When Does Conflict Resolution Help Downstream ML Models?

Wenfei Fan[1,2,3], Xiaoyu Han[4], Hufsa Khan[1], Weilong Ren[1], Yaoshu Wang[1], Min Xie[1], Zihuan Xu[1]
[1]Shenzhen Institute of Computing Sciences [2]Beihang University [3]University of Edinburgh [4]Fudan University
wenfei@inf.ed.ac.uk,xyhan22@m.fudan.edu.cn,{hufsa,renweilong,yaoshuw,xiemin,xuzihuan}@sics.ac.cn

ABSTRACT

The paper studies the impact of inconsistencies in datasets on the accuracy of machine learning (ML) classification models $\mathcal{M}$, when $\mathcal{M}$ is trained and evaluated with the datasets. While previous work reports that inconsistencies typically have an insignificant impact, we show that conflicts have substantial impact on $\mathcal{M}$ if appearing in influential attributes and/or tuples. Hence, we study how to identify influential attributes and tuples that affect the accuracy of $\mathcal{M}$. We show that both problems are intractable. This said, we develop effective algorithms to pinpoint the two factors. Moreover, we propose an iterative creator-critic framework that integrates model training and conflict resolution; the creator trains the models, and the critic resolves conflicts and provides the creator with improved data for incremental training. We show that the iterative process improves ML accuracy and converges at a stable state. Using real-life datasets, we empirically verify that the approach improves the (relative) accuracy of various ML classification models by 43.1% on average.

1 INTRODUCTION

Data cleaning has been studied for decades [29, 30]. Traditionally it takes a dataset $\mathcal{D}$ as input, detects and fixes errors in $\mathcal{D}$, and returns an improved dataset $\mathcal{D}_c$ for subsequent queries and applications. Intuitively, since $\mathcal{D}_c$ has a better quality than $\mathcal{D}$, the answers to the queries in $\mathcal{D}_c$ are more accurate and reliable than in $\mathcal{D}$.

There has been recent interest in *data cleaning for machine learning (ML)* [13, 27, 28, 32, 40, 45, 47, 48, 52, 54, 55, 65, 70, 71, 73, 78]. It takes an ML model $\mathcal{M}$ and a dataset $\mathcal{D}$ as input, where $\mathcal{D}$ is used to train and evaluate $\mathcal{M}$. It aims to deduce a "cleaned" dataset $\mathcal{D}_{\mathcal{M}}$ from $\mathcal{D}$, such that the accuracy of the downstream model $\mathcal{M}$ trained with $\mathcal{D}_{\mathcal{M}}$ is maximumly improved over with $\mathcal{D}$. The need for this is evident. It is reported that 80% of data scientists' time is spent on cleaning datasets for ML applications [6, 20, 44]. Using data cleaning tool Cleanlab [4], a Berkeley lab improves the accuracy of ML models by 15% and reduces the training time by 33%.

Rather than the one-size-fit-all $\mathcal{D}_c$ from traditional data cleaning, data cleaning for ML tailors $\mathcal{D}_{\mathcal{M}}$ for improving the accuracy of a given model $\mathcal{M}$. It is known that dataset $\mathcal{D}_c$ "could in fact degrade" the accuracy of $\mathcal{M}$ [60], due to overfitting and low tolerance to noise, among other things. However, the computation of $\mathcal{D}_{\mathcal{M}}$ is rather delicate, depending on the type of model $\mathcal{M}$, the distribution of data in $\mathcal{D}$, and the accuracy of cleaning methods.

What data cleaning operations can we employ to deduce $\mathcal{D}_{\mathcal{M}}$ from $\mathcal{D}$? A routine operation of traditional data cleaning is *conflict resolution* (*CR*). It is to detect and correct inconsistencies in the data, *e.g.,* mismatching area code and city in an address. There has been a large body of work on CR, by using logic rules [7, 10, 11, 16, 21–23, 25, 34–36, 66, 80], ML [41, 56, 57, 75, 79], and hybrid of the two [24, 26]. Despite the efforts, it is reported [52] that CR "is more likely to have insignificant impact and unlikely to have negative impact on ML classification models". Thus, the community has focused

tid	category	gender	education	workhour	occupation	income
	(A_1)	(A_2)	(A_3)	(A_4)	(A_5)	(Y)
t_1	Private	M	HS-grad	60	Farming	>50K
t_2	Private	M	Bachelor	30	Transport-moving	<=50K
t_3	Private	F	Bachelor	35	Sales	<=50K
t_4	Local-gov	M	PhD	35	Adm-clerical	<=50K
t_5	Local-gov	F	PhD	70	Prof.	>50K
t_6	Part-time	M	HS-graduate	10	Tech-support	>50K
...	...	...	...	...	...	...

Table 1: An Adult **Table**

on other cleaning operators for ML, *e.g.,* category deduplication [70] and missing value imputation [61]; to the best of our knowledge, no prior work has studied critical/categorical attributes for CR.

Is CR always useless for training downstream ML models? The answer is negative, as illustrated below for ML classification.

Example 1: Consider Table 1 of schema Adult, which has category, gender, education, workhour, occupation as attributes and income as the label. Values colored in red are incorrect *e.g.,* t_1[occupation] should be `Exec-managerial` instead of `Farming`, t_4[education] should be `Bachelor`, not `PhD`, and t_6[workhour] should be `50`, not `10`. Such errors may deteriorate the accuracy of a ML model, *e.g.,* the model learns that `PhD` comes with low salary (t_4); this is usually not true. As will be seen in Section 2, cleaning such errors improves the accuracy of some classification models by 14.4% on average. □

A related topic is *data cleaning for AutoML* [31, 42, 49, 50, 61], which is to automatically equip ML pipelines (with embedded data cleaning) to maximize the accuracy of $\mathcal{M}$. We focus on data cleaning for ML in this paper since compared to AutoML, it not only helps train a better $\mathcal{M}$ but also produces a cleaned dataset $\mathcal{D}_{\mathcal{M}}$ for other use, *e.g.,* $D_{\mathcal{M}}$ can serve as a better starting point of data preparation (*e.g.,* feature engineering and smoothing) for AutoML [70]; as will be seen in Section 2, a CR method [56, 57] improves the accuracy of a state-of-the-art (SOTA) AutoML method [61] by 18% on average.

To make effective use of CR for improving downstream ML models, several questions have to be answered. When does CR help improve the accuracy of ML models? What conflicts should we fix? How can we make practical use of CR when training ML models? To simplify the discussion, we focus on differential ML classification.

Contributions & Organization. This paper aims to answer these questions, develop a better understanding of the impact of CR on ML, and make practical use of it. We report that CR can indeed help improve the accuracy of ML classification models if we do it right. The work differs from the prior work in that it conducts CR on partial data (see Section 7 for more); it suggests that the community rethink about the need for *CR for ML in a fine-grained manner.*

(1) Impactful factors (Section 2). We find that when CR is accurate, it indeed improves the accuracy of ML classifications. It is most effective when conflicts appear in (a) influential attributes, *e.g.,* area and city attributes for predicting house rent, and/or (b) influential tuples for under-represented classes of entities in an unbalanced dataset $\mathcal{D}$, *e.g.,* unmarried female house owners among all unmarried fe-

males; we refer to such a class as *a minority group*. Conflicts in these attributes/tuples are responsible for 7.6% of accuracy reduction on average for classification. Moreover, CR on these attributes/tuples makes the ML models more accurate than on all attributes/tuples.

The findings motivate us to study the following problems.

(2) A framework (Section 3). We propose a creator-critic framework, Rock4ML, to integrate CR and ML training. It takes a differentiable model $\mathcal{M}$ and a dataset $\mathcal{D}$ of schema $\mathcal{R}$ as input, and iteratively cleans $\mathcal{D}$ and trains $\mathcal{M}$, in rounds. In the k-th round, the critic identifies influential attributes and tuples (Sections 4-5) by analyzing error signals, resolves conflicts, and obtains a dataset $\mathcal{D}_k$ from $\mathcal{D}$; the creator then incrementally trains $\mathcal{M}$ with $\mathcal{D}_k$. We show that Rock4ML converges at a stable state, and returns a dataset $\mathcal{D}_{\mathcal{M}}$ and a trained model $\mathcal{M}$ with $\mathcal{D}_{\mathcal{M}}$ that is more accurate than with $\mathcal{D}$.

One may employ Rock4ML to train any differential classification model $\mathcal{M}$, and may plug any CR method in Rock4ML. From our experimental study (Sections 2 and 6), we find that Rock4ML with an accurate CR method can indeed make $\mathcal{M}$ better, notably by fixing errors in influential attributes and influential tuples.

(3) Identifying influential attributes (Section 4). We formulate a problem that, given a set $\mathcal{D}$ of schema $\mathcal{R}$, a differentiable classification model $\mathcal{M}$ and a validation set C (selected by the critic of Rock4ML), identifies a set of influential attributes of $\mathcal{R}$ such that resolving their conflicts maximumly improves the accuracy of $\mathcal{M}$ on C. We show that unfortunately, its decision problem is intractable. Nonetheless, we develop a genetic algorithm that employs the boosting strategy [81] to select a sub-optimal set of attributes from the huge search space, without frequently retraining $\mathcal{M}$.

(4) Identifying influential tuples (Section 5). We formulate another problem that, given $\mathcal{M}$ and C, pinpoints tuples in minority groups of $\mathcal{D} \setminus C$ on which CR can maximumly improve the accuracy of $\mathcal{M}$ on C. We show that the decision version of this problem is also intractable. This said, we develop an algorithm to identify a set $\mathcal{T}$ of influential tuples by employing influence functions [5, 46]. We resolve conflicts in $\mathcal{T}$, estimate the update to $\mathcal{M}$'s parameters via influence functions and validate the updated $\mathcal{M}$ on C, such that the accuracy of $\mathcal{M}$ on C is maximumly improved, without actually retraining $\mathcal{M}$.

(5) Experimental evaluation (Section 6). Using different datasets, models and CR methods, we empirically find the following. On average, (a) Rock4ML improves the (relative) accuracy of various models by 43.1%. (b) By cleaning errors in both influential attributes and influential tuples, Rock4ML is 39.6% more accurate than cleaning the entire $\mathcal{D}$. (c) It is efficient given an efficient and accurate CR, *e.g.,* it takes 451.6s on $\mathcal{D}$ of 13K tuples with CR method Rock [9]; it even beats non-iterative baselines in some cases. (d) The more accurate CR is, the better Rock4ML improves the accuracy of $\mathcal{M}$.

We discuss related and future work in Sections 7-8 (proofs in [2]).

2 TWO IMPACTFUL FACTORS

This section reports an experimental study to reveal when conflicts (inconsistencies) may substantially affect the accuracy of models.

Preliminaries. We start with basic notations.

Datasets. Consider a database schema $\mathcal{R} = (R_1, \ldots, R_m)$, where R_j is a relation schema $R(A_1, \ldots, A_n, L)$, each A_i is an attribute, and L is an attribute denoting the classification label. A dataset $\mathcal{D}$ of $\mathcal{R}$ is $(D_1, \ldots, D_m)$, where D_i is a relation of R_i. Following [15], we assume *w.l.o.g.* that each tuple t in $\mathcal{D}$ represents an entity.

Accuracy. For an ML model $\mathcal{M}$ and a dataset $\mathcal{D}$, we use $\text{acc}(\mathcal{M}, \mathcal{D})$ to denote the accuracy of $\mathcal{M}$ that is trained (resp. evaluated) with a training (resp. testing) set of $\mathcal{D}$. More specifically, we train $\mathcal{M}$ with a subset $\mathcal{D}_{\text{train}}$ of $\mathcal{D}$, whose attributes and labels are denoted by X_{train} and Y_{train}, respectively. We evaluate $\mathcal{M}$ with a subset $\mathcal{D}_{\text{test}}$ of $\mathcal{D}$, denoted by $(X_{\text{test}}, Y_{\text{test}})$. Following [32, 45, 78], we assume that $\mathcal{D}_{\text{test}}$ is clean, while $\mathcal{D}_{\text{train}}$ may be erroneous.

We measure $\text{acc}(\mathcal{M}, \mathcal{D})$ in terms of macro-averaged F_1 score over all classes [1], *i.e.,* $\text{acc}(\mathcal{M}, \mathcal{D}) = \frac{1}{N} \sum_l 2 \times \frac{\text{recall}_l \times \text{precision}_l}{\text{recall}_l + \text{precision}_l}$, where N is the number of classes, l is a classification label, recall_l is the ratio of l-class tuples (*i.e.,* tuples with label l) correctly predicted by $\mathcal{M}$ to all l-class tuples, and precision_l is the ratio of l-class tuples correctly predicated by $\mathcal{M}$ to all tuples with predicated class l.

Given a dataset $\mathcal{D}$ of schema $\mathcal{R}$, an ML model $\mathcal{M}$ and a CR method $\mathcal{A}_{\text{CR}}$, we denote by $\mathcal{D}_{\text{CR}}$ the cleaned $\mathcal{D}$ via $\mathcal{A}_{\text{CR}}$. The difference between $\text{acc}(\mathcal{M}, \mathcal{D})$ and $\text{acc}(\mathcal{M}, \mathcal{D}_{\text{CR}})$ is the accuracy improvement of $\mathcal{M}$. In particular, if $\mathcal{A}_{\text{CR}}$ only cleans a subset $\mathcal{S}$ of attributes of $\mathcal{R}$ in all tuples of $\mathcal{D}$ (resp. all attributes of a subset $\mathcal{T}$ of tuples in $\mathcal{D}$), we denote the cleaned set by $\mathcal{D}^{\mathcal{S}}_{\text{CR}}$ (resp. $\mathcal{D}^{\mathcal{T}}_{\text{CR}}$), written as $\mathcal{D}^{\mathcal{S}}$ (resp. $\mathcal{D}^{\mathcal{T}}$) when $\mathcal{A}_{\text{CR}}$ is clear in the context.

We consider a set $\mathcal{S}$ (resp. $\mathcal{T}$) of *influential attributes* (resp. *influential tuples*) that help improve the accuracy of $\mathcal{M}$, as follows.

Influential attributes. Given a dataset $\mathcal{D}$ of schema $\mathcal{R}$, we say that $\mathcal{S} \subseteq \mathcal{R}$ is a set of *influential attributes* for $\mathcal{M}$ *w.r.t.* $\mathcal{D}$ if for any set $\mathcal{S}' \subseteq \mathcal{R}$, $\text{acc}(\mathcal{M}, \mathcal{D}^{\mathcal{S}}_{\text{CR}}) \geq \text{acc}(\mathcal{M}, \mathcal{D}^{\mathcal{S}'}_{\text{CR}})$. Intuitively, it means that cleaning the $\mathcal{S}$-attribute values in $\mathcal{D}$ maximumly improves the accuracy of $\mathcal{M}$, compared with other attributes. In Example 1, the set $\mathcal{S}$ in Adult schema consists of education (A_3) and occupation (A_5), since these attributes contain errors and have a strong impact on income. These errors may easily degrade the accuracy of $\mathcal{M}$, *e.g.,* $t_1[A_5]$ may mislead $\mathcal{M}$ to conclude that `Farming` is a high-paying job; and $t_4[A_3]$ may deceive $\mathcal{M}$ that `PhD` comes with low income.

Influential tuples. Dataset $\mathcal{D}$ can be partitioned into disjoint groups based on some attributes (*e.g.,* gender [39]), where each groups contain tuples from different classes. A *minority group* of $\mathcal{D}$ is a group of tuples with class l such that there exists another group of tuples with class l', and the number of l-class tuples is γ-fold smaller than the number of l'-class tuples, where γ is a user-defined threshold. Intuitively, l-class tuples are under-represented, *e.g.,* unmarried female house owners among all unmarried females.

We say that a dataset $\mathcal{D}$ is *unbalanced* if it contains minority groups. Trained with such $\mathcal{D}$, $\mathcal{M}$ typically has a lower precision for l-class tuples than for l'-class tuples and thus, $\mathcal{M}$ is often inaccurate, unfair and less robust to adversarial attacks in such datasets.

Given a dataset $\mathcal{D}$ of schema $\mathcal{R}$ and a subset $\mathcal{T} \subseteq \mathcal{D}$, we say that $\mathcal{T}$ is a set of *influential tuples* for $\mathcal{M}$ *w.r.t.* $\mathcal{D}$, if for any subset $\mathcal{T}' \subseteq \mathcal{D}$, $\text{acc}(\mathcal{M}, \mathcal{D}^{\mathcal{T}}_{\text{CR}}) \geq \text{acc}(\mathcal{M}, \mathcal{D}^{\mathcal{T}'}_{\text{CR}})$. Intuitively, cleaning tuples in $\mathcal{T}$ maximumly improves the accuracy of $\mathcal{M}$, compared with cleaning other tuples in $\mathcal{D}$. In Example 1, $\mathcal{T}$ is $\{t_1, t_6\}$, by partitioning the tuples based on category, where high-income individuals account for less than 30% in either private or part-time groups. Errors in $\mathcal{T}$ worsens the accuracy of $\mathcal{M}$, regardless of whether these errors are in influential attributes (*e.g.,* t_1) or non-influential ones (*e.g.,* t_6). For instance, $\mathcal{M}$ may be misled to conclude that private people in `farming` are well paid by t_1, and Part-time jobs get $> 50K$ salary by t_6.

Justification. The concepts of influential attributes and influential tuples defined above are different from the traditional notions of feature importance [14] and data valuation [63] in ML. While the latter assesses the contribution of features and data points to the model accuracy, we specifically target erroneous attributes and tuples to identify and rectify those that most significantly improve model accuracy after applying CR to them. Our notions are distinguished by being (a) *error-focused, i.e.,* we identify and correct errors in dedicated attributes and tuples to boost the model accuracy; (b) *data-dependent*, since the identification of influential attributes and tuples is guided by the actual data distribution, which may vary across different datasets; and (c) *flexible*, since multiple subsets of attributes or tuples can maximize accuracy improvement at the same time. Thus, defining influential attributes and influential tuples is essential for enhancing the model accuracy via precise data quality improvement. Furthermore, finding these influential attributes and tuples is uniquely challenging because it involves not only identifying and correcting errors in dirty data but also assessing the impact of these corrections on model performance, adding a distinct layer of intractability discussed in detail in Sections 4 and 5, respectively.

Experimental setting. We start with our experimental setting.

Datasets. Table 2 shows the tested datasets (see [2] for details). We checked all the datasets from previous work and tested those that are (a) (relative) clean and complete, and (b) imbalanced. As observed in [61], there is no "good" publicly available benchmark that contains (a) ground truth of real errors and (b) designated errors on certain attributes/tuples that show significant impacts on ML models. Therefore, following [5], we treat the released version of each dataset $\mathcal{D}$ as "clean", and manually inject noises into a few attributes (see below), where the amount of noises injected is controlled by a noise ratio. We will show that a small ratio of errors in $\mathcal{D}$ can significantly affect the accuracy of $\mathcal{M}$. We randomly split each $\mathcal{D}$ into $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ in $8:2$ ratio for training and testing, respectively, with seed = 42 for reproduction. We partition $\mathcal{D}_{\text{train}}$ based on one attribute, and obtain minority groups with $\gamma \geq 2$.

Noise injection. We defer the identification of influential attributes and tuples to Sections 4 and 5, respectively. We denote the dirty version of $\mathcal{D}_{\text{train}}$ as $\mathcal{D}^-_{\text{train}}$, created by injecting noise into (1) influential, non-influential, or random attributes and/or (2) influential, non-influential, or random tuples. For each value $t[A]$ to be corrupted, we randomly replace its original value c with another value c' ($\neq c$) in the domain $\mathsf{dom}(A)$ of attribute A (see [2] for details).

ML models. We test two widely-used anti-noise differential classification models MLPClassifier [62] and FT-Transformer [37] in their default setting. We use one-hot encoding to encode relation data for its popularity. We will report results on more models in Section 6.

CR. We use five CR methods $\mathcal{A}_{\text{CR}}$: (1) RB that uses Raha [57] to detect errors and Baran [56] to correct errors; (2) HoloClean [66] for error detection and correction; (3) T5 that employs the transformer-based model T5 [64] for error detection and correction; (4) Rock, a data cleaning system [9] for error detection and correction; and (5) Oracle, a "perfect" method that manually fixes all errors correctly by experts. Note that Oracle may not be feasible in all applications; we use it here just as a theoretical upper bound of CR methods. As reported in [9, 56], Rock and RB are considered accurate.

Datasets	$\|\mathcal{D}\|$	$\|\mathcal{R}\|$	Noise Ratio (%)	#classes
Adult [5]	48,842	15	2	2
Nursery [69]	12,960	8	2	4
Default [77]	30,000	24	2	2
German [39, 77]	1,000	20	3.2	2

Table 2: The tested datasets

Accuracy evaluation. For an ML model $\mathcal{M}$, we trained $\mathcal{M}$ with different training sets, *e.g.,* $\mathcal{D}^-_{\text{train}}$ (with injected noises), $\mathcal{D}^+_{\text{train}}$ ($\mathcal{D}^-_{\text{train}}$ cleaned by a CR method) and $\mathcal{D}_{\text{train}}$ (the "ground-truth" clean version), and tested $\mathcal{M}$ on $\mathcal{D}_{\text{test}}$. We treated the accuracy of $\mathcal{M}$ that is trained with $\mathcal{D}_{\text{train}}$ as the baseline, *e.g.,* when $\mathcal{M}$ trained with $\mathcal{D}^-_{\text{train}}$ has a lower accuracy than the baseline, the gap indicates that $\mathcal{M}$ is negatively affected by the injected noises in $\mathcal{D}^-_{\text{train}}$; equivalently, it suggests that CR can improve $\mathcal{M}$ by correcting such errors.

