231.

[16] Raul Castro Fernandez and Ziawasch Abedjan et al. 2018. Aurum: A Data Discovery System. In *ICDE*. 1001–1012.

[17] 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.

[18] Google Dataset Search API. 2022. https://developers.google.com/search.

[19] Shuang Hao, Chengliang Chai, Guoliang Li, Nan Tang, Ning Wang, and Xiang Yu. 2020. Outdated Fact Detection in Knowledge Bases. In *ICDE 2020*. IEEE, 1890–1893.

[20] Y. Hirashima, Y. Iiguni, A. Inoue, and S. Masuda. 1999. Q-learning algorithm using an adaptive-sized Q-table. In *IEEE CDC*, Vol. 2. 1599–1604 vol.2.

[21] ImageNet. 2022. https://image-net.org/. Accessed: 2022-03-14.

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

[23] Guoliang Li and Chengliang Chai et al. 2018. CDB: A Crowd-Powered Database System. *PVLDB* 11, 12 (2018), 1926–1929.

[24] Xin Li and Yuhong Guo. 2013. Adaptive Active Learning for Image Classification. In *IEEE CVPR*. 859–866.

[25] Yifan Li, Xiaohui Yu, and Nick Koudas. 2021. Data Acquisition for Improving Machine Learning Models. *CoRR* abs/2105.14107 (2021).

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

[27] Yuyu Luo, Chengliang Chai, Xuedi Qin, Nan Tang, and Guoliang Li. 2020. Interactive Cleaning for Progressive Visualization through Composite Questions. In *ICDE 2020*. IEEE, 733–744.

[28] 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' 21*. 1235–1247.

[29] Baharan Mirzasoleiman and Jeff A. Bilmes et al. 2020. Coresets for Data-efficient Training of Machine Learning Models. In *ICML*, Vol. 119. 6950–6960.

[30] Volodymyr Mnih and Kavukcuoglu et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

[31] Volodymyr Mnih and Koray Kavukcuoglusilvesilver et al. 2013. Playing Atari with Deep Reinforcement Learning. *CoRR* abs/1312.5602 (2013).

[32] Rémi Munos and Tom et al. Stepleton. 2016. Safe and efficient off-policy reinforcement learning. *arXiv preprint arXiv:1606.02647* (2016).

[33] Fatemeh Nargesian and Abolfazl Asudeh et al. 2021. Tailoring Data Source Distributions for Fairness-aware Data Integration. *PVLDB* 14, 11 (2021), 2519–2532.

[34] Fatemeh Nargesian and Ken Q. Pu et al. 2020. Organizing Data Lakes for Navigation. In *SIGMOD*. 1939–1950.

[35] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2018. Table Union Search on Open Data. *PVLDB* 11, 7 (2018), 813–825.

[36] Felix Neutatz, Binger Chen, Ziawasch Abedjan, and Eugene Wu. 2021. From Cleaning before ML to Cleaning for ML. *IEEE Data Eng. Bull.* (2021).

[37] Andrew Ng. 2021. MLOPs: From Model-centric to Data-centric AI.

[38] NYU Auctus. 2022. https://auctus.vida-nyu.org/. Accessed: 2022-01-14.

[39] Danny Pfeffermann and Calyampudi Radhakrishna Rao. 2009. *Sample surveys: design, methods and applications*. Elsevier.

[40] Xuedi Qin, Chengliang Chai, Yuyu Luo, Nan Tang, and Guoliang Li. [n.d.]. Interactively Discovering and Ranking Desired Tuples without Writing SQL Queries. In *SIGMOD Conference 2020*. 2745–2748.

[41] Xuedi Qin and Chengliang Chai et al. 2021. Ranking Desired Tuples by Database Exploration. In *ICDE 2021*. IEEE, 1973–1978.

[42] Alexander Ratner and Christopher De Sa et al. 2017. Data Programming: Creating Large Training Sets, Quickly. arXiv:1605.07723

[43] Alexander Ratner and Stephen H. Bach et al. 2020. Snorkel: rapid training data creation with weak supervision. *VLDB J.* 29, 2-3 (2020), 709–730.

[44] Nicholas Roy and Andrew McCallum. 2001. Toward Optimal Active Learning through Sampling Estimation of Error Reduction. In *(ICML*. 441–448.

[45] Sunita Sarawagi and Anuradha Bhamidipaty. 2002. Interactive deduplication using active learning. In *SIGKDD*. ACM, 269–278.

[46] 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 TODS* 42, 3 (2017), 1–21.

[47] Ozan Sener and Silvio Savarese. 2018. Active Learning for Convolutional Neural Networks: A Core-Set Approach. In *ICLR*.

[48] David Silver and Aja Huang et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nat.* 529, 7587 (2016), 484–489.

[49] Sklearn. 2022. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.estimate_bandwidth.html. Accessed: 2022-03-14.

[50] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement learning - an introduction*. MIT Press.

[51] Ki Hyun Tae and Steven Euijong Whang. 2021. Slice Tuner: A Selective Data Collection Framework for Accurate and Fair Machine Learning Models. In *SIGMOD*.

[52] Laurens Van Der Maaten, Eric Postma, Jaap Van den Herik, et al. 2009. Dimensionality reduction: a comparative. *J Mach Learn Res* 10, 66-71 (2009), 13.

[53] Joannès Vermorel and Mehryar Mohri. 2005. Multi-armed Bandit Algorithms and Empirical Evaluation. In *ECML*, Vol. 3720. 437–448.

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

[55] Hai Yu and Pankaj K. Agarwal et al. 2004. Practical methods for shape fitting and kinetic data structures using core sets. In *SCG*. ACM, 263–272.