Cost-effective Missing Value Imputation for Data-effective Machine Learning

CHENGLIANG CHAI, Computer Science and Technology, Beijing Institute of Technology, Beijing, China
KAISEN JIN, Computer Science and Technology, Beijing Institute of Technology, Beijing, China
NAN TANG, Computer Scisence and Technology, HKUST (GZ), Guangzhou, China
JU FAN, Renmin University of China, Beijing, China
DONGJING MIAO, Faculty of Computing, Harbin Institute of Technology, Harbin, China
JIAYI WANG, Computer Secience and Technology, Tsinghua University, Beijing, China
YUYU LUO, HKUST(GZ), Guangzhou, China
GUOLIANG LI*, Computer Science, Tsinghua University, Beijing, China
YE YUAN, Beijing Institute of Technology, Beijing, China
GUOREN WANG, School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China

Given a dataset with incomplete data (e.g., missing values), training a machine learning model over the incomplete data requires two steps. First, it requires a data-effective step that cleans the data in order to improve the data quality (and the model quality on the cleaned data). Second, it requires a data-efficient step that selects a core subset of the data (called coreset)

*Guoliang Li is the corresponding author.

Guoliang Li is supported by National Key R&D Program of China (2023YFB4503600), NSF of China (61925205, 62232009, 62102215), Shenzhen Project (CJGJZD20230724093403007), Zhongguancun Lab, Huawei, and Beijing National Research Center for Information Science and Technology (BNRist). Chengliang Chai is supported by National Key Research and Development Program of China (2024YFC3308200), NSFC (62472031), CCF-Baidu Open Fund (CCF-Baidu202402) and Huawei. Ju Fan is supported by the NSFC (62436010 and 62441230), the Beijing Natural Science Foundation (L244010 and L222006), and the Research Funds of Renmin University of China. Dongjing Miao is supported by NSFC (62372138), Heilongjiang Natural Science Foundation (HSF20230095, 2024ZXJ01A04) and CCF- Huawei Populus Grove Fund(CCF-HuaweiDB202406). Ye Yuan is supported by Beijing Natural Science Foundation (L241010), the National Key R&D Program of China (Grant No.2022YFB2702100), and the NSFC (Grant Nos. 61932004, 62225203, U21A20516). Nan Tang is supported by Guangzhou Municipality Big Data Intelligence Key Lab (2023A03J0012). Yuyu Luo is supported by NSFC (62402409), Guangdong Basic and Applied Basic Research Foundation (2023A1515110545), and Guangzhou-HKUST(GZ) Joint Funding Program (2025A03J3714). Guoren Wang is supported by the NSFC (62427808, U2001211), the Liaoning Revitalization Talents Program (Grant No.XLYC2204005).

Authors' Contact Information: Chengliang Chai, Computer Science and Technology, Beijing Institute of Technology, Beijing, Beijing, China; e-mail: ccl@bit.edu.cn; Kaisen Jin, Computer Science and Technology, Beijing Institute of Technology, Beijing, Beijing, China; e-mail: jks@bit.edu.cn; Nan Tang, Computer Scisence and Technology, HKUST (GZ), Guangzhou, Guangdong, China; e-mail: nantang@hkust-gz.edu.cn; Ju Fan, Renmin University of China, Beijing, China; e-mail: fanj@ruc.edu.cn; Dongjing Miao, Faculty of Computing, Harbin Institute of Technology, Harbin, Heilongjiang, China; e-mail: miaodongjing@hit.edu.cn; Jiayi Wang, Computer Secience and Technology, Tsinghua University, Beijing, Beijing, China; e-mail: jiayi-wa20@mails.tsinghua.edu.cn; Yuyu Luo, HKUST(GZ), Guangzhou, Guangdong, China; e-mail: yuyuluo@hkust-gz.edu.cn; Guoliang Li, Computer Science, Tsinghua University, Beijing, Beijing, China; e-mail: liguoliang@tsinghua.edu.cn; Ye Yuan, Beijing Institute of Technology, Beijing, Beijing, China; e-mail: yuan-ye@bit.edu.cn; Guoren Wang, School of Computer Science and Technology, Beijing Institute of Technology, Beijing, Beijing, China; e-mail: wanggrbit@126.com.

ACM 1557-4644/2025/3-ART
https://doi.org/10.1145/3716376

such that the trained models on the entire data and the coreset have similar model quality, in order to save the computational cost of training. The first-data-effective-then-data-efficient methods are too costly, because they are expensive to clean the whole data; while the first-data-efficient-then-data-effective methods have low model quality, because they cannot select high-quality coreset for incomplete data.

In this paper, we investigate the problem of coreset selection over incomplete data for data-effective and data-efficient machine learning. The essential challenge is how to model the incomplete data for selecting high-quality coreset. To this end, we propose the GoodCore framework towards selecting a good coreset over incomplete data with low cost. To model the unknown complete data, we utilize the combinations of possible repairs as possible worlds of the incomplete data. Based on possible worlds, GoodCore selects an expected optimal coreset through gradient approximation without training ML models. We formally define the expected optimal coreset selection problem, prove its NP-hardness, and propose a greedy algorithm with an approximation ratio. To make GoodCore more efficient, we propose optimization methods that incorporate human-in-the-loop imputation or automatic imputation method into our framework. Moreover, a group-based strategy is utilized to further accelerate the coreset selection with incomplete data given large datasets. Experimental results show the effectiveness and efficiency of our framework with low cost.

CCS Concepts: • **Computing methodologies → Machine learning**; *Machine learning approaches*; *Machine learning algorithms*; • **Information systems → Data cleaning**.

Additional Key Words and Phrases: data-centric AI; machine learning; data cleaning; coreset selection

1 Introduction

Data-effective machine learning (ML) (a.k.a. data-centric AI [66]) aims at obtaining high-quality training data to release the value of AI, because it is well-known that dirty data may severely degrade the performance of ML models [22, 65].

Data-efficient ML focuses on saving the training cost, i.e., making the training process more efficient. A commonly used strategy is to select a core subset of training data (or coreset) [34, 62] to represent the entire dataset such that ML models trained on the coreset can achieve similar performance to the ML models trained on the entire dataset.

Apparently, users desire both data-effective ML (for training better ML models) and data-efficient ML (for saving training cost). In this work, our main goal is to support both data-effective and data-efficient ML over *incomplete data* where there are many missing values, which is very common in real-world scenarios [22, 57, 78].

Running data-effective and data-efficient tools sequentially. Intuitively, we can either run data imputation methods first for data-effective and then run coreset selection algorithms denoted by $\mathbf{C}(\cdot)$ for data-efficient, or vice versa. Moreover, for data-effective solutions through data cleaning, we generally consider two cases, either human-based solutions denoted by $\mathbf{H}(\cdot)$ or automatic solutions denoted by $\mathbf{A}(\cdot)$. In summary, we have the following four cases, as shown in Figure 1:

- *First data-effective (impute) then data-efficient (coreset):*
  (1) Impute-Human: $\mathbf{H}(D) \rightarrow$ Coreset: $\mathbf{C}(\mathbf{H}(D))$
  (2) Impute-Auto: $\mathbf{A}(D) \rightarrow$ Coreset: $\mathbf{C}(\mathbf{A}(D))$
- *First data-efficient (coreset) then data-effective (impute):*
  (3) Coreset: $\mathbf{C}(D) \rightarrow$ Impute-Human: $\mathbf{H}(\mathbf{C}(D))$
  (4) Coreset: $\mathbf{C}(D) \rightarrow$ Auto-Human: $\mathbf{A}(\mathbf{C}(D))$

Next let's discuss the pros and cons of the above approaches.

Case (1) has high human cost, low machine cost, and high accuracy in terms of the trained ML models. Case (2) has zero human cost, low machine cost, but with low accuracy because automatic imputation may not be good enough. Case (3) has low human cost, low machine cost, but with low accuracy because corset selection over a

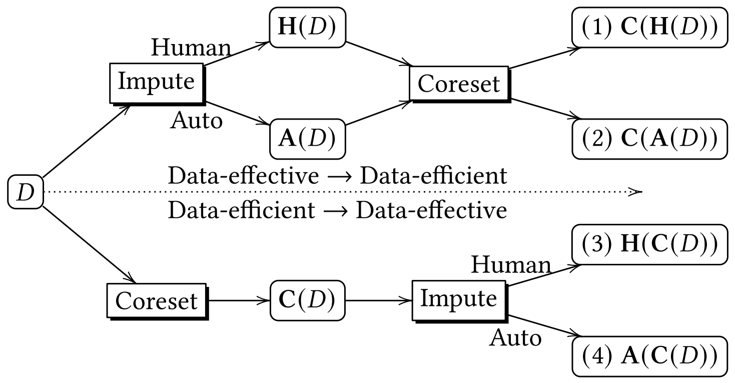


Fig. 1. Sequential methods.

Solution	**Accuracy**	**Human Cost**	**Machine Cost**
(1) $\mathbf{C}(\mathbf{H}(D))$	High	High	Low
(2) $\mathbf{C}(\mathbf{A}(D))$	Low	None	Low
(3) $\mathbf{H}(\mathbf{C}(D))$	Low	Low	Low
(4) $\mathbf{A}(\mathbf{C}(D))$	Low	None	Low
Our goal	High	None or Low	Low or Very Low
(5) $\mathbf{H}(\mathbf{G}(D))$	High	Low	High
(6) $\mathbf{A}(\mathbf{G}(D))$	Medium	None	High
(7) $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$	High	Low	Low
(8) $\mathbf{G}(D, \circlearrowleft^{\mathbf{A}})$	Medium	None	Low
(9) $\mathbf{G}^{+}(D, \circlearrowleft^{\mathbf{H}})$	High	Low	Very Low
(10) $\mathbf{G}^{+}(D, \circlearrowleft^{\mathbf{A}})$	Medium	None	Very Low

Fig. 2. A comparison of different approaches (1–4: sequential methods; 5–10: our solutions).

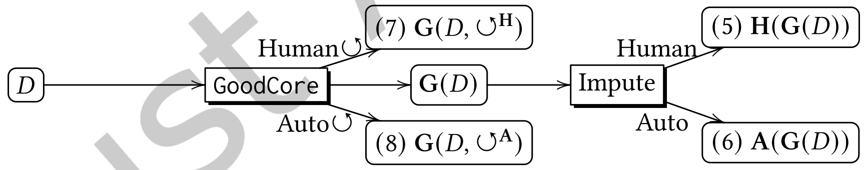


Fig. 3. Our proposal and its variants.

dirty dataset may not ensure to compute a "good" coreset. Case (4) has no human cast, low machine cost, but with low accuracy with the similar reason as (3). The comparison of the above four methods can be found in Figure 2.

Our goal. Clearly, a primary goal is to achieve high accuracy for ML models, where only case (1) can achieve. Case (2) achieves low accuracy because automatic imputation is hard to be accurate. The main obstacle for making (1) practical is its high human cost. Hence, our *main goal* is to achieve high accuracy with no or low human cost, and with low machine cost.

Consider cases (3) and (4), the main reason for them to achieve low accuracy is because they cannot compute a good coreset directly from the dirty data. Intuitively, if we can compute a good coreset directly from the dirty data, we can cheaply clean the coreset to achieve high accuracy, where the “goodness” means that the subset of tuples selected from the dirty data is similar to the subset of tuples selected from the clean data.

Challenge. The main challenge of computing a good coreset from dirty data is to accurately estimate the ground truth of each missing value; otherwise, we cannot select a coreset to well represent the clean data. This is a known hard problem because each missing value may have multiple possible repairs. Also, because a coreset selection algorithm is typically iterative that each tuple is selected per iteration [58], selecting a bad tuple may cause cascade amplification to the following iterations, resulting in a bad coreset.

Our proposal. To tackle the above challenge, we model the combinations of possible repairs as possible worlds of the original dirty data D. We then formulate it as an optimization problem for selecting an expected optimal coreset that can represent the possible worlds of D via gradient approximation without training in advance. We prove this problem to be NP-hard. We propose an approximate algorithm, called GoodCore, denoted by $\mathbf{G}(\cdot)$, with the main idea to iteratively add a tuple with the highest utility into the coreset. After a good coreset is computed, we can either use human imputation or automatic imputation to impute the data, as shown in Figure 3. We further elaborate these two methods below:

(5) GoodCore: $\mathbf{G}(D) \rightarrow$ Impute-Human: $\mathbf{H}(\mathbf{G}(D))$
(6) GoodCore: $\mathbf{G}(D) \rightarrow$ Impute-Auto: $\mathbf{A}(\mathbf{G}(D))$

However, one main drawback is that modeling possible worlds of D is computationally expensive, which hinders the practicability of the GoodCore algorithm. To address this high computational cost problem, we further propose imputation-in-the-loop optimization (with either humans or automatic methods) into the GoodCore algorithm (see methods 7 and 8 in Figure 3). To this end, the optimized algorithms can significantly reduce the number of possible worlds, thus achieving low computational cost.

(7) GoodCore with human-in-the-loop imputation: $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$
(8) GoodCore with machine-in-the-loop imputation: $\mathbf{G}(D, \circlearrowleft^{\mathbf{A}})$

Besides, since the above methods for coreset selection incorporate at least one iteration over the entire dataset, it is not very efficient when the dataset is large, so we propose a group-based acceleration strategy to further reduce the machine cost. The key idea is to assign similar tuples in D into a group and select a coreset to represent these groups. Since the groups can still represent the distribution of D, the selected coreset is still well-performed. In this way, we only need to iterate these groups, with a much smaller number than the tuples of D, and thus the efficiency is improved. We also provide a theoretical analysis with respect to the group-based strategy. Hence, we further have the following 2 methods.

(9) Group-based GoodCore (GoodCore^{+}) with human-in-the-loop imputation: $\mathbf{G}^{+}(D, \circlearrowleft^{\mathbf{H}})$.
(10) GoodCore^{+} with machine-in-the-loop imputation: $\mathbf{G}^{+}(D, \circlearrowleft^{\mathbf{A}})$.

A comparison of methods (5)–(10) is given in Figure 2. Note that method (10) is likely to be a good choice because it can achieve a high ML accuracy with low human cost and low machine cost.

Contributions We make the following contributions.

(i) *Two birds with one stone.* We study the problem of solving both data-effective and data-efficient ML in one framework, which is an important but not addressed problem. (Section 3)
(ii) *NP-hardness and approximate solutions.* We prove the NP-hardness of the problem. We propose a greedy algorithm with an approximate ratio. (Section 4)
(iii) *Imputation-in-the-loop optimizations.* We develop optimization techniques that integrate imputation-in-the-loop into the coreset selection process, to improve the efficiency while achieving high accuracy. We also analyze the convergence rate of our method and theoretically prove that it can converge fast. (Section 5)

(iv) *Group-based acceleration.* We develop group-based techniques to further improve the efficiency. We also analyze the theoretical guarantee and convergence of the proposed techniques. (Section 6)

(v) *Experiments.* We conduct extensive experiment on 8 real-world datasets and compare with 10 baselines to show that GoodCore can select a well-performed coreset to achieve both data-effective and data-efficient ML while consuming a low human cost. (Section 7)

2 Background of Coreset Selection

In this section, we introduce the background of coreset selection on complete data, denoted by D_c.

2.1 Gradient Descent for Machine Learning

Gradient descent [51] is the most typical optimization algorithm to train ML models. At a high level, it tweaks the parameters iteratively to minimize a given convex and differentiable function to its local minimum.

Let $D_c = \{t_1, t_2, ..., t_n\}$ be a set of train tuples (without missing values), where $t_i = (\mathbf{x}_i, \mathbf{y}_i)$, $\mathbf{x}_i \in \mathbb{R}^d$ denotes the vector of features and $\mathbf{y}_i$ denotes the corresponding label. The goal of training on D_c is to find the best parameter θ^* of a model by minimizing the loss:

$$\theta^* = \arg\min_{\theta \in \vartheta} f(\theta), f(\theta) = \frac{1}{n}\sum_{i=1}^{n} f_i(\theta, t_i) \tag{1}$$

where ϑ is the parameter space. For ease of representation, we abbreviate $f_i(\theta, t_i)$ as $f_i(\theta)$ to represent the loss of the i-th train example. Generally speaking, the gradient descent approach is always applied to find the minimizer of Eq. 1, where the **full gradient** (sum of the gradients over all training tuples), denoted by $\nabla f(\theta) = \sum_{i=1}^{n} \nabla f_i(\theta)$, has to be computed iteratively.

Besides incremental gradient methods like stochastic gradient descent (SGD) that can be leveraged to accelerate the iterative gradient computation, there are other popular and orthogonal methods, such as coreset, which will be discussed next.

2.2 Coreset over Complete Data

Coreset. To make training more efficient, instead of learning from entire D_c, one research question is that whether we can compute a small subset $\mathbf{C}(D_c)$ of D_c such that learning with $\mathbf{C}(D_c)$ can hopefully achieve the same performance as learning with D_c. This small selected subset is called **coreset** [34, 62]. In the following, we simply write $\mathbf{C}(D_c)$ as C when it is clear from the context.

The state-of-the-art coreset selection solutions are mostly based on gradient approximation [44, 58]. Suppose that θ denotes the parameter of an ML model trained over the full dataset, and θ' denotes the parameter of the same model trained over the coreset. Intuitively, the objective of gradient approximation for coreset selection is to make $\nabla f(\theta')$ as close as possible to $\nabla f(\theta)$. To this end, existing solutions focus on *selecting the coreset that minimizes the upper bound of gradient approximation error* ($\|\nabla f(\theta) - \nabla f(\theta')\|$). Next, let's formally define it from scratch.

Gradient-based coreset selection is to minimize the **gradient approximation error (GA error)** between the full gradient w.r.t. D_c and the weighted sum of gradients w.r.t. the coreset C (or coreset gradient). Formally, Eq. 2 tries to minimize the GA error by considering all possible parameters $\theta \in \vartheta$ (*i.e.*, $\max\limits_{\theta\in\vartheta}$), where "$\|\cdot\|$" denotes the normed difference. Next, we introduce the coreset gradient.