Configuration. We ran the experiments on a machine powered by 504GB RAM and 104 processors with Intel(R) Xeon(R) Gold 5320 CPU @2.20GHz. Each test was run 3 times; the average is reported.

Experimental results. We next report our findings. We present in Figure 1 (resp. Figure 2) the accuracy of the tested ML models trained with datasets $\mathcal{D}^-_{\text{train}}$ (resp. $\mathcal{D}^+_{\text{train}}$) with conflicts in different attributes/tuples (resp. cleaned with different CR methods)

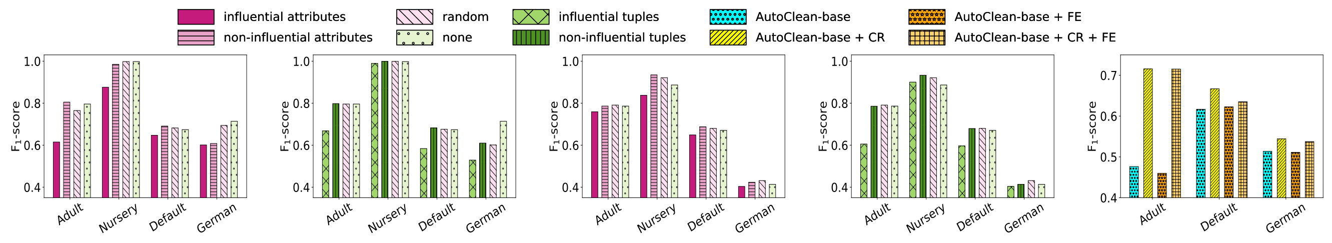
(1) Influential attributes. We generated dirty datasets $\mathcal{D}^-_{\text{train}}$ by injecting the same amount of noises into influential, non-influential and random attributes, respectively. In Figures 1(a) and 1(c), (a) conflicts in influential attributes (*e.g.,* education in Adult) indeed make $\mathcal{M}$ worse; its accuracy drops by 6.9% on average, up to 18.1%. (b) In contrast, conflicts in non-influential attributes (*e.g.,* gender in Adult) may even make $\mathcal{M}$ better, *e.g.,* the accuracy of FT-Transformer is even 1.7% higher than the baseline accuracy on Default dataset.

Finding #1: Errors in influential attributes have the most negative impacts on the accuracy of $\mathcal{M}$, since these attributes are essential to $\mathcal{M}$'s decision-making; in contrast, errors in non-influential ones are not decisive and may even make $\mathcal{M}$ more robust to noises.

(2) Influential tuples. We produced dirty datasets $\mathcal{D}^-_{\text{train}}$ by injecting the same amount of noises into (a) influential tuples (*i.e.,* tuples with a particular label l) in minority groups, (b) tuples in non-minority groups and (c) tuples in random groups, respectively. As shown in Figures 1(b) and 1(d), (a) conflicts in influential tuples (*e.g.,* private workers with income above 50K in Adult such as t_1 in Table 1) indeed make $\mathcal{M}$ less accurate, *e.g.,* the accuracy of $\mathcal{M}$ drops by 8.3% on average, up to 18.5%. (b) Conflicts in tuples of non-minority groups (*e.g.,* workers with Bachelor degrees) may even make $\mathcal{M}$ better, *e.g.,* the accuracy of FT-Transformer increases by 1% on Default.

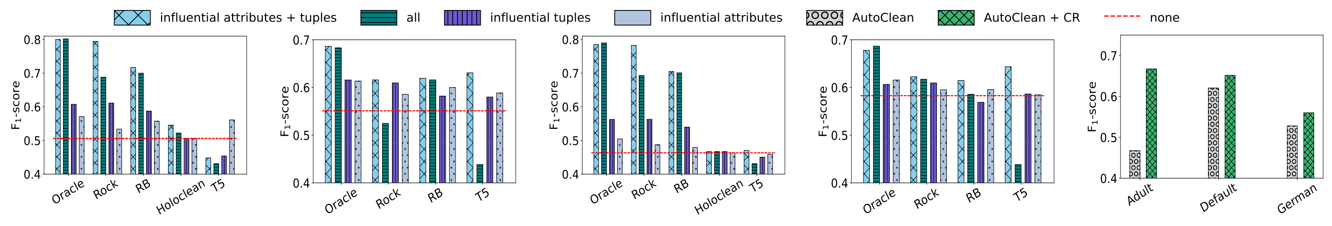
Finding #2: The accuracy of model $\mathcal{M}$ can be most negatively affected by errors in influential tuples, since these errors can largely disturb the distribution of small-sized minority groups; in contrast, errors in non-minority groups do not significantly disturb the data distribution and thereby make $\mathcal{M}$ more resilient to noises.

(3) Data cleaning via CR for ML. In Figure 2, we cleaned $\mathcal{D}^-_{\text{train}}$ of Adult and Default (consistent for other datasets), by fixing errors in influential attributes, influential tuples, both of them, and all tuples, respectively, using different CR methods, where the red dashed



(a) Attributes (Transformer) (b) Tuples (Transformer) (c) Attributes (MLP) (d) Tuples (MLP) (e) RB (AutoClean-base)

Figure 1: Errors on different attributes/tuples (accuracy)



(a) Adult (Transformer) (b) Default (Transformer) (c) Adult (MLP) (d) Default (MLP) (e) RB (AutoClean)

Figure 2: Cleaning errors via different CR methods (accuracy)

line denotes the accuracy of $\mathcal{M}$ trained with $\mathcal{D}^-_{\text{train}}$. The result of Holoclean on Default is not reported as it cannot fit in memory.

We find the following. (a) As opposed to prior work that claims CR does not help $\mathcal{M}$, we find that CR indeed helps no matter whether it cleans all tuples in $\mathcal{D}$ or part of $\mathcal{D}$, when CR is accurate, *e.g.,* Oracle, Rock and RB; the three improve the accuracy of FT-Transformer by 14.4%, 9.2% and 7.6% on average, up to 29.6%, 24.3% and 21.1%, respectively. (b) If CR is perfect (*i.e.,* 100% accurate like Oracle), CR on influential attributes and influential tuples is as effective as CR on entire $\mathcal{D}$, *e.g.,* Oracle improves the accuracy of MLPClassifier by 20.9% on average when it applies to influential attributes and influential tuples, only 0.7% lower than that when it applies to all tuples and attributes. (c) When CR is accurate but not perfect (*e.g.,* the accuracy is high but not 100%), it improves the accuracy of $\mathcal{M}$ the most if it is applied to both influential attributes and influential tuples, *e.g.,* Rock improves the accuracy of MLPClassifier by 18.0% on average, versus 13.2% when Rock is applied to all tuples and attributes. (d) If CR is inaccurate, it may not help as it may introduce more errors than it fixes, *e.g.,* T5 degrades the accuracy of $\mathcal{M}$ by 9.1% on average when applied to the entire $\mathcal{D}$. Only point (d) was observed in [51], but [51] makes the claim indistinguishably for all CR methods, not just for inaccurate CR.

Finding #3: It is practical to clean both influential attributes and influential tuples with a relatively accurate CR method, since practical CR methods in the real world are typically accurate but not perfect.

(4) Data cleaning via CR for AutoML. We studied whether CR can be replaced by non-cleaning operations (*e.g.,* feature engineering) in AutoClean [61], a SOTA AutoML approach. Denote by AutoClean-base the version of AutoClean with the feature engineering (FE) and data preparation components disabled. We equipped AutoClean-base with RB, FE and both of them, respectively, and fed it with dirty $\mathcal{D}^-_{\text{train}}$. As shown in Figure 1(e), RB improves the accuracy of AutoClean-base by 21.5% on average, while unexpectedly FE may degrade the accuracy, *e.g.,* by 3.5% on Adult. Moreover, we studied the impact of CR on improving AutoML, by feeding AutoClean [61] with $\mathcal{D}^-_{\text{train}}$ and its cleaned version $\mathcal{D}^+_{\text{train}}$ via RB, respectively. As shown in Figure 2(e), RB makes AutoClean more accurate by 42.9%, 5%, 6.1% on Adult, Default and German, respectively.

Finding #4: It helps AutoML to conduct CR prior to feature engineering, especially when conflicts exit in influential data. Hence, CR plays an irreplaceable role in enhancing AutoML, improving its accuracy via effective data preparation, boosting the impact of feature engineering and smoothing supported by AutoML.

Takeaway message. (1) As opposed to previous reports, we find that (1) the accuracy of $\mathcal{M}$ is substantially affected by conflicts in (a) influential attributes, and/or (b) influential tuples in minority groups in $\mathcal{D}$. These indicate when CR can have substantial impact on $\text{acc}(\mathcal{M}, \mathcal{D})$. (2) An accurate CR method helps $\mathcal{M}$, especially when applying to influential attributes/tuples. (3) When a CR method is not very accurate, it may do more harm than good for $\mathcal{M}$, since it may introduce more errors than they correct. (4) Accurate CR can improve AutoML; better still, as will be shown in Section 6, CR on influential data above can even beat sophisticated AutoML systems.

This suggests that we should identify attributes vertically and tuples horizontally in $\mathcal{D}$ to improve the downstream ML models. We study these two impactful factors in Sections 4 and 5, respectively.

3 A CREATOR-CRITIC FRAMEWORK

This section proposes Rock4ML, a creator-critic framework that integrates ML training and data cleaning. Given a model $\mathcal{M}$, Rock4ML aims to clean training data and train $\mathcal{M}$ with the data, such that $\mathcal{M}$ is more robust and accurate on unseen data. We focus on CR here, and consider differentiable classification ML models in this paper.

Overview. Rock4ML takes as input a model $\mathcal{M}$, a CR method $\mathcal{A}_{\text{CR}}$, a schema $\mathcal{R}$, a dirty dataset $\mathcal{D}$ of $\mathcal{R}$, two thresholds η_1 and η_2 for identifying clean training data for $\mathcal{M}$, and a number e of epochs. One can plug in any model $\mathcal{M}$ and any (accurate) method $\mathcal{A}_{\text{CR}}$, no matter whether it is based on rules, ML, or a hybrid of the two.

Rock4ML iterates a creator and a critic in *rounds* (Figure 3). In the k-th round, it works on a dirty dataset $\mathcal{D}_k$ and aims to produce a cleaner set $\mathcal{D}_{k+1}$ from $\mathcal{D}_k$, so that $\mathcal{D}_{k+1}$ provides more clean data to train $\mathcal{M}$. In particular, Rock4ML integrates CR and ML training,

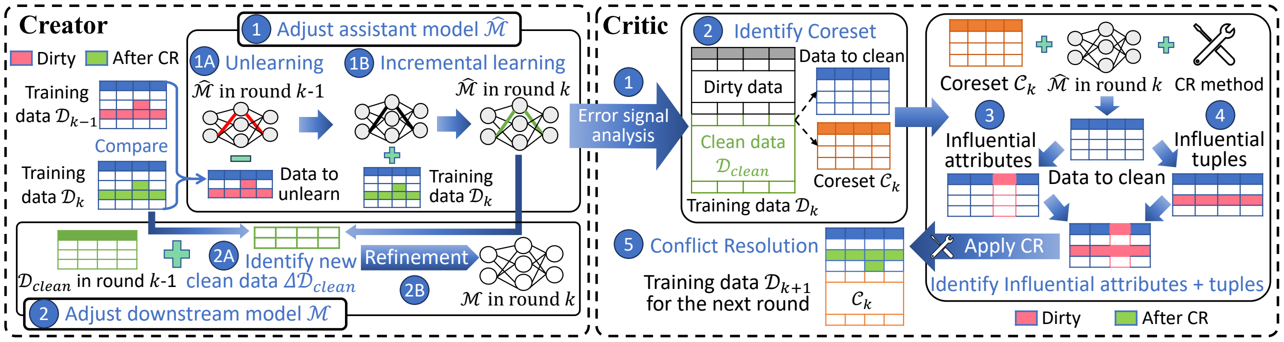

Creator
Dirty
After CR
Training data $\mathcal{D}_{k-1}$
Compare
Training data $\mathcal{D}_k$
1 Adjust assistant model $\hat{\mathcal{M}}$
1A Unlearning
1B Incremental learning
$\hat{\mathcal{M}}$ in round k-1
$\hat{\mathcal{M}}$ in round k
Data to unlearn
Training data $\mathcal{D}_k$
$\mathcal{D}_{clean}$ in round k-1
2A Identify new clean data $\Delta\mathcal{D}_{clean}$
Refinement
2B
$\mathcal{M}$ in round k
2 Adjust downstream model $\mathcal{M}$
Critic
1 Error signal analysis
2 Identify Coreset
Data to clean
Dirty data
Clean data $\mathcal{D}_{clean}$
Coreset $\mathcal{C}_k$
Training data $\mathcal{D}_k$
Coreset $\mathcal{C}_k$
$\hat{\mathcal{M}}$ in round k
CR method
3 Influential attributes
4 Influential tuples
Data to clean
5 Conflict Resolution
Apply CR
Training data $\mathcal{D}_{k+1}$ for the next round
$\mathcal{C}_k$
Identify Influential attributes + tuples
Dirty
After CR

Figure 3: Overview of Rock4ML

where the creator focuses on ML training and the critic targets data cleaning. Initially, we set $\mathcal{D}_0 = \mathcal{D}$. Besides, we maintain the following datasets/models during the iterative process of Rock4ML:

- A set $\mathcal{D}_{\text{clean}}$, which accumulates clean data obtained so far.
- Model $\mathcal{M}$, the downstream model that we ultimately want and $\mathcal{M}$ is only trained on the clean data $\mathcal{D}_{\text{clean}}$ in each round.
- Model $\hat{\mathcal{M}}$, an assistant model with the same architecture and initial parameters as $\mathcal{M}$. Here $\hat{\mathcal{M}}$ is incrementally trained on (dirty) $\mathcal{D}_k$ in each round, and is used to identify (a) influential data in $\mathcal{D}_k$ to clean and (b) new clean data to be added to $\mathcal{D}_{\text{clean}}$.

This process continues until $\mathcal{D}_k$ cannot be further cleaned; the set $\mathcal{D}_k$ and the tuned $\mathcal{M}$ are returned as the results. More specifically, in the k-th round, the creator and the critic work as follows.

The creator is responsible for the *model training* for $\mathcal{M}$ and $\hat{\mathcal{M}}$:

(1) Given the assistant model $\hat{\mathcal{M}}$ that has been trained on dirty data $\mathcal{D}_{k-1}$ in the previous round, we first update $\hat{\mathcal{M}}$ on $\mathcal{D}_k$ (*i.e.,* the dataset cleaned from $\mathcal{D}_{k-1}$) to identify influential data.

(2) For the downstream model $\mathcal{M}$, we first *identify new clean data from* $\mathcal{D}_k$ based on the updated $\hat{\mathcal{M}}$, and accumulate them in $\mathcal{D}_{\text{clean}}$. Then, we conduct *model refinement* on $\mathcal{M}$.

The critic is responsible for *data cleaning* in five steps:

(1) It first computes the *error signal* of each tuple in $\mathcal{D}_k$ using $\hat{\mathcal{M}}$.
(2) It finds a set of relatively clean data from $\mathcal{D}_k$ as *coreset* [12, 58].
(3) It identifies *influential attributes* with error signals and coresets.
(4) Similarly, it identifies *influential tuples* in $\mathcal{D}_k$.
(5) Finally, it uses $\mathcal{A}_{\text{CR}}$ to fix errors in those attributes/tuples, and returns a cleaner $\mathcal{D}_{k+1}$ to train $\mathcal{M}$ and $\hat{\mathcal{M}}$ in the next round.

Rock4ML guarantees to converge at a stable state. Below we present its model training and data cleaning in Section 3.1 and its workflow and property in Section 3.2 (see [2] for more details).

3.1 Model Training and Data Cleaning

Creator. The creator incrementally (a) trains $\hat{\mathcal{M}}$ with the updated $\mathcal{D}_k$ such that $\hat{\mathcal{M}}$ can provide more accurate signals to distinguish dirty/clean tuples, and (b) trains $\mathcal{M}$ with clean data so that it can generalize to unseen data. Its input and output are as follows.

- *Input*: Models $\hat{\mathcal{M}}$ and $\mathcal{M}$ trained in (k-1)-th round, $\mathcal{D}_k$ and $\mathcal{D}_{k\text{-}1}$ from the (k-1)-th and (k-2)-th rounds, respectively, $\mathcal{D}_{\text{clean}}$ accumulated so far, thresholds η_1 and η_2, and a number e of epochs.
- *Output*: An extended $\mathcal{D}_{\text{clean}}$ with more clean data (controlled by thresholds η_1 and η_2), two updated models $\hat{\mathcal{M}}$ and $\mathcal{M}$ incrementally trained using $\mathcal{D}_k$ and the extended $\mathcal{D}_{\text{clean}}$, respectively.

Below we outline the training of $\hat{\mathcal{M}}$ and $\mathcal{M}$ (see more in [2]).

(1) Assistant model $\hat{\mathcal{M}}$. The training for $\hat{\mathcal{M}}$ consists of two parts: (A) *model unlearning* and (B) *incremental learning*, to eliminate the negative effect of dirty tuples in $\mathcal{D}_{k-1}$ (on which $\hat{\mathcal{M}}$ is previously trained) and to update $\hat{\mathcal{M}}$ with cleaned tuples in $\mathcal{D}_k$, respectively.

(1A) Model unlearning. Denote by $\Delta\mathcal{D}_k$ the set of tuples in $\mathcal{D}_{k-1}$ that are cleaned by $\mathcal{A}_{\text{CR}}$ in $\mathcal{D}_k$: $\Delta\mathcal{D}_k = \{t \mid t \in \mathcal{D}_{k-1}$ and $t \notin \mathcal{D}_k\}$; *i.e.,* $\Delta\mathcal{D}_k$ contains biased/dirty tuples. To debias the negative effects of $\Delta\mathcal{D}_k$ from $\hat{\mathcal{M}}$, we adopt the influence function of [46], an effective tool to estimate parameter change of a model when tuples are removed/modified in the training data. Intuitively, it mimics the scenario that we retrain $\hat{\mathcal{M}}$ with a smaller training data, $\mathcal{D}_{k-1} \setminus \Delta\mathcal{D}_k$, in the $(k-1)$-th round, by removing dirty tuples in $\Delta\mathcal{D}_k$ from $\mathcal{D}_{k-1}$.

(1B) Incremental learning. Previous incremental strategies [53, 76] mainly focus on retaining the performance of $\hat{\mathcal{M}}$ on both old and new datasets, *i.e.,* $\mathcal{D}_{k-1}$ and $\mathcal{D}_k$. However, since $\mathcal{D}_k$ is cleaned from $\mathcal{D}_{k-1}$, we adopt a simple strategy that continuously fine-tunes $\hat{\mathcal{M}}$ with updated $\mathcal{D}_k$ in e epochs, where $\hat{\mathcal{M}}$ inherits the parameters from the model unlearning step. This strategy works well since the negative effects of dirty data in $\mathcal{D}_{k-1}$ have already been removed.

(2) Downstream model $\mathcal{M}$. We train $\mathcal{M}$ on accumulated $\mathcal{D}_{\text{clean}}$ by (A) *identifying new clean data* and (B) *model refinement* as follows.

(2A) Identifying new clean data. The creator first identifies new clean data $\Delta\mathcal{D}_{\text{clean}}$ from $\mathcal{D}_k$ with the help of $\hat{\mathcal{M}}$. To do this, we initialize $\Delta\mathcal{D}_{\text{clean}}$ to be empty and compute the dynamic loss $L_k(t)$ for each tuple t in $\mathcal{D}_k$ [82], using the exponential moving average (EMA):

$$L_k(t) = \lambda \cdot l(\hat{\mathcal{M}}(x;\theta_k), y) + (1-\lambda) \cdot L_{k-1}(t),$$

where $t = (x, y) \in \mathcal{D}_k$, $\lambda \in [0, 1]$ is a discounting factor, and $L_k(t_i)$ is the accumulated loss of t from 0 to k epochs. Intuitively, the larger $L_k(t)$, the more likely t is dirty. For each tuple $t \in \mathcal{D}_k$, we monitor the dynamic loss $L_k(t)$ of $\hat{\mathcal{M}}$. Given thresholds η_1 and η_2, we add a tuple t into $\Delta\mathcal{D}_{\text{clean}}$ if one of the following is satisfied:

- $L_{k-1}(t) - L_k(t) \geq \eta_1$, indicating that $\mathcal{A}_{\text{CR}}$ has effectively enhanced t's quality and thus t can be considered as a clean tuple.
- $L_k(t) \leq \eta_2$, showing that no matter whether t has been cleaned by $\mathcal{A}_{\text{CR}}$ or not, it is likely to be clean due to its low $L_k(t)$.

(2B) Model refinement. Rock4ML adopts Stochastic Gradient Descent (SGD) [68] for iterative model refinement, to leverage insights from both new clean data $\Delta\mathcal{D}_{\text{clean}}$ and the accumulated clean data $\mathcal{D}_{\text{clean}}$. We adapt SGD for continuous integration of clean data, such that $\mathcal{M}$ can be progressively improved with more clean data accumulated.

Finally, we accumulate $\mathcal{D}_{\text{clean}}$, *i.e.,* $\mathcal{D}_{\text{clean}} = \Delta\mathcal{D}_{\text{clean}} \cup \mathcal{D}_{\text{clean}}$.

Critic. Based on updated $\hat{\mathcal{M}}$ from the creator, the critic analyzes the error signals and retrieves a relatively clean subset from $\mathcal{D}_k$ as the *coreset* [12, 58]. Here we adopt the concept of coreset in the ML community, and use it as validation data for identifying influential attributes (see Section 4) and influential tuples (Section 5) from $\mathcal{D}_k$ on which the CR method $\mathcal{A}_{\text{CR}}$ is applied. It returns a cleaner set $\mathcal{D}_{k+1}$ for incrementally training $\hat{\mathcal{M}}$ and $\mathcal{M}$ in the next round.

- *Input:* Updated $\hat{\mathcal{M}}$, $\mathcal{D}_k$ from the $(k-1)$-th round, and $\mathcal{A}_{\text{CR}}$.
- *Output:* A cleaner set $\mathcal{D}_{k+1}$ for the next round.

More specifically, the critic consists of the following five steps.

(1) Error signal analysis. Intuitively, the error signal of a tuple $t \in \mathcal{D}_k$, denoted by $\text{err}(t)$, represents the degree of error possibly incurred by training $\hat{\mathcal{M}}$ with $t \in \mathcal{D}_k$. To compute the error signal of t, we transform the dynamic loss $L_k(t)$ to a softmax probability, *i.e.,*

$$\text{err}(t) = \mathcal{P}(t) = \frac{\exp[-\pi L_k(t)]}{\sum_{t' \in \mathcal{D}_k} \exp[-\pi L_k(t')]}$$

where π is a parameter that controls the probability distribution.

(2) Coreset identification. Based on the error signals, the critic identifies a coreset $C_k \subseteq \mathcal{D}_k$. Following [82], we add a tuple $t \in \mathcal{D}_k$ into C_k with probability $\mathcal{P}(t) = \text{err}(t)$, such that C_k retains the underlying distribution and the diversity of $\mathcal{D}_k$. Note that if a dirty tuple is added into C_k, it has a high probability to be excluded from C_{k+1} in the next round, and thus it can be eventually detected and cleaned.

(3) Identifying influential attributes. Given the coreset C_k, the critic identifies a set of influential attributes in $\mathcal{R}$ so that if we resolve conflicts in those attributes using $\mathcal{A}_{\text{CR}}$ and fine-tune $\hat{\mathcal{M}}$ on $\mathcal{D}_k \setminus C_k$, the accuracy of $\hat{\mathcal{M}}$ is maximumly improved on C_k. However, this is non-trivial: (a) frequently fine-tuning $\hat{\mathcal{M}}$ is costly. To address this, we adopt the boosting strategy [81] to train a lightweight model for updating $\hat{\mathcal{M}}$. (b) The problem of identifying influential attribute is NP-hard. We propose a genetic algorithm to find a sub-optimal solution.

(4) Pinpointing influential tuples. Detecting influential tuples is another NP-hard problem, which is also costly since we need to frequently update the model (due to data modification) to decide what tuples are truly influential. To reduce the cost, the critic adopts the influence function of [5, 46] to update $\hat{\mathcal{M}}$ when $\mathcal{D}_k$ is partially modified, and develops a greedy strategy to iteratively select tuples from $\mathcal{D}_k \setminus C_k$ until the accuracy of $\hat{\mathcal{M}}$ on C_k cannot be further improved.

(5) CR. Finally, we adopt $\mathcal{A}_{\text{CR}}$ to resolve the conflicts in the influential attributes of all tuples and all the attributes of influential tuples, yielding a cleaner set $\mathcal{D}_{k+1}$ for model training in the next round.

Example 2: Consider the set $\mathcal{D} = \{t_1\text{-}t_6\}$ in Table 1. In the first round, the creator trains $\hat{\mathcal{M}}$ using dirty $\mathcal{D}_0 = \mathcal{D}$ and outputs $\hat{\mathcal{M}}$ to the critic, which executes the five steps above as follows. It first (1) computes the error signal $\text{err}(t)$ of each tuple t and (2) samples tuples with probability $\text{err}(t)$. Assume that $C_0 = \{t_2, t_3, t_5\}$ is identified as the coreset. The critic then (3) identifies a set of influential attributes, *e.g.,* $\{A_3, A_5\}$, and (4) pinpoints the set of influential tuples, *e.g.,* $\mathcal{T} = \{t_1\}$, using C_0 as the validation data. Finally, (5) the critic resolves the conflicts in those attributes/tuples via $\mathcal{A}_{\text{CR}}$ and obtains a cleaner set $\mathcal{D}_1$ by updating $t_1[A_5]$ = `Exec-managerial` and $t_4[A_3]$ = `Bachelor` in $\mathcal{D}_1$. The first iteration ends. □

```
Input: An ML model $\mathcal{M}$, a data schema $\mathcal{R}$, a set $\mathcal{D}$ of $\mathcal{R}$, a CR method $\mathcal{A}_{\text{CR}}$,
       parameters $\eta_1$ and $\eta_2$ to identify clean data, the number $e$ of epochs.
Output: A fine-tuned ML model $\mathcal{M}$ and a cleaned set $\mathcal{D}_{\text{clean}}$.
1.  $k := 0; \mathcal{D}_0 := \mathcal{D}; \hat{\mathcal{M}} := \mathcal{M}; L_k := \emptyset; \mathcal{D}_{\text{clean}} := \emptyset; \Delta\mathcal{D}_{\text{clean}} := \emptyset$;
2.  while true do
      /* Creator */
3.      $\Delta\mathcal{D}_k$ := CheckUpdates($\mathcal{D}_k, \mathcal{D}_{k-1}$);
4.      Model unlearning of $\hat{\mathcal{M}}$ with $\Delta\mathcal{D}_k$;
5.      Incrementally fine-tune $\hat{\mathcal{M}}$ with $\mathcal{D}_k$ in $e$ epochs;
6.      Compute $L_k(t)$ for all $t \in \mathcal{D}_k / \mathcal{D}_{\text{clean}}$;
7.      $\Delta\mathcal{D}_{\text{clean}}$ := CleanDataIdentify($\mathcal{D}_k/\mathcal{D}_{\text{clean}}, L_k, L_{k-1}, \eta_1, \eta_2$);
8.      if $\Delta\mathcal{D}_{\text{clean}} \neq \emptyset$ then
9.          Train $\mathcal{M}$ with $\mathcal{D}_{\text{clean}}$ and $\Delta\mathcal{D}_{\text{clean}}$ in $e$ epochs;
10.     $\mathcal{D}_{\text{clean}} := \mathcal{D}_{\text{clean}} \cup \Delta\mathcal{D}_{\text{clean}}$;   $\Delta\mathcal{D}_{\text{clean}} := \emptyset$;
      /* Critic */
11.     err := ErrSignalsAnalysis($\hat{\mathcal{M}}, \mathcal{D}_k$);
12.     $C_k$ := CoresetIdentify($\mathcal{D}_k$, err);
13.     $\mathcal{S}$ := InfluentialAttributes($\mathcal{D}_k, \mathcal{R}, \hat{\mathcal{M}}, \mathcal{A}_{\text{CR}}, C_k, \mathcal{D}_{\text{clean}}$);
14.     $\mathcal{T}$ := InfluentialTuples($\mathcal{D}_k, \hat{\mathcal{M}}, \mathcal{A}_{\text{CR}}, C_k, \mathcal{D}_{\text{clean}}$);
15.     $\mathcal{D}_{k+1} := \mathcal{A}_{\text{CR}}(\mathcal{D}_k, \mathcal{T}, \mathcal{S})$;
16.     if $\mathcal{D}_{k+1}$ remains unchanged from $\mathcal{D}_k$ then
17.         $\mathcal{D}_{\text{clean}} := \mathcal{D}_k$;  Train $\mathcal{M}$ with $\mathcal{D}_{\text{clean}}$ in $e$ epochs;
18.         return ($\mathcal{M}, \mathcal{D}_{\text{clean}}$);
19.     $k := k + 1$;
```

Figure 4: The workflow of Rock4ML

3.2 Workflow and Property

As shown in Figure 4, Rock4ML takes as input $\mathcal{M}$, $\mathcal{R}$, $\mathcal{D}$, $\mathcal{A}_{\text{CR}}$, η_1, η_2, and the number e of epochs. It outputs a cleaned $\mathcal{D}_{\mathcal{M}}$ (*i.e.,* $\mathcal{D}_{\text{clean}}$) and a fine-tuned $\mathcal{M}$ with $\mathcal{D}_{\mathcal{M}}$. It initializes $\mathcal{D}_0 = \mathcal{D}$, $\hat{\mathcal{M}}=\mathcal{M}$, $\mathcal{D}_{\text{clean}}=\emptyset$ and $\Delta\mathcal{D}_{\text{clean}}=\emptyset$ (line 1). It trains $\hat{\mathcal{M}}$ and $\mathcal{M}$ with creator, and cleans $\mathcal{D}$ with critic, in rounds (lines 2-15). In each round, the creator first computes $\Delta\mathcal{D}_k$, the differences between $\mathcal{D}_k$ and $\mathcal{D}_{k-1}$ (line 3). Then it eliminates the effect of $\Delta\mathcal{D}_k$ from $\hat{\mathcal{M}}$ via model unlearning (line 4) and incrementally trains $\hat{\mathcal{M}}$ with $\mathcal{D}_k$ in e epochs (line 5). It computes the dynamic loss $L_k(t)$ for each tuple in $\mathcal{D}_k$ (line 6), and identifies new clean data $\Delta\mathcal{D}_{\text{clean}}$ (line 7). If $\Delta\mathcal{D}_{\text{clean}} \neq \emptyset$, it trains $\mathcal{M}$ with $\mathcal{D}_{\text{clean}}$ and $\Delta\mathcal{D}_{\text{clean}}$ in e epochs (lines 8-9). It updates $\mathcal{D}_{\text{clean}}=\mathcal{D}_{\text{clean}} \cup \Delta\mathcal{D}_{\text{clean}}$, and $\Delta\mathcal{D}_{\text{clean}}=\emptyset$ (line 10).

The critic is then executed to find what data in $\mathcal{D}_k$ should be cleaned to improve the accuracy of $\hat{\mathcal{M}}$. To this end, the critic first analyzes the error signals (line 11), based on which it identifies a relatively clean subset C_k of $\mathcal{D}_k$ as the coreset (line 12). The coreset C_k is used as the validation data to identify a subset $\mathcal{S}$ of influential attributes and a subset $\mathcal{T}$ of influential tuples (lines 13-14). The CR method $\mathcal{A}_{\text{CR}}$ is then applied on these attributes and tuples to get a better dataset $\mathcal{D}_{k+1}$ for the next round (line 15).

The process proceeds until it reaches a *stable state*, *i.e.,* $\mathcal{D}_k$ is no longer changed (lines 16-18). Then we take $\mathcal{D}_k$ as the cleaned set $\mathcal{D}_{\text{clean}}$ tailored for $\mathcal{M}$ and fine-tune $\mathcal{M}$ on $\mathcal{D}_{\text{clean}}$ (line 17). Finally, the improved $\mathcal{M}$ and the cleaned $\mathcal{D}_{\text{clean}}$ are returned (line 18).

Termination. Rock4ML converges at a stable state provided that $\mathcal{A}_{\text{CR}}$ is α-accurate ($\alpha \in (0.5, 1]$), *i.e.,* it correctly fixes erroneous data with probability α; the cleaned data does not raise error signal.

Theorem 1: *Given an α-accurate $\mathcal{A}_{\text{CR}}$,* Rock4ML *guarantees to terminate and returns (a) a cleaned set $\mathcal{D}_{\text{clean}}$ as $\mathcal{D}_{\mathcal{M}}$, and (b) a model $\mathcal{M}$ trained with $\mathcal{D}_{\text{clean}}$ such that* $\text{acc}(\mathcal{M}, \mathcal{D}_{\text{clean}}) > \text{acc}(\mathcal{M}, \mathcal{D})$. □

Proof sketch: Observe the following: (1) $\mathcal{A}_{\text{CR}}$ is α-accurate ($\alpha >$ 50%); hence it cleans data and approaches Oracle (Section 2) that fixes all errors through iterations; (2) the convergence of the SGD is guaranteed by unbiased gradient estimates and Robbins-Monro step-size conditions, and (3) training models with clean data, as opposed to dirty data, approximates the true data distribution and better test accuracy [48, 59, 72]. Taken together, these guarantee that by iteratively applying $\mathcal{A}_{\text{CR}}$ and optimizing model parameters, Rock4ML terminates with $\mathcal{D}_{\mathcal{M}}$ that improves the accuracy of $\mathcal{M}$. □

Example 3: Continuing with Example 2, given that that tuples t_1 and t_4 are updated in the second round ($\Delta\mathcal{D}_1 = \{t_1, t_4\}$), the creator first eliminates their effect from $\hat{\mathcal{M}}$ by model unlearning, and incrementally learns $\hat{\mathcal{M}}$ on $\mathcal{D}_1$. The critic receives $\hat{\mathcal{M}}$ and re-computes the error signals, based on which a new C_1 is identified. Assume that η_1=0.04, η_2=0.02, $L_1(t_1)$=0.04, $L_0(t_1)$=0.08, $L_1(t_5)$=0.01. Then t_1 and t_5 are added to $\Delta\mathcal{D}_{\text{clean}}$ since (a) t_1 is enhanced by $\mathcal{A}_{\text{CR}}$ with high loss reduction (*i.e.,* $L_1(t_1) - L_0(t_1) \geq \eta_1$), and (b) t_5 has a low loss (*i.e.,* $L_1(t_5) \leq \eta_2$). After identifying $\Delta\mathcal{D}_{\text{clean}}$, $\mathcal{M}$ is trained with $\Delta\mathcal{D}_{\text{clean}}$ and $\mathcal{D}_{\text{clean}}$ (= $\emptyset$) and we update $\mathcal{D}_{\text{clean}} = \mathcal{D}_{\text{clean}} \cup \Delta\mathcal{D}_{\text{clean}}$. After identifying influential attributes and influential tuples as before, we get a cleaned set $\mathcal{D}_2$ by fixing $t_6[A_4]$ = 50. Since all conflicts are now resolved, $\mathcal{D}_2$ is stable in the next round. Finally, we incrementally train $\mathcal{M}$ on $\mathcal{D}_{\mathcal{M}} = \mathcal{D}_2$, to make $\mathcal{M}$ better. □

Complexity. Rock4ML takes $O(|R|\cdot|\mathcal{D}_k|\cdot(|\mathcal{D}_k|\cdot c_{\text{ul}}+2e\cdot c_{\text{ft}}+5|\mathcal{D}_k|+ c_{\text{IA}}+c_{\text{IT}}+c_{\text{CR}})+e\cdot c_{\text{ft}})$ time, where c_{ul} is the cost of eliminating the effect of a dirty tuple in $\mathcal{D}_{k-1}$ from $\hat{\mathcal{M}}$, c_{ft} is the cost of fine-tuning $\hat{\mathcal{M}}$ (resp. $\mathcal{M}$) with $\mathcal{D}_k$ (resp. $\mathcal{D}_{\text{clean}} \cup \Delta\mathcal{D}_{\text{clean}}$) in one epoch, e is the number of epochs, c_{IA} is the cost of finding a set of influential attributes in $\mathcal{R}$, c_{IT} is the cost of getting a set of influential tuples in $\mathcal{D}_k$, and c_{CR} is the cost of cleaning $\mathcal{D}_k$ by $\mathcal{A}_{\text{CR}}$. Here c_{CR} is often the most costly factor. We will see in Section 6 that the Rock4ML is efficient when the embedded $\mathcal{A}_{\text{CR}}$ and $\mathcal{M}$ are efficient.

4 IDENTIFYING INFLUENTIAL ATTRIBUTES

This section formulates the problem to identify influential attributes, show its intractability, and develops an algorithm to solve it.

Problem. For each dataset $\mathcal{D}_k$ of schema $\mathcal{R}$, we split it into a training set $\mathcal{B}_k$ and a validation set C_k; here C_k is the coreset identified via error signals, and $\mathcal{B}_k = \mathcal{D}_k \setminus C_k$. With slight abuse of notations, we also use $\text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k)$ to denote the accuracy of $\hat{\mathcal{M}}$ that is trained/fine-tuned (resp. evaluated) with $\mathcal{B}_k$ (resp. C_k).

Given $\mathcal{D}_k = (\mathcal{B}_k, C_k)$, a CR method $\mathcal{A}_{\text{CR}}$, and a model $\hat{\mathcal{M}}$, we identify a set $\mathcal{S}$ of influential attributes, such that if we resolve conflicts in the $\mathcal{S}$-attributes in $\mathcal{B}_k$, the accuracy of $\hat{\mathcal{M}}$ on C_k is maximumly improved. Denote the cleaned $\mathcal{B}_k$ by $\mathcal{B}_k^{\mathcal{S}}$. Then we obtain a cleaned version of $\mathcal{D}_k$, denoted by $\mathcal{D}_k^{\mathcal{S}}$, *i.e.,* $\mathcal{D}_k^{\mathcal{S}} = (\mathcal{B}_k^{\mathcal{S}}, C_k)$.

The *problem of identifying influential attributes* (IA) is as follows.

- *Input*: $\mathcal{R}, \mathcal{D}_k = (\mathcal{B}_k, C_k)$, $\hat{\mathcal{M}}$ and $\mathcal{A}_{\text{CR}}$ as described above.
- *Output*: A set $\mathcal{S}$ of influential attributes from schema $\mathcal{R}$.
- *Objective*: Maximize $\text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{S}}) - \text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k)$.

NP-hardness. It is nontrivial to find the optimal set of influential attributes. The decision problem, also referred to as IA, is to decide, given $\mathcal{R}, \mathcal{D}_k = (\mathcal{B}_k, C_k)$, $\hat{\mathcal{M}}$, $\mathcal{A}_{\text{CR}}$ and a bound δ, whether there is a set $\mathcal{S}$ of attributes such that $\text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{S}}) - \text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) \geq \delta$. This is intractable, even when training/testing $\hat{\mathcal{M}}$ takes polynomial time.

Theorem 2: *Problem* IA *is* NP*-hard.* □

Proof sketch: We prove the NP-hardness of IA by reduction from the NP-complete problem 3-SAT [17]. Given a 3-SAT instance ϕ, we construct a set $\mathcal{D}_k$ of schema $\mathcal{R}$ so that each $A \in \mathcal{R}$ encodes a variable in ϕ; we also develop a model $\hat{\mathcal{M}}$ and a CR method $\mathcal{A}_{\text{CR}}$. We show that there is a set $\mathcal{S} \subseteq \mathcal{R}$ with $\text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{S}}) - \text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) \geq \delta$ if and only if there exists a truth assignment that satisfies ϕ. □

Naive approach. A naive approach to solving IA is to greedily and iteratively add attributes $A \in \mathcal{R}$ into $\mathcal{S}$, one attribute per round, so that cleaning the A-attributes of $\mathcal{B}_k$ via $\mathcal{A}_{\text{CR}}$ can maximumly improve the accuracy of $\hat{\mathcal{M}}$ on C_k in each round. This process terminates when cleaning any more attribute cannot make $\hat{\mathcal{M}}$ better. Unfortunately, this method is costly, since $\hat{\mathcal{M}}$ needs to be retrained $O(|\mathcal{R}||\mathcal{S}|)$ times, where $|\mathcal{S}|$ (resp. $|\mathcal{R}|$) is the size of $\mathcal{S}$ (resp. $\mathcal{R}$).

Our approach. Adapting the boosting strategy [81], we do not retrain $\hat{\mathcal{M}}$ every time when an attribute is added. Instead, we first pre-clean $\mathcal{B}_k$ as $\mathcal{B}_k^{\mathcal{R}}$, by cleaning all attributes in $\mathcal{R}$ via $\mathcal{A}_{\text{CR}}$; for a set $\mathcal{S}$ of attributes, we train a lightweight tree-based model $\hat{\mathcal{M}}_{\text{add}}$ (*e.g.,* GBDT [81]) with the cleaned $\mathcal{S}$-attributes of $\mathcal{B}_k^{\mathcal{R}}$ (*i.e.,* a projected set of $\mathcal{B}_k^{\mathcal{R}}$ with schema $\mathcal{S}$), where $\hat{\mathcal{M}}_{\text{add}}$ aims to fit the misalignment between actual labels and predicated labels by $\hat{\mathcal{M}}$. We combine (two weaker) $\hat{\mathcal{M}}$ and $\hat{\mathcal{M}}_{\text{add}}$ as an integrated/stronger model $\hat{\mathcal{M}}_{\text{int}}$ by weighted averaging of predicated labels from $\hat{\mathcal{M}}$ and $\hat{\mathcal{M}}_{\text{add}}$, and validate $\hat{\mathcal{M}}_{\text{int}}$ in C_k without retaining $\hat{\mathcal{M}}$. The set $\mathcal{S}$ that gives the most accurate $\hat{\mathcal{M}}_{\text{int}}$ on C_k is returned as the solution.

However, since there are $|\mathcal{R}|!$ attribute combinations for $\mathcal{S}$, it can still be costly to train numerous $\hat{\mathcal{M}}_{\text{add}}$. To reduce the search space, we develop a genetic algorithm to find a sub-optimal set $\mathcal{S}$, by running a generation step and a selection step iteratively.