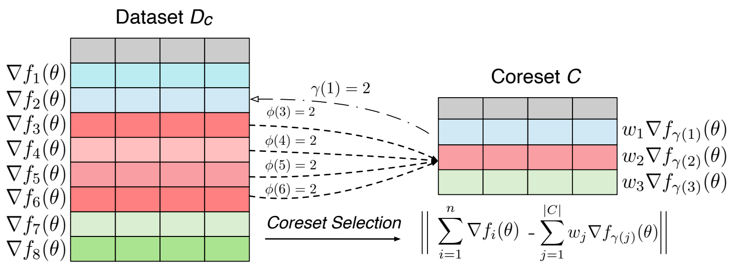


Fig. 4. Example of coreset selection.

$$C^* = \underset{C\subseteq D_c, w_j\geq 0}{\arg\min}\ \max_{\theta\in\vartheta} \underbrace{\| \underbrace{\sum_{i=1}^{n}\nabla f_i(\theta)}_{\text{full gradient}} - \underbrace{\sum_{j=1}^{|C|} w_j \nabla f_{\gamma(j)}(\theta)}_{\text{coreset gradient}} \|,}_{\textbf{gradient approximation error}} \quad s.t.\ |C| \leq K \tag{2}$$

Because the coreset is a subset of the complete dataset (*i.e.*, $C \subseteq D_c$), we use $\gamma(j) = i$ (where $j \in [1, |C|], i \in [1, n]$) to denote that the j-th tuple in C (denoted by c_j) is the i-th tuple in D_c, *i.e.*, t_i. In other words, γ is an index mapping from C to D_c.

Recall that the key idea of the coreset is to *use a subset of tuples to represent the entire set.* Eq. 2 potentially contains another important mapping ϕ from D_c to C to indicate this, *i.e.*, $\phi(i) = j, i \in [1, n], j \in [1, |C|]$, which is highly related to the weight. Specifically, let $\phi(i) = j$ denote that we will assign t_i to c_j (use c_j to represent t_i) and use $\nabla f_{\gamma(j)}$ to represent ∇f_i. Each t_i will be assigned to one and only one c_j, but each c_j might be assigned with multiple tuples in D_c. Based on ϕ, w_j is defined as the weight of the c_j, which is the number of tuples in D_c assigned to the c_j, *i.e.*, $w_j = |\{t_i|\phi(i) = j, i \in [1, n]\}|$ (c_j is utilized to represent w_j tuples in D_c).

Next let's use an example to better illustrate Eq. 2.

Example 1. *Let's consider a case of the gradients of each tuple, as shown in Figure 4. Suppose that for any* θ, $\nabla f_1(\theta) \approx \nabla f_2(\theta)$, $\nabla f_3(\theta) \approx \nabla f_4(\theta) \approx \nabla f_5(\theta) \approx \nabla f_6(\theta)$ *and* $\nabla f_7(\theta) \approx \nabla f_8(\theta)$. *In this case, based on Eq. 2, if one wants to find an optimal coreset with a size of 3, i.e.,* $K = 3$, *the solution can be* $C^* = \{t_2, t_5, t_7\}$ ($\gamma(1) = 2, \gamma(2) = 5$ *and* $\gamma(3) = 7$), *associated with* $w_1 = 2, w_2 = 4, w_3 = 2$ *because* $\phi(1) = \phi(2) = 1, \phi(3) = \phi(4) = \phi(5) = \phi(6) = 2$ *and* $\phi(7) = \phi(8) = 3$. *In this way,* C^* *can be one of the optimal coresets that can well approximate the full gradient because* $\|\sum_{i=1}^{8}\nabla f_i(\theta) - \sum_{j=1}^{3} w_j \nabla f_{\gamma(j)}(\theta)\|$ *is minimized, which is close to 0.*

Key observation. We can observe from Example 1 that in order to minimize the GA error, we should set $\phi(i) = j$, where ∇f_i and $\nabla f_{\gamma(j)}$ are likely to be close. Therefore, computing the coreset is similar to computing the K exemplars [67] of the gradients, if all the gradients of tuples can be computed.

Upper bound minimization of GA error. We can see from Eq. 2 that to solve the equation, the gradients have to be computed, which have a close relationship with the parameter θ. However, the main bottleneck is that the entire parameter space ϑ is too expensive to explore. Hence, a typical solution is to first compute the upper bound of GA error (Eq. 3), then generalize [8, 37, 58] the upper bound computation to the entire parameter space

(Eq. 4), and finally select the coreset to minimize the bound. To be specific, using the triangle equation, for any particular θ, we have:

$$\|\sum_{i=1}^{n} \nabla f_i(\theta) - \sum_{j=1}^{|C|} w_j \nabla f_{Y(j)}(\theta)\| \leq \sum_{i=1}^{n} \|\nabla f_i(\theta) - \nabla f_{Y(\phi_\theta(i))}(\theta)\| \tag{3}$$

Together with the aforementioned observation, given a coreset C, the upper bound is minimized when ϕ assigns every tuple t_i to the tuple in C with most gradient similarity, *i.e.*, $\|\sum_{i=1}^{n} \nabla f_i(\theta) - \sum_{j=1}^{|C|} w_j \nabla f_{Y(j)}(\theta)\| \leq \sum_{i=1}^{n} \min_{c_j \in C} \|\nabla f_i(\theta) - \nabla f_{Y(j)}(\theta)\|$.

For the entire space ϑ, it has been proved in recent works [8, 37, 58] that for convex ML problems (corresponding to an optimization problem in which the objective function is a convex function), the normed gradient difference between tuples can be efficiently bounded by:

$$\forall i, j, \max_{\theta \in \vartheta} \|\nabla f_i(\theta) - \nabla f_j(\theta)\| \leq \max_{\theta \in \vartheta} O(\|\theta\|) \cdot \|\mathbf{x}_i - \mathbf{x}_j\| \tag{4}$$

where $\|\mathbf{x}_i - \mathbf{x}_j\|$ is the Euclidean distance between feature vectors of two tuples, namely *feature distance*, and $O(\|\theta\|)$ is a constant. Hence, we can conclude that **GA error can be bounded independent of the optimization problem in practice,** *i.e.*, **any particular** θ. Finally, considering Eq. 3 and Eq. 4 together, *the coreset selection problem can be converted to:*

$$C^* = \arg\min_{C \subseteq D_c} \sum_{i=1}^{n} \min_{c_j \in C} s_{ij}, \text{ s.t. } |C| \leq K \tag{5}$$

where $s_{ij} = \|\mathbf{x}_i - \mathbf{x}_{Y(j)}\|$ for ease of representation. The above equation indicates that given a train data D_c and a coreset C, we use $S = \sum_{i=1}^{n} \min_{c_j \in C} s_{ij}$ to score the coreset. The lower the score, the smaller upper bound of the GA error we can get, which indicates a better coreset. To summarize, solving Eq. 5 is to minimize the upper bound of the GA error (*i.e.*, select the coreset with the lowest score) by just considering the feature vectors of the training tuples without training in advance.

Note that Eq. 4 holds for tuples associated with the same label [8, 37]. Therefore, in practice, we respectively select coresets for tuples with different labels and combine them. Suppose that we aim to select a coreset with size K for a binary classification task (label 1: 60%, label 0: 40%), so we select a coreset with size $60\%K$ for tuples with label 1 and another one with $40\%K$ for tuples with label 0.

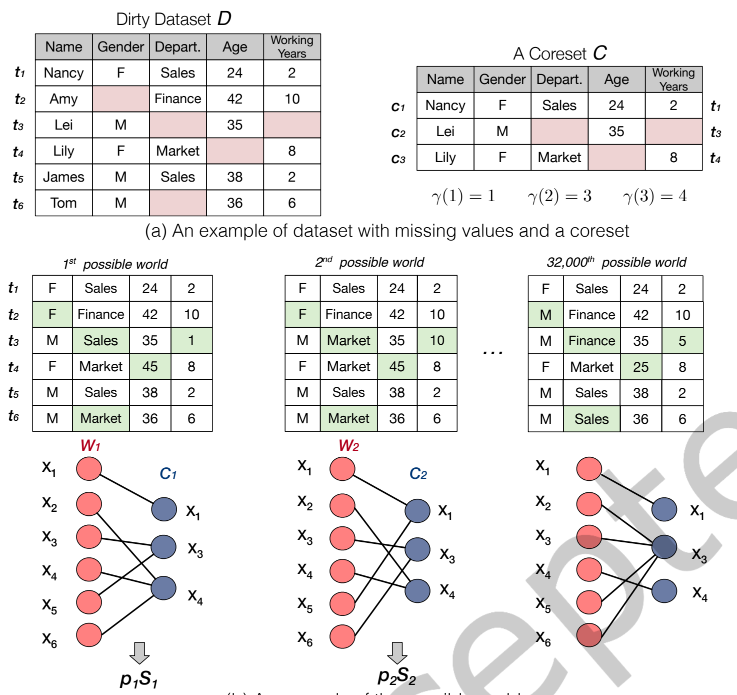
Our scope. In this paper, we focus on the convex problems (*e.g.*, logistic regression, support vector machine, etc.) because for such problems the gradient difference can be well bounded by the difference between feature vectors. Note that, for other ML algorithms such as deep neural networks, they can also be trained using selected coreset to achieve good training accuracy (see Section 7 for our experimental findings).

3 Coreset Over Incomplete Data

In this section, we will formally define the problem of coreset selection over incomplete data (Section 3.1) and then describe our proposed framework to solve the problem (Section 3.2).

3.1 Problem Definition

As discussed above, we have to compute the coreset score S, so as to produce a good coreset. To this end, the feature distances can be computed as a pre-processing step, based on which the coreset score can be computed. However, when there exists incomplete data with missing values, even the feature distances are hard to compute accurately, let alone selecting a proper coreset.



	Name	Gender	Depart.	Age	Working Years
t_1	Nancy	F	Sales	24	2
t_2	Amy		Finance	42	10
t_3	Lei	M		35	
t_4	Lily	F	Market		8
t_5	James	M	Sales	38	2
t_6	Tom	M		36	6

	Name	Gender	Depart.	Age	Working Years	
c_1	Nancy	F	Sales	24	2	t_1
c_2	Lei	M		35		t_3
c_3	Lily	F	Market		8	t_4

(a) An example of dataset with missing values and a coreset

t_1	F	Sales	24	2
t_2	F	Finance	42	10
t_3	M	Sales	35	1
t_4	F	Market	45	8
t_5	M	Sales	38	2
t_6	M	Market	36	6

F	Sales	24	2
F	Finance	42	10
M	Market	35	10
F	Market	45	8
M	Sales	38	2
M	Market	36	6

F	Sales	24	2
M	Finance	42	10
M	Finance	35	5
F	Market	25	8
M	Sales	38	2
M	Sales	36	6

(b) An example of the possible worlds

Fig. 5. Example of coreset selection with missing values

Incomplete data. Formally, suppose that D has M attributes, denoted by $\{\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_M\}$. Each attribute $\mathcal{A}_m, m \in [1, M]$ represents a domain set including the `Null`, (*i.e.*, $\texttt{Null} \in \mathcal{A}_m$), in which each tuple in D can take value on this attribute. $|\mathcal{A}_m|$ denotes the domain size. Then, each tuple $t_i \in \mathcal{A}_1 \times \mathcal{A}_2 \times, ..., \times \mathcal{A}_m$. Let $t_i[m]$ denote the value of the $m-$th attribute of t_i, *i.e.*, $t_i[m] \in \mathcal{A}_m$.

For a tuple $t_i \in D$, if $\exists\, t_i[m] = \texttt{Null}, m \in [1, M]$, t_i is an incomplete tuple, denoted by $\mathbb{I}[t_i] = 1$, otherwise $\mathbb{I}[t_i] = 0$. Let us better illustrate this using an example.

Example 2. *As shown in Figure 5(a), there are 6 tuples in the table D with five attributes (an excerpt from a large table). For example, $\mathcal{A}_2$ is the* `Gender` *attribute, i.e.,* $\mathcal{A}_2 = \{\texttt{M}, \texttt{F}, \texttt{Null}\}$. *Among these tuples, t_2, t_3, t_4, t_6 have missing values, e.g., $\mathbb{I}[t_2] = 1, \mathbb{I}[t_1] = 0$. Given a coreset as shown on the right side, if there are no missing values, we can assign each tuple $t_i \in D$ to its most similar tuple in C (compute $\min_{c_j \in C} s_{ij}$), and then sum these feature distances up to compute the coreset score S. However, given these missing values, the feature distances cannot be computed accurately (e.g., s_{12}, s_{13}, s_{22}, etc.), and thus the assignment of tuples in D cannot be determined precisely. Hence, the coreset score is not precise, and thereby leads to a coreset that cannot well represent the full complete (clean) data.*

As discussed above, *imputation before coreset selection* suffers from either large cost (human imputation) or large number of possible repairs (automatic imputation), while *imputation after coreset selection* cannot obtain a good coreset because of the inaccurate feature distance computation (see Example 2).

Therefore, an essential problem is to select a good coreset that can represent the complete dataset D_c, which relies on accurate coreset score computation given D_c that is the unknown ground truth. Fortunately, the possible repairs of D can be modeled by possible worlds [11–13, 26], based on which we can effectively select the coreset over incomplete data.

Possible worlds. Given the incomplete dataset D, $\forall t \in D$ and $\mathbb{I}[t] = 1$, $\forall t[m] = \texttt{Null}$, $m \in [1, M]$, we assign a value in $\mathcal{A}_m \setminus \{\texttt{Null}\}$ to $t[m]$ as an imputation (a.k.a. a possible repair). Thus, we have an assignment for all the missing values in D, which corresponds to a possible world W. Since there exist a large number of possible assignments, we define the set of possible worlds as $\mathcal{I}_W = \{W_k | k \in [1, |\mathcal{I}_W|]\}$.

Let us better illustrate this using an example.

Example 3. *Given D, for tuples t_2, t_3, t_4, t_6 with missing values, we have a large number of possible assignment as shown in Figure. 5(b), each of which corresponds to a possible world (we omit the* `Name` *attribute because there is no missing value on this attribute). Suppose that there are 2 (4/100/10) types of values of the attribute* `Gender` *(*`Department/Age/Working years`*), there exist 32,000 possible worlds in total.*

Note that for numerical attributes, we will bin them into different buckets, such that we can treat them as categorical values and avoid the unlimited number of possible worlds.

Even with possible worlds, the score computation of coreset remains challenging. Each possible world of D is a complete dataset, and thus given a coreset, the score can be directly computed considering the feature distances, as discussed in Section 2.2. However, the crucial issue is that each possible world could be the ground truth, *i.e.*, D_c, but each one leads to a different score.

Example 4. *As shown in Figure. 5(b), the two possible worlds W_1 and W_2 are only different in t_3, leading to a different feature vector $\mathbf{x}_3$, which makes the score computation a difference. To be specific, given the same coreset C with tuples $t_1(c_1)$, $t_3(c_2)$ and $t_4(c_3)$, because of a different $\mathbf{x}_3$, the closest feature distance of $\mathbf{x}_5$ in W_2 becomes $\mathbf{x}_1$, rather than $\mathbf{x}_3$ in W_1. And the closest feature distance of $\mathbf{x}_6$ in W_2 becomes $\mathbf{x}_3$, rather than $\mathbf{x}_4$ in W_1. Therefore, the coreset scores, i.e., the sum of these closest feature distances of tuples are different among possible worlds.*

Example 4 shows that different possible worlds make the mapping ϕ different, which leads to different scores. Hence, to get a good coreset without the ground truth, an intuitive solution is to compute the expected coreset score considering all possible worlds. By doing so, although we cannot get the complete data (D_c) in advance, we can focus on how to select an informative coreset that can represent the possible worlds of D.

Next, we formally define the studied problem.

Expected optimal coreset selection over incomplete data. Given D, we have a number of possible worlds $\mathcal{I}_W = \{W_k\}$. Then given a subset (coreset) $C \subset D$, for different W_k, we have the corresponding C_k with the same tuples as C but probably different imputations. For C_k, we can compute a score $S_k = \sum_{i=1}^{n} \min_{c_j \in C_k} s_{ij}$, where $s_{ij} = \|\mathbf{x}_i - \mathbf{x}_{\gamma(j)}\|$ and both feature vectors are from $\{W_k\}$. Then, we have the expectation $\mathrm{E}[C] = \sum_{k=1}^{|\mathcal{I}_W|} p_k S_k$, where p_k denotes the probability of the appearance of $\{W_k\}$. Finally, our problem becomes how to compute the coreset C with the lowest expectation of GA error upper-bound. Formally, we have

$$C^* = \arg\min_{C \subseteq D} \mathrm{E}[C], \text{ s.t. } |C| \leq K \tag{6}$$

For example, given D, the corresponding possible worlds and a coreset C in Figure 5, we have different C_k with the same tuples (containing t_1, t_3, t_4) but probably different imputations. For each C_k, we will compute S_k, and finally compute $\mathrm{E}[C]$. Solving Eq. 6 can result in an informative coreset with incomplete tuples being selected. After these tuples imputed by a human, *i.e.*, Case (5), or state-of-the-art automatic method, *i.e.*, Case (6), we can derive a good coreset.

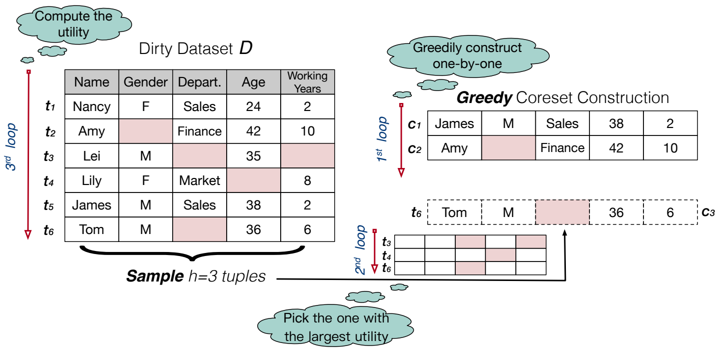


Fig. 6. The GoodCore framework.

3.2 Goodcore Framework

Next, we will introduce our proposed GoodCore framework to solve Eq. 6, which is non-trivial because it is NP-hard. But fortunately, we prove that it has the sub-modular property (see Section 4). Hence, GoodCore uses a greedy framework with three loops to solve the problem with an approximate ratio.

At a high level, the greedy strategy adds one tuple with the largest "utility" to the coreset iteratively, which can be considered as the first loop. In each iteration, we have to iterate tuples in D to select the one with the largest utility, which is the second loop. Naturally, we have to compute the utility of each tuple, where all tuples in D have to be considered, leading to the third loop.

Next, we will further illustrate the framework using Figure 6 and Algorithm 1.

The first loop (lines 3-9) of the greedy algorithm is to add the tuple t^* with the maximum *utility* (*i.e.*, $\mathrm{E}[t|C] = \mathrm{E}[C] - \mathrm{E}[C \cup \{t\}]$) into the coreset iteratively for K times. To be specific, the "utility" of a tuple t denotes the reduction of expectation of GA error after adding t into the coreset C.