<u>*(a) Candidate set generation.*</u> In each round, we generate a set $\mathcal{G} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_i\}$ of at most $\frac{m\cdot(m-1)}{2}$ attribute sets, where m is a bound picked by users. In the first round, each $\mathcal{S}_i$ in $\mathcal{G}$ consists of a single attribute in $\mathcal{R}$ while in subsequent rounds, $\mathcal{G}$ is generated from the sets in previous rounds via two operators: mutation and crossover; here mutation randomly replaces an attribute in a set with another attribute (*e.g.,* we replace A_2 in $\{A_1, A_2\}$ with A_3 and get $\{A_1, A_3\}$), and crossover merges two attribute sets into one (*e.g.,* we merge $\{A_1, A_2\}$ and $\{A_1, A_3\}$ to get $\{A_1, A_2, A_3\}$). Following the evolution principle of reproduction [8, 18], each iteration starts with crossover and then invokes mutation with a probability.

<u>*(b) Candidate set selection.*</u> We then adopt the boosting strategy [81] to evaluate each set in $\mathcal{G}$ and retain the best (at most) m ones in $\mathcal{G}$ for set generation in the next round. More specifically, for each $\mathcal{S}_i$ in $\mathcal{G}$, we train the corresponding model $\hat{\mathcal{M}}_{\text{add}}$, combine it with $\hat{\mathcal{M}}$ to get the integrated model $\hat{\mathcal{M}}_{\text{int}}$, and evaluate the accuracy of $\hat{\mathcal{M}}_{\text{int}}$ with C_k. At most m sets with the highest accuracy are retained in $\mathcal{G}$. Moreover, we also maintain the set $\mathcal{S}_{\text{best}}$ with highest accuracy so far, and return it as the final solution when we reach the maximum number of rounds. Note that the best accuracy of $\hat{\mathcal{M}}$ is improved monotonically in this process, since (a) C_k is fixed, and (b) $\mathcal{S}_{\text{best}}$ is updated only when there exists another set with a higher accuracy.

```
Input: A schema $\mathcal{R}$, a dataset $\mathcal{D}_k = (\mathcal{B}_k, C_k)$ of schema $\mathcal{R}$, an ML model $\hat{\mathcal{M}}$,
    a CR method $\mathcal{A}_{\text{CR}}$, the clean data $\mathcal{D}_{\text{clean}}$, and thresholds $\text{ite}_{\max}$ and $m$.
Output: A set $\mathcal{S}$ of influential attributes.
1.  $\mathcal{G} := \{\{A\} \mid A \in \mathcal{R}\}$; idx := 1; $\mathcal{S}_{\text{best}} := \emptyset$; $\text{acc}_{\text{best}} := 0$;
2.  $\mathcal{B}_k^{\mathcal{R}}$ := the resulting set by cleaning tuples in $\mathcal{B}_k \setminus \mathcal{D}_{\text{clean}}$ via $\mathcal{A}_{\text{CR}}$;
3.  while idx $\leq \text{ite}_{\max}$ do
      /* candidate set selection */
4.      for each $\mathcal{S}_i$ in $\mathcal{G}$ do
5.          $\hat{\mathcal{M}}_{\text{add}}$ := train($\mathcal{B}_k^{\mathcal{R}}[\mathcal{S}_i]$) where $\mathcal{B}_k^{\mathcal{R}}[\mathcal{S}_i]$ is a projected set of $\mathcal{B}_k^{\mathcal{R}}$;
6.          $\hat{\mathcal{M}}_{\text{int}}$ := boosting($\hat{\mathcal{M}}, \hat{\mathcal{M}}_{\text{add}}$);
7.          if acc($\hat{\mathcal{M}}_{\text{int}}, C_k$) > $\text{acc}_{\text{best}}$ do
8.              $\mathcal{S}_{\text{best}} := \mathcal{S}_i$; $\text{acc}_{\text{best}}$ := acc($\hat{\mathcal{M}}_{\text{int}}, C_k$);
9.      if $\mathcal{S}_{\text{best}}$ is not updated do
10.         break
      /* candidate set generation */
11.     $\mathcal{G}$ := select($\mathcal{G}, m$); // retain at most $m$ sets with the best $\hat{\mathcal{M}}_{\text{int}}$
12.     $\mathcal{G}$ := generate($\mathcal{G}, m$) via mutation and crossover; idx := idx + 1;
13. return $\mathcal{S}_{\text{best}}$;
```

Figure 5: Algorithm AlgIA

Algorithm. We develop an algorithm for IA, denoted by AlgIA and shown in Figure 5. AlgIA takes as input $\mathcal{R}$, $\mathcal{D}_k = (\mathcal{B}_k, C_k)$, $\hat{\mathcal{M}}$, $\mathcal{A}_{\text{CR}}$, the clean data $\mathcal{D}_{\text{clean}}$ and two thresholds $\text{ite}_{\max}$ and m (the number of rounds and the maximum number of candidate sets in each round, respectively). It returns a set $\mathcal{S}$ of influential attributes.

AlgIA first initializes the set $\mathcal{G}$ of candidate attribute sets and the attribute set $\mathcal{S}_{\text{best}}$ (initially empty) with the highest accuracy acc_{best} observed so far (line 1), and pre-cleans $\mathcal{B}_k$ as $\mathcal{B}_k^{\mathcal{R}}$ via $\mathcal{A}_{\text{CR}}$ (line 2), where we only clean tuples in $\mathcal{B}_k \setminus \mathcal{D}_{\text{clean}}$. Then AlgIA iteratively conducts two steps (lines 3-12). (1) *Selection* (lines 4-10). For each $\mathcal{S}_i$ in $\mathcal{G}$, it trains the lightweight $\hat{\mathcal{M}}_{\text{add}}$ (line 5), combines it with $\hat{\mathcal{M}}$ as $\hat{\mathcal{M}}_{\text{int}}$ (line 6), and evaluates the accuracy of $\hat{\mathcal{M}}_{\text{int}}$ on C_k (line 7). If acc($\hat{\mathcal{M}}_{\text{int}}, C_k$) > acc_{best}, we take $\mathcal{S}_i$ as $\mathcal{S}_{\text{best}}$ (line 8). If none of the sets in $\mathcal{G}$ outperforms $\mathcal{S}_{\text{best}}$, it indicates that $\mathcal{S}_{\text{best}}$ is a good one and AlgIA converges (lines 9-10). (2) *Generation* (lines 11-12). At most m sets in $\mathcal{G}$ with the most accurate $\hat{\mathcal{M}}_{\text{int}}$ are then used to generate at most $\frac{m\cdot(m-1)}{2}$ attribute sets for the next iteration, via mutation and crossover. Finally, when it reaches the maximum number of iterations (*i.e.,* $\text{ite}_{\max}$), AlgIA returns $\mathcal{S}_{\text{best}}$ (line 13).

Example 4: Consider $\mathcal{D}_k = (\mathcal{B}_k, C_k)$ of schema $\mathcal{R}$ in Table 1, where $m = 8$. Initially, $\mathcal{G} = \{\{A_1\}, \{A_2\}, \{A_3\}, \{A_4\}, \{A_5\}\}$ and $\mathcal{S}_{\text{best}} = \emptyset$. AlgIA first pre-cleans all A_1-A_5 attributes of $\mathcal{B}_k$ via $\mathcal{A}_{\text{CR}}$ to get $\mathcal{B}_k^{\mathcal{R}}$. In the first iteration, AlgIA trains a model $\hat{\mathcal{M}}_{\text{add}}$, for each projected set of $\mathcal{B}_k^{\mathcal{R}}$ on $\{A_i\}$ ($1 \leq i \leq 5$). Assume that cleaning A_5 gives the model with the best acc_{best}. We update $\mathcal{S}_{\text{best}}$ as $\{A_5\}$. All sets in $\mathcal{G}$ are retained to generate more candidates for the next iteration, *e.g.,* $\{A_3\}$ and $\{A_5\}$ are merged into $\{A_3, A_5\}$. If cleaning $\{A_3, A_5\}$ gives an accuracy higher than acc_{best}, we update $\mathcal{S}_{\text{best}} = \{A_3, A_5\}$. This process proceeds until either $\mathcal{S}_{\text{best}}$ no longer changes or AlgIA reaches $\text{ite}_{\max}$ rounds and $\mathcal{S}_{\text{best}}$ is returned as the result. □

Analyses. AlgIA takes $O(c_{\text{CR}} + \text{ite}_{\max} \cdot m^2 \cdot (c_{\text{train}} + c_{\text{val}} + \log(m)) + c_{\text{cross}} + c_{\text{muta}}))$ time, where $\text{ite}_{\max}$ and m are given above, c_{CR} is the cost of cleaning $\mathcal{B}_k$ by $\mathcal{A}_{\text{CR}}$, c_{train} is the cost of training $\hat{\mathcal{M}}_{\text{add}}$, c_{val} is the cost of validating $\hat{\mathcal{M}}_{\text{int}}$ on C_k, $O(m^2 \cdot \log(m))$ is the cost of selecting the top-m sets from $\mathcal{G}$, and c_{cross} (resp. c_{muta}) is the cost of a crossover (resp. mutation) operation in $\mathcal{G}$. Among these, c_{CR} is often the most costly. We will see in Section 6 that AlgIA is fast.

```
Input: A schema $\mathcal{R}$, a dataset $\mathcal{D}_k = (\mathcal{B}_k, C_k)$ of $\mathcal{R}$, an ML model $\hat{\mathcal{M}}$,
    a CR method $\mathcal{A}_{\text{CR}}$, the cleaned data sets $\mathcal{D}_{\text{clean}}$, and
    the maximum number $T_{\max}$ of influential tuples.
Output: A set $\mathcal{T}$ of at most $\mathcal{T}_{\max}$ influential tuples.
1.  $\mathcal{T} := \emptyset$; $\mathcal{B}_k$ := the cleaned $\mathcal{B}_k$ by cleaning all tuples in $\mathcal{B}_k$ via $\mathcal{A}_{\text{CR}}$;
2.  for $t \in \mathcal{B}_k \setminus \mathcal{D}_{\text{clean}}$ do
3.      $t_{\text{old}}$ := the version of $t$ with which $\mathcal{M}$ is trained;
4.      $t_{\text{new}}$ := the new version of $t$ in $\mathcal{B}_k$ after pre-cleaning via $\mathcal{A}_{\text{CR}}$;
5.      if $t_{\text{old}} \neq t_{\text{new}}$ do
6.          $\Delta\theta_t := \frac{1}{|\mathcal{D}_k|}(H_\theta^{-1}(\nabla_\theta l(t_{\text{new}}, \theta) - \nabla_\theta l(t_{\text{old}}, \theta)))$;
7.          $\hat{\mathcal{M}}_t$ := update($\hat{\mathcal{M}}, \Delta\theta_t$); $\Delta\text{acc}_t$ := acc($\hat{\mathcal{M}}_t, C_k$) − acc($\hat{\mathcal{M}}, C_k$);
8.          if $\Delta\text{acc}_t > 0$ do
9.              $\mathcal{T}$ := update($\mathcal{T}, t, T_{\max}$);
10. return $\mathcal{T}$;
```

Figure 6: Algorithm AlgIT

5 PINPOINTING INFLUENTIAL TUPLES

This section formulates the problem of identifying influential tuples, proves its intractability, and develops an effective method.

Problem. Given a dataset $\mathcal{D}_k = (\mathcal{B}_k, C_k)$ of schema $\mathcal{R}$, a CR method $\mathcal{A}_{\text{CR}}$ and a model $\hat{\mathcal{M}}$ (Section 4), we aim to identify the set $\mathcal{T}$ of *influential tuples* of $\mathcal{B}_k$, such that if we resolve conflicts in such tuples via $\mathcal{A}_{\text{CR}}$, the accuracy of $\hat{\mathcal{M}}$ is maximumly improved when evaluated with C_k. Denote the cleaned $\mathcal{B}_k$ by $\mathcal{B}_k^{\mathcal{T}}$. Then we obtain a cleaned version of $\mathcal{D}_k$, denoted by $\mathcal{D}_k^{\mathcal{T}}$, *i.e.,* $\mathcal{D}_k^{\mathcal{T}} = (\mathcal{B}_k^{\mathcal{T}}, C_k)$.

The *problem of pinpointing influential tuples* (IT) is as follows.

- *Input*: $\mathcal{R}$, $\mathcal{D}_k = (\mathcal{B}_k, C_k)$, $\hat{\mathcal{M}}$, $\mathcal{A}_{\text{CR}}$ and $\mathcal{D}_{\text{clean}}$ as in Section 3.
- *Output*: A set $\mathcal{T}$ of influential tuples of $\mathcal{B}_k$.
- *Objective*: Maximize acc($\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{T}}$) − acc($\hat{\mathcal{M}}, \mathcal{D}_k$)).

Here acc($\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{T}}$) (resp. acc($\hat{\mathcal{M}}, \mathcal{D}_k$)) is the accuracy of $\hat{\mathcal{M}}$ that is trained with $\mathcal{B}_k^{\mathcal{T}}$ (resp. $\mathcal{B}_k$) and is evaluated with C_k.

NP-hardness. It is also nontrivial to identify the desired set of influential tuples. The decision problem, also referred to as IT, is to determine, given $\mathcal{R}$, $\mathcal{D}_k = (\mathcal{B}_k, C_k)$, $\hat{\mathcal{M}}$, $\mathcal{A}_{\text{CR}}$ and a bound δ, whether there is a set $\mathcal{T}$ of tuples such that acc($\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{T}}$) − acc($\hat{\mathcal{M}}, \mathcal{D}_k$) $\geq \delta$.

Theorem 3: *Problem* IT *is* NP-*hard.* □

Proof sketch: We show the intractability of IT also by reduction from 3-SAT. Given a 3-SAT instance ϕ with n variables, we define a schema $\mathcal{R}$ and construct a set $\mathcal{D}_k$, where each tuple in $\mathcal{D}_k$ encodes a variable in ϕ. We also pick $\hat{\mathcal{M}}$ and $\mathcal{A}_{\text{CR}}$. We show that there exists a set $\mathcal{T}$ of tuples in $\mathcal{D}_k$ such that acc($\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{T}}$)−acc($\hat{\mathcal{M}}, \mathcal{D}_k$) $\geq \delta$ if and only if there exists a truth assignment that satisfies ϕ. □

Naive approach. One might want to adopt a greedy algorithm to solve IT as follows. (a) Clean a dirty tuple t in $\mathcal{B}_k$ via $\mathcal{A}_{\text{CR}}$, and denote the cleaned $\mathcal{B}_k$ as $\mathcal{B}_k^t$. (2) Retrain a model $\hat{\mathcal{M}}$, denoted by $\hat{\mathcal{M}}_t$, on each $\mathcal{B}_k^t$. (3) Iteratively find the most influential tuple t' in $\mathcal{B}_k$ such that cleaning t' in $\mathcal{B}_k$ improves the accuracy of $\hat{\mathcal{M}}$ on C_k the most, *i.e.,* $t' = \arg\max_{t \in \mathcal{D}_k} \left(\text{acc}(\hat{\mathcal{M}}_t, \mathcal{D}_k^t) - \text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k)\right)$; add t' to $\mathcal{T}$ and update $\mathcal{D}_k$ and $\hat{\mathcal{M}}$ to be $\mathcal{D}_k^{t'}$ and $\hat{\mathcal{M}}_{t'}$, respectively, until cleaning any tuple in $\mathcal{B}_k$ cannot further improve the accuracy (*i.e.,* acc($\hat{\mathcal{M}}_{t'}, \mathcal{D}_k^{t'}$) − acc($\hat{\mathcal{M}}, \mathcal{D}_k$) ≤ 0). (4) The set $\mathcal{T}$ is returned as the set of influential tuples. However, this algorithm incurs a large cost, since $\hat{\mathcal{M}}$ needs to be retrained $O(|\mathcal{D}_k||\mathcal{T}|)$ times.

Our approach. In contrast, we identify a set $\mathcal{T}$ of influential tuples in $\mathcal{B}_k$ by estimating the effect of cleaning tuples in $\mathcal{B}_k$ via $\mathcal{A}_{\mathsf{CR}}$ without actually retraining $\hat{\mathcal{M}}$. To achieve this, we adopt the influence function [5, 46] to estimate the change to the parameters of model $\hat{\mathcal{M}}$ when tuples in $\mathcal{B}_k$ are removed or modified.

More specifically, cleaning a tuple $t \in \mathcal{B}_k$ via $\mathcal{A}_{\mathsf{CR}}$ can be regarded as removing an old version t_{old} of t from $\mathcal{D}$ and adding a new version t_{new} to $\mathcal{D}$. Denote by θ_{old} and θ_{new} the parameters of $\hat{\mathcal{M}}$ trained with the set containing t_{old} and t_{new}, respectively.

We adopt the influence function as in Section 3. More specifically, for each $t = (x, y)$, where x and y are the attributes and label of t, respectively, we upweight t by an infinitesimal amount ϵ, where removing (resp. adding) t from $\mathcal{D}$ is the same as upweighting its old version t_{old} (resp. t_{new}) by $-\epsilon = -\frac{1}{|\mathcal{D}_k|}$ (resp.ϵ). As shown in [46], the change of $\hat{\mathcal{M}}$'s parameters, *i.e.,* $\Delta\theta_t = \theta_{\text{new}} - \theta_{\text{old}}$, by cleaning t via $\mathcal{A}_{\mathsf{CR}}$ can be estimated in a closed form: $-H_\theta^{-1}(\nabla_\theta l(t_{\text{new}}, \theta) - \nabla_\theta l(t_{\text{old}}, \theta))$, where $\theta = \theta_{\text{old}}$, ∇_θ is the partial derivation of θ *w.r.t.* ϵ, H_θ is the Hession Matrix, and $l(t, \theta)$ is the loss of tuple t.

When a tuple t in $\mathcal{B}_k$ is cleaned via $\mathcal{A}_{\mathsf{CR}}$, we can approximate its impact on $\hat{\mathcal{M}}$ by adjusting the parameters of $\hat{\mathcal{M}}$ as $\theta_{\text{new}}(= \theta_{\text{old}} + \Delta\theta_t)$ and then validate $\hat{\mathcal{M}}$ on C_k, without actually retaining $\hat{\mathcal{M}}$. Based on the accuracy improvement, we can derive the set $\mathcal{T}$.

Algorithm. We develop an algorithm for IT, denoted by AlgIT and shown in Figure 6. AlgIT takes as input $\mathcal{R}$, $\mathcal{D}_k = (\mathcal{B}_k, C_k)$, $\hat{\mathcal{M}}$, $\mathcal{A}_{\mathsf{CR}}$, the clean data $\mathcal{D}_{\text{clean}}$, and the maximum number $T_{\max}$ of tuples. It returns a set $\mathcal{T}$ of at most $T_{\max}$ influential tuples.

More specifically, AlgIT initializes $\mathcal{T}$ as an empty set, and cleans all tuples in $\mathcal{B}_k \setminus \mathcal{D}_{\text{clean}}$ via $\mathcal{A}_{\mathsf{CR}}$ (line 1). For each tuple t in $\mathcal{B}_k$, it checks whether t is cleaned by $\mathcal{A}_{\mathsf{CR}}$ (lines 3-5). If so (line 5), it uses the influence function to estimate the impact of cleaning $t \in \mathcal{B}_k$ on $\hat{\mathcal{M}}$. To do this, AlgIT produces a copy $\hat{\mathcal{M}}_t$ of $\hat{\mathcal{M}}$ by updating the parameters of $\hat{\mathcal{M}}$ with the change $\Delta\theta_t$, estimated with the influence function (line 6). Then it computes $\Delta\text{acc}_t = \text{acc}(\hat{\mathcal{M}}_t, C_k) - \text{acc}(\hat{\mathcal{M}}, C_k)$ (line 7), the accuracy improvement of cleaning t on C_k, and adds t to $\mathcal{T}$ if $\Delta\text{acc}_t > 0$ and Δacc_t is among the top-$T_{\max}$ highest for tuples in $\mathcal{T}$ (lines 8-9). Finally, $\mathcal{T}$ is returned (line 10).

There is a trade-off between the efficiency and accuracy with $T_{\max}$. Given a smaller $T_{\max}$, $\hat{\mathcal{M}}$ is often more accurate (Section 6). This is because influence function estimates impact to $\hat{\mathcal{M}}$ when a single tuple in $\mathcal{D}_k$ is removed or modified [46], but not the joint influence when multiple tuples in $\mathcal{D}_k$ are changed. Thus a smaller $T_{\max}$ indicates more fine-tuning rounds of Rock4ML, which more accurately estimates the joint influence, but incurs a larger cost.

Example 5: Continuing with Example 4 with $\mathcal{D}_{\text{clean}} = \emptyset$ and $T_{\max} = 1$, we assume that $\hat{\mathcal{M}}$ is trained with $\mathcal{B}_k = \{t_1\text{-}t_6\}$ in Table 1 and after pre-cleaning, t_1 and t_4 are updated via $\mathcal{A}_{\mathsf{CR}}$. Their accuracy improvements Δacc on C_k are 0.03 and 0.01, respectively. As a consequence, the set $\mathcal{T} = \{t_1\}$ is returned as output. □

Analyses. AlgIT takes $O(c_{\mathsf{CR}} + |\mathcal{D}_k| \cdot (c_{\Delta\theta_t} + c_{\hat{\mathcal{M}}_t} + c_{\Delta\text{acc}_t} + \log(T_{\max})))$ time, where c_{CR} is the cost of cleaning $\mathcal{B}_k$ by $\mathcal{A}_{\mathsf{CR}}$, $c_{\Delta\theta_t}$ is the cost of computing $\hat{\mathcal{M}}$'s parameter changes via the influence function, $c_{\hat{\mathcal{M}}_t}$ is the cost of updating $\hat{\mathcal{M}}$ to $\hat{\mathcal{M}}_t$, $c_{\Delta\text{acc}_t}$ is the cost of computing the accuracy improvement of $\hat{\mathcal{M}}_t$ on C_k, and $O(\log(T_{\max}))$ is the cost of maintaining the top-$T_{\max}$ tuples in $\mathcal{T}$.

6 EXPERIMENTAL STUDY

This section experimentally evaluates the (1) effectiveness and (2) efficiency of Rock4ML for different (a) ML differential classification models, (b) CR methods, and (c) configurable parameters.

Experimental setting. We start with our settings.

Datasets. We used six datasets (Tables 2-3): (1) Adult, Default, Nursery and German (Section 2); (2) Bank, a dataset from marketing campaigns of a Portuguese banking institute; and (3) RoadSafety, which is among the largest datasets from OpenML [3]. We injected noise in a way similar to Section 2, except that we set the noise ratio at 10% for simulating real-world error rates following previous work: (a) [61] tested CleanML benchmarks [51] and concluded that these datasets contain few errors and the errors do not have significant impact on the downstream models; thus they called for datasets with impactful errors with large error rates; and (b) [56, 57, 66, 70] set error rates above 10% on these real-world datasets. We denote by $\mathcal{D}^-_{\text{train}}$, $\mathcal{D}^+_{\text{train}}$ and $\mathcal{D}_{\text{train}}$ the dirty, cleaned and "ground-truth" version of the training data of $\mathcal{D}$, respectively, as in Section 2.

ML models $\mathcal{M}$. We evaluated three classical differential ML classification models commonly used in practice and frequently considered in the literature: (1) MLPClassifier [62] and FT-Transformer [37] tested in Section 2, and (2) logistic regression (LR) [62].

CR methods $\mathcal{A}_{\mathsf{CR}}$. We tested five CR methods: (1) HoloClean [66], RB [56, 57] and Rock [9] (see Section 2). In particular, for RB, we set its labeling budget to 20 as in [56]; and for HoloClean, we fed it with the same set of rules (transformed to denial constraint) as Rock, and when its factor graph cannot fit in memory, we randomly selected a maximal subset of rules that works. (2) RT that uses Raha [57] and T5 [64] to detect and fix errors, respectively. (3) Mode [45, 47] that fills in empty cells with the most frequent values from the domains.

Baselines. We implemented Rock4ML in python and used the following baselines with their released codes. (1) Base, a "lower-bound" baseline that cleans nothing. (2) Oracle$_{\text{full}}$, an "upper-bound" baseline that corrects all errors by experts with 100% accuracy. (3) CR$_{\text{full}}$, an approach that cleans all errors via $\mathcal{A}_{\mathsf{CR}}$. (4) Picket [54], which detects corrupted tuples via reconstruction loss and removes dirty tuples from the training data. (5) DiffPrep-fix and DiffPrep-flex [50], two data preprocessing pipelines that select suitable cleaning operators for differential ML models with and without pre-defined transformation orders, respectively. (6) SAGA [71], a framework that automatically generates the top-K most effective data cleaning pipelines; we set K as 1 in the tests. (7) ActiveClean [48], an approach that treats data cleaning and model training in a form of stochastic gradient descent. (8) CPClean [45], a data imputation method for KNN models. (9) BoostClean [47], a data cleaning approach that enhances the accuracy of ML models by boosting.

Note that (a) BoostClean and CPClean impute missing values only, and we only compared Rock4ML's efficiency with them; (b) we fed ActiveClean with dirty training data, the same as Rock4ML; and (c) we configured DiffPrep-fix, DiffPrep-flex, SAGA, CPClean and BoostClean with selected cleaning tools as in their released codes.

Here Base, Oracle$_{\text{full}}$, CR$_{\text{full}}$ and Picket are independent of models $\mathcal{M}$, while the others are $\mathcal{M}$-aware, *i.e.,* they target a given $\mathcal{M}$.

Configurable parameters. We set default parameters as follows: (1)

Datasets	$\lvert\mathcal{D}\rvert$	# of Attrib.	Injected Noise Ratio (%)	# of Classes
Bank [77]	49,732	16	10	2
RoadSafety [38]	363,243	67	10	3

Table 3: More tested datasets

for Rock4ML, $e = 3$ for the number of epochs; and following [43], we let Q_1^1 and Q_3^1 (resp. Q_1^2 and Q_3^2) be the first and third quartiles of the distribution of dynamic loss $L_{k-1}(t)-L_k(t)$ (resp. $L_k(t)$) of tuples in $\mathcal{D}$, respectively (see Section 3 for $L_k(t)$) and then we set η_1 and η_2 (for identifying clean data for $\mathcal{M}$ in Rock4ML) as $Q_3^1+1.5\cdot\mathsf{IQR}^1$ and $Q_1^2-1.5\cdot\mathsf{IQR}^2$, respectively, where $\mathsf{IQR}^1 = Q_3^1 - Q_1^1$, and $\mathsf{IQR}^2 = Q_3^2 - Q_1^2$; (2) for AlgIA, $\mathsf{ite}_{\max} = 2$ as the maximum number of rounds and $m = |\mathcal{R}|$ as the bound on candidate sets; and for AlgIT, $T_{\max} = 2\%|\mathcal{D}|$. By default, we used LR as $\mathcal{M}$ and Rock as $\mathcal{A}_{\mathsf{CR}}$.

Configuration. We adopted the same setting as in Section 2.

Experimental findings. We next report our findings.

Exp-1 Effectiveness. We first evaluated the accuracy of Rock4ML and baselines with various ML models and CR methods. Following [45], we adopt the relative accuracy of a baseline X, defined as

$$\mathsf{ACC}_{\mathsf{re}}(X) = \frac{\mathsf{acc}(X) - \mathsf{acc}(\mathsf{Base})}{\mathsf{acc}(\mathsf{Oracle}_{\mathsf{full}}) - \mathsf{acc}(\mathsf{Base})},$$

where $\mathsf{acc}(X) = \mathsf{acc}(\mathcal{M}, \mathcal{D}_{\mathsf{CR}})$ and $\mathcal{D}_{\mathsf{CR}}$ is the dataset cleaned by CR method X (*e.g.,* Rock4ML). Since $\mathsf{Oracle}_{\mathsf{full}}$ and Base are the upper and lower bounds, respectively (whose accuracy shown in Table 4), $\mathsf{ACC}_{\mathsf{re}}(X) \in (-\infty, 1]$ quantifies the improvement in the accuracy relative to the upper bound; note that $\mathsf{ACC}_{\mathsf{re}}(X) < 0$ if the accuracy of cleaning via X is worse than cleaning nothing (*i.e.,*Base).

Accuracy vs. baselines. As shown in Figures 7(a)-7(c), Rock4ML has the highest $\mathsf{ACC}_{\mathsf{re}}$. Besides, we find the following.

(1) The released code of all $\mathcal{M}$-aware baselines can only support LR. Rock4ML outperforms all these baselines for LR *e.g.,* its $\mathsf{ACC}_{\mathsf{re}}$ is 45.7% on average, as opposed to 7.7%, -1.1%, -20.4% and -1.8% of DiffPrep-fix, DiffPrep-flex, ActiveClean and SAGA, respectively. This is because the creator-critic framework of Rock4ML (a) identifies dirty data by monitoring and analyzing error signals, and (b) selectively cleans errors only in influential attributes and influential tuples that maximumly affect the relative accuracy of the given model $\mathcal{M}$, *e.g.,* Rock4ML detected and fixed 1‰$|\mathcal{D}|$ influential tuples on Adult, which improves $\mathsf{ACC}_{\mathsf{re}}$ by 2.63% alone, and these tuples were not detected by the baselines. The $\mathcal{M}$-aware baselines clean the entire dataset $\mathcal{D}$, and may introduce more errors. This justifies the need for cleaning influential data, instead of the entire $\mathcal{D}$.

(2) Rock4ML consistently beats Base, $\mathsf{CR}_{\mathsf{full}}$ and Picket for all models; on average its $\mathsf{ACC}_{\mathsf{re}}$ is 43.1%, 39.6% and 78.8% higher than the three, respectively. These further show the benefits of cleaning influential partial data; moreover, guided by influence function and boosting strategy, Rock4ML selectively fixes errors that it can confidently address, reducing the negative impact of noise introduced by CR method $\mathcal{A}_{\mathsf{CR}}$, *e.g.,* the $\mathsf{ACC}_{\mathsf{re}}$ of $\mathsf{CR}_{\mathsf{full}}$ and Rock4ML on Adult for MLPClassifier is 27.4% and 42.5%, respectively.

We next tested the impact of various parameters on the accuracy.

Varying $\mathcal{A}_{\mathsf{CR}}$. We tested the impact of $\mathcal{A}_{\mathsf{CR}}$ (except Holoclean on Default, whose features are too large to fit in memory). Besides, Picket, DiffPrep-fix, DiffPrep-flex, ActiveClean and SAGA are not compared since they do not rely on any particular CR methods.

As shown in Figures 7(d)-7(g), (1) Rock4ML performs better with more accurate $\mathcal{A}_{\mathsf{CR}}$, *e.g.,* the $\mathsf{ACC}_{\mathsf{re}}$ of Rock4ML with Rock is 53.7% higher on average than Holoclean. This is because more accurate CR corrects more errors while introducing less noise. (2) Equipped with accurate $\mathcal{A}_{\mathsf{CR}}$ methods, Rock4ML improves the $\mathsf{ACC}_{\mathsf{re}}$ better than directly cleaning $\mathcal{D}$, *e.g.,* Rock4ML with Rock improves $\mathsf{ACC}_{\mathsf{re}}$ of $\mathcal{M}$ by 70% on German instead of 48.8% by cleaning the entire $\mathcal{D}$. This further verifies the effectiveness of selective data cleaning. (3) With inaccurate $\mathcal{A}_{\mathsf{CR}}$ methods (*e.g.,* Holoclean), neither Rock4ML nor $\mathsf{CR}_{\mathsf{full}}$ can make substantial improvement on $\mathsf{ACC}_{\mathsf{re}}$ ($\approx$ 0%).

$\mathcal{M}$	$\mathcal{A}_{\mathsf{CR}}$	German	Adult	Nursery	Default	Bank
LR	Base	0.481	0.653	0.727	0.523	0.487
	$\mathsf{Oracle}_{\mathsf{full}}$	0.691	0.784	0.901	0.673	0.684
MLP	Base	0.578	0.617	0.775	0.478	0.53
	$\mathsf{Oracle}_{\mathsf{full}}$	0.788	0.801	1	0.678	0.746
FT-Transformer	Base	0.434	0.47	0.259	0.46	0.471
	$\mathsf{Oracle}_{\mathsf{full}}$	0.51	0.493	0.26	0.484	0.496

Table 4: acc (Base) **and** $\mathsf{acc}(\mathsf{Oracle}_{\mathsf{full}})$

Varying m. We varied the number m of attribute sets from $0.5|\mathcal{R}|$ to $2.5|\mathcal{R}|$ (Section 4). As shown in Figure 7(h), the $\mathsf{ACC}_{\mathsf{re}}$ of Rock4ML increases when m gets larger, *e.g.,* it is improved by 17.6% when m is changed from $0.5|\mathcal{R}|$ to $2|\mathcal{R}|$. This is because a larger m allows more explorations in the search space, increasing the chance of identifying a good set of influential attributes. However, a fairly small m suffices for Rock4ML to find a good set, *e.g.,* the $\mathsf{ACC}_{\mathsf{re}}$ of $\mathcal{M}$ with $m = |\mathcal{R}|$ is just 5% lower than with $m = 2.5|\mathcal{R}|$.

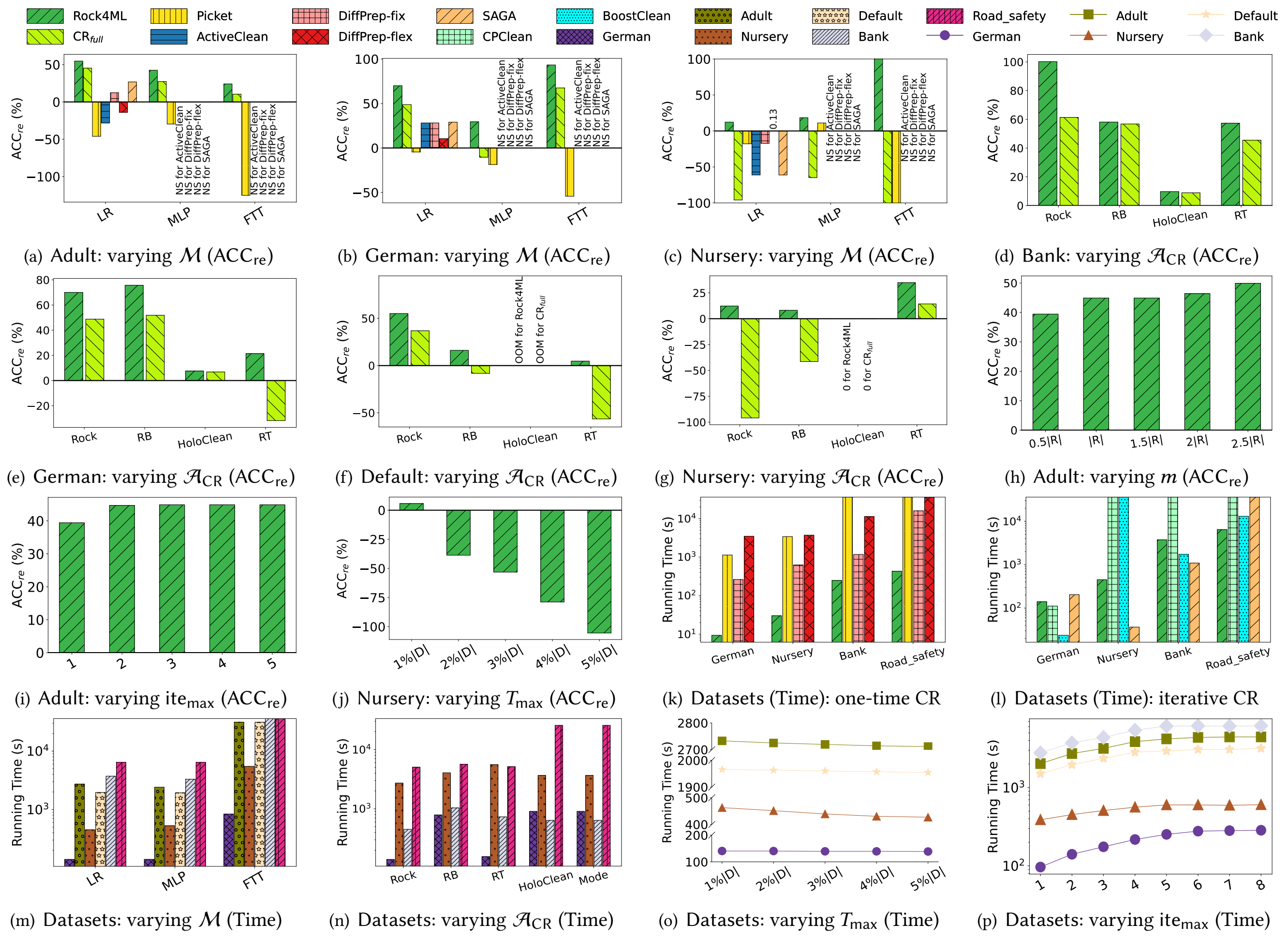
Varying $\mathsf{ite}_{\max}$. In Figure 7(i), we varied $\mathsf{ite}_{\max}$ from 1 to 5 (Section 4). As shown there, the $\mathsf{ACC}_{\mathsf{re}}$ initially increases and then stabilizes when $\mathsf{ite}_{\max}$ exceeds 2. This is because although Rock4ML checks more candidate attribute sets when $\mathsf{ite}_{\max}$ increases, a good set of influential attributes can be identified with a small $\mathsf{ite}_{\max}$.

Varying $T_{\max}$. We tested the impact of the number $T_{\max}$ of influential tuples on Rock4ML, varying it from $1\%|\mathcal{D}|$ to $5\%|\mathcal{D}|$ (Section 5). To better illustrate, we disabled the component of identifying influential attributes (*i.e.,* Algorithm AlgIA). As shown in Figure 7(j), when $T_{\max}$ increases, the $\mathsf{ACC}_{\mathsf{re}}$ of Rock4ML decreases, *e.g.,* its $\mathsf{ACC}_{\mathsf{re}}$ decreases from 5.7% to -78.9% when $T_{\max}$ varies from $1\%|\mathcal{D}|$ to $4\%|\mathcal{D}|$, since Rock4ML estimates the joint influence less accurately in less fine-tuning rounds (Section 5). From our empirical study, we suggest to set $T_{\max}$ at a small value (*e.g.,* $\leq 2\%|\mathcal{D}|$).

Varying e. We also varied the number e of epochs for incremental learning (not shown). The relative accuracy of Rock4ML is not very sensitive to e when it reaches a certain threshold (*e.g.,* the default). This is because $\mathcal{M}$ can fit $\mathcal{D}$ within a small number of epochs. We find that a small e (*e.g.,* = 3) suffices for incremental training.

Exp-2 Efficiency. Below we first compared the overall time of Rock4ML and baselines in the default setting. We then evaluated the impact of $\mathcal{M}$, $\mathcal{A}_{\mathsf{CR}}$ and configurable parameters on the efficiency of Rock4ML. We report the performance of Rock4ML for both one-round and multi-round iterations, and compared it with the non-iterative baselines (Picket, DiffPrep-fix and DiffPrep-flex) and iterative ones (BoostClean, CPClean and SAGA), respectively; we did not compare with ActiveClean, since it needs experts to fix errors and is hard to estimate its time. We do not report the time of a baseline if it ran out of memory or could not finish within 10 hours.

Comparison vs. baselines. In Figures 7(k)-7(l), (a) one-round execution of Rock4ML outperforms all non-iterative baselines, *e.g.,* 16.8X, 121.6X and 96.5X faster than DiffPrep-fix, DiffPrep-flex and Picket on average, respectively, and its $\mathsf{ACC}_{\mathsf{re}}$ is 24%, 38.8% and 57% better than the three for LR. This is because Rock4ML selectively cleans



(a) Adult: varying $\mathcal{M}$ (ACC$_{\text{re}}$) (b) German: varying $\mathcal{M}$ (ACC$_{\text{re}}$) (c) Nursery: varying $\mathcal{M}$ (ACC$_{\text{re}}$) (d) Bank: varying $\mathcal{A}_{\text{CR}}$ (ACC$_{\text{re}}$)

(e) German: varying $\mathcal{A}_{\text{CR}}$ (ACC$_{\text{re}}$) (f) Default: varying $\mathcal{A}_{\text{CR}}$ (ACC$_{\text{re}}$) (g) Nursery: varying $\mathcal{A}_{\text{CR}}$ (ACC$_{\text{re}}$) (h) Adult: varying m (ACC$_{\text{re}}$)

(i) Adult: varying ite$_{\max}$ (ACC$_{\text{re}}$) (j) Nursery: varying $T_{\max}$ (ACC$_{\text{re}}$) (k) Datasets (Time): one-time CR (l) Datasets (Time): iterative CR

(m) Datasets: varying $\mathcal{M}$ (Time) (n) Datasets: varying $\mathcal{A}_{\text{CR}}$ (Time) (o) Datasets: varying $T_{\max}$ (Time) (p) Datasets: varying ite$_{\max}$ (Time)

Figure 7: Performance evaluation

data. (b) Multi-round Rock4ML is faster than or comparable to iterative baselines, *e.g.,* it only takes 451.6s on Nursery and is at least 66.4X faster than CPClean. This is because Rock4ML only detects and fixes errors in influential data, and stops as soon as these errors are fixed. Indeed, Rock4ML converges after 7 rounds on average. Note that BoostClean is often the fastest one since it enhances $\mathcal{M}$ by training weaker classifiers, without directly cleaning $\mathcal{D}$ using $\mathcal{A}_{\text{CR}}$, which dominates the cost. This said, Rock4ML is able to not only improve the accuracy of $\mathcal{M}$ but also tune datasets for training more models. (c) Multi-round Rock4ML is even comparable to some non-iterative baselines, *e.g.,* Rock4ML takes 140.5s on German with 15 rounds, and is 8.1X, 1.9X and 24.8X faster than Picket, DiffPrep-fix and DiffPrep-flex, respectively. In short, Rock is practical since (a) it is efficient given efficient $\mathcal{M}$ and $\mathcal{A}_{\text{CR}}$, and (b) its creator-critic framework stops as soon as $\mathcal{A}_{\text{CR}}$ fixes errors in all influential data.

Varying $\mathcal{M}$. We tested the impact of different models $\mathcal{M}$ on the efficiency. As shown in Figure 7(m), Rock4ML takes longer when $\mathcal{M}$ is more complex, *e.g.,* with FT-Transformer it takes 5,487.2s on Nursery, 12.1X longer than with LR. This is because with a more complex $\mathcal{M}$, Rock4ML needs more time to train $\mathcal{M}$, *e.g.,* it takes longer to compute the hessian matrix for $\mathcal{M}$. This said, Rock4ML is fast when $\mathcal{M}$ is fast, *e.g.,* with LR it only takes 140.5s on German.

Varying $\mathcal{A}_{\text{CR}}$. As shown in Figure 7(n), the efficiency of Rock4ML is heavily dependent on its embedded CR method $\mathcal{A}_{\text{CR}}$, *e.g.,* it takes 1,801.8s (resp. 5,669.2s) total time on average for data cleaning when $\mathcal{A}_{\text{CR}}$ is Rock (resp. HoloClean). When $\mathcal{A}_{\text{CR}}$ is fast, so is Rock4ML, *e.g.,* Rock4ML with Mode takes only 6.3s and 307s on German and Adult, respectively, compared to 156.1s and 5,575.5s with RB.