Suppose that $K = 3$. Figure 6 (the 1^{st} loop part) shows the situation that there already have been 2 tuples in C, and we are going to add the third tuple into the coreset.

The second loop (lines 6-7) computes the utilities of tuples that are not in coreset C, based on which the best one is picked for the first loop. An ideal solution is to consider all tuples in $D \setminus C$, which is prohibitively expensive, so in practice we use an efficient method to accelerate this loop by uniformly sampling h tuples as T_{sample} (line 5) and then selecting the best one from T_{sample} (line 14). The difference is that theoretically, considering all tuples has an approximate ratio 1-$\frac{1}{e}$ (because of the sub-modular property), while the sampling method holds a (1-$\frac{1}{e} - \epsilon$) ratio [61], where ϵ is related to the sampling ratio.

As shown in Figure 6, suppose that $h = 3$, and we sample $\{t_3, t_4, t_6\}$ from $\{t_1, t_3, t_4, t_6\}$. Then the second loop iterates the three tuples and computes the utility for each one (the third loop).

The third loop (line 7) will loop through all tuples in D, so as to compute the utility of tuple t used in the second loop. To be specific, the core part of the utility computation (*i.e.*, ComputeUtility) is to compute $\mathrm{E}[C] = \sum_{k=1}^{|\mathcal{I}_W|} p_k S_k = \sum_{k=1}^{|\mathcal{I}_W|} p_k (\sum_{i=1}^{n} \min_{c_j \in C_k} s_{ij})$, from which we can see that it is inevitable to iterate the n tuples in D. However, the most challenging part is that we also have to enumerate a large number of possible worlds. We will illustrate how to solve this in details in Section 4.

Algorithm 1: GoodCore Framework

```
Input: Incomplete train data $D$, coreset size $K$, sample size $h$.
Output: A coreset $C \subseteq D$, weight $\mathbb{W} = \{w_j\}$,$|C| = |\mathbb{W}| = K$.
1  $C = \emptyset$;
2  while $|C| < K$ do
3      /*1st loop*/
4      Sample $h$ tuples as $T_{sample} \subseteq D \setminus C$
5      for each tuple $t \in T_{sample}$ do
6          /*2nd loop*/
7          $\mathrm{E}[t|C]$ = ComputeUtility($t, C, D$); /*3rd loop*/
8      $t^* = \arg\max_{t \in T_{sample}} \mathrm{E}[t|C]$ ;
9      $C = C \cup \{t^*\}$;
10 for $t \in C$ do
11     if $\mathbb{I}[t] = 1$ then
12         Impute $t$ by a human or automatic method.
13 for $j = 1$ to $|C|$ do
14     for $i = 1$ to $n$ do
15         if $c_j = \arg\min_{c_{j'} \in C} \max_{\theta \in \vartheta} \|\nabla f_i(\theta) - \nabla f_{\gamma(j')}(\theta)\|$ then
16             $w_j$ += 1;
17 return $C, \mathbb{W}$;
```

The imputation step (line 12). After GoodCore selects the coreset C using the above 3 loops, we can leverage a human or automatic method to impute the tuples that are incomplete in C, which correspond to Case (5) and Case (6) in Section 1 respectively.

Weights computation (lines 20-23). It computes the weight of each tuple in C, which will be used to approximate the full gradient during training. For training, tuples in the coreset are randomly shuffled. Afterwards, suppose that in each step of the gradient decent, when we use $c_j \in C$ to update the gradient, we compute the gradient (∇f_j) of c_j first, and then use $w_j \nabla f_j$ to update the model parameters. w_j is the number of tuples in D assigned to c_j. The above steps repeat until the model converges.

Imputation-in-the-loop optimizations. Unfortunately, the 3-loop computation of the strategy is rather expensive due to the large number of possible worlds (Section 4). To address this, we can integrate either human-in-the-loop or the automatic method into GoodCore framework (Section 5). It iteratively imputes one incomplete tuple or a mini-batch of incomplete tuples. Once the tuple(s) is (are) computed and added to the coreset within the first loop, the number of possible worlds can be significantly reduced, and so does the computational cost.

Group-based acceleration. As discussed above, we have to iterate all tuples of D in the third loop to compute the utility of a tuple t. Given a large train set with missing values, it is still inefficient to compute the coreset. To address this, we propose to assign tuples in D to multiple groups, and use these groups to represent the entire dataset. Since the number of groups are smaller than n, the efficiency can be much improved (Section 6).

4 Goodcore Algorithm

In this section, we will illustrate `GoodCore` algorithm in details for solving Eq. 6, which is proven to be prohibitively expensive (Section 4.1). Then we focus on how to compute the expectation using possible worlds (Section 4.2) in the algorithm.

4.1 Problem Complexity

Let us first discuss the time complexity of finding the optimum of Eq. 6.

Theorem 1. *The problem of expected optimal coreset selection over incomplete data is NP-hard.*

Proof. Let us consider a special case that there is no missing value in D. Our problem becomes the typical coreset selection problem over complete data, which has been proven to be NP-hard by reduction from the Minimum Vertex Cover problem [32, 58, 59]. Hence, our problem is also NP-hard. □

Theorem 2. *The problem of expected optimal coreset selection over incomplete data has the sub-modular property.*

Proof. First, we regard $\mathrm{E}[C] = \sum_{k=1}^{|\mathcal{I}_W|} p_k S_k$ as a utility function, where $S_k = \sum_{i=1}^{n} \min_{c_j \in C_k} s_{ij}$. In fact, S_k can be regarded as a function of the coreset score computation over complete data, which has already proven to have the sub-modular property [42, 58, 59]. Therefore, consider the property that a non-negative linear combination of sub-modular functions is also sub-modular [53]. To be specific, given any sub-modular function $g_1, g_2, \ldots, g_k$ and non-negative numbers $\alpha_1, \alpha_2, \ldots, \alpha_k$. Then the function $\mathcal{G}$ defined by $\mathcal{G} = \sum_{i=1}^{k} \alpha_i g_i$ is sub-modular. Hence, we can conclude that our studied problem is a sub-modular problem because $\mathrm{E}[C] = \sum_{k=1}^{|\mathcal{I}_W|} p_k S_k$, where $p_k > 0$. □

The greedy algorithm. Given the sub-modular property, naturally, we can design a greedy algorithm with an approximate ratio. As shown in Algorithm 1, we greedily add one tuple to the coreset at each iteration. The added tuple should have the largest utility computed by $\mathrm{E}[t|C] = \mathrm{E}[C] - \mathrm{E}[C \cup \{t\}]$. Hence, the key component is that given the original train data (D) and a coreset (C or $C \cup \{t\}$), how to compute the expectation of GA error ($\mathrm{E}[C]$ or $\mathrm{E}[C \cup \{t\}]$) of the coreset. However, it is non-trivial because of the large number of possible worlds. We will first introduce how to compute the probability p_k, and describe the expectation computation in Section 4.2. After K tuples are added, we can impute missing tuples in the coreset generated by `GoodCore`.

4.2 Expectation Computation

Possible world probability. To compute the expectation, it is inevitable to derive the probability of each possible world, which can be taken as a pre-processing step in our framework. To be specific, since tuples with missing values are always imputed independently [57], given a possible world W_k, the probability p_k can be computed by $p_k = \prod_{t \in W_k, \mathbb{I}[t]=1} p_k^t$, where p_k^t denotes the probability of the appearance of tuple t with $\mathbb{I}[t] = 1$. Besides, apparently $p_k^t = 1$ when $\mathbb{I}[t] = 0$, so $p_k = 1$ if there are only complete tuples. Therefore, our focus is on how to get the value of p_k^t, which can be solved by many approaches, like statistic methods and learning-based methods (see [57] for a survey). In this paper, we use the learning-based method [14] with a Python library [1] to generate the probability, which can be easily replaced by other libraries or domain-specific methods. During training, learning-based methods take D as input and learn a model $\mathcal{M}$ to describe the joint data distribution. For inference, we have $P(\mathcal{A}_i|\mathbf{x}, v_{mask}) = \mathcal{M}(\mathbf{x}, v_{mask}, \omega^*)$, where the model takes as input the feature vector $\mathbf{x}$ of t, the mask vector v_{mask} (indicating which attributes are missing) and the model parameter ω^*, outputs the probability distribution of a missing attribute $\mathcal{A}_i$.

Suppose that t just has one missing attribute $\mathcal{A}_i$, and then v_{mask} is a one-hot vector with $v_{mask}[i] = 0$. Hence, we can directly obtain p_k^t from the distribution $P(\mathcal{A}_i|\mathbf{x}, v_{mask})$. For t with multiple missing attributes, we can also

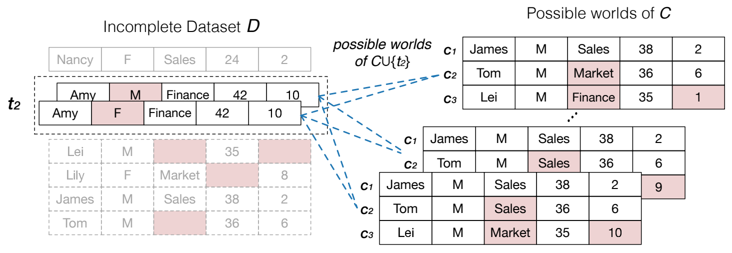


Fig. 7. Tuple-based expectation computation.

compute p_k^t using the chain rule. If t has two missing values of $\mathcal{A}_i$ and $\mathcal{A}_j$, to compute p_k^t, we have to compute $P(\mathcal{A}_i, \mathcal{A}_j|\mathbf{x}, v_{mask})$, abbreviated as $P(\mathcal{A}_i, \mathcal{A}_j) = P(\mathcal{A}_i)P(\mathcal{A}_j|\mathcal{A}_i)$. $P(\mathcal{A}_i)$ can be obtained by masking the i-th and j-th attribute in v_{mask}. Then, we only mask the j-th attribute and impute different values of $\mathcal{A}_i$ to obtain $P(\mathcal{A}_j|\mathcal{A}_i)$.

Example 5. *In Figure 5(a), suppose that for the first possible world, we have to compute* $p_1 = p_1^2 \times p_1^3 \times p_1^4 \times p_1^6$. *For instance, to compute* p_1^3*, given the trained deep learning model, we feed* {Lei, M, Mask, 35, Mask} *and a one-hot vector* $\{1, 1, 0, 1, 0\}$ *into the model and compute the probability distribution of this tuple, from which we can get* p_1^3*, i.e., the probability of* {Lei, M, Sales, 35, 1}.

Compared with statistical approaches, deep learning-based methods use more powerful models with good learning capacity and consider the correlation between attributes. For practitioners, they can use any ad-hoc method to compute the probability.

Brute-force expectation computation. Recap that $\mathrm{E}[C] = \sum_{k=1}^{|\mathcal{I}_W|} p_k(\sum_{i=1}^{n} \min_{c_j \in C_k} s_{ij})$. Intuitively, the brute-force method is to enumerate each possible world, compute the probability and finally get the expectation. However, there are a huge number of possible worlds, which makes the computation prohibitively expensive. Specifically, we assume the attribute number M and $|\mathcal{A}_m|, m \in [1, M]$ are constants, so the number of possible worlds of each tuple is a constant, denoted by L. Suppose that the number of tuples with missing values is $O(n)$, so the number of possible worlds ($|\mathcal{I}_W|$) is $O(L^n)$. Given a coreset C, the time complexity to compute $\mathrm{E}[C]$ is $O(nL^n)$, which is rather expensive.

Tuple-based expectation computation. To further elaborate, we can easily expand $\mathrm{E}[C]$ as follows:

$$\begin{aligned}\mathrm{E}[C] =& p_1(\underset{\sim\sim\sim}{\min_{c_j \in C_1} s_{1j}} + \min_{c_j \in C_1} s_{2j} + \cdots + \min_{c_j \in C_1} s_{nj}) \\ &+ p_2(\underset{\sim\sim\sim}{\min_{c_j \in C_2} s_{1j}} + \min_{c_j \in C_2} s_{2j} + \cdots + \min_{c_j \in C_2} s_{nj}) + \cdots \\ &+ p_{|\mathcal{I}_W|}(\underset{\sim\sim\sim}{\min_{c_j \in C_{|\mathcal{I}_W|}} s_{1j}} + \min_{c_j \in C_{|\mathcal{I}_W|}} s_{2j} + \cdots + \min_{c_j \in C_{|\mathcal{I}_W|}} s_{nj}).\end{aligned}$$

We can see from the above equation that these underlined terms are only related to $t_1 \in D$ as well as $\{C_1, C_2, \cdots, C_{|\mathcal{I}_W|}\}$, *i.e.*, the coresets corresponding to the $|\mathcal{I}_W|$ possible worlds. However, as the coreset C is much smaller than the full data D, the number of possible worlds of C will be also much smaller than $|\mathcal{I}_W|$, and thus there will be many duplicates among $\{C_1, C_2, \cdots, C_{|\mathcal{I}_W|}\}$. Therefore, many of these underlined terms have identical variable parts, *i.e.*, $\min_{c_j \in C_k} s_{1j}$, when they are associated with the same C_k. These terms are *like terms*.

Combining these like terms (*i.e.*, $\sum_{k=1}^{|\mathcal{I}_W|} p_k \min_{c_j \in C_k} s_{1j}$), we can get the expectation of $\min_{c_j \in C} s_{1j}$, denoted by $\mathrm{E}[\min_{c_j \in C} s_{1j}]$.

In short, we can convert the expectation computation over the possible worlds of the entire training set D to the sum of expectation of each tuple in D, as follows:

$$\mathrm{E}[C] = \sum_{k=1}^{|\mathcal{I}_W|} p_k (\sum_{i=1}^{n} \min_{c_j \in C_k} s_{ij}) = \sum_{i=1}^{n} \mathrm{E}[\min_{c_j \in C} s_{ij}] \tag{7}$$

Example 6. *Figure 7 shows how to compute* $\mathrm{E}[\min_{c_j \in C} s_{2j}]$. *Instead of enumerating* $|\mathcal{I}_W|$ *possible worlds by the brute-force method, we can enumerate a much smaller number of possible worlds of* $C \cup t_2$, *compute the corresponding probabilities and finally get the tuple expectation. Specifically, The left part of Figure 7 shows the possible worlds of the tuple, the right part shows the possible worlds of the coreset, and their combination is the possible worlds of* $C \cup t_2$. *Then, following Eq. 7, we can iterate the tuples in D, compute their expectations and sum them up to derive* $\mathrm{E}[C]$.

Time complexity. Since the coreset size is K, and the number of tuples with missing values in the coreset is $O(K)$, the time complexity of computing $\mathrm{E}[C]$ using tuple-based method is $O(nL^K)$, where K is much smaller than n, compared with the brute-force method. However, note that computing $\mathrm{E}[C]$ is just the third loop in the entire framework. Besides, the first two loops incrementally add K tuples into the coreset, and sample h tuples for tuple selection respectively. Hence, the overall time complexity of coreset selection over incomplete data is $O(KhnL^K)$, which is still expensive when K is not small enough. In the next section, we involve the imputation-in-the-loop strategies to achieve further improvement.

5 Optimized Goodcore with Imputation-in-the-loop

As discussed above, it is rather expensive to directly compute all the K tuples in the coreset. Hence, in this section, we propose to involve the imputation-in-the-loop mechanism that asks the human, *i.e.*, Case (7), or automatic method, *i.e.*, Case (8) to impute these missing values iteratively while they are generated by Algorithm 1.

The advantages of this optimization are two-fold. First, with more and more missing values being imputed, the number of possible worlds is greatly reduced, which reduces the machine cost a lot. Second, for human-in-the-loop imputation, it allows us to gradually impute the tuples accurately, and thus the coreset score computation can be more and more accurate, which produces a better coreset.

5.1 One Tuple Each Iteration

In fact, we can just slightly modify Algorithm 1 to achieve the imputation-in-the-loop strategy. To be specific, in the first loop, we will iteratively impute the tuple once an incomplete tuple t^* is computed by `GoodCore`, rather than conducting the imputation after K tuples are computed, as discussed in Section 4. To this end, we move the imputation step (lines 11-12 in Algorithm 1) inside the first loop of Algorthm 1, *i.e.*, imputing each selected t^* by a human or automatic method in each iteration after line 9.

Afterwards, we will add the next tuple into the coreset, so another loop starts and h tuples are sampled. In the following, we will expand the third loop, *i.e.*, the function `ComputeUtility` (line 7) of Algorithm 1 under this one tuple per iteration scenario.

As shown in Algorithm 2, at the beginning, we temporarily add the sampled tuple t to the current coreset, so as to compute the benefit of t, *i.e.*, $\mathrm{E}[t|C]$. To this end, we have to first compute the expectation of GA error bound of $\hat{C}$ (*i.e.*, computing $\mathrm{E}[\hat{C}]$ in the for-loop lines 3-12). And the expectation w.r.t. C (*i.e.*, $\mathrm{E}[C]$) has been computed in the last loop. Then we can compute $\mathrm{E}[t|C] = \mathrm{E}[C] - \mathrm{E}[\hat{C}]$ (line 10).

Algorithm 2: `ComputeUtility` (3rd-loop to compute $\mathrm{E}[t|C]$)

```
Input: Incomplete train data $D$, current coreset $C$, a sampled tuple $t$.
Output: The expectation $\mathrm{E}[t|C]$.
1  $\hat{C} = C \cup \{t\}$;
2  $\mathrm{E}[\hat{C}] = 0$;
3  for each tuple $t_i \in D$ do
4      if $\mathbb{I}[t] = 0$ and $\mathbb{I}[t_i] = 0$ then
5          $\mathrm{E}[\hat{C}]$ += $\min_{c_j \in \hat{C}} s_{ij}$;
6      else
7          Get the possible worlds of $\hat{C} \cup \{t_i\}$;
8          Compute $\mathrm{E}[\min_{c_j \in \hat{C}} s_{ij}]$ using these possible worlds and their probabilities;
9          $\mathrm{E}[\hat{C}]$ += $\mathrm{E}[\min_{c_j \in \hat{C}} s_{ij}]$;
10 $\mathrm{E}[t|C] = \mathrm{E}[C] - \mathrm{E}[\hat{C}]$;
11 return $\mathrm{E}[t|C]$;
```

Specifically, to compute $\mathrm{E}[\hat{C}]$, we will use the tuple-based expectation computation method proposed in Section 4.2. For each tuple $t_i \in D$, if t_i and t are both complete, we can directly compute $\min_{c_j \in \hat{C}} s_{ij}$ because there is no incomplete data in $\hat{C}$ (lines 4-5). Otherwise, we will enumerate the possible worlds of $\hat{C} \cup \{t_i\}$, compute their probabilities and compute $\mathrm{E}[\min_{c_j \in \hat{C}} s_{ij}]$ (lines 7-8). Note that since there are at most two tuples (*i.e.*, t_i and t) have missing values, the number of possible worlds is small because other missing values in $\hat{C}$ have been imputed by humans in previous iterations.