Varying $T_{\max}$. We varied the maximum number $T_{\max}$ of influential tuples returned in a round, from 1%$|\mathcal{D}|$ to 5%$|\mathcal{D}|$. As shown in Figure 7(o), it takes less time when $T_{\max}$ gets larger as above, from 463s to 427.2s on Nursery. This is because with larger $T_{\max}$, algorithm AlgIT may find more tuples to fix, and thus make $\mathcal{M}$ converge quicker towards a local optima. However, larger $T_{\max}$ may make Rock4ML less accurate as shown in Exp-1 (see also Section 5).

Varying ite$_{\max}$. We varied the maximum number ite$_{\max}$ of iterations for algorithm AlgIA from 1 to 8. As shown in Figure 7(p), the cost of Rock4ML first increases and then stabilizes, *e.g.,* it increases from 2,025s to 4,379s on Adult when ite$_{\max}$ varies from 1 to 6, and then remains stable when ite$_{\max}$ $\geq$ 6. This is because given a larger ite$_{\max}$, Rock4ML explores more attribute combinations to fine-tune the models, and takes longer as expected. This said, a small ite$_{\max}$ suffices for AlgIA to find a good $\mathcal{S}$ and converge.

The impact of m is similar and hence is not shown.

Varying e. We also varied the number e of epochs for incremental learning from 1 to 5 (not shown). When e gets larger, the training time of $\mathcal{M}$ increases slightly, *e.g.,* when e varies from 1 to 5, Rock4ML only takes 1.06X longer to converge. This tells us that it only needs a few epochs such that $\mathcal{M}$ fits the newly cleaned tuples in $\mathcal{D}$, *i.e.,* a small e suffices; moreover, incremental training is efficient.

Summary. We find the following. (1) Rock4ML improves the relative accuracy ACC_{re} of the differential ML classification models by 43.1% on average; *e.g.,* it improves the ACC_{re} of LR by 45.7% on average, versus -22.8%, -20.4%, 7.7%, -1.1% and -1.8% of Picket, ActiveClean, DiffPrep-fix, DiffPrep-flex and SAGA, respectively. This verifies that accurate CR indeed improves the accuracy of $\mathcal{M}$. Rock4ML with accurate CR works better or comparably to AutoML approaches (*e.g.,* DiffPrep-fix). (2) Rock4ML is 39.6% more accurate than cleaning the entire dataset, which may degrade the accuracy of models. These justify the need for selective cleaning errors. (3) Rock4ML improves ML models better when CR is more accurate, *e.g.,* the accuracy of $\mathcal{M}$ improved by Rock4ML with Rock is 53.7% higher than with HoloClean on average for LR. (4) Rock4ML is efficient when it is equipped with an efficient CR method, *e.g.,* with Rock, 5 rounds of Rock4ML are 18.2X faster than non-iterative Picket and iterative CPClean, with 58.2% higher ACC_{re}. Moreover, a small number of iterations suffice to improve the accuracy of $\mathcal{M}$.

7 RELATED WORK

Data cleaning for ML. This line of work aims to improve the performance of downstream ML models by cleaning training data.

White-box. ActiveClean [48] treats data cleaning and model training as a form of stochastic gradient descent. CPClean [45] studies the impact of data incompleteness on the quality of KNN models trained with the data. GoodCore [13] selects a coreset over incomplete data through gradient approximation. CleanML [52] empirically investigates the impact of data cleaning on ML classification tasks on structured data. It concludes that CR "is more likely to have insignificant impact and unlikely to have negative impact".

Black-box. BoostClean [47] automatically selects an ensemble of error detection and repairing operations from a library. Picket [54] proposes a simple framework to safeguard against data corruptions during both training and deployment phases of ML models. MLClean [73] proposes a data cleaning framework that simultaneously handles data cleaning, unfairness mitigation and sanitation. Snorkel [65] enables users to train ML models via data programming with labeling functions. Coco [27, 28] assesses unfriendly tuples for ML models by using constraints defined with arithmetic expressions on numerical attributes. Amalur [40] unifies data integration and ML. Complaint-driven methods [32, 78] leverage users' complaints to remove/revise a minimum set of training examples for a better ML model. Explanation-based [55] models treat query explanation as a hyper-parameter tuning problem and adopt Random tree and Bayesian optimization to solve it. CategDedup [70] empirically studies the impact of categorical duplicates on ML models.

This work differs from the prior work. (1) We aim to develop a better understanding of the impact of CR on downstream ML models. We show that accurate CR indeed helps ML classification models, by identifying two impactful factors for CR to help. (2) Instead of an one-shot data cleaning workflow for $\mathcal{M}$ (*e.g.,* MLClean [73]), we propose Rock4ML, a creator-critic framework for differentiable models $\mathcal{M}$, which iteratively conducts data cleaning and model training. It selectively cleans errors on influential data, rather than indistinguishably applying CR to the entire data as practiced by all previous frameworks. (3) Rock4ML provides a concrete solution to improving $\mathcal{M}$ via CR, beyond an empirical analysis as in [70]. (4) Rock4ML does not restrict the choice of (perfect) CR methods as in [47, 48]. (5) Rock4ML detects and fixes errors in $\mathcal{D}$, rather than deleting dirty tuples entirely as in [54]. (6) Rock4ML does not require user interaction in the cleaning/training process, as opposed to Snorkel [65] and complaint-driven methods [32, 78].

Data cleaning for AutoML. This approach automatically selects ML pipelines to maximumly improve the accuracy of downstream ML models, in which data cleaning serves as a component.

White-box. DiffML [42] enables differentiable ML pipelines by assigning adjustable parameters to each training record and each error detection/repairing approach in a pool of data cleaning tools. AutoClean [61] re-evaluates several CleanML benchmarks using AutoML systems and extends AutoSklearn [31] with more advanced cleaning strategies for both white-box and black-box models; it concludes that most existing benchmarks contain few errors that substantially impact the downstream models, and thus, calls on the community for benchmarks with impactful real-world errors so as to study data cleaning for ML/AutoML on the attribute level.

Black-box. AlphaClean [49] studies parameter tuning for data cleaning pipelines to optimize an objective function. AutoSklearn [31] improves AutoML by assessing its historical performance on similar datasets. DiffPrep [50] proposes an automatic data preprocessing method for any dataset and differentiable model. SAGA [71] automatically generates the top-K most effective data cleaning pipelines.

More generally, [67] shows how different aspects of data quality propagate through various stages of machine learning development; [60] reviews recent progress in data cleaning for ML and presents a holistic cleaning framework; and Rein [5] introduces a comprehensive benchmark to evaluate the impact of data cleaning methods on various ML models. None of the previous work considers when CR may actually help improve the accuracy of ML classifications, and how to make practical use of CR in ML model training.

This work differs from the prior works on AutoML. (1) Rather than replying on other components (*e.g.,* feature engineering [31, 61]) in AutoML, we study the impact of CR itself on ML; moreover, our results can help AutoML prepare data for feature engineering and smoothing, beyond serving as embedded data cleaning. (2) Rock4ML allows users to plug in any CR method, as opposed to those (*e.g.,* [42, 50]) that only support restrictive CR methods.

8 CONCLUSION

The novelty of the work consists of the following. (1) In contrast to previous work, it finds that CR can help improve the accuracy of differential classification models. (2) It advocates to selectively clean data, rather than to clean the entire dataset; it identifies two impactful factors for CR to help, settles their complexity, and develops effective algorithms. (3) It proposes Rock4ML, a creator-critic framework for jointly cleaning data and training models. Our experimental study has verified that Rock4ML is promising in practice.

Topics for future work include (1) extending Rock4ML to improve models beyond differential classification, and (2) designing benchmarks with impactful real-life errors with the ground truth.

REFERENCES

[1] 2023. Scikit-learn Metrics. https://scikit-learn.org/stable/modules/model_evaluation.html.

[2] 2024. Code, datasets and full version. https://github.com/HatsuneHan/Rock4ML-VLDB25.

[3] 2024. OpenML. *https://www.openml.org/*.

[4] 2024. Why Users Love Cleanlab. https://cleanlab.ai/love/.

[5] Mohamed Abdelaal, Christian Hammacher, and Harald Schöning. 2023. REIN: A Comprehensive Benchmark Framework for Data Cleaning Methods in ML Pipelines. In *EDBT*. OpenProceedings.org, 499–511.

[6] Ashvin Agrawal, Rony Chatterjee, Carlo Curino, Avrilia Floratou, Neha Godwal, Matteo Interlandi, Alekh Jindal, Konstantinos Karanasos, Subru Krishnan, Brian Kroth, Jyoti Leeka, Kwanghyun Park, Hiren Patel, Olga Poppe, Fotis Psallidas, Raghu Ramakrishnan, Abhishek Roy, Karla Saur, Rathijit Sen, Markus Weimer, Travis Wright, and Yiwen Zhu. 2019. Cloudy with high chance of DBMS: A 10-year prediction for Enterprise-Grade ML. *CoRR* abs/1909.00084 (2019).

[7] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *PODS*. 68–79.

[8] Oluleye H Babatunde, Leisa Armstrong, Jinsong Leng, and Dean Diepeveen. 2014. *A Genetic Algorithm-Based Feature Selection.* Technical Report. Edith Cowan University.

[9] Xianchun Bao, Zian Bao, Qingsong Duan, Wenfei Fan, Hui Lei, Daji Li, Wei Lin, Peng Liu, Zhicong Lv, Mingliang Ouyang, Jiale Peng, Jing Zhang, Runxiao Zhao, Shuai Tang, Shuping Zhou, Yaoshu Wang, Qiyuan Wei, Min Xie, Jing Zhang, Xin Zhang, Runxiao Zhao, and Shuping Zhou. 2024. Rock: Cleaning Data by Embedding ML in Logic Rules. In *SIGMOD (industrial track)*.

[10] Leopoldo Bertossi. 2011. *Database Repairing and Consistent Query Answering.* Morgan & Claypool Publishers.

[11] George Beskales, Ihab F. Ilyas, Lukasz Golab, and Artur Galiullin. 2013. On the relative trust between inconsistent data and inaccurate constraints. In *ICDE*. IEEE, 541–552.

[12] Zalán Borsos, Mojmir Mutny, and Andreas Krause. 2020. Coresets via Bilevel Optimization for Continual Learning and Streaming. In *NeurIPS*.

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

[14] Rung-Ching Chen, Christine Dewi, Su-Wen Huang, and Rezzy Eko Caraka. 2020. Selecting critical features for data classification based on machine learning methods. *Journal of Big Data* (2020).

[15] E. F. Codd. 1979. Extending the Database Relational Model to Capture More Meaning. *TODS* 4, 4 (1979), 397–434.

[16] Gao Cong, Wenfei Fan, Floris Geerts, Xibei Jia, and Shuai Ma. 2007. Improving Data Quality: Consistency and Accuracy. In *VLDB*. 315–326.

[17] Stephen A Cook. 2023. The complexity of theorem-proving procedures. In *Logic, Automata, and Computational Complexity: The Works of Stephen A. Cook*. 143–152.

[18] Charles Darwin. 2023. Origin of the Species. In *British Politics and the Environment in the Long Nineteenth Century*. Routledge, 47–55.

[19] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. 2012. Optimal Distributed Online Prediction Using Mini-Batches. *Journal of Machine Learning Research* 13, 1 (2012).

[20] Dong Deng, Raul Castro Fernandez, Ziawasch Abedjan, Sibo Wang, Michael Stonebraker, Ahmed K. Elmagarmid, Ihab F. Ilyas, Samuel Madden, Mourad Ouzzani, and Nan Tang. 2017. The Data Civilizer System. In *CIDR*. www.cidrdb.org.

[21] Xiaoou Ding, Hongzhi Wang, Jiaxuan Su, Muxian Wang, Jianzhong Li, and Hong Gao. 2020. Leveraging currency for repairing inconsistent and incomplete data. *TKDE* (2020).

[22] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional functional dependencies for capturing data inconsistencies. *ACM Trans. Database Syst.* 33, 2 (2008), 6:1–6:48.

[23] Wenfei Fan, Floris Geerts, Nan Tang, and Wenyuan Yu. 2014. Conflict resolution with data currency and consistency. *J. Data and Information Quality* 5, 1-2 (2014), 6:1–6:37.

[24] Wenfei Fan, Ziyan Han, Weilong Ren, Ding Wang, Yaoshu Wang, Min Xie, and Mengyi Yan. 2024. Splitting Tuples of Mismatched Entities. *Proc. ACM Manag. Data* (2024).

[25] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. 2012. Towards certain fixes with editing rules and master data. *VLDB J.* 21, 2 (2012), 213–238.

[26] Wenfei Fan, Ping Lu, and Chao Tian. 2020. Unifying Logic Rules and Machine Learning for Entity Enhancing. *Sci. China Inf. Sci.* 63, 7 (2020).

[27] Anna Fariha, Ashish Tiwari, Alexandra Meliou, Arjun Radhakrishna, and Sumit Gulwani. 2021. CoCo: Interactive Exploration of Conformance Constraints for Data Understanding and Data Cleaning. In *SIGMOD*. ACM, 2706–2710.

[28] Anna Fariha, Ashish Tiwari, Arjun Radhakrishna, Sumit Gulwani, and Alexandra Meliou. 2021. Conformance Constraint Discovery: Measuring Trust in Data-Driven Systems. In *SIGMOD*. ACM, 499–512.

[29] I.P. Fellegi and D. Holt. 1976. A Systematic Approach to Automatic Edit and Imputation. *J. of the American Statistical Association* 71, 353 (1976), 17–35.

[30] Ivan Fellegi and Alan B. Sunter. 1969. A theory for record linkage. *J. American Statistical Association* (1969).

[31] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and Robust Automated Machine Learning. In *NIPS*. 2962–2970.

[32] Lampros Flokas, Weiyuan Wu, Yejia Liu, Jiannan Wang, Nakul Verma, and Eugene Wu. 2022. Complaint-Driven Training Data Debugging at Interactive Speeds. In *SIGMOD*. 369–383.

[33] Michael Garey and David Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company.

[34] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. 2013. The LLUNATIC data-cleaning framework. *PVLDB* 6, 9 (2013), 625–636.

[35] Stella Giannakopoulou, Manos Karpathiotakis, and Anastasia Ailamaki. 2020. Cleaning denial constraint violations through relaxation. In *SIGMOD*. 805–815.

[36] Amir Gilad, Daniel Deutch, and Sudeepa Roy. 2020. On multiple semantics for declarative database repairs. In *SIGMOD*. 817–831.

[37] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. 2021. Revisiting deep learning models for tabular data. *NeurIPS* 34 (2021), 18932–18943.

[38] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. 2022. Why do tree-based models still outperform deep learning on typical tabular data?. In *Advances in Neural Information Processing Systems*.

[39] Shubha Guha, Falaah Arif Khan, Julia Stoyanovich, and Sebastian Schelter. 2023. Automated Data Cleaning Can Hurt Fairness in Machine Learning-based Decision Making. In *ICDE*. IEEE, 3747–3754.

[40] Rihan Hai, Christos Koutras, Andra Ionescu, Ziyu Li, Wenbo Sun, Jessie van Schijndel, Yan Kang, and Asterios Katsifodimos. 2023. Amalur: Data Integration Meets Machine Learning. In *ICDE*. IEEE, 3729–3739.

[41] Alireza Heidari, Joshua McGrath, Ihab F. Ilyas, and Theodoros Rekatsinas. 2019. HoloDetect: Few-Shot Learning for Error Detection. In *SIGMOD*. 829–846.

[42] Benjamin Hilprecht, Christian Hammacher, Eduardo Souza dos Reis, Mohamed Abdelaal, and Carsten Binnig. 2023. DiffML: End-to-end Differentiable ML Pipelines. In *DEEM@SIGMOD*. ACM, 7:1–7:7.

[43] David C Hoaglin and Boris Iglewicz. 1987. Fine-tuning some resistant rules for outlier labeling. *Journal of the American statistical Association* 82, 400 (1987), 1147–1149.

[44] Sean Kandel, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. 2012. Enterprise Data Analysis and Visualization: An Interview Study. *IEEE Trans. Vis. Comput. Graph.* 18, 12 (2012), 2917–2926.

[45] Bojan Karlas, Peng Li, Renzhi Wu, Nezihe Merve Gürel, Xu Chu, Wentao Wu, and Ce Zhang. 2020. Nearest Neighbor Classifiers over Incomplete Information: From Certain Answers to Certain Predictions. *CoRR* abs/2005.05117 (2020). arXiv:2005.05117 https://arxiv.org/abs/2005.05117

[46] Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning (ICML)*. PMLR, 1885–1894.

[47] 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).

[48] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J. Franklin, and Ken Goldberg. 2016. ActiveClean: Interactive Data Cleaning For Statistical Modeling. *PVLDB* 9, 12 (2016), 948–959.

[49] Sanjay Krishnan and Eugene Wu. 2019. AlphaClean: Automatic Generation of Data Cleaning Pipelines. *CoRR* abs/1904.11827 (2019). http://arxiv.org/abs/1904.11827

[50] Peng Li, Zhiyi Chen, Xu Chu, and Kexin Rong. 2023. DiffPrep: Differentiable Data Preprocessing Pipeline Search for Learning over Tabular Data. *Proc. ACM Manag. Data* 1, 2 (2023), 183:1–183:26.

[51] Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. 2019. CleanML: A Benchmark for Joint Data Cleaning and Machine Learning [Experiments and Analysis]. *CoRR* abs/1904.09483 (2019).

[52] Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. 2021. CleanML: A Study for Evaluating the Impact of Data Cleaning on ML Classification Tasks. In *ICDE*. IEEE, 13–24.

[53] Hanmo Liu, Shimin Di, and Lei Chen. 2023. Incremental Tabular Learning on Heterogeneous Feature Space. *Proc. ACM Manag. Data* 1, 1 (2023), 18:1–18:18.

[54] Zifan Liu, Zhechun Zhou, and Theodoros Rekatsinas. 2020. Picket: Self-supervised Data Diagnostics for ML Pipelines. *CoRR* abs/2006.04730 (2020). https://arxiv.org/abs/2006.04730

[55] Brandon Lockhart, Jinglin Peng, Weiyuan Wu, Jiannan Wang, and Eugene Wu. 2021. Explaining Inference Queries with Bayesian Optimization. *PVLDB* 14, 11 (2021), 2576–2585.

[56] Mohammad Mahdavi and Ziawasch Abedjan. 2020. Baran: Effective error correction via a unified context representation and transfer learning. *PVLDB* 13, 12 (2020), 1948–1961.

[57] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Mad-

den, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha: A Configuration-Free Error Detection System. In *SIGMOD*. 865–882.

[58] Baharan Mirzasoleiman, Jeff A. Bilmes, and Jure Leskovec. 2020. Coresets for Data-efficient Training of Machine Learning Models. In *ICML (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 6950–6960.

[59] Baharan Mirzasoleiman, Kaidi Cao, and Jure Leskovec. 2020. Coresets for Robust Training of Deep Neural Networks against Noisy Labels. In *NeurIPS*.

[60] Felix Neutatz, Binger Chen, Ziawasch Abedjan, and Eugene Wu. 2021. From Cleaning before ML to Cleaning for ML. *IEEE Data Eng. Bull.* 44, 1 (2021), 24–41.

[61] Felix Neutatz, Binger Chen, Yazan Alkhatib, Jingwen Ye, and Ziawasch Abedjan. 2022. Data Cleaning and AutoML: Would an optimizer choose to clean? *Datenbank-Spektrum* (2022).

[62] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[63] Neoklis Polyzotis, Martin Zinkevich, Sudip Roy, Eric Breck, and Steven Whang. 2019. Data validation for machine learning. *MLSys* (2019).

[64] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *J. Mach. Learn. Res.* 21 (2020), 140:1–140:67.

[65] Alexander Ratner, Stephen H. Bach, Henry R. Ehrenberg, Jason Alan Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid Training Data Creation with Weak Supervision. *PVLDB* 11, 3 (2017), 269–282.

[66] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *PVLDB* 10, 11 (2017), 1190–1201.

[67] Cédric Renggli, Luka Rimanic, Nezihe Merve Gürel, Bojan Karlas, Wentao Wu, and Ce Zhang. 2021. A Data Quality-Driven View of MLOps. *IEEE Data Eng. Bull.* 44, 1 (2021), 11–23.

[68] Herbert Robbins and Sutton Monro. 1951. A Stochastic Approximation Method. *The Annals of Mathematical Statistics* 22, 3 (1951), 400–407.

[69] Yevgeny Seldin and Naftali Tishby. 2008. Multi-classification by categorical features via clustering. In *ICML (Proceedings of Machine Learning Research)*. PMLR, 920–927.

[70] Vraj Shah, Thomas Parashos, and Arun Kumar. 2024. How do Categorical Duplicates Affect ML? A New Benchmark and Empirical Analyses. *PVLDB* 17, 6 (2024), 1391–1404.

[71] Shafaq Siddiqi, Roman Kern, and Matthias Boehm. 2024. SAGA: A Scalable Framework for Optimizing Data Cleaning Pipelines for Machine Learning Applications. *Proc. ACM Manag. Data* (2024).

[72] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. 2017. PixelDefend: Leveraging Generative Models to Understand and Defend against Adversarial Examples. arXiv:1710.10766 [cs.LG]

[73] Ki Hyun Tae, Yuji Roh, Young Hun Oh, Hyunsu Kim, and Steven Euijong Whang. 2019. Data Cleaning for Accurate, Fair, and Robust Models: A Big Data - AI Integration Approach. In *DEEM@SIGMOD 2019*. ACM, 5:1–5:4.