Time complexity analysis. As discussed above, using this human-in-the-loop strategy, the number of possible worlds to be considered is greatly reduced. For Algorithm 2, the time complexity is $O(nL^2)$ because there are at most two incomplete tuples in $\hat{C}$. For the entire three loops framework, the time complexity can be regarded as $O(Khn)$ because L is a constant, which is much lower than the solution without imputation in the loop.

However, if we utilize the human for imputation, the above method will incorporate many human iterations. In the following, we propose to ask human to impute a small batch of missing tuples in each iteration, so as to reduce the number of human iterations.

5.2 One Batch Each Iteration with Human-in-the-loop

In Section 5.1, one tuple per iteration by humans requires many human iterations. However, if we just incorporate a single human iteration like Section 4.2, it is infeasible to compute the tuples to be imputed due to the large number of possible worlds. Therefore, in this subsection, we propose a trade-off solution that asks the human to impute a small batch of tuples per human iteration.

To be specific, as shown in Algorithm 3, compared with the one tuple per human iteration algorithm (*i.e.*, the modified Algorithm 1 at the beginning of Section 5.1), we additionally take the batch size b as input (when $b = 1$, Algorithm 3 is in fact the modified Algorithm 1). Algorithm 3 also incorporates 3 loops, but the main difference is that we do not instantly ask the human to impute the most beneficial tuple t^* among T_{sample}. Instead, we just add t^* into the coreset C (line 7). When there have been b incomplete tuples, we ask the human to impute these tuples together (line 17-19). Finally we compute the weight (line 13), same as Algorithm 1. Although this approach reduces the number of human iterations, it takes a longer time to compute $\mathrm{E}[t|C]$ (line 5) than Algorithm 1 because there are more incomplete tuples, which indicates more possible worlds. Specifically, the time complexity

Algorithm 3: Batch algorithm of `GoodCore`

```
   Input: $D$, $K$, $h$, batch size $b$.
   Output: A coreset $C$, weight $\mathbb{W}$.
1  $C = \emptyset$, $cnt = 0$;
2  while $|C| < K$ do
3      Sample $h$ tuples as $T_{sample} \subseteq D \setminus C$
4      for each tuple $t \in T_{sample}$ do
5          $\mathrm{E}[t|C]$ = ComputeUtility($t, C, D$);
6      $t^* = \arg\max_{t \in T_{sample}} \mathrm{E}[t|C]$ ;
7      $C = C \cup \{t^*\}$;
8      if $\mathbb{I}[t^*] = 1$ then
9          $cnt++$;
10     if $cnt = b$ then
11         Ask the human to impute the incomplete tuples;
12         $cnt = 0$;
13 Compute the weight $\mathbb{W}$.
14 return $C, \mathbb{W}$;
```

of computing $\mathrm{E}[t|C]$ is $O(nL^b)$, which is also expensive. Hence, we propose a heuristic method to accelerate this process as follows.

Reducing the number of possible worlds. A straightforward method of improving the efficiency is to reduce the number of possible worlds. To this end, intuitively, we should focus more on the possible world with a high probability, so these possible worlds with low probabilities can be pruned without sacrificing the accuracy of expectation computation much. Note that for each possible world, the probability is computed by the multiplication of the probabilities of incomplete tuples in the world because the tuples can be considered independent [57]. Therefore, we can remove the possible worlds of each tuple with low probabilities (*i.e.*, reducing L), and thus the number of possible worlds of the entire coreset is greatly reduced. For example, we can keep top-l (*e.g.*, $l = 3$) possible worlds (*i.e.*, 3 different possible imputations of t with high probabilities) of a tuple t. Then for the batch of b incomplete tuples, the number of possible worlds is l^b and the complexity of computing $\mathrm{E}[t|C]$ is $O(nl^b)$, where both l and b are small enough. Therefore, the time complexity is $O(Khn)$. Besides, we can also apply this heuristic method to make the algorithm in Section 4 practical, which is evaluated in Section 7.5.

5.3 Convergence Rate Analysis

Convergence rate is often used to reflect the speed of finding the optimal parameters for the machine learning algorithm. With a higher convergence rate, we can take fewer epochs to make the model converge. To compute the convergence rate, we have to compute the distance between the parameter θ and the optimal parameter θ^* in the t-th and the $(t+1)$-th epoch. Since f is a strongly convex function, $\forall \theta, \theta'$ we have

$$f(\theta) - f(\theta') \geq \nabla f(\theta')(\theta - \theta') + \frac{\eta}{2}\|\theta - \theta'\|^2 \tag{8}$$

where η is a constant. We denote the stepsize as $\zeta_t = \frac{\zeta_0}{k^\tau}$ for the t-th epoch, where τ is a constant. After using gradient descent in each step, we have $\|\theta^{t+1} - \theta^*\|^2 = \|\theta^t - \zeta_k \sum_{j=1}^{|C|} w_j \nabla f_{\gamma(j)}(\theta^t_{j-1}) - \theta^*\|^2$. Then, following Eq. 8, we have

$$\begin{aligned}\|\theta^{t+1}-\theta^*\|^2 \leq \|\theta^t-\theta^*\|^2 - 2\zeta_t\sum_{j=1}^{|C|}(f_j(\theta^t)-f_j(\theta^*))\\ +2\zeta_t\sum_{j=1}^{|C|}(f_j(\theta^t_{j-1})-f_j(\theta^t))+\zeta_t^2\sum_{j=1}^{|C|}\|w_j\nabla f_j(\theta^t_{j-1})\|^2\end{aligned} \tag{9}$$

Recap that we select a coreset that minimizes $\mathrm{E}[C]$ through converting gradient difference to feature distance (s_{ij}) computation. Obviously, given a dataset, s_{ij} can be bounded (suppose that $s_{ij} \leq s_0$). Then we have $\mathrm{E}[\min_{c_j\in C} s_{ij}] = \sum_{k=1}^{|\mathcal{I}_W|} p_k(\min_{c_j\in C_k} s_{ij}) \leq \sum_{k=1}^{|\mathcal{I}_W|} p_k * s_0 = s_0$, and thus $\mathrm{E}[C] = \sum_{i=1}^{n}\mathrm{E}[\min_{c_j\in C} s_{ij}] \leq n * s_0 = \kappa_1$. Besides, we also have $max_{\theta\in\vartheta}\|\sum_{i=1}^{n}\nabla f_i(\theta)-\sum_{j=1}^{|C|}w_j\nabla f_{\gamma(j)}(\theta)\| \leq \sum\limits_{i=1}^{n}\min\limits_{c_j\in C}\|\nabla f_i(\theta)-\nabla f_{\gamma(j)}(\theta)\| \leq \sum\limits_{i=1}^{n}\min\limits_{c_j\in C} s_{ij} \leq \kappa_1$. Following the definition of convex function, we have $f_j(\theta^t)-f_j(\theta^*) \leq w_j\nabla f_j(\theta^*)(\theta^t-\theta^*)+\frac{\eta}{2}\|\theta^t-\theta^*\|^2$. Based on the above things, we can apply Cauchy-Schwarz inequlity [74] and derive

$$\begin{aligned}&-2\zeta_t\sum_{j=1}^{|C|}(f_j(\theta^t)-f_j(\theta^*))\\ &\leq -\eta\zeta_t\|\theta^t-\theta^*\|^2+2\zeta_t\|\sum_{j=1}^{|C|}w_j\nabla f_j(\theta^*)\|\|(\theta^t-\theta^*)\|\\ &\leq -\eta\zeta^t\|\theta^t-\theta^*\|^2+\frac{2\zeta^t|C|\kappa_1\kappa_2}{\eta}\end{aligned} \tag{10}$$

where κ_2 can be regarded as the upper bound of $\|\theta_t-\theta^*\|$. Since f is convex, thus, for item $f_j(\theta^t_{j-1})-f_j(\theta^t)$, we have $f_j(\theta^t_{j-1})-f_j(\theta^t) \leq \|w_j\nabla f_j(\theta^t)\|\zeta_t\sum_{i=1}^{j-1}\|w_i\nabla f_i(\theta^t_{i-1})\|$. In addition, we can assume that $max_{j\in\{1,\cdots,|C|\}}\|\nabla f_j(\theta)\| \leq \kappa_3$. Then, we have

$$\begin{aligned}&2\zeta_t\sum_{j=1}^{|C|}(f_j(\theta^t_{j-1})-f_j(\theta^t))+\zeta_t^2\sum_{j=1}^{|C|}\|w_j\nabla f_j(\theta^t_{j-1})\|^2\\ \leq&2\zeta_t\sum_{j=1}^{|C|}\|w_j\nabla f_j(\theta^t)\|\zeta_t\sum_{i=1}^{j-1}\|w_i\nabla f_i(\theta^t_{i-1})\|+\zeta_t^2\sum_{j=1}^{|C|}\|w_j\nabla f_j(\theta^t_{j-1})\|^2\\ \leq&2\zeta_t^2(|C|^2-|C|)w_{max}^2\kappa_3^2+\zeta_t^2|C|w_{max}^2\kappa_3^2\end{aligned} \tag{11}$$

Thus, from Eq. 9 to Eq. 11, we can get

$$\|\theta^{t+1}-\theta^*\| \leq (1-\eta\zeta_t)\|\theta^t-\theta^*\|^2+\frac{2\zeta_t|C|\kappa_1\kappa_2}{\eta}+\zeta_t^2|C|^2w_{max}^2\kappa_3^2 \tag{12}$$

Finally, following Lemma 4 in [25], the convergence rate of Algorithm 1 is at the same rate of $O(\frac{1}{\sqrt{k}})$ as the convergence rate on the entire dataset [64]. Therefore, theoretically, the selected coreset can converge with the same number of epochs as training on the full data. In this way, since coreset has a much smaller size than the full data, the efficiency can be much improved.

6 GoodCore$^+$: Group-based Acceleration

As discussed above, we can observe that in Section 5.1, even with the most efficient imputation-in-the-loop strategy, *i.e.*, one tuple in each iteration, the time complexity is $O(Khn)$, where K is the size of the coreset, h is the sample size, and n is the cardinality of the entire dataset. Therefore, obviously, the efficiency is dominated by n, which is still low when n is large, and thus it is necessary to further accelerate this process.

Key observation. Recap that in Figure 4, we can observe that given a tuple c in the coreset, the tuples in the origin full train set D represented by c are likely to be closer to each other than other tuples not represented by c. Based on this observation, we propose to first group the full train set, and then compute the coreset based on the groups. This can achieve much acceleration because the number of groups is much smaller than n.

At the following, we will theoretically and empirically show that the group-based solution can accelerate the coreset selection process without sacrificing the effectiveness much.

6.1 Solution Overview

One of the core parts of coreset computation is to compute the tuple-tuple distance, *i.e.*, s_{ij}. For the group-based solution, we just need to consider the relationship between tuples and these pre-computed groups, namely tuple-group distance, rather than the large amount of tuple-tuple distances. As we will discuss below, the computation of tuple-group distance does not need to iterate all tuples in the group, and thus the overall efficiency can be much improved. At a high level, the overall process of group-based `GoodCore` solution with imputation in the loop is shown in Algorithm 4.

To be specific, as shown in Algorithm 4 Line 2, we first group D using the efficient local sensitive hash (LSH) approach, where each group $G_u, u \in [1, U]$ includes the indexes of tuples in D. In this way, every pair of tuples in the same group is close to each other in the feature distance. Afterwards, the major difference between group-based `GoodCore` and original `GoodCore` lies in the 3rd loop. Instead of selecting a coreset to represent all tuples in the train set, group-based `GoodCore` selects a coreset to represent all groups. As these groups can well capture the train set distribution, the selected coreset contains enough information to approximate the full gradient of the entire train set.

To this end, recap that the typical coreset selection algorithm relies the tuple-tuple distances to approximate the full gradient, while for group-based `GoodCore`, we just need to consider the tuple-group distances, *i.e.*, $\bar{s}_{\gamma(j)u} = \max_{v \in G_u} s_{\gamma(j)v}, s_{\gamma(j)v} = \|\mathbf{x}_v - \mathbf{x}_{\gamma(j)}\|, \gamma(j) \in [1, n]$, which denotes the maximum feature distance between the tuple c_j in the coreset and all tuples in G_u. As tuples in G_u are close to each other, $\bar{s}_{\gamma(j)u}$ can represent the relationship between c_j and tuples in G_u to a large extent. We will theoretically show that using this maximum distance can still derive a bounded GA error. However, since computing $\bar{s}_{\gamma(j)u}$ needs to iterate the tuples in G_u, which is time-consuming, we finally estimate an upper bound $\hat{s}_{ju}$ to compute the coreset score (as shown in Line 11), which still leads to a well-performed coreset.

6.2 Group-based GA Error Bound

In this section, following the equations in previous sections, we deduce the GA error bound for our group-based solution. If we group D to $\{G_1, G_2, ...G_U\}$, considering Equation 7, we can rewrite the total sum of n feature distances (*i.e.*, $\min_{c_j \in C_k} \|\mathbf{x}_i - \mathbf{x}_{\gamma(j)}\|$) to U summations as follows:

$$\mathrm{E}[C] = \sum_{k=1}^{|\mathcal{I}_W|} p_k \Big(\sum_{i=1}^{n} \min_{c_j \in C_k} \|\mathbf{x}_i - \mathbf{x}_{\gamma(j)}\|\Big) = \sum_{k=1}^{|\mathcal{I}_W|} p_k \Big(\sum_{u=1}^{U} \sum_{v \in G_u} \min_{c_j \in C_k} \|\mathbf{x}_v - \mathbf{x}_{\gamma(j)}\|\Big) \tag{13}$$

Afterwards, each summation is the sum of U feature distances, as shown in Equation 14, the sum of each group can be bounded by the maximum distance ($\max_{v \in G_u} \min_{c_j \in C_k} s_{\gamma(j)v}$) multiplying the group size, but the bound is expensive to compute because of iterating D. To address this, we further apply the max-min inequality [16] to simplify the computations.

Algorithm 4: GoodCore$^+$ (imputation-in-the-loop by humans)

```
Input: Incomplete train data $D$, coreset size $K$, sample size $h$, batch size $b$.
Output: A coreset $C \subseteq D$, weight $\mathbb{W} = \{w_j\}, |C| = |\mathbb{W}| = K$.
1  $C = \emptyset$;
2  Group $D$ into groups $\mathcal{G} = \{G_1, G_2, ..., G_U\}$;
3  while $|C| < K$ do
4      /*1st loop*/
5      Sample $h$ tuples as $T_{sample} \subseteq D \setminus C$
6      for each tuple $t \in T_{sample}$ do
7          /*2nd loop*/
8          $\hat{C} = C \cup \{t\}$;
9          for each group $G_u \in \mathcal{G}$ do
10             /*3rd loop*/
11             $E[\hat{C}]$+= $E[\min_{c_j \in \hat{C}} \hat{s}_{ju} \times |G_u|]$, where $\hat{s}_{ju}$ is the estimated upper bound of
12                 $\bar{s}_{\gamma(j)u} = \max_{v \in G_u} s_{\gamma(j)v}, s_{\gamma(j)v} = \|\mathbf{x}_v - \mathbf{x}_{\gamma(j)}\|, \gamma(j) \in [1, n]$;
13         $E[t|C] = E[C] - E[\hat{C}]$;
14     $t^* = \arg\max_{t \in T_{sample}} E[t|C]$ ;
15     if $\mathbb{I}[t^*] = 1$ then
16         $cnt$ ++;
17     if $cnt = b$ then
18         Ask the human to impute the incomplete tuples;
19         $cnt = 0$;
20 for $j = 1$ to $|C|$ do
21     for $i = 1$ to $n$ do
22         if $c_j = \arg\min_{c_{j'} \in C} \max_{\theta \in \vartheta} \|\nabla f_i(\theta) - \nabla f_{\gamma(j')}(\theta)\|$ then
23             $w_j$ += 1;
24 return $C, \mathbb{W}$;
```

$$\begin{aligned}
\sum_{k=1}^{|\mathcal{I}_W|} p_k \Big(\sum_{u=1}^{U} \sum_{v \in G_u} \min_{c_j \in C_k} s_{\gamma(j)v}\Big) &\le \sum_{k=1}^{|\mathcal{I}_W|} p_k \Big(\sum_{u=1}^{U} |G_u| \max_{v \in G_u} \min_{c_j \in C_k} s_{\gamma(j)v}\Big) \\
&\le \sum_{k=1}^{|\mathcal{I}_W|} p_k \Big(\sum_{u=1}^{U} |G_u| \min_{c_j \in C_k} \max_{v \in G_u} s_{\gamma(j)v}\Big) \\
&= \sum_{k=1}^{|\mathcal{I}_W|} p_k \Big(\sum_{u=1}^{U} |G_u| \min_{c_j \in C_k} \bar{s}_{\gamma(j)u}\Big)
\end{aligned} \tag{14}$$

Therefore, we can iterate over smaller group G to compute the maximum feature distance (*i.e.*, $\bar{s}_{\gamma(j)u} = \max_{v \in G_u} s_{\gamma(j)v}, j \in [1, |C_k|]$) between each $c_j \in C_k$ and tuples in each group G_u. Then, similar to assigning tuples of the full train set to the tuple of the coreset in previous sections, we can assign the group G_u to the tuple with the minumum distance, *i.e.*, $\min_{c_j \in C_k} \bar{s}_{\gamma(j)u}$. To enable efficient coreset selection, given D and these groups $\mathcal{G}$, we should precompute all the maximum feature distances $\{\overline{s}_{Y(j)u} | j \in [1,n], u \in [1,U]\}$. In this way, we can directly get the value of $\overline{s}_{Y(j)u}$.