[74] Panos Toulis, Thibaut Horel, and Edoardo M Airoldi. 2021. The proximal robbins–monro method. *Journal of the Royal Statistical Society Series B: Statistical Methodology* 83, 1 (2021), 188–212.

[75] Larysa Visengeriyeva and Ziawasch Abedjan. 2018. Metadata-driven error detection. In *SSDBM*. 1:1–1:12.

[76] Zhikai Wang and Yanyan Shen. 2023. Incremental Learning for Multi-Interest Sequential Recommendation. In *ICDE*. IEEE, 1071–1083.

[77] Zichong Wang, Yang Zhou, Meikang Qiu, Israat Haque, Laura Brown, Yi He, Jianwu Wang, David Lo, and Wenbin Zhang. 2023. Towards Fair Machine Learning Software: Understanding and Addressing Model Bias Through Counterfactual Thinking. *arXiv preprint arXiv:2302.08018* (2023).

[78] Weiyuan Wu, Lampros Flokas, Eugene Wu, and Jiannan Wang. 2020. Complaint-driven Training Data Debugging for Query 2.0. In *SIGMOD*. ACM, 1317–1334.

[79] Mohamed Yakout, Laure Berti-Équille, and Ahmed K. Elmagarmid. 2013. Don't Be SCAREd: Use SCalable Automatic REpairing with Maximal Likelihood and Bounded Changes. In *SIGMOD*. ACM.

[80] Mohamed Yakout, Ahmed K. Elmagarmid, Jennifer Neville, Mourad Ouzzani, and Ihab F. Ilyas. 2011. Guided data repair. *PVLDB* 4, 5 (2011), 279–289.

[81] Tianping Zhang, Zheyu Aqa Zhang, Zhiyuan Fan, Haoyan Luo, Fengyuan Liu, Qian Liu, Wei Cao, and Li Jian. 2023. OpenFE: Automated Feature Generation with Expert-level Performance. In *ICML (Proceedings of Machine Learning Research, Vol. 202)*. PMLR, 41880–41901.

[82] Tianyi Zhou, Shengjie Wang, and Jeff A. Bilmes. 2021. Robust Curriculum Learning: from clean label detection to noisy label self-correction. In *ICLR*.

A DATASETS

As shown in [61], the real-world benchmark datasets contain errors in a few (*e.g.,* 1-3) attributes. Thus, we manually inject errors in 1-3 attributes for each tested dataset; in particular, for a selected cell $t[A]$ $(= c)$ to be corrupted, we randomly replace its original value c with another valid value c' $(\neq c)$ in the domain $\text{dom}(A)$ of attribute A. The details of the tested datasets in Table 2 are as follows.

- **Adult:** It contains demographic information of individuals. The ML prediction is to determine whether a person's income exceeds 50K per year. The class labels are <=50k or >50k. In order to inject noises, we select education and race as the influential and non-influential attribute, respectively; we set marital-status as the attribute to inject noise for influential/non-influential/random tuples, where such tuples are from groups of $\mathcal{D}_{\text{train}}$ partitioned on attribute marital-status. Then, we generate the dirty version $\mathcal{D}^-_{\text{train}}$ of $\mathcal{D}_{\text{train}}$ by injecting noises into the selected attributes/tuples, with the error rate in Table 2.
- **Nursery:** It includes the family and financial circumstances of children. The prediction is to assign a ranking label to each child for kindergartens. The class labels include not recommend, recommend, strongly recommend, priority, and top priority. We select health/social as the influential/non-influential attribute; we set health, has_nurs and parents as the attributes to inject noises for influential/non-influential/random tuples, where different types of tuples are from groups of $\mathcal{D}_{\text{train}}$ partitioned on parents. Then, we produce the $\mathcal{D}^-_{\text{train}}$ by injecting noises into the selected attributes/tuples.
- **Default:** It records default payments of customers in Taiwan from April 2005 to September 2005. The ML prediction is to estimate whether a customer will default payments in the next month. The class labels are Yes or No. We select PAY_0 and PAY_5 as the influential and non-influential attribute, respectively; we set PAY_AMT1 and PAY_0 as the attributes to inject noises for influential/non-influential/random tuples, where the tuples are from groups of $\mathcal{D}_{\text{train}}$ partitioned on SEX. Then, we inject noises into the selected attributes/tuples.
- **German:** It contains financial information of bank account holders. The prediction is to determine the creditworthiness of a holder. The class labels are Good or Bad. We select status-of-existing-checking-account and present-residence-since as the influential and non-influential attribute, respectively; we set status-of-existing-checking -account and savings-account/bonds as the attributes to inject noises for influential/non-influential/random tuples, where the tuples are from groups of $\mathcal{D}_{\text{train}}$ partitioned on other-installment-plans. Then, we produce the $\mathcal{D}^-_{\text{train}}$ by injecting noises into the selected attributes/tuples.

B MORE ABOUT MODEL TRAINING

Below we present the incremental training for $\hat{\mathcal{M}}$ and $\mathcal{M}$.

(1) Assistant model $\hat{\mathcal{M}}$. The training for $\hat{\mathcal{M}}$ consists of two parts: (1) *model unlearning* and (2) *incremental learning*, to eliminate the negative effect of dirty tuples in $\mathcal{D}_{k-1}$ (which $\hat{\mathcal{M}}$ is previously trained on) and to update $\hat{\mathcal{M}}$ with cleaned tuples in $\mathcal{D}_k$, respectively.

(1A) Model unlearning. Denote by $\Delta\mathcal{D}_k$ the set of tuples in $\mathcal{D}_{k-1}$ that are cleaned by $\mathcal{A}_{\text{CR}}$ in $\mathcal{D}_k$: $\Delta\mathcal{D}_k = \{t \mid t \in \mathcal{D}_{k-1} \text{ and } t \notin \mathcal{D}_k\}$; *i.e.,* $\Delta\mathcal{D}_k$ contains biased/dirty tuples. To debias the negative effects of $\Delta\mathcal{D}_k$ from $\hat{\mathcal{M}}$, we adopt the influence function of [46], an effective tool to estimate parameter change of a model when tuples are removed/modified in the training data. Intuitively, it mimics the scenario that we retrain $\hat{\mathcal{M}}$ with a smaller training data, $\mathcal{D}_{k-1} \setminus \Delta\mathcal{D}_k$, in the $(k-1)$-th round, by removing dirty tuples in $\Delta\mathcal{D}_k$ from $\mathcal{D}_{k-1}$. To achieve this, for each $t = (x, y) \in \Delta\mathcal{D}_k$, where x and y are the attributes and label of t, respectively, we upweight t by an infinitesimal amount ϵ; the influence of upweighting t on parameters of $\hat{\mathcal{M}}$ is

$$\mathcal{I}_{\text{up,params}}(t) \overset{\text{def}}{=} \frac{\mathrm{d}\theta_{\epsilon,t}}{\mathrm{d}\epsilon}\Big|_{\epsilon=0} = -H_\theta^{-1}\nabla_{\theta_{k-1}} l(\hat{\mathcal{M}}(x;\theta_{k-1}), y),$$

where θ_k is the parameter of $\hat{\mathcal{M}}$ in the k-th round, $\nabla_{\theta_{k-1}}$ is the partial derivation of θ_{k-1} *w.r.t.* ϵ, H_θ is the Hession Matrix, and $l(\hat{\mathcal{M}}(x;\theta_k), y)$ is the loss of tuple t in the k-th round. Note that removing t from the training set is the same as upweighting it by $\epsilon = -\frac{1}{|D_{k-1}|}$, such that the parameter θ_k of $\hat{\mathcal{M}}$ in the k-th round can be updated as $\theta_k = \theta_{k-1} - \frac{1}{|D_{k-1}|}\sum_{t\in\Delta\mathcal{D}_k}\mathcal{I}_{\text{up,params}}(t)$. In particular, when $\hat{\mathcal{M}}$ is non-convex, H_θ may contain negative eigenvalues and H_θ^{-1} does not exist. To cope with this, we follow [46] and add a small dampening coefficient to the matrix's diagonal to ensure H_θ^{-1} exists.

(1B) Incremental learning. Previous incremental strategies [53, 76] mainly focus on retaining the performance of $\hat{\mathcal{M}}$ on both old and new datasets, *i.e.,* $\mathcal{D}_{k-1}$ and $\mathcal{D}_k$. However, since $\mathcal{D}_k$ is cleaned from $\mathcal{D}_{k-1}$, we adopt a simple strategy that continuously fine-tunes $\hat{\mathcal{M}}$ with updated $\mathcal{D}_k$ in e epochs, where $\hat{\mathcal{M}}$ inherits the parameters from the model unlearning step. This strategy works well since the negative effects of dirty data in $\mathcal{D}_{k-1}$ have already been removed.

(2) Downstream model $\mathcal{M}$. We train $\mathcal{M}$ on accumulated $\mathcal{D}_{\text{clean}}$ by (1) *identifying new clean data* and (2) *model refinement* as follows.

(2A) Identifying new clean data. The creator first identifies new clean data $\Delta\mathcal{D}_{\text{clean}}$ from $\mathcal{D}_k$ with the help of $\hat{\mathcal{M}}$. To do this, we initialize $\Delta\mathcal{D}_{\text{clean}}$ to empty and compute the dynamic loss $L_k(t)$ for each tuple t in $\mathcal{D}_k$ [82], using the exponential moving average (EMA):

$$L_k(t) = \lambda \cdot l(\hat{\mathcal{M}}(x;\theta_k), y) + (1-\lambda)\cdot L_{k-1}(t),$$

where $t = (x, y) \in \mathcal{D}_k$, $\lambda \in [0, 1]$ is a discounting factor, and $L_k(t_i)$ is the accumulated loss of t from 0 to k epochs. Intuitively, the larger $L_k(t)$, the more likely t is dirty. For each tuple $t \in \mathcal{D}_k$, we monitor the dynamic loss $L_k(t)$ of $\hat{\mathcal{M}}$. Given thresholds η_1 and η_2, we add a tuple t into $\Delta\mathcal{D}_{\text{clean}}$ if one of the following is satisfied:

- $L_{k-1}(t) - L_k(t) \geq \eta_1$, indicating that $\mathcal{A}_{\text{CR}}$ has effectively enhanced t's quality and thus t can be considered as a clean tuple.
- $L_k(t) \leq \eta_2$, suggesting that no matter whether t has been cleaned by $\mathcal{A}_{\text{CR}}$ or not, it is likely to be clean due to its low $L_k(t)$.

(2B) Model refinement. Rock4ML adopts Stochastic Gradient Descent (SGD) [68] for iterative model refinement, to leverage insights from both new clean data $\Delta\mathcal{D}_{\text{clean}}$ and the accumulated clean data $\mathcal{D}_{\text{clean}}$. We adapt SGD for continuous integration of clean data, such that $\mathcal{M}$ can be progressively improved with more clean data accumulated.

We apply full-gradient computation to the cleaned dataset $\mathcal{D}_{\text{clean}}$, denoted as $g_{\text{clean}}(\cdot)$, taking into account all data points in $\mathcal{D}_{\text{clean}}$:

$$g_{\text{clean}}(\theta_{k-1}) = \frac{1}{|\mathcal{D}_{\text{clean}}|} \sum_{(x,y)\in\mathcal{D}_{\text{clean}}} \nabla l(\mathcal{M}(x;\theta_{k-1}), y).$$

The idea is to ensure a continuous training of $\mathcal{M}$ by effectively incorporating the data cleaned after the critic (see below).

We calculate the gradient from the newly cleaned data $\Delta\mathcal{D}_{\text{clean}}$ after each round of critic as follows, denoted by $g_{\Delta\text{clean}}(\cdot)$:

$$g_{\Delta\text{clean}}(\theta_{k-1}) = \frac{1}{|\Delta\mathcal{D}_{\text{clean}}|} \sum_{(x,y)\in\Delta\mathcal{D}_{\text{clean}}} \frac{\nabla l(\mathcal{M}(x;\theta_{k-1}), y)}{p(x,y)}.$$

Here $p(x,y)$ denotes the probability for tuple (x,y) in $\mathcal{D}_k$ to be picked to clean (see below). We use $p(x,y)$ to adjust any bias that may arise from non-uniform sampling or selective cleaning. By weighting the gradient contribution of each newly cleaned tuple based on the inverse of its selection probability, the gradient computation accurately reflects the entire dataset, and thus provides an unbiased foundation for the subsequent updates of model parameters.

We compute the parameter updates by taking into account the gradients from both data sources, which are weighted by their respective proportions in the entire dataset $\mathcal{D}_k$, as follows:

$$g(\theta_{k-1}) = \frac{|\mathcal{D}_{\text{clean}}|}{|\mathcal{D}_k|} g_{\text{clean}}(\theta_{k-1}) + \frac{|\mathcal{D}_k| - |\mathcal{D}_{\text{clean}}|}{|\mathcal{D}_k|} g_{\Delta\text{clean}}(\theta_{k-1}).$$

Intuitively, the weighting scheme balances the influence of (a) the stable, already cleaned data $\mathcal{D}_{\text{clean}}$, and (b) the dynamic, newly cleaned data sampled from $\mathcal{D}_k \setminus \mathcal{D}_{\text{clean}}$ on the model evolution.

Using the combined gradient g, we update the parameters of $\mathcal{M}$:

$$\theta_k = \theta_{k-1} - \zeta_k g(\theta_{k-1}),$$

where ζ_k denotes the learning rate for round k, modulating the extent of each update in response to the calculated gradient.

To ensure the convergence of SGD, the learning rate ζ_k is adjusted at each round following the Robbins-Monro conditions [68], *i.e.,* on the one hand, ζ_k does not decrease too rapidly to allow sufficient exploration of the parameter space and on the other hand, it decreases fast enough to ensure convergence to an optimum.

Finally, we accumulate $\mathcal{D}_{\text{clean}}$, *i.e.,* $\mathcal{D}_{\text{clean}} = \Delta\mathcal{D}_{\text{clean}} \cup \mathcal{D}_{\text{clean}}$.

C PROOF OF THEOREM 1

Theorem 1: *Given an α-accurate $\mathcal{A}_{\text{CR}}$,* Rock4ML *guarantees to terminate and returns (a) a cleaned dataset $\mathcal{D}_{\text{clean}}$ as $\mathcal{D}_{\mathcal{M}}$, and (b) model $\mathcal{M}$ trained with $\mathcal{D}_{\text{clean}}$ such that* $\text{acc}(\mathcal{M}, \mathcal{D}_{\text{clean}}) > \text{acc}(\mathcal{M}, \mathcal{D})$. $\square$

Proof: We start with three observations.

- (F1). The CR method $\mathcal{A}_{\text{CR}}$ is α-accurate ($\alpha \in (0.5, 1]$), *i.e.,* it correctly fixes erroneous data with probability α. We assume that the data correctly fixed by $\mathcal{A}_{\text{CR}}$ raises no error signals by the ML model $\mathcal{M}$ as in practice, and hence remains unchanged. Intuitively, iteratively applying $\mathcal{A}_{\text{CR}}$ approaches the "perfect method" Oracle, which correctly fixes all errors in the data.
- (F2). The Stochastic Gradient Descent (SGD) process converges on unbiased gradient estimates with a step size sequence $\{\gamma_t\}$ under the Robbins-Monro conditions, as verified by [19, 74].
- (F3). Training $\mathcal{M}$ with clean data makes a closer approximation of the true data distribution and better test accuracy than training with dirty data, as observed in [48, 59, 72].

Based on the observations, the proof consists of three steps.

- (S1) **Iterative data cleaning**: For part (a) of Theorem 1, we show that iterative applications of $\mathcal{A}_{\text{CR}}$ reduces the error rate in the dataset (F1) such that $\mathcal{D}_{\mathcal{M}}$ obtained more accurately reflects the true data distribution and improves the accuracy of $\mathcal{M}$ (F3).
- (S2) **Convergence by unbiased gradient estimation**: For part (b) of the theorem, we prove the convergence of model refinement with progressively cleaned data by (F2); that is, Rock4ML ensures unbiased gradient estimation in SGD when incorporating gradients from both the accumulated and newly cleaned datasets. Rock4ML approaches this by adjusting gradients based on the probability of the data points being cleaned.
- (S3) **Accuracy improvement**: Also for part (b) of the theorem, we show that $\mathcal{M}$ trained with $\mathcal{D}_{\text{clean}}$ becomes more accurate. We first examine model convergence results in both convex and non-convex optimization scenarios. We then explore the impact of Rock4ML's cleaning on the accuracy of $\mathcal{M}$ in terms of dynamic loss reduction. We show that the selection criteria of $\Delta\mathcal{D}_{\text{clean}}$ yield lower dynamic losses, thereby demonstrating the improved model accuracy of $\mathcal{M}$ trained with $\mathcal{D}_{\mathcal{M}}$.

Below we provide the details of each step in the proof.

(S1) **Iterative data cleaning**: Given an initial dataset $\mathcal{D}$ with an error rate e_0, we inductively show that iterative applications of α-accurate $\mathcal{A}_{\text{CR}}$ can reduce the error rate below a threshold ϵ in k iterations, where k is determined by α, ϵ and $\mathbb{E}[p(x,y)]$ (see below).

For a dataset $\mathcal{D}_k$ in the k-th iteration, the likelihood of selecting its erroneous data points for cleaning is:

$$\mathbb{E}[p(x,y)] = \frac{1}{|\mathcal{D}_k|} \sum_{(x,y)\in\mathcal{D}_k} p(x,y),$$

where $|\mathcal{D}_k|$ is the size of $\mathcal{D}_k$, and $p(x,y)$ is determined by AlgIA and AlgIT. Assuming that the error rate is e_k after k iterations, then by incorporating $\mathbb{E}[p(x,y)]$, the error rate e_{k+1} is

$$e_{k+1} = e_k \cdot (1 - \alpha \cdot \mathbb{E}[p(x,y)]),$$

where α is the accuracy of $\mathcal{A}_{\text{CR}}$. To satisfy $e_k < \epsilon$, the equation:

$$e_0 \cdot (1 - \alpha \cdot \mathbb{E}[p(x,y)])^k < \epsilon$$

determines the necessary number k of iterations. That is,

$$k > \frac{\log(\epsilon/e_0)}{\log(1 - \alpha \cdot \mathbb{E}[p(x,y)])}.$$

This shows the connection between the number k of iterations and (a) the average error selection probability $\mathbb{E}[p(x,y)]$, and (b) the accuracy α of CR method $\mathcal{A}_{\text{CR}}$. For example, when $\alpha = 0.6$ and $\mathbb{E}[p(x,y)] = 0.6$, to reduce $e_0 = 0.1$ below $\epsilon = 0.01$, it typically requires about 4 iterations. That is, we need k rounds in the creator-critic framework of Rock4ML to get the cleaned dataset $\mathcal{D}_{\mathcal{M}}$. In practice, Rock4ML typically converges in much fewer rounds.

(S2) **Convergence via unbiased gradient estimation**: Rock4ML converges when the gradient estimation during the training process is unbiased. The gradient from the already cleaned data $\mathcal{D}_{\text{clean}}$ in the k-th round, denoted by $g_{\text{clean}}(\theta_{k-1})$, is computed as follows:

$$g_{\text{clean}}(\theta_{k-1}) = \frac{1}{|\mathcal{D}_{\text{clean}}|} \sum_{(x,y)\in\mathcal{D}_{\text{clean}}} \nabla l(\mathcal{M}(x;\theta_{k-1}), y),$$

where $\nabla l(\mathcal{M}(x;\theta_{k-1}), y)$ denotes the gradient of the loss function l *w.r.t.* model $\mathcal{M}$'s parameters θ_{k-1} in the (k-1)-th round, evaluated at the predicted label $\mathcal{M}(x;\theta_{k-1})$ and the truth label y of a data point (x,y). That is, the continuous training of $\mathcal{M}$ integrates all tuples in $\mathcal{D}_{\text{clean}}$ and is unbiased by the use of the full-gradient method.