Similar to Sec 4.2, directly computing the probability and getting the expectation is extremely expensive, and thus we still can convert the expectation computation over the possible worlds associated with all groups to the sum of expectation of each group, as follows:

$$\begin{aligned}\sum_{k=1}^{|\mathcal{I}_W|} p_k \Big(\sum_{u=1}^{U} |G_u| \min_{c_j \in C_k} \overline{s}_{Y(j)u}\Big) = {} & \underline{p_1\big(\, |G_1| \min_{c_j \in C_1} \overline{s}_{Y(j)1}} + |G_2| \min_{c_j \in C_1} \overline{s}_{Y(j)2} + \cdots + |G_u| \min_{c_j \in C_1} \overline{s}_{Y(j)u}\big) \\ & + \underline{p_2\big(\, |G_1| \min_{c_j \in C_2} \overline{s}_{Y(j)1}} + |G_2| \min_{c_j \in C_2} \overline{s}_{Y(j)2} + \cdots + |G_u| \min_{c_j \in C_2} \overline{s}_{Y(j)u}\big) + \cdots \\ & + \underline{p_{|\mathcal{I}_W|}\big(\, |G_1| \min_{c_j \in C_{|\mathcal{I}_W|}} \overline{s}_{Y(j)1}} + |G_2| \min_{c_j \in C_{|\mathcal{I}_W|}} \overline{s}_{Y(j)2} + \cdots + |G_u| \min_{c_j \in C_{|\mathcal{I}_W|}} \overline{s}_{Y(j)u}\big) = \sum_{u=1}^{U} |G_u| \times \mathrm{E}[\min_{c_j \in \hat{C}} \overline{s}_{Y(j)u}].\end{aligned} \tag{15}$$

6.3 Technical Details of GoodCore^{+}

6.3.1 Grouping. As discussed in Algorithm 4 Line 2, we need to group the entire train set as a pre-processing step. To achieve this efficiently, we adopt locality sensitive hashing (LSH) [10] to assign similar tuples to the same group, with a time complexity linear with $|D|$. Note that D contains some tuples with missing values, an ideal way is to first impute these tuples precisely and then group, but we do not know the ground truth in advance. Therefore, we just apply a typical algorithm *i.e.*, MICE [68] to impute these missing values, and then conduct the grouping. Although the imputation results may not be accurate enough, it does not influence much because we just need closer tuples to be included in the same group, and these missing cells do not have a large impact on determining whether two tuples are close. Finally, tuples with the same hash code are considered highly similar and grouped together.

6.3.2 Computing the expected maximum distance. In this part, we focus on computing Equation 15, where the key part is the expected maximum distance, *i.e.*, $\mathrm{E}[\min_{c_j \in \hat{C}} \overline{s}_{Y(j)u}]$. To be specific, we expand $\mathrm{E}[\min_{c_j \in \hat{C}} \overline{s}_{Y(j)u}] = q_1 \min_{c_j \in \hat{C}} \overline{s}_{Y(j)u}[1] + q_2 \min_{c_j \in \hat{C}} \overline{s}_{Y(j)u}[2] + \ldots + q_{card(u)} \min_{c_j \in \hat{C}} \overline{s}_{Y(j)u}[card(u)]$, where $card(u)$ denotes the number of possible world of $G_u \cup \hat{C}$, $q_x(x \in [1, card(u)])$ denotes the probability of the $x-$th possible world and $\overline{s}_{Y(j)u}[x]$ denotes the maximum distance corresponding to the $x-$th possible world (each possible world does not contain missing values). Therefore, to compute the expected maximum distance $\mathrm{E}[\min_{c_j \in \hat{C}} \overline{s}_{Y(j)u}]$, we should know how to compute $\overline{s}_{Y(j)u}[x]$, *i.e.*, the maximum feature distance between a tuple c_j and a group G_u within a possible world.

Estimating an upper bound for each possible world. For ease of representation, we just use $\overline{s}_{ju}$ to represent $\overline{s}_{Y(j)u}[x]$, indicating the maximum feature distance of a possible world. Recap that the reason why we do not directly compute the $\overline{s}_{Y(j)u}$ is that iterating G_u to compute the maximum distance is expensive. To solve this, we propose to leverage the quantization technique to estimate an upper bound $\hat{s}_{ju}$ of $\overline{s}_{ju}$, and then use $\hat{s}_{ju}$ to compute the coreset score that still very likely leads to a bounded GA error.

Basic idea. At a high level, if we consider each tuple individually, it is time-consuming as discussed above. However, if we take all tuples in a group as a whole, we cannot distinguish these tuples from each other and the maximum distance is impossible to estimate. Therefore, we propose a more refined method that partitions the m-dimensional feature space into M low-dimensional subspaces, and then quantize each subspace separately. The quantization is conducted by applying k-means algorithm [36] over the vectors in each subspace, where R clusters are generated.

In this way, a short code will represent a feature vector, where the z-th element corresponds to the quantization index (*i.e.*, cluster ID) of the z-th subspace, and thus the short codes of two vectors can be used to efficiently estimate their Euclidean distance. In our scenario, we use the short codes to efficiently estimate an upper bound. To be specific, we also split $\mathbf{x}_j$ of c_j into M subvectors, each of which is represented as $\mathbf{x}_j^z$. If we can respectively compute the maximum distance (denoted by $\overline{s}_{ju}^z$) between the z-th subvector and vectors in the z-th subspace of G_u, $z \in [1, M]$, and sum them up, we can derive an upper bound between c_j and G_u, *i.e.*, $\overline{s}_{ju} \leq \sum_{z=1}^{M} \overline{s}_{ju}^z$.
<u>*Computing* $\hat{s}_{ju}$.</u> As discussed before, directly computing $\overline{s}_{ju}^z$ is time-consuming, so we leverage these clusters in each subspace to represent all the vectors in the z-th subspace.

Specifically, we use $\{r_1^1, r_1^2, ..., r_1^R\}$ to represent the cluster centers in the first subspace. Hence, we can build a matrix mr_1 to store the feature distances (denoted by $mr_1[x][y]$) between every two cluster centers, in total M matrices are built. In this way, we can quantize each $t_i(\mathbf{x}_i) \in \mathcal{T}$ to a short code d_i, where $d_i^z, z \in [1, M]$ denotes the z-th element, indicating that the d_i^z-th center has the shortest distance with $\mathbf{x}_i^z$ among all clusters of the z-th subspace. Then, the feature distance between t_i and c_j can then be approximated by $\sum_{z=1}^{M} mr_z[d_i^z][d_j^z]$.

Given c_j and G_u, we approximate $\overline{s}_{ju}^z$ by first quantizing $\mathbf{x}_j$ and $\forall \mathbf{x} \in G_u$ to short codes. Based on these matrices, for the zth subspace, we calculate the maximum distance between the code corresponding to c_j and codes of tuples in G_u, i.e., $\hat{s}ju^z = \max v \in G_u mr_z[d_j^z][d_v^z]$ as the approximation. Then, we approximate the upper limit by summing up the M distances $\hat{s}_{ju} = \sum_{z=1}^{M} \hat{s}_{ju}^z$. Although $\hat{s}_{ju}^z$ may slightly underestimate $\overline{s}_{ju}^z$ due to the quantization bias, the summation $\hat{s}_{ju}$ always overestimates $\overline{s}_{ju}$ since each $\hat{s}_{ju}^z$ is close to $\overline{s}_{ju}^z$, and thus the GA error can be always bounded.

Reducing the number of possible worlds. Recap that $\mathrm{E}[\min_{c_j \in \hat{C}} \hat{s}_{ju}] = \sum_{x=1}^{card(u)} q_x \min_{c_j \in \hat{C}} \hat{s}_{ju}[x]$. Hence, since each group contains multiple tuples, possibly multiple missing values, it is expensive to enumerate $card(u)$ possible worlds and to get the expectation. To address this, we can reduce the number (L) of possible worlds of each tuple, and just keep several top possible worlds, say l, with the highest probabilities as discussed before. Next, suppose that there are $y+1$ tuples (y tuples in G_u and one tuple in $\hat{C}$) with missing values, and thus there exist l^{y+1} possible worlds for $G_u \cup \hat{C}$. To achieve further acceleration, we can just select top-l_g (*e.g.*, 3) possible worlds and normalize them to compute the expectation. Similarly, the above framework is easy to generalize to the scenario of one batch per iteration, where the only difference is that in the coreset, we have a small batch b of tuples with missing values rather than just one, indicating that the number of possible worlds that should be considered increases.

Furthermore, we can continue to reduce the number of possible world considering the following entropy-based method. Considering a missing cell value with 3 possible values to be imputed, if the probability distribution predicted by an imputation algorithm is (0.8, 0.1, 0.1), *i.e.*, with a low entropy, it is certain enough to directly impute the value corresponding to the probability 0.8 rather than considering the possible worlds of this value. Formally, we use $H(X) = -\sum_{i=1}^{n_X} p(x_i) \log p(x_i)$ to denote the entropy [70] of a cell value X, where n_x denotes the number of possible values of X, and $H(X) \in [0, \log n_x]$. Therefore, we can set a threshold (*e.g.*, $10\% \log n_x$) that if $H(X) \leq 10\% \log n_x$, we directly impute the values. In this way, the number of possible world can be further reduced.

Time complexity analysis. For the quantization step, we can sample a small subset to compute the clusters in each subspace, and thus the time complexity of this part can be ignored. Then the matrices can be computed in $O(mR^2)$ and the short codes of D can be computed in $O(mnR)$. As discussed above, the largest distance between corresponding codes of c_j and codes of tuples in G_u can be computed by $\hat{s}_{ju}^z = \max_{v \in G_u} mr_z[d_j^z][d_v^z]$ in the z-th subspace. As d_j^z only takes from R different values $\{1, 2, \ldots, R\}$, we can precompute $\max_{v \in G_u} mr_z[i][d_v^z], i \in [1, R]$ for each G_u, which takes $O(UR^2)$. In this way, we can compute each $\hat{s}_{ju} = \sum_{z=1}^{M} \hat{s}_{ju}^z$ in $O(M)$, and considering that

Table 1. Statistics of datasets

Dataset	$\|D\|$	m	# **Incomp. Tuples**	**Task**
`Nursery`	10960	9	3218	Multi-Class.
`HR`	18287	12	5475	Binary Class.
`Adult`	32842	14	10752	Binary Class.
`Credit`	131,000	11	76813	Binary Class.
`BikeShare`	13300	15	4821	Regression
`Air`	437,200	18	128,372	Regression
`IMDB`	1,000,000	40	331,189	Multi-Class.
`IMDB-Large`	4,000,000	40	1,312,908	Multi-Class.

l, l_g and b are all small constants, all the upper bounds $\hat{s}_{ju}$, $j \in [1,n]$, $u \in [1,U]$ can be computed in $O(MnU)$, which is much faster than $O(mn^2)$ that enumerates every tuple and tuples in each group to compute the upper bounds, because $m \ll M$ and $U \ll n$. Then, considering the three-loop framework, the overall time complexity is $O(KhU)$, which is much faster than the $O(Khn)$ because $U \ll n$.

Convergence analysis. Considering the proof in Section 5.3, obviously, given a dataset, $\bar{s}_{\gamma(j)u}$ can be bounded (suppose that $\bar{s}_{\gamma(j)u} \le s_0$). Then we have $\mathrm{E}[\min_{c_j \in \hat{C}} \bar{s}_{\gamma(j)u}] = \sum_{x=1}^{card(u)} q_x \min_{c_j \in \hat{C}} \bar{s}_{ju}[x] \le \sum_{x=1}^{card(u)} q_x * s_0 = s_0$, and thus $\mathrm{E}[C] = \sum_{u=1}^{U} |G_u| \times \mathrm{E}[\min_{c_j \in \hat{C}} \bar{s}_{\gamma(j)u}] \le n * s_0 = \kappa_1$. And we also have $max_{\theta \in \vartheta} \| \sum_{i=1}^{n} \nabla f_i(\theta) - \sum_{j=1}^{|C|} w_j \nabla f_{\gamma(j)}(\theta) \| \le \sum\limits_{i=1}^{n} \min\limits_{c_j \in C} \| \nabla f_i(\theta) - \nabla f_{\gamma(j)}(\theta) \| \le \sum\limits_{i=1}^{n} \min\limits_{c_j \in C} s_{ij} \le \kappa_1$. Then we can still apply Cauchy-Schwarz inequlity [74] to justify the convergence of the group-based method following the proof in Section 5.3.

7 Experiment

In this section, we sufficiently compare our proposed methods with multiple baselines on real datasets to demonstrate our effectiveness and efficiency.

7.1 Experimental Settings

Dataset. We evaluate on 6 real-world datasets that are often used in the field of data imputation [40, 48, 52, 54], as shown in Table 1, where M denotes the number of attributes.

(1) `Nursery` [2] is a multi-classification task, which predicts "*the level of recommendation for whether a child goes to school*". There are five different levels, *i.e.*, {`not_recom`, `priority`, `recommend`, `spec_prior`, `very_recom`}. **(2)** `HR` [20] is a binary classification task of "*predicting whether an employee would change the job*". **(3)** `Adult` [3] is a binary classification task that predicts "*if the annual revenue of a people is over 50000 dollars*". **(4)** `Credit` [4] is a binary classification task that predicts "*whether the loan will be deferred based on a person's economic situation*". **(5)** `BikeShare` [5] is a regression task that predicts "*the number of bike sharing in a given time*". **(6)** `Air` [6] is a regression task that predicts "*the air quality at a certain time*". **(7)** `IMDB` [50] refers to a dataset that *predicts the rating (1-10) of movies*, which contains the basic information of movies, *e.g.*, `movie_id, title, production_year`. **(8)** `IMDB-Large` [50] is the large vision of `IMDB`, which contains 4,000,000 records with the same attributes.

For datasets (1)-(3) and (7)-(8), we follow existing works [41, 77, 78] to manually inject missing values until the rate of missing tuples is 30%, and we will vary the percentage of incomplete tuples in Section 7.8. Datasets (4)-(6) already contain missing values. For all datasets, we randomly split them for 80%/10%/10% as train/validation/test sets.

Evaluation metrics. We mainly evaluate the effectiveness and efficiency of GoodCore and baselines. For effectiveness, we use the *prediction accuracy* for the classification task and use the *mean square error* ($MSE = \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)}{N}$, where N denotes the size of test set) for the regression task.

For efficiency, we focus on the machine cost (*i.e.*, the runtime of machine) as well as the human cost (the number of tuples imputed by human for human-involved methods). For datasets (1)-(3) and (7)-(8), we have the ground truth of missing tuples, so we use them to simulate the human imputation. For datasets (4)-(6), we leverage the expert to impute missing values in the coreset by looking at the top-5 values recommended by the automatic method as a reference. Note that we only involve humans when it is affordable. For baselines that require humans to impute a lot of missing tuples (*i.e.*, `Complete` and $\mathbf{C}(\mathbf{H}(D))$ as below), we will not apply them on datasets (4)-(6).

Baselines. We compare GoodCore and GoodCore $^+$ with a variety of baselines.

(1) `Origin` refers to just training on D.

(2) `ActiveClean` [48] is an iterative data cleaning framework, which estimates the impact of tuples and prioritizes cleaning the tuples that much affect the model performance. In each iteration, it can ask the human to clean a sample subset of tuples. We set the sample size to 50, same as the paper.

(3) `BoostClean` [49] is an automatic data cleaning method that iteratively selects a cleaning method from several pre-defined algorithms, applies to the train dataset and updates the model. We use MICE [68], MISSForest [73], GAIN [78] as pre-defined algorithms.

(4) `Best-Auto` uses MICE [68], MISSForest [73], GAIN [78] to respectively impute the train set and selects the one that achieves the highest accuracy on the validation set.

(5) `Complete` is an ideal case that trains on the ground truth, *i.e.*, D_c. Note that only datasets (1)-(3) have the ground truth to evaluate this baseline. Datasets (4)-(6) do not have the ground truth and it is too expensive to ask the human to impute so many missing values.

(6) `MixCore` is a baseline that selects a coreset from all complete tuples, and then we randomly select some incomplete tuples to impute. We set the number of incomplete tuples to be imputed equal to that of other baselines for fair comparison. Finally we train with the tuples in the coreset plus the imputed ones.

(7) $\mathbf{C}(\mathbf{H}(D))$ first involves human to impute the dataset D and then selects a coreset. Similar to `Complete`, only datasets (1)-(3) can be evaluated on it because they have the ground truth. The coreset selection solution is the algorithm in [58], which is a greedy algorithm by modifying Algorithm 1 without considering the possible worlds.

(8) $\mathbf{C}(\mathbf{A}(D))$ first uses automatic data imputation methods to impute the dataset D, and then selects a coreset using the same method of baseline (7).

(9) $\mathbf{H}(\mathbf{C}(D))$ directly selects a coreset based on D and then asks human to impute the incomplete tuples of the coreset.

(10) $\mathbf{A}(\mathbf{C}(D))$ also directly selects a coreset from D, it then uses MICE [68] to impute the incomplete tuples in the coreset.

Our solutions. We compare GoodCore and its variants.

(11) $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$ uses GoodCore to select the coreset and iteratively asks human to impute incomplete tuples (one tuple per human iteration) during the coreset selection process.

(12) $\mathbf{G}(D, \circlearrowleft^{\mathbf{A}})$ is similar to $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$, but the automatic MICE method is used.

(13) $\mathbf{G}^{+}(D, \circlearrowleft^{\mathbf{H}})$ uses group-based method to accelerate GoodCore, which selects the coreset and iteratively asks human to impute incomplete tuples (one tuple per human iteration) during the coreset selection process.

(14) $\mathbf{G}^{+}(D, \circlearrowleft^{\mathbf{A}})$ is similar to $\mathbf{G}^{+}(D, \circlearrowleft^{\mathbf{H}})$, but the automatic MICE method is used.

Besides, since the coreset of $\mathbf{H}(\mathbf{G}(D))$ (or $\mathbf{A}(\mathbf{G}(D))$) is too expensive to compute due to the large number of possible worlds, we do not directly compare with it. Instead, we will limit the number of possible worlds of each tuple to 3 as discussed in Section 5.2 and evaluate in Section 7.5.

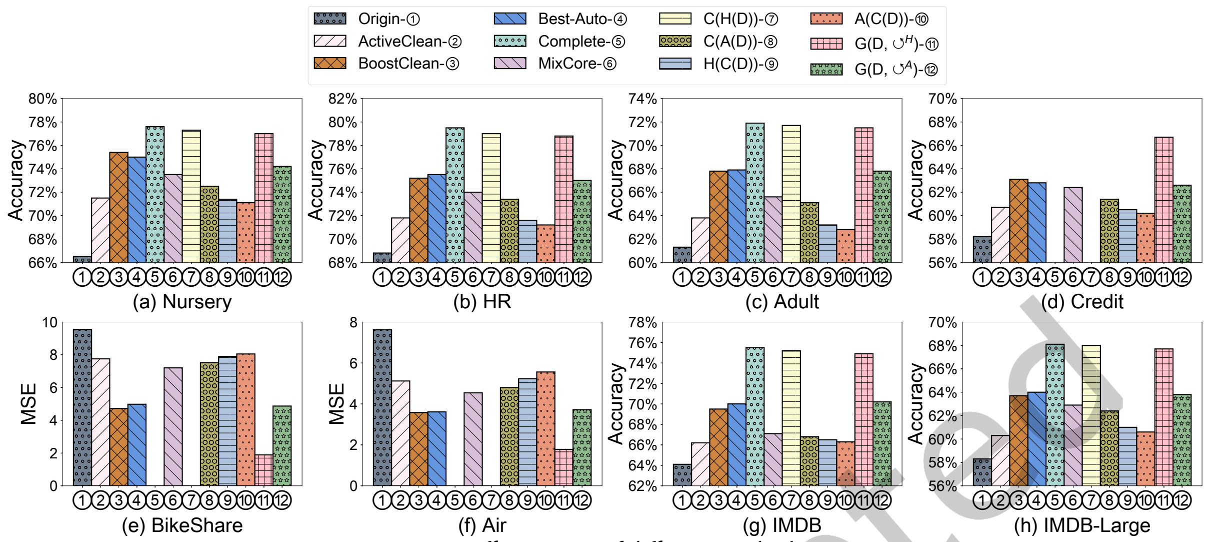


Fig. 8. Effectiveness of different methods.

Hyper-parameter setting. We use SVM and linear regression as the default downstream model for classification and regression tasks, respectively. We vary the downstream models in Section 7.8. For model training, we use SGD and k-inverse decay scheduling, *i.e.*, $\alpha_k = \alpha_0/(1+bk)$ (α_0 and b are hyper-parameters to be tuned independently for different methods). The sample size h is set to 200 as default and we vary the size in Section 7.8. The number of training epochs is set as 20. We also impute the test data using the same method that is applied to the train data before testing.

7.2 Overall Evaluation

In this part, we compare GoodCore solutions with baselines. We use $\rho = \frac{K}{|D|}$ to denote the proportion of coreset to the entire train set. We set $\rho = 0.005$ for datasets (1)-(4), $\rho = 0.001$ for datasets (5) and $\rho = 0.0005$ for larger datasets (6)-(8). We will further conduct evaluation by varying the coreset size in Section 7.4.

7.2.1 Evaluation of model accuracy. The results are provided in Figure 8. To summarize, the result could be generally ranked as $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$/$\mathbf{C}(\mathbf{H}(D))$/Complete > $\mathbf{G}(D, \circlearrowleft^{\mathbf{A}})$/BoostClean/Best − Auto > $\mathbf{C}(\mathbf{A}(D))$ > MixCore > ActiveClean > $\mathbf{H}(\mathbf{C}(D))$/$\mathbf{A}(\mathbf{C}(D))$ > Origin. Next, we explain the results.

In general, on all datasets, our method $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$, Complete and $\mathbf{C}(\mathbf{H}(D))$ perform the best. Complete and $\mathbf{C}(\mathbf{H}(D))$ achieve a high accuracy because they ask the human to impute missing values accurately, but incur a high human cost. For example, Complete and $\mathbf{C}(\mathbf{H}(D))$ achieve accuracy of 71.9% and 71.7% on Adult. $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$ is competitive with them because it selects a good coreset that can well represent the unknown ground truth via gradient approximation. In addition, we can observe that $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$ performs better than $\mathbf{G}(D, \circlearrowleft^{\mathbf{A}})$ because human imputation is more accurate than automatic methods. For example, on Adult, $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$ has an accuracy of 71.7%, while $\mathbf{G}(D, \circlearrowleft^{\mathbf{A}})$ and others are below 68%. $\mathbf{G}(D, \circlearrowleft^{\mathbf{A}})$, BoostClean and Best-Auto have competitive performance on accuracy. BoostClean and Best-Auto can have a not bad performance because they impute all tuples and train on the entire dataset, but they cannot achieve efficient training (see 7.2.2). But we can train on the much smaller coreset generated by $\mathbf{G}(D, \circlearrowleft^{\mathbf{A}})$ with a good accuracy, because GoodCore considers the possible repairs to derive the coreset that can approximate the full gradient of the entire dataset. Given the same number of tuples to be imputed by human, $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$ also outperforms ActiveClean because we have theoretical guarantees on the gradient approximation. For other baselines, $\mathbf{H}(\mathbf{C}(D))$ and $\mathbf{A}(\mathbf{C}(D))$ do not perform well

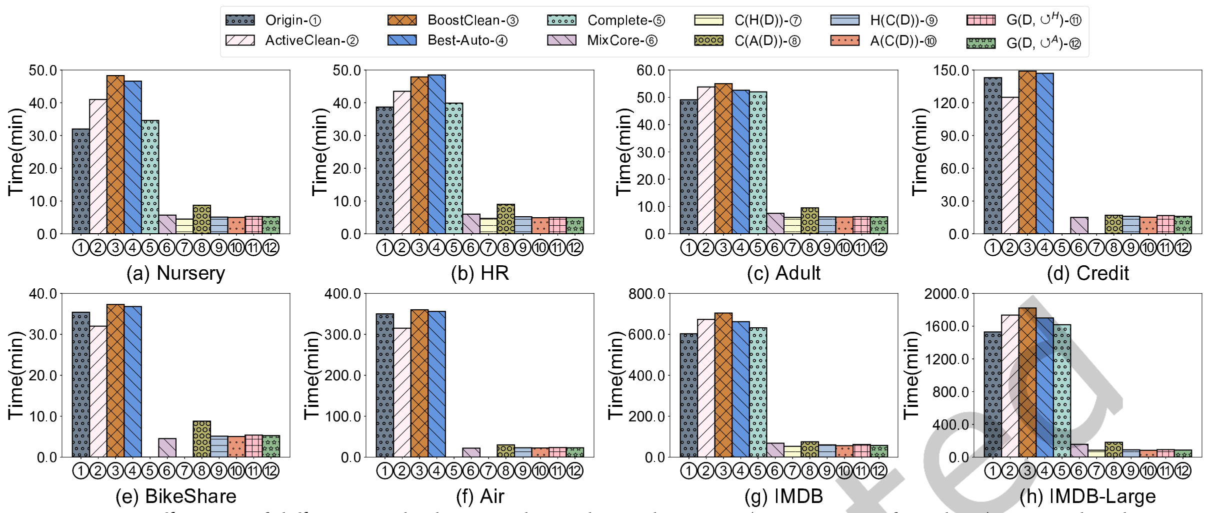


Fig. 9. Efficiency of different methods. Note that only machine cost (i.e., runtime of machine) is considered.

Table 2. Human cost of different methods

Dataset	$\mathbf{G}^+(D, \circlearrowleft^{\mathbf{H}})$	$\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$	$\mathbf{H}(\mathbf{C}(D))$	$\mathbf{C}(\mathbf{H}(D))$
Nursery	35	37	22	3278
HR	48	44	32	5475
Adult	60	63	81	10752
Credit	57	52	67	-
BikeShare	35	38	25	-
Air	100	98	102	-
IMDB	215	220	230	331189
IMDB-Large	520	511	530	1312908

because they select the coreset from an incomplete dataset. $\mathbf{C}(\mathbf{A}(D))$ cannot achieve a good performance because the selected coreset can not well represent the complete entire dataset, as it does not consider possible repairs as our method. MixCore does not perform well (*e.g.*, 65.2% on Adult) because $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$ and $\mathbf{G}(D, \circlearrowleft^{\mathbf{A}})$ select a better coreset considering the full data. For Origin, on Adult, the model has an accuracy of 61.3% because of the incomplete tuples.

7.2.2 Evaluation of the efficiency. We evaluate the efficiency of all methods, including the machine cost and human cost.

Machine cost. Machine cost is shown in Figure 9. The results could be ranked as $\mathbf{H}(\mathbf{C}(D))/\mathbf{A}(\mathbf{C}(D))/\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})/\mathbf{G}(D, \circlearrowleft^{\mathbf{A}})/\mathbf{C}(\mathbf{H}(D))/$MixCore $<$ $\mathbf{C}(\mathbf{A}(D))$ $<$ Complete $<$ Origin $<$ ActiveClean $<$ BoostClean/Best − Auto. We can observe that the first 5 methods in the ranking have low machine cost, mainly because they train based on the selected coreset and do not need iterative training. $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$ and $\mathbf{G}(D, \circlearrowleft^{\mathbf{A}})$ are slightly slower because they need to iterate several possible repairs during the process of coreset selection. But $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$ is still more efficient than Origin, Complete, BoostClean and Best-Auto by more than one order of magnitude, because they need to train on the entire training data. Moreover, ActiveClean and BoostClean are not efficient either because they incorporate multiple training times, so as to estimate the gradient while data imputation. Best-Auto is slow because training multiple imputation models takes time.

Human cost. In terms of the human cost, $\mathbf{C}(\mathbf{H}(D))$, $\mathbf{H}(\mathbf{C}(D))$, $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$ and ActiveClean involve human. As shown in Table 2, $\mathbf{C}(\mathbf{H}(D))$ is very expensive because it asks the human to impute all missing tuples. For example, on datset Adult, 10752 tuples have to be imputed. We do not compare Credit , BikeShare and Air

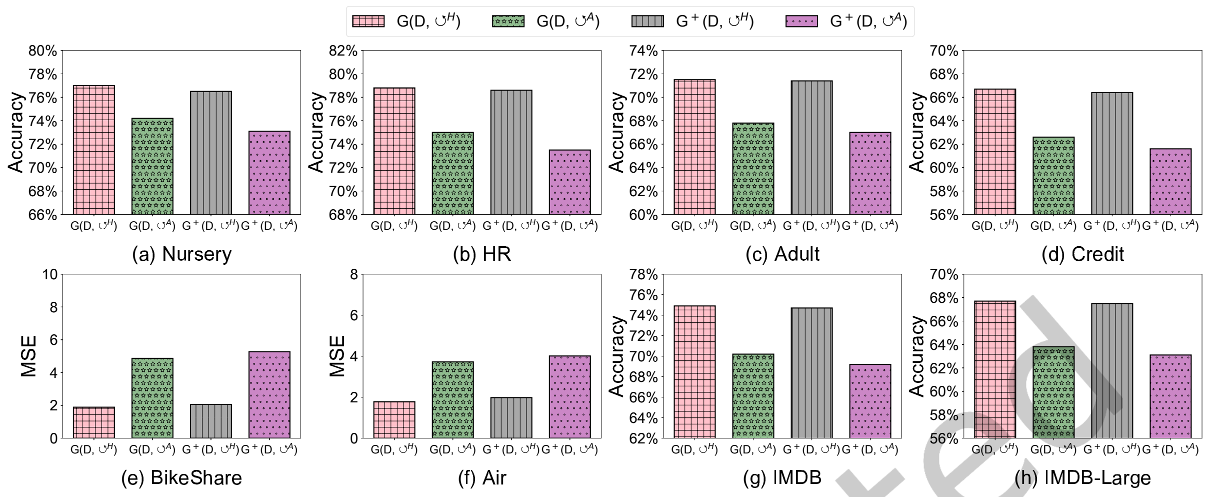


Fig. 10. Effectiveness of G vs G^+.

for $\mathbf{C}(\mathbf{H}(D))$ because they do not have the ground truth. But $\mathbf{H}(\mathbf{C}(D))$ and $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$ are cost-effective because human just needs to impute missing tuples in the much smaller coreset. For example, they only cost 81 and 63 tuples on dataset `Adult` respectively. `ActiveClean` asks the human to iteratively impute the data. Given the same number of tuples to impute, our method can achieve much higher accuracy. We will evaluate it in details in next subsection.

Summary. Based on the results, we have the following conclusions. (1) Our proposed methods $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$ and $\mathbf{G}(D, \circlearrowleft^{\mathbf{A}})$ can achieve high model accuracy because the selected coreset can well represent the underlying ground truth by gradient approximation considering possible repairs. Meanwhile, they are practical because of the low machine cost. (2) Compared with $\mathbf{C}(\mathbf{H}(D))$ that involves human to impute the entire dataset D, the human cost of $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$ is much lower, as observed in Table 2, e.g., 37 vs. 3278 on the `Nursery` dataset. Thus, we can choose $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$ when we want to achieve high model accuracy and afford a certain human cost. (3) If we neither care very much about the accuracy nor consider to incur human cost, the much more efficient $\mathbf{G}(D, \circlearrowleft^{\mathbf{A}})$ is a good choice.

7.3 Evalution of GoodCore$^+$

In this part, we evaluate the efficacy of GoodCore$^+$.

7.3.1 Evaluation of model accuracy. The results are provided in Figure 10. We can found that the accuracy of GoodCore$^+$ and GoodCore are roughly the same on all datasets. For example, on dataset `IMDB-Large`, $\mathbf{G}^+(D, \circlearrowleft^{\mathbf{H}}$)and $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$ achieve accuracy of 74.7% and 74.9% and they differ from each other by 0.2%. This is because both of them select a good coreset because of a bounded GA error. In addition, we can observe that $\mathbf{G}^+(D, \circlearrowleft^{\mathbf{H}})$ performs better than $\mathbf{G}^+(D, \circlearrowleft^{\mathbf{A}})$ because human imputation is more accurate than automatic methods. For example, on `IMDB-Large`, $\mathbf{G}^+(D, \circlearrowleft^{\mathbf{H}})$ has an accuracy of 74.7%, higher than that of $\mathbf{G}(D, \circlearrowleft^{\mathbf{A}})$.

7.3.2 Evaluation of the efficiency. We evaluate the efficiency of GoodCore$^+$ and GoodCore, including the machine cost and human cost.

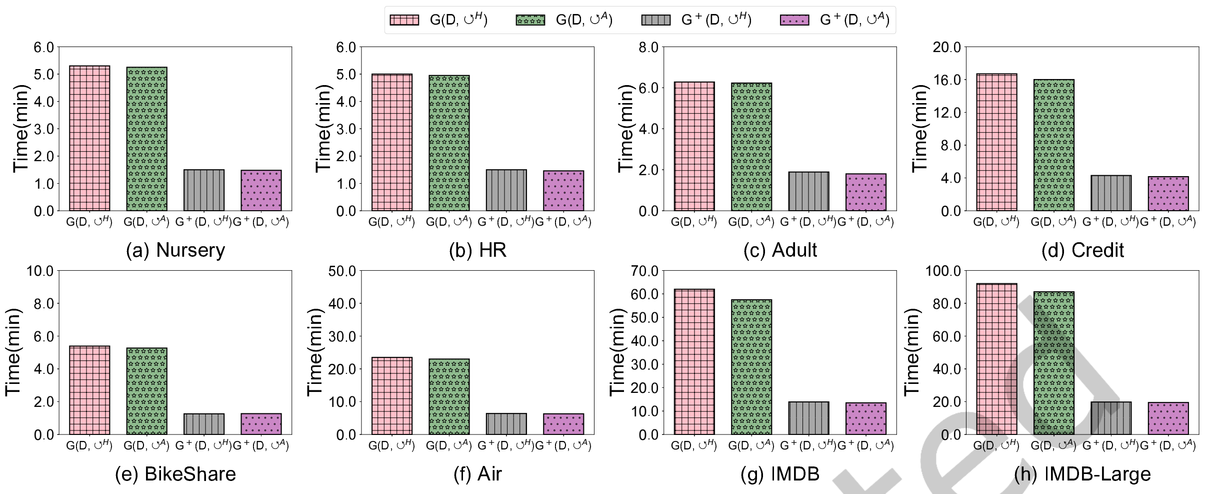


Fig. 11. Efficiency of G vs G^{+}. Note that only machine cost (i.e., runtime of machine) is considered.

Machine cost. Machine cost is shown in Figure 11. GoodCore^{+} is more efficient than GoodCore. For example, on IMDB-Large, $\mathbf{G}^{+}(D, \circlearrowleft^{\mathbf{H}})$ spends about 12min, which is 4.8× faster than $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$. That is because $\mathbf{G}^{+}(D, \circlearrowleft^{\mathbf{H}})$ have the lower time complexity than $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$, which is discussed in Section 6.3.2.
Human cost. In terms of the human cost, as shown in Table 2, $\mathbf{G}^{+}(D, \circlearrowleft^{\mathbf{H}})$ and $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$ are cost-effective because humans just need to impute missing tuples in a much smaller coreset. For example, they only cost 520 and 511 tuples on dataset IMDB-Large respectively.
Summary. Based on the results, we have the following conclusions. (1) Although we group tuples over the entire train set, our proposed methods $\mathbf{G}^{+}(D, \circlearrowleft^{\mathbf{H}})$ and $\mathbf{G}^{+}(D, \circlearrowleft^{\mathbf{A}})$ still achieve high accuracy because the gradient approximation error can still be bounded. (2) The efficiency is much improved compared with GoodCore because we just need to iterate these groups rather than the entire train set within the 3-loop coreset selection process. (3) The human cost is competitive with $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$ because the group-based solution has low impact on the number of tuples to be imputed by humans.

7.4 Coreset Size Selection of GoodCore

Recap that GoodCore needs the user-specified coreset size as input. Thus, we discuss how to select an appropriate coreset size. We adopt a simple yet effective solution that starts from a coreset with a small size, train over it and evaluate via a validation set, enlarge the coreset and iteratively train until the performance does not improve much. To be specific, initially, we begin with $\rho = 10^{-4}$, and enlarge the coreset by 2 times iteratively. If the performance on validation set varies no more than 0.5% within three successive iterations, we will stop. Figure 12 shows the performance on dataset HR , Adult and BikeShare when varying the coreset size. We can see that the performance first improves rapidly, then remains stable just after several iterations. For example, on dataset Adult, when $\rho = 5 \times 10^{-3}$, the accuracy has improved to 72.85% on the validation set. Empirically, an ideal coreset size is between $\rho = 10^{-3}$ to 10^{-2}.
Summary. The results show that coreset size is not difficult to determine. If the user is willing to specify a coreset size like in Section 7.2 based on the empirical finding, we can directly compute a coreset without training. If she cannot, we can also get a good coreset with just several training iterations over small coresets, which is also efficient.

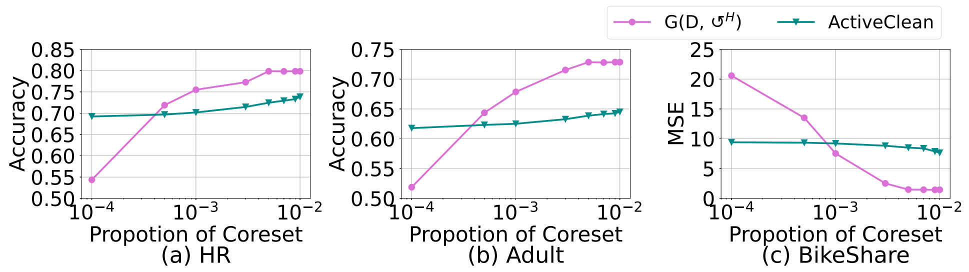


Fig. 12. Coreset size selection of GoodCore.

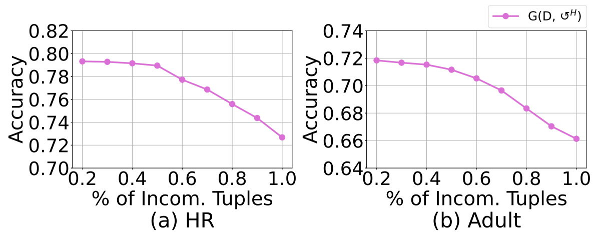


Fig. 13. Varying missing tuple rate.

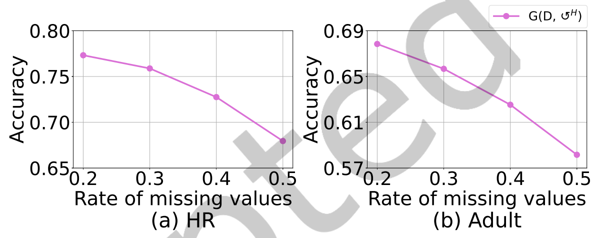


Fig. 14. Varying missing value rate.

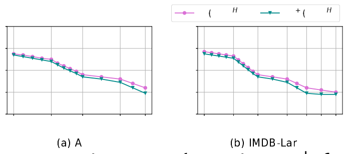


Fig. 15. GoodCore and GoodCore $^+$ for batch algorithm.

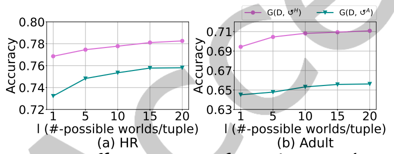


Fig. 16. Effectiveness of GoodCore when varying l.

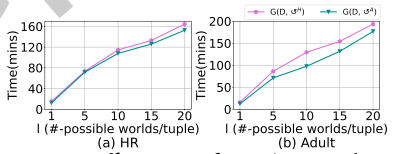


Fig. 17. Efficiency of GoodCore when varying l.

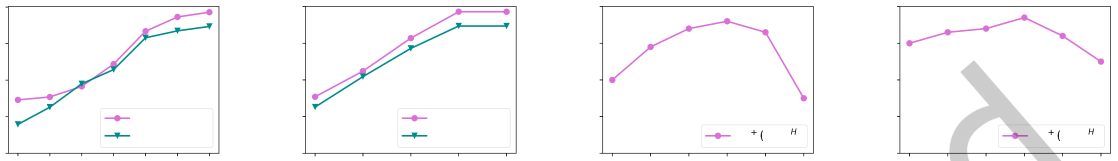
Compare with ActiveClean. Figure 12 also reports an interesting comparison with ActiveClean. Specifically, in ActiveClean, we use the coreset size K as the budget, i.e., number of tuples to be imputed by human in each active cleaning iteration. We can observe that at the beginning, when the coreset size is very small, ActiveClean is better because it trains with the entire dataset including the imputed tuples, while we train the model using only few tuples in the coreset. However, as with the increase of the coreset size, we can see that $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$ outperforms ActiveClean. This is because ActiveClean uses a heuristic method to estimate the impact of tuples to the overall gradient, which is not theoretically bounded (e.g., with gradient bounds like Coreset) and thus not accurate enough. For $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$, it can achieve high accuracy with a proper coreset size, which is not large.

7.5 Batch Algorithm of GoodCore and GoodCore$^+$

In Section 7.2 and Section 7.3, $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$ and $\mathbf{G}^+(D, \circlearrowleft^{\mathbf{H}})$ outperform other baselines on accuracy, but require many human iterations. In this part, we evaluate the batch algorithm of GoodCore and GoodCore$^+$ by varying the batch size b, *i.e.*, Algorithm 3 to reduce the number of iterations. Intuitively, the algorithm is $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$ when $b = 1$. Then we increase b until a single batch with a size b can contain all incomplete tuples in the coreset with size K, which is in fact the algorithm $\mathbf{H}(\mathbf{G}(D))$. Due to the large number of possible worlds, we adopt the heuristic method in Section 5.2 to set $l = 3$ when $b > 1$ for GoodCore and we set $l = 3, l_g = 3$ for GoodCore $^+$ according to Section 6.3.2.

Table 3. The number of possible worlds on different datasets

Method	`Nursery`	`HR`	`Adult`
$\mathbf{H}(\mathbf{G}(D))$	10^{201}	10^{201}	10^{202}
$\mathbf{A}(\mathbf{G}(D))$	10^{201}	10^{201}	10^{202}
$\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$	10^{3}	10^{3}	10^{4}
$\mathbf{G}(D, \circlearrowleft^{\mathbf{A}})$	10^{3}	10^{3}	10^{4}



(a) Hyperparameters in LSH (b) Hyperparameters in (c) V M(m = 12) (d) V M(m = 40)

Fig. 18. Varying Hyperparameters in LSH and Quantization-based Method.

In Figure 15, the x-axis denotes the batch size and the y-axis denotes the test performance on dataset `Adult` and `IMDB-Large`. We can see that when b is small (*i.e.*, $b \leq 5$), the performance does not significantly decrease (*e.g.*, on `IMDB-Large`, the accuracy decreases from 71.5% to 70.9% with $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$). However, when b keeps increasing, the performance slightly decreases. Thus, `GoodCore` and `GoodCore` $^{+}$ are not very sensitive to the batch size b and we can reduce the number of human iterations without sacrificing much model performance.

In this part, we also vary the number of possible worlds by varying l, which is the number of possibles world per tuple. The larger l, the larger number of possible worlds we have. The results are shown in Figures 16 and 17. In terms of the accuracy, we can see that with l increasing (fixing $b = 10$), the accuracy increases first and then remains stable soon, but the time keeps increasing because more possible worlds indicate more computation. Hence, we do not need a large l.

When it comes to the number of possible worlds, we would like to clarify that we do not compare with $\mathbf{H}(\mathbf{G}(D))$ and $\mathbf{A}(\mathbf{G}(D))$ because the number of possible worlds of D is very large, which is infeasible to compute. We show the number in Table 3, where we also report the numbers of possible worlds of $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$ and $\mathbf{G}(D, \circlearrowleft^{\mathbf{A}})$ in each iteration, which are practical to compute.

7.6 Ablation Studies of GoodCore^{+}

Hyperparamaters for grouping.

We use LSH to effectively group the entire dataset and test the impact of different numbers of hyperplanes, which is an important parameter in LSH. As shown in Figure 18 (a), as the number increases, more groups are generated and the tuples within each group become closer, resulting in an increase in initial accuracy. Afterwards, the accuracy remains stable because the tuples in each cluster are similar enough to approximate the gradient. Therefore, based on experience, using 64 hyperplanes is the most suitable, as more groups will reduce efficiency.

Hyperparamaters in quantization-based method. In Section 6.3.2, we use quantization-based method to estimate the upper bound $\hat{s}_{ju}$ of $\bar{s}_{ju}$. Recap that `GoodCore`$^{+}$ needs the user-specified cluster centers size R, which is important for computing the maximum feature distances. To choose a proper R, we adopt a simple yet effective solution that selects different R and obtain different coresets. Then we train over these coresets and evaluate via a validation set to get different results. Specifically, we select R from 32 to 512 for each dataset. Figure 18 (b) shows the performance on dataset `HR` and `IMDB-Large` when varying the cluster centers size R. We can see that

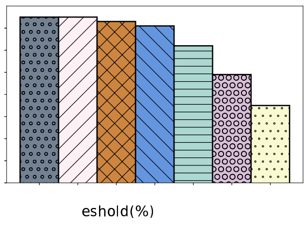


Fig. 19. Effectiveness of GoodCore$^+$ for different entropy thresholds.

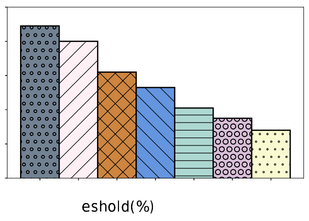


Fig. 20. Efficiency of GoodCore$^+$ for different entropy thresholds.

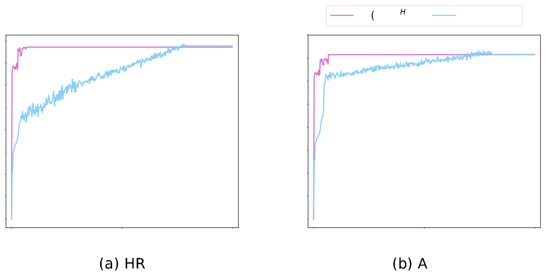


Fig. 21. Convergence of GoodCore.

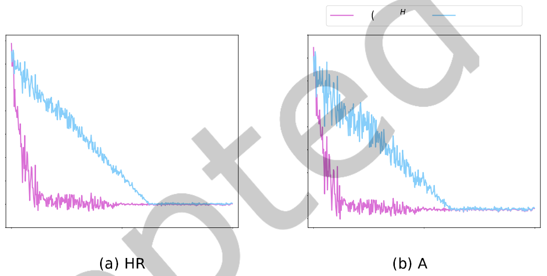


Fig. 22. Loss of GoodCore.

as R increases, the accuracy of the dataset also gradually increases, because when R increases, the upper bound $\hat{s}_{ju}$ is closer to $\bar{s}_{ju}$, which can help us to select a good coreset.

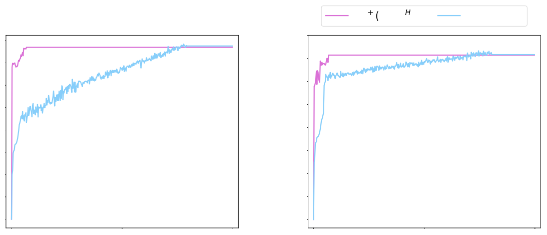
We also tested the performance of different feature segmentation methods (corresponding to different M). In Figure 18 (c)-(d), the initial $M = 12$ indicates that in each subspace, the length of all sub-vectors is 1. As M decreases, the accuracy first improves because each sub-vector becomes longer, containing more information when adding these $\hat{s}^z_{ju}$, resulting in more accurate boundaries. But if each sub-vector is too long, which means that each vector is quantized into a very short code, the accuracy will decrease because in this case, the quantization based method does not have enough information to give accurate distance estimates. Based on experience, when M is around 3, it is always a good choice.

Varying the entropy threshold. In this part, we start to use the entropy to further eliminate the number of possible world by imputing the missing cells with low entropy in advance, as discussed in Section 6.3.1. Specifically, we test the impact of different entropy thresholds. As shown in Figure 19 and Figure 20, when the threshold is less than 20% (*i.e.*, we directly impute the cell if its corresponding entropy is no larger than 20% $\log n_x$), the accuracy almost remains unchanged, while the efficiency is improved by almost 30%. However, when the threshold exceeds 20%, the accuracy begins to decrease because in this situation, directly imputing a value is not accurate enough. In short, setting an appropriate threshold (*e.g.*, 20%) is helpful to improve the efficiency without sacrificing the accuracy.

7.7 Convergence Evaluation

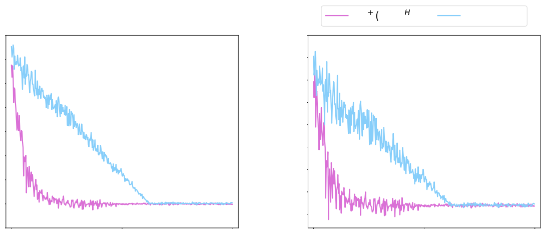
In Section 5.3, we have shown the convergence rate of GoodCore theoretically. In this part, we test the convergence of training over the coreset ($\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$) and entire data (`Complete`) empirically. Figure 21 shows the test accuracy of two methods with the number of training iterations increasing. We can observe that on both datasets, training on the coreset converges much faster than training on the full data.

For example, on dataset `Adult`, it takes ~40 iterations for GoodCore to converge, which is 180× faster than `Complete`. This is because GoodCore has the same convergence rate with training over the entire dataset as



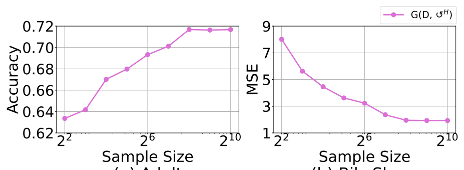
(a) HR (b) A

Fig. 23. Convergence of GoodCore^{+}.



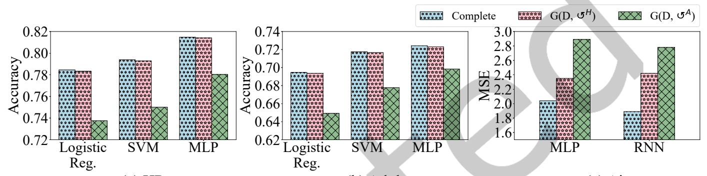
(a) HR (b) A

Fig. 24. Loss of GoodCore^{+}.



(a) Adult (b) BikeShare

Fig. 25. Varying sample size.



(a) HR (b) Adult (c) Air

Fig. 26. Varying downstream model.

discussed in the theoretical result of Section 5.3, but the entire dataset (*e.g.*, Adult) is 200× (similar to 180×) larger than the coreset ($\rho = 0.005$). That is, GoodCore converges with the same number of epochs as training on the entire dataset. Since the size of coreset is much smaller, GoodCore is more efficient. Also, we can achieve competitive accuracy as training on full data by approximating the full gradient with a theoretical bound.

Furthermore, we report the loss change to reflect the relation between actual convergence rate and theoretical results. In Figure 22, on dataset HR , the initial loss is 8.4. According to the theoretical convergence rate $O(\frac{1}{\sqrt{k}})$ (this k denotes the k-th epoch), the loss should decrease to around 3.8 at the end of 5-th epoch ($\approx$ 3200-th iteration). Actually, the actual loss decreases to 3.25 at that time, which is close to the theoretical value.

We also test the convergence of GoodCore^{+}, as shown in Figure 23 and 24. The results validate that the group-based method can also converge fast.

7.8 Sensitivity Analysis

Varying the sample size. In this part, we vary the sample size h and evaluate the performance. The experimental results are shown in Figure 25. We vary the sample size h from 2^2 to 2^{10}. We can see that when h is too small, the performance is low. The reason is that GoodCore cannot precisely estimate the utilities of tuples when h is small. When the sample size increases, we can see that the performance improves rapidly and finally becomes stable, which indicates that GoodCore is not much sensitive to the sample size when h is not too small.

Varying the downstream models. Recap that GoodCore can be used on different convex models. Thus, in this part, we apply GoodCore on different convex models and evaluate the performance. We evaluate logistic regression and SVM for classification tasks. For regression tasks, we evaluate linear regression, ridge regression and SVM regression. We can see that in Figure 26 (a) and (b), on dataset Adult, $\mathbf{G}(D, \circlearrowleft^{\mathbf{H}})$ achieved 71.7% accuracy for SVM, better than on logistic regression (69.4%). Although different downstream models may have different performance, GoodCore can improve the model performance for the specific downstream model. In order to show that GoodCore can be used for other types of models like neural networks, we compare with Multilayer Perceptron (MLP, fully connected networks of 2 hidden layers with 256 nodes for each layer), although GoodCore does not hold theoretical guarantee for this non-convex model. As shown in Figure 26, we can see that MLP achieves almost the same performance as the ground truth. This is because the coreset selected by GoodCore can also represent the full dataset. However, in Figure 26(c), on a large dataset Air (with metric MSE, the lower the

better), neural network based methods (we also implement RNN, 2 hidden layers with 128 nodes for each layer) can have a better accuracy but the coreset cannot perfectly achieve the same performance. This may because this large dataset has more informative things to learn, and it is hard for the coreset-based solution to well represent the dataset without the theoretical guarantee.
Varying the percentage of incomplete tuples. In this part, we vary the rate of missing tuples and evaluate the performance, as shown in Figure 13. Note that the rate denotes the percentage of incomplete tuples rather than the cell values. Even if a tuple just has one missing attribute, it is regarded as incomplete. We vary the percentage from 20% to 100%. We can observe that the performance does not decrease much with the percentage increasing from 20% to 50%, which indicates that GoodCore is not very sensitive to the percentage of incomplete tuples in this range. After that, the accuracy decreases because there are more missing tuples.

Besides, we also vary the rate of missing cell values in Figure 14. In this scenario, for example, 50% missing values of a dataset indicates more number of missing cell values than the scenario of 50% missing tuples. Hence, we can see that the accuracy decreases more quickly than Figure 13.

8 Related Work

Task-agnostic incomplete data imputation. Data imputation has been widely studied for years. Existing methods can be divided into two categories: statistic-based methods and learning-based methods. The former one always uses the statistic information [33, 55] (like mean, median or mode) to impute the missing values. Also, some methods compute the similarity of the incomplete tuples to the complete tuples and use the most similar one to impute the missing values [9, 39, 75]. Recently, to improve the imputation accuracy, many learning-based methods focus on how to use ML to learn the data distribution (*e.g.*, MissForest imputation [73], MICE [68], IIM [79]), and then use the trained model to predict the missing values. Besides traditional ML models, some deep learning models are also used for data imputation (*e.g.*, autoencoder [35, 56, 63], GANs [72, 78]).
Coreset selection. A previous work [21] has studied how to select a well-performed coreset over incomplete tuples. Another work [23] studies to use group-based method to accelerate the coreset selection process without incomplete data. The extension to the previous studies in this work is five-fold. First, we propose a new framework that incorporates the group-based strategy into the coreset selection process with incomplete data. Second, we theoretically analyze that incorporating the group-based strategy still leads to a bounded GA error, considering the possible worlds produced by the incomplete data. Third, given these groups, the number of possible worlds further increases, more strategies are proposed to reduce the number of possible worlds to improve the efficiency. Fourth, we theoretically and empirically prove that incorporating the group-based method still guarantees the convergence. Fifth, two large datasets and ten new experiments are added to demonstrate the efficacy of our proposed methods. In addition, Huang et al. [38] studied how to compute and continuously update the coreset while training, but it is rather time-consuming because of the training process. To solve this problem, works [17, 18] selected the coreset without training in advance, but they can only be customized to particular model types respectively. Some works [7, 24, 69, 71] study how to select a subset of data only considering the data distribution rather than the model performance. Gradient-based methods [46, 58, 60, 76] focused on selecting the coreset to approximate the full gradient without training in advance for multiple model types, which is regarded as an optimization problem as discussed in Section 2.2. Moreover, Deng et al. [29] choose an optimal coreset under label uncertainty, particularly when encountering a deep learning training set that contains mislabeled data. Coreset selection can also be formulated as a bilevel optimization problem [15, 43, 45], where the outer objective involves choosing a subset and the inner objective entails optimizing model parameters on the subset, the ultimate goal is to select a subset such that the model empirical risk on this subset approximates that of the model trained on the complete dataset. However it is rather time-consuming because it requires solving an inner optimization problem during each outer iteration.

In short, none of the above methods except [21] consider coreset selection over incomplete data.

Data cleaning for ML. Recently, there have been several works that clean the data to optimize the ML model. In contrast to the above discussion about task-agnostic incomplete data imputation, data cleaning for ML is task-aware, which triggers new technical challenges. SampleClean [47] focuses on cleaning selected samples, so as to answer SQL aggregate queries more efficiently, but it is not for any model. CPClean [40] proposes certain prediction to impute missing data for optimizing ML models. Different from us, it is customized to nearest neighbor classifiers rather than convex models solved by the gradient descent algorithm. BoostClean [49] regards data cleaning as a boosting problem that iteratively selects from a predefined set of cleaning algorithms, so as to continuously maximize the accuracy of a validation set with training iteratively. MisDetect [27] and IDE [30] focus on detecting mislabeled data instances using early loss signals and influence functions. Closer to our work is ActiveClean [48], which progressively cleans the data tuples that are likely to much influence the model measured by the gradients. Different from us, given a budget K, we can select the coreset without training, but ActiveClean needs to train iteratively and label a set of validation dataset. We empirically show that our method outperforms ActiveClean on model accuracy and efficiency in Section 7.

Data preparation for ML. Recent studies have concentrated on enhancing data preparation within the machine learning field. LakeBench [28] provides a benchmark for discovering joinable and unionable tables, while LakeCompass [19] offers a comprehensive system for data search and improves ML model performance. Conversely, STAIR [31] presents a technique for summarizing outliers through interpretable rules, which improves the management of dataset anomalies. Together, these works advance the effectiveness and clarity of data preparation for machine learning.

9 Conclusion

In this paper, we propose the GoodCore framework to select a good coreset over the incomplete data, which achieves data-effective and data-efficient ML. We formulate it as an expected optimal coreset selection problem, which is NP-hard. Then we propose a greedy algorithm with an approximation ratio. We also propose to involve imputation-in-the-loop strategies into GoodCore to improve the efficiency. Furthermore, a group-based acceleration method is incorporated to further accelerate the coreset selection process. We conduct experiments on real-world datasets to verify the effectiveness and efficiency of GoodCore and GoodCore$^+$.

References

[1] 2022. https://github.com/awslabs/datawig.
[2] 2022. https://archive.ics.uci.edu/ml/datasets/nursery.
[3] 2022. https://archive.ics.uci.edu/ml/datasets/adult.
[4] 2022. https://www.kaggle.com/.
[5] 2022. https://ride.capitalbikeshare.com/system-data.
[6] 2022. https://auctus.vida-nyu.org/.
[7] Sharat Agarwal, Himanshu Arora, Sandeep Anand, and Chetan Arora. 2020. Contextual Diversity for Active Learning. Cornell University - arXiv,Cornell University - arXiv (Aug 2020).
[8] Zeyuan Allen-Zhu, Yang Yuan, and Karthik Sridharan. 2016. Exploiting the structure: Stochastic gradient methods using raw clusters. NeurIPS 29 (2016).
[9] Naomi S Altman. 1992. An introduction to kernel and nearest-neighbor nonparametric regression. The American Statistician 46, 3 (1992), 175–185.
[10] Alexandr Andoni and Piotr Indyk. 2006. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. In 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings. IEEE Computer Society, 459–468. doi:10.1109/FOCS.2006.49
[11] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In PODS. ACM Press, 68–79.
[12] Leopoldo E. Bertossi. 2011. Database Repairing and Consistent Query Answering. Morgan & Claypool Publishers.

[13] Leopoldo E. Bertossi. 2019. Database Repairs and Consistent Query Answering: Origins and Further Developments. In PODS. ACM, 48–58.
[14] Felix Biessmann and Tammo Rukat et al. 2019. DataWig: Missing Value Imputation for Tables. JMLR 20, 175 (2019), 1–6.
[15] Zalán Borsos, Mojmir Mutny, and Andreas Krause. 2020. Coresets via bilevel optimization for continual learning and streaming. Advances in neural information processing systems 33 (2020), 14879–14890.
[16] Stephen P Boyd and Lieven Vandenberghe. 2004. Convex optimization. Cambridge university press.
[17] Vladimir Braverman, Dan Feldman, and Harry Lang. 2016. New Frameworks for Offline and Streaming Coreset Constructions. CoRR abs/1612.00889 (2016).
[18] Trevor Campbell and Tamara Broderick. 2018. Bayesian Coreset Construction via Greedy Iterative Geodesic Ascent. In ICML 2018, Vol. 80. PMLR, 697–705.
[19] Chengliang Chai, Yuhao Deng, Yutong Zhan, Ziqi Cao, Yuanfang Zhang, Lei Cao, Yu-Ping Wang, Zhiwei Zhang, Ye Yuan, Guoren Wang, and Nan Tang. 2024. LakeCompass: An End-to-End System for Table Maintenance, Search and Analysis in Data Lakes. Proc. VLDB Endow. 17, 12 (2024), 4381–4384. doi:10.14778/3685800.3685880
[20] Chengliang Chai and Jiabin Liu et al. 2022. Selective data acquisition in the wild for model charging. PVLDB 15, 7 (2022), 1466–1478.
[21] Chengliang Chai and Jiabin Liu et al. 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. doi:10.1145/3589302
[22] Chengliang Chai and Jiayi Wang et al. 2022. Data management for machine learning: A survey. TKDE (2022).
[23] Chengliang Chai and Jiayi Wang et al. 2023. Efficient Coreset Selection with Cluster-based Methods. In Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023. ACM, 167–178. doi:10.1145/3580305.3599326
[24] Yutian Chen, Max Welling, and Alex Smola. 2012. Super-samples from kernel herding. arXiv preprint arXiv:1203.3472 (2012).
[25] Kai Lai Chung. 1954. On a stochastic approximation method. The Annals of Mathematical Statistics (1954), 463–483.
[26] Ting Deng, Wenfei Fan, and Floris Geerts. 2016. Capturing Missing Tuples and Missing Values. ACM Trans. Database Syst. 41, 2 (2016), 10:1–10:47.
[27] Yuhao Deng, Chengliang Chai, Lei Cao, Nan Tang, Jiayi Wang, Ju Fan, Ye Yuan, and Guoren Wang. 2024. MisDetect: Iterative Mislabel Detection using Early Loss. Proc. VLDB Endow. 17, 6 (2024), 1159–1172. doi:10.14778/3648160.3648161
[28] Yuhao Deng, Chengliang Chai, Lei Cao, Qin Yuan, Siyuan Chen, Yanrui Yu, Zhaoze Sun, Junyi Wang, Jiajun Li, Ziqi Cao, Kaisen Jin, Chi Zhang, Yuqing Jiang, Yuanfang Zhang, Yuping Wang, Ye Yuan, Guoren Wang, and Nan Tang. 2024. LakeBench: A Benchmark for Discovering Joinable and Unionable Tables in Data Lakes. Proc. VLDB Endow. 17, 8 (2024), 1925–1938. doi:10.14778/3659437.3659448
[29] Yuhao Deng, Chengliang Chai, Kaisen Jin, Linan Zheng, Lei Cao, Ye Yuan, and Guoren Wang. 2025. Two Birds with One Stone: Efficient Deep Learning over Mislabeled Data through Subset Selection. In SIGMOD.
[30] Yuhao Deng, Deng Qiyan, Chengliang Chai, Lei Cao, Nan Tang, Ju Fan, Jiayi Wang, Ye Yuan, and Guoren Wang. 2024. IDE: A System for Iterative Mislabel Detection. In Companion of the 2024 International Conference on Management of Data, SIGMOD/PODS 2024, Santiago AA, Chile, June 9-15, 2024, Pablo Barceló, Nayat Sánchez-Pi, Alexandra Meliou, and S. Sudarshan (Eds.). ACM, 500–503. doi:10.1145/3626246.3654737
[31] Yuhao Deng, Yu Wang, Lei Cao, Lianpeng Qiao, Yuping Wang, Xu Jingzhe, Yizhou Yan, and Samuel Madden. 2024. Outlier Summarization via Human Interpretable Rules. Proc. VLDB Endow. 17, 7 (2024), 1591–1604. doi:10.14778/3654621.3654627
[32] Irit Dinur and Samuel Safra. 2005. On the hardness of approximating minimum vertex cover. Annals of mathematics (2005), 439–485.
[33] Alireza Farhangfar, Lukasz A. Kurgan, and Witold Pedrycz. 2007. A Novel Framework for Imputation of Missing Values in Databases. IEEE Trans. Syst. Man Cybern. Part A 37, 5 (2007), 692–709.
[34] Dan Feldman. 2020. Introduction to Core-sets: an Updated Survey. CoRR abs/2011.09384 (2020).
[35] Lovedeep Gondara and Ke Wang. 2017. Multiple Imputation Using Deep Denoising Autoencoders. CoRR abs/1705.02737 (2017).
[36] John A Hartigan and Manchek A Wong. 1979. Algorithm AS 136: A k-means clustering algorithm. Journal of the royal statistical society. series c (applied statistics) 28, 1 (1979), 100–108.
[37] Thomas Hofmann, Aurelien Lucchi, Simon Lacoste-Julien, and Brian McWilliams. 2015. Variance reduced stochastic gradient descent with neighbors. Advances in Neural Information Processing Systems 28 (2015).
[38] Jiawei Huang and Ruomin Huang et al. 2021. A Novel Sequential Coreset Method for Gradient Descent Algorithms. In ICML 2021, Vol. 139. PMLR, 4412–4422.
[39] José M. Jerez and Ignacio Molina et al. 2010. Missing data imputation using statistical and machine learning methods in a real breast cancer problem. Artif. Intell. Medicine 50, 2 (2010), 105–115.
[40] Bojan Karlas and Peng Li et al. 2020. Nearest Neighbor Classifiers over Incomplete Information: From Certain Answers to Certain Predictions. Proc. VLDB Endow. 14, 3 (2020), 255–267.
[41] Shahidul Islam Khan and Abu Sayed Md. Latiful Hoque. 2020. SICE: an improved missing data imputation technique. J. Big Data 7, 1 (2020), 37.

[42] Krishnateja Killamsetty and S Durga et al. 2021. Grad-match: Gradient matching based data subset selection for efficient deep model training. In ICML. 5464–5474.
[43] Krishnateja Killamsetty, Durga Sivasubramanian, Ganesh Ramakrishnan, and Rishabh Iyer. 2022. GLISTER: Generalization based Data Subset Selection for Efficient and Robust Learning. Proceedings of the AAAI Conference on Artificial Intelligence (Sep 2022), 8110–8118. doi:10.1609/aaai.v35i9.16988
[44] KrishnaTeja Killamsetty, Durga Sivasubramanian, Ganesh Ramakrishnan, and Rishabh K. Iyer. 2021. GLISTER: Generalization based Data Subset Selection for Efficient and Robust Learning. In AAAI 2021,. AAAI Press, 8110–8118.
[45] Krishnateja Killamsetty, Xujiang Zhao, Feng Chen, and Rishabh Iyer. 2021. Retrieve: Coreset selection for efficient and robust semi-supervised learning. Advances in neural information processing systems 34 (2021), 14488–14501.
[46] Katrin Kirchhoff and Jeff A. Bilmes. 2014. Submodularity for Data Selection in Machine Translation. In EMNLP 2014. ACL, 131–141.
[47] Sanjay Krishnan and Jiannan Wang et al. 2015. SampleClean: Fast and Reliable Analytics on Dirty Data. IEEE Data Eng. Bull. 38, 3 (2015), 59–75.
[48] Sanjay Krishnan and Jiannan Wang et al. 2016. ActiveClean: Interactive Data Cleaning For Statistical Modeling. Proc. VLDB Endow. 9, 12 (2016), 948–959.
[49] Sanjay Krishnan, Michael J. Franklin, Ken Goldberg, and Eugene Wu. 2017. BoostClean: Automated Error Detection and Repair for Machine Learning. CoRR abs/1711.01299 (2017).
[50] Viktor Leis and Andrey Gubichev et al. 2015. How Good Are Query Optimizers, Really? Proc. VLDB Endow. 9, 3 (2015), 204–215.
[51] Claude Lemaréchal. 2012. Cauchy and the gradient method. Doc Math Extra 251, 254 (2012), 10.
[52] Peng Li and Xi Rao et al. 2021. CleanML: A Study for Evaluating the Impact of Data Cleaning on ML Classification Tasks. In ICDE. 13–24.
[53] Hui Lin and Jeff Bilmes. 2011. A class of submodular functions for document summarization. In Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies. 510–520.
[54] Tongyu Liu, Ju Fan, and Yinqing Luo et al. 2021. Adaptive data augmentation for supervised learning over missing data. Proc. VLDB Endow. 14, 7 (2021), 1202–1214.
[55] Chris Mayfield, Jennifer Neville, and Sunil Prabhakar. 2010. ERACER: a database approach for statistical inference and data cleaning. In SIGMOD. ACM, 75–86.
[56] John T McCoy, Steve Kroon, and Lidia Auret. 2018. Variational autoencoders for missing data imputation with application to a simulated milling circuit. IFAC-PapersOnLine 51, 21 (2018), 141–146.
[57] Xiaoye Miao and Yangyang Wu et al. 2022. An Experimental Survey of Missing Data Imputation Algorithms. TKDE (2022).
[58] Baharan Mirzasoleiman, Jeff A. Bilmes, and Jure Leskovec. 2020. Coresets for Data-efficient Training of Machine Learning Models. In ICML 2020, Vol. 119. 6950–6960.
[59] Baharan Mirzasoleiman, Kaidi Cao, and Jure Leskovec. 2020. Coresets for robust training of deep neural networks against noisy labels. NeurIPS 33 (2020), 11465–11477.
[60] Baharan Mirzasoleiman, Kaidi Cao, and Jure Leskovec. 2020. Coresets for Robust Training of Neural Networks against Noisy Labels. arXiv: Learning,arXiv: Learning (Nov 2020).
[61] Baharan Mirzasoleiman and Ashwinkumar Badanidiyuru et al. 2015. Lazier than lazy greedy. In AAAI, Vol. 29.
[62] Alexander Munteanu and Chris Schwiegelshohn. 2018. Coresets-methods and history: A theoreticians design pattern for approximation and streaming algorithms. KI-Künstliche Intelligenz 32, 1 (2018), 37–53.
[63] Alfredo Nazábal, Pablo M. Olmos, Zoubin Ghahramani, and Isabel Valera. 2020. Handling incomplete heterogeneous data using VAEs. Pattern Recognit. 107 (2020), 107501.
[64] Angelia Nedić and Dimitri Bertsekas. 2001. Convergence rate of incremental subgradient algorithms. In Stochastic optimization: algorithms and applications. Springer, 223–264.
[65] Felix Neutatz, Binger Chen, Ziawasch Abedjan, and Eugene Wu. 2021. From Cleaning before ML to Cleaning for ML. IEEE Data Eng. Bull. (2021).
[66] Andrew Ng. 2021. MLOPs: From Model-centric to Data-centric AI.
[67] LKPJ Rdusseeun and P Kaufman. 1987. Clustering by means of medoids. In Proceedings of the statistical data analysis based on the L1 norm conference, neuchatel, switzerland, Vol. 31.
[68] Patrick Royston and Ian R White. 2011. Multiple imputation by chained equations (MICE): implementation in Stata. Journal of statistical software 45 (2011), 1–20.
[69] Ozan Sener and Silvio Savarese. 2017. Active learning for convolutional neural networks: A core-set approach. arXiv preprint arXiv:1708.00489 (2017).
[70] Claude Elwood Shannon. 1948. A mathematical theory of communication. The Bell system technical journal 27, 3 (1948), 379–423.
[71] Samarth Sinha and Han Zhang et al. 2019. Small-GAN: Speeding Up GAN Training Using Core-sets. arXiv: Machine Learning,arXiv: Machine Learning (Oct 2019).

[72] Indro Spinelli, Simone Scardapane, and Aurelio Uncini. 2020. Missing data imputation with adversarially-trained graph convolutional networks. Neural Networks 129 (2020), 249–260.
[73] Daniel J. Stekhoven and Peter Bühlmann. 2012. MissForest - non-parametric missing value imputation for mixed-type data. Bioinform. 28, 1 (2012), 112–118.
[74] Gilbert Strang. 2006. Linear algebra and its applications. Belmont, CA: Thomson, Brooks/Cole.
[75] Bhekisipho Twala, Michelle Cartwright, and Martin J. Shepperd. 2005. Comparison of various methods for handling incomplete data in software engineering databases. In ISESE 2005. 105–114.
[76] Jiayi Wang and Chengliang Chai et al. 2022. Coresets over Multiple Tables for Feature-rich and Data-efficient Machine Learning. Proc. VLDB Endow. 16, 1 (2022), 64–76. doi:10.14778/3561261.3561267
[77] Richard Wu, Aoqian Zhang, Ihab F. Ilyas, and Theodoros Rekatsinas. 2020. Attention-based Learning for Missing Data Imputation in HoloClean. In MLSys 2020. mlsys.org.
[78] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. 2018. GAIN: Missing Data Imputation using Generative Adversarial Nets. In ICML, Vol. 80. 5675–5684.
[79] Aoqian Zhang, Shaoxu Song, Yu Sun, and Jianmin Wang. 2019. Learning Individual Models for Imputation. In ICDE. IEEE, 160–171.

Received 16 April 2024; revised 16 April 2024; accepted 3 February 2025