Denote by $g_{\Delta\text{clean}}(\theta_{k-1})$ the gradient from the newly cleaned data $\Delta\mathcal{D}_{\text{clean}}$ in the k-th round. Given probability $p(x,y)$ for selecting data to clean, Rock4ML conducts gradient estimation as follows:

$$g_{\Delta\text{clean}}(\theta_{k-1}) = \frac{1}{|\Delta\mathcal{D}_{\text{clean}}|}\sum_{(x,y)\in\Delta\mathcal{D}_{\text{clean}}}\frac{\nabla l(\mathcal{M}(x;\theta_{k-1}),y)}{p(x,y)}.$$

To show the unbiasedness of $g_{\Delta\text{clean}}(\theta_{k-1})$, we consider its expectation *w.r.t.* the data selection process for cleaning:

$$\mathbb{E}[g_{\Delta\text{clean}}(\theta_{k-1})] = \mathbb{E}\left[\frac{1}{|\Delta\mathcal{D}_{\text{clean}}|}\sum_{(x,y)\in\Delta\mathcal{D}_{\text{clean}}}\frac{\nabla l(\mathcal{M}(x;\theta_{k-1}),y)}{p(x,y)}\right].$$

Given that each data point (x,y) is independently chosen from $\mathcal{D}_k \setminus \mathcal{D}_{\text{clean}}$ with probability $p(x,y)$, the expected contribution of the gradient for any selected data is adjusted by its selection probability. This ensures that the overall expectation reflects the true gradient contribution from the subset $\mathcal{D}_k \setminus \mathcal{D}_{\text{clean}}$:

$$\mathbb{E}\left[\frac{\nabla l(\mathcal{M}(x;\theta_{k-1}),y)}{p(x,y)}\right] = \sum_{(x,y)\in\mathcal{D}_k\setminus\mathcal{D}_{\text{clean}}}\nabla l(\mathcal{M}(x;\theta_{k-1}),y).$$

In light of this, the combined gradient $g(\theta_{k-1})$ for model parameter update integrates gradients from both sources, appropriately weighted by their proportions in the total dataset $\mathcal{D}_k$:

$$g(\theta_{k-1}) = \frac{|\mathcal{D}_{\text{clean}}|}{|\mathcal{D}_k|}g_{\text{clean}}(\theta_{k-1}) + \frac{|\mathcal{D}_k| - |\mathcal{D}_{\text{clean}}|}{|\mathcal{D}_k|}g_{\Delta\text{clean}}(\theta_{k-1}).$$

Given the unbiasedness of $g_{\text{clean}}(\theta_{k-1})$ and the proven unbiased expectation of $g_{\Delta\text{clean}}(\theta_{k-1})$, the expectation of the combined gradient $E[g(\theta_{k-1})]$ accurately reflects the true gradient of the entire dataset $\mathcal{D}_k$ (note that $|\mathcal{D}_k| = |\mathcal{D}_{\text{clean}}| = |\mathcal{D}_{\mathcal{M}}|$ in the end). This ensures unbiased gradient estimation and model parameter updates:

$$\mathbb{E}[g(\theta_{k-1})] = \frac{1}{|\mathcal{D}_{\mathcal{M}}|}\sum_{(x,y)\in\mathcal{D}_{\mathcal{M}}}\nabla l(\mathcal{M}(x;\theta_{k-1}),y).$$

As a consequence, model parameters are updated according to:

$$\theta_k = \theta_{k-1} - \zeta_k g(\theta_{k-1}),$$

where the learning rate ζ_k satisfies Robbins-Monro conditions. From this it follows Rock4ML's convergence:

$$\sum_{k=1}^{\infty}\zeta_k = \infty, \quad \sum_{k=1}^{\infty}\zeta_k^2 < \infty.$$

That is, Rock4ML ensures stable convergence in its integrated model training and data processing of the creator-critic process.

(S3) **Accuracy improvement**: We next show that when Rock4ML converges, $\mathcal{M}$ is more accurate than $\mathcal{M}$ trained with the original dataset $\mathcal{D}$, given that $\mathcal{M}$ is trained with progressively cleaned datasets. We prove that the process reduces dynamic loss by considering convex and non-convex optimization cases.

Convex optimization. In convex optimization scenarios, Rock4ML ensures that model $\mathcal{M}$ converges at the global optimum θ^* via gradient descent. Employing unbiased gradient updates $g(\theta_{k-1})$, $\mathcal{M}$ iteratively refines its parameters with the cleaned data in $\mathcal{D}_{\text{clean}}$. As more data is cleaned and integrated into the training process, $\mathcal{M}$ gradually approaches θ^*, demonstrating convergence as:

$$\lim_{k\to\infty}\theta_k = \theta^*.$$

Non-convex optimization. In non-convex optimization, gradient descent on $\mathcal{M}$ typically converges at local optima or saddle points. However, by (F3), Rock4ML enhances the exploration of the loss function's landscape by progressively incorporating cleaned data during training. This increases the chances of finding lower local minima, and improves the accuracy of $\mathcal{M}$ trained with uncleaned data in non-convex settings. We show this by dynamic loss analysis.

Given a dataset $\mathcal{D}$, let $D_{\text{clean}}^{(k)}$ and $\Delta\mathcal{D}_{\text{clean}}^{(k)}$ denote the dataset D_{clean} and $\Delta\mathcal{D}_{\text{clean}}$ after k rounds, respectively. The conditions for a data point t to be added to $\Delta\mathcal{D}_{\text{clean}}^{(k)}$ are either $L_{k-1}(t) - L_k(t) \geq \eta_1$ or $L_k(t) \leq \eta_2$, where $L_k(t)$ is the dynamic loss of t in round k.

The total dynamic loss of $\mathcal{M}$ trained with $\mathcal{D}_{\text{clean}}^{(k)}$ at k-th round is:

$$\mathcal{L}_{\text{clean}}^{(k)} = \sum_{t\in\mathcal{D}_{\text{clean}}^{(k)}} L_k(t),$$

while the total dynamic loss of $\mathcal{M}$ trained with $\mathcal{D}$ is:

$$\mathcal{L}_{\mathcal{D}}^{(k)} = \sum_{t\in\mathcal{D}} L_k(t).$$

In the dynamic loss analysis, $\mathcal{D}_{\text{clean}}^{(k)} = \bigcup_{i=0}^{k-1}\Delta\mathcal{D}_{\text{clean}}^{(i)}$, where $\Delta\mathcal{D}_{\text{clean}}^{(i)}$ consists of tuples t categorized by their original state in dataset $\mathcal{D}$, either clean or dirty, processed in the i-th round.

Inherently clean tuples are incorporated into $\Delta\mathcal{D}_{\text{clean}}^{(k)}$ when they meet the condition $L_k(t) \leq \eta_2$, due to their inherent cleanliness and alignment with the model's accuracy criteria, thus resulting in a low dynamic loss. Their impact on the dynamic loss $\mathcal{L}_{\text{clean}}^{(k)}$ aligns with their effect on the dynamic loss $\mathcal{L}_{\mathcal{D}}^{(k)}$, since their contribution to the model's learning is both stable and predictable.

On the other hand, newly-cleaned dirty tuples are selected for $\Delta\mathcal{D}_{\text{clean}}^{(k)}$ when they satisfy $L_{k-1}(t) - L_k(t) \geq \eta_1$, indicating a significant reduction in the dynamic loss by $\mathcal{A}_{\text{CR}}$. Such tuples decrease $\mathcal{L}_{\text{clean}}^{(k)}$ more substantially compared to $\mathcal{L}_{\mathcal{D}}^{(k)}$. Denote the quantity of such tuples by d. Its reduction in dynamic loss is quantified as:

$$\Delta\mathcal{L}_{\text{clean}}^{(k)} = \sum_{i=1}^{k}\sum_{t\in\Delta\mathcal{D}_{\text{clean}}^{(i)}}(L_{i-1}(t) - L_i(t)) \geq \eta_1 \cdot d \geq 0.$$

Therefore, suppose that Rock4ML terminates at round k. Then we have the following by taking into account the reduction in loss from cleaning and the contribution of inherently clean data:

$$\mathcal{L}_{\text{clean}}^{(k)} + \Delta\mathcal{L}_{\text{clean}}^{(k)} \leq \mathcal{L}_{\mathcal{D}}^{(k)}.$$

Thus $\mathcal{L}_{\text{clean}}^{(k)} \leq \mathcal{L}_{\mathcal{D}}^{(k)} - \eta_1 \cdot d$. From this it follows that $\text{acc}(\mathcal{M}, \mathcal{D}_{\text{clean}}) > \text{acc}(\mathcal{M}, \mathcal{D})$, *i.e.,* part (b) of Theorem 1 holds. □

D PROOF OF THEOREM 2

Theorem 2: *Problem* IA *is NP-hard.* □

Proof: We prove the NP-hardness of problem IA by reduction from the 3-SAT problem, which is NP-complete (cf. [33]). The 3-SAT problem is to decide, given a Boolean formula ϕ in the conjunctive normal form with n variables and m clauses, each containing three literals (a variable x or its negation $\neg x$), whether there exists a truth value assignment to the variables that makes ϕ true. The reduction from a 3-SAT instance to a IA instance is given as follows.

(1) Schema $\mathcal{R}$: We use a schema $(A_1, A_2, \ldots, A_{2n}, \text{id}, \text{label})$, where for all $i \in [1, 2n]$, A_i is a Boolean attribute, id is in the range $[1, m]$, and label is in the range $[1, 2m]$.

(2) Dataset $D_k = (\mathcal{B}_k, C_k)$: Construct relations $\mathcal{B}_k$ and C_k of the

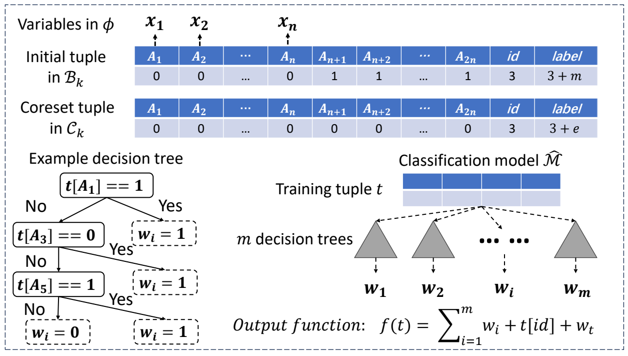

Variables in ϕ
x_1
x_2
x_n
Initial tuple in $\mathcal{B}_k$
A_1 A_2 … A_n A_{n+1} A_{n+2} … A_{2n} id label
0 0 … 0 1 1 … 1 3 $3+m$
Coreset tuple in $\mathcal{C}_k$
A_1 A_2 … A_n A_{n+1} A_{n+2} … A_{2n} id label
0 0 … 0 0 0 … 0 3 $3+e$
Example decision tree
$t[A_1] == 1$
No
Yes
$t[A_3] == 0$
$w_i = 1$
Yes
No
$t[A_5] == 1$
$w_i = 1$
Yes
No
$w_i = 0$
$w_i = 1$
Classification model $\widehat{\mathcal{M}}$
Training tuple t
m decision trees
w_1
w_2
w_i
w_m
Output function: $f(t) = \sum_{i=1}^{m} w_i + t[id] + w_t$

Figure 8: Problem IA reduction example

schema above, each having m tuples. For tuples t in $\mathcal{B}_k$, $t[\text{id}]$ takes values from 1 to m, respectively, $t[A_i] = 0$ ($i \in [1,n]$), $t[A_j] = 1$ ($j \in [n+1, 2n]$), and $t[\text{label}] = t[\text{id}]+m$. Intuitively, we use attributes $A_1, A_2, \ldots, A_n$ of tuples in $\mathcal{B}_k$ to encode the n variables of ϕ.

For tuples s in $\mathcal{C}_k$, their id attributes are instantiated in the same way as their counterparts in $\mathcal{B}_k$. However, we set $s[A_i] = 0$ ($i \in [1, 2n]$), and $s[\text{label}] = s[\text{id}] + e$, where e is the number of satisfied clauses in ϕ when all variables are set false.

Intuitively, we will use a CR method $\mathcal{A}_{\text{CR}}$ to simulate the truth assignment of ϕ, an ML model $\hat{\mathcal{M}}$ to check the satisfaction of the clauses of ϕ, and the correspondence between attributes A_i and A_{n+i} ($i \in [1,n]$) in $\mathcal{B}_k$ to identify influential attributes.

(3) CR method $\mathcal{A}_{\text{CR}}$: We use a rule-based method $\mathcal{A}_{\text{CR}}$ to adjust attributes in $\mathcal{B}_k$. It employs a single rule that makes use of a set $\mathcal{S}$ of influential attributes. The rule states that for each tuple $t \in \mathcal{B}_k$ and each $i \in [1,n]$, if the attribute A_i is identified as a influential attribute, then $t[A_i]$ takes the value of $t[A_{i+n}]$; while for $i > n$, $t[A_i]$ remains unchanged. That is, $\mathcal{A}_{\text{CR}}$ aligns correlated attributes such that the attribute pair A_i and A_{i+n} have the same value.

(4) Model $\hat{\mathcal{M}}$: We use a classification tree-based (regression) model $\hat{\mathcal{M}}$ to evaluate clause satisfaction based on the attribute values in $\mathcal{D}_k$. It consists of m decision trees, each for a clause C_i in ϕ. The i-th tree yields $w_i = 1$ if clause C_i is satisfied, otherwise $w_i = 0$. Figure 8 shows an example decision tree for the clause $C_i = (x_1 \vee \neg x_3 \vee x_5)$.

More specifically, the model $\hat{\mathcal{M}}$ predicts labels for each tuple $t \in D_k$ using the regression function $f(t) = \sum_{i=1}^{m} w_i + t[\text{id}] + w_t$, where w_t is a trainable parameter, making f differentiable on the input tuple t. The label is predicated as $\lfloor f(t) \rfloor$.

Observe that before applying $\mathcal{A}_{\text{CR}}$ to $\mathcal{B}_k$, for any tuple $t \in \mathcal{B}_k$, the function $f(t)$ is evaluated as $e + t[\text{id}] + w_t$. Since $t[\text{label}] = t[\text{id}] + m$ for tuples $t \in \mathcal{B}_k$, the training process adjusts w_t to $m - e$ such that $f(t) = t[\text{label}]$. When $\hat{\mathcal{M}}$ predicts any tuple $s \in \mathcal{C}_k$ in the validation process, $f(s) = e + s[\text{id}] + m - e$, which is not equal to $s[\text{label}] = s[\text{id}] + e$; as a consequence, initially $\text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) = 0$.

(5) Parameter δ: Define $\delta = 1$, which is the maximum possible improvement under an initial $\text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) = 0$.

The reduction can be obviously constructed in PTIME. We next show that there exists a set $\mathcal{S}$ of attributes that makes $\text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) = 1$ if and only if the 3-SAT instance ϕ is satisfiable.

$\Rightarrow$ First, we assume that there exists a set $\mathcal{S}$ such that $\mathcal{A}_{CR}$ on $\mathcal{S}$ makes $\text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{S}}) \geq \text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) + 1$. More specifically, the application of $\mathcal{A}_{CR}$ ensures that for any tuple t in $\mathcal{B}_k^{\mathcal{S}}$ and for any attribute $A_i \in \mathcal{S}$, $t[A_i] = 1$; and for any attribute $A_j \notin \mathcal{S}$, $t[A_j] = 0$. In fact, $\text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{S}}) \geq \text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) + 1$ implies that $\hat{\mathcal{M}}$ classifies all tuples in $\mathcal{C}_k$ correctly. Given $f(s) = \sum_{i=1}^{m} w_i + s[\text{id}] + w_t$ for each $s \in \mathcal{C}_k$ and $s[\text{label}] = e + s[\text{id}]$, it necessitates $w_t = 0$. Recall that during training on tuples t in $\mathcal{B}_k^{\mathcal{S}}$ using mean squared error, the optimal value for w_t is determined by the formula $w_t = \frac{\sum_{t \in \mathcal{B}_k^{\mathcal{S}}} (t[\text{label}] - t[\text{id}] - \sum_{i=1}^{m} w_i)}{m}$. If this optimal w_t is found to be 0, it implies that $\sum_{i=1}^{m} w_i = m$, indicating that each of the m clauses has been satisfied by the current attribute settings in $\mathcal{B}_k^{\mathcal{S}}$.

We give the truth assignment in ϕ as follows. For each tuple t in $\mathcal{B}_k^{\mathcal{S}}$ and $i \in [1,n]$, if $t[A_i] = 1$, then variable x_i is assigned true. Conversely, if $t[A_i] = 0$, then x_i is set to false.

One can verify that this truth assignment satisfies ϕ. Observe that the truth assignment is determined by $\hat{\mathcal{M}}$'s decision trees, each aligned with a specific clause C_i from ϕ. A tree yields $w_i = 1$ iff for any tuple $t \in \mathcal{B}_k^{\mathcal{S}}$ and $i \in [1,n]$, at least one $t[A_i]$ value meets the decision criterion, equivalently satisfying C_i by ensuring a literal's truth. This leads to $\sum_{i=1}^{m} w_i = m$, with $w_i = 1$ for all trees, which verifies ϕ's satisfiability via the truth assignment derived from $\mathcal{S}$.

$\Leftarrow$ Conversely, we show that the existence of a satisfying truth assignment for ϕ guarantees the existence of a set $\mathcal{S}$ of attributes such that $\mathcal{A}_{\text{CR}}$ on $\mathcal{S}$ in $\mathcal{B}_k$ ensures $\text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{S}}) \geq \text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k)+1$. Given a satisfying truth assignment for ϕ, the set $\mathcal{S}$ is identified as follows: For each variable x_i in ϕ, if x_i is true in the truth assignment, then we include the attribute A_i into the influential attribute set $\mathcal{S}$. Then we perform $\mathcal{A}_{\text{CR}}$ on $\mathcal{S}$ to obtain $\mathcal{D}_k^{\mathcal{S}}$, where for each $t \in \mathcal{B}_k^{\mathcal{S}}$ and $A_i \in \mathcal{S}$, $t[A_i] = 1$, while $t[A_j] = 0$ for $A_j \notin \mathcal{S}$. Suppose by contradiction that $\text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{S}}) < \text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) + 1$, *i.e.,* some $\mathcal{C}_k$ tuples are incorrectly classified. Given $f(s) = e + s[\text{id}] + w_t$ for $s \in \mathcal{C}_k$ where $s[\text{label}] = e+s[\text{id}]$, w_t cannot be 0 in order to produce incorrect predictions. A non-zero w_t contradicts the premise that ϕ is satisfiable. Indeed, during the training, for a tuple $t \in \mathcal{B}_k^{\mathcal{S}}$ with $t[\text{label}] = t[\text{id}]+m$, $\hat{\mathcal{M}}$ outputs $f(t) = m+t[\text{id}]+w_t$; this indicates that w_t must be 0 to train an accurate classifier $\hat{\mathcal{M}}$. □

E PROOF OF THEOREM 3

Theorem 3: *Problem* IT *is NP-hard.* □

Proof: The NP-hardness of IT is also shown by reduction from the 3-SAT problem. The reduction is given as follows. Consider a 3-SAT instance ϕ that has n variables and m conjunctive clauses.

(1) Schema $\mathcal{R}$: We use a ternary schema $(A_1, \text{id}, \text{label})$, where A_1 is a Boolean, id is in the range $[1, n]$, and label is in the range $[1, n+m]$.

(2) Dataset $\mathcal{D}_k = (\mathcal{B}_k, \mathcal{C}_k)$. We create relations $\mathcal{B}_k$ and $\mathcal{C}_k$ of the schema above, each having n tuples. For tuples t in $\mathcal{B}_k$, $t[\text{id}]$ takes values from 1 to n, respectively, $t[A_1] = 0$, and $t[\text{label}] = t[\text{id}] + m$. Intuitively, we use the n tuples in $\mathcal{B}_k$ to encode the n variables of ϕ.

For tuples s in $\mathcal{C}_k$, their id attributes are instantiated in the same way as their counterparts in $\mathcal{B}_k$. However, we set $s[A_1] = 1$, and $s[\text{label}] = s[\text{id}] + e$, where e is the number of satisfied clauses in ϕ when all variables in the 3-SAT instance ϕ are set true.

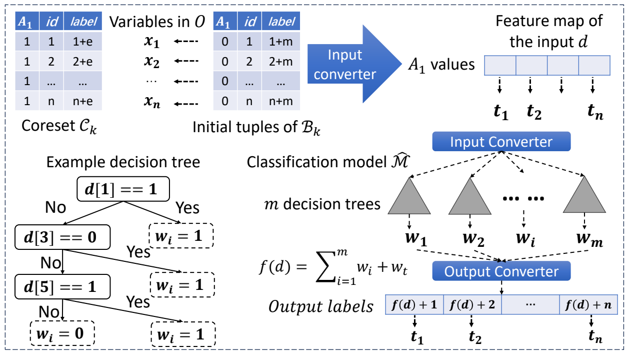


Figure 9: Problem IT reduction example

As will be seen shortly, we will use a CR method $\mathcal{A}_{\rm CR}$ to simulate the truth assignment of ϕ, an ML model $\hat{\mathcal{M}}$ to check the satisfaction of the clauses of ϕ, and the pairwise correspondence between tuples in $\mathcal{B}_k$ and those in C_k to identify influential tuples.

(3) CR method $\mathcal{A}_{\rm CR}$: We use $\mathcal{A}_{\rm CR}$ that, for each tuple t in $\mathcal{B}_k$, if t is identified as influential, then $t[A_1]$ takes the value of $s[A_1]$, where s is the tuple in C_k such that $t[\text{id}] = s[\text{id}]$. That is, $\mathcal{A}_{\rm CR}$ ensures that for any pair $t \in \mathcal{B}_k$ and $s \in C_k$ of tuples, if they have the same id, then the two must have the same A_1-attribute value.

(4) Model $\hat{\mathcal{M}}$: Design a classification tree-based (regression) model $\hat{\mathcal{M}}$ that uses decision trees for each 3-SAT clause, where each tree decides whether its clause is satisfied based on the A_1 values. The input to the model regression function is an array of A_1 values in the size of n, arranged in order by the tuple identifiers; that is, $\hat{\mathcal{M}}$ processes a batch d of n tuples simultaneously. The trees yield $w_i = 1$ for satisfied clauses C_i, 0 otherwise. Figure 9 shows an example for the clause $C_i = (x_1 \vee \neg x_3 \vee x_5)$, where $d[i]$ indicates the i-th A_1 value of the input d. The regression function $f(d) = \sum_{i=1}^{m} w_i + w_t$ aggregates the trees' outputs and $\hat{\mathcal{M}}$ predicts the label of each tuple t within the batch by $f(d) + t[\text{id}]$, where w_t is the trainable parameter, making f differentiable on the input d.

Observe that before applying $\mathcal{A}_{\rm CR}$ on $\mathcal{B}_k$, for a tuple $t \in \mathcal{B}_k$ with label $t[\text{id}]+m$, its predicted label is $f(d)+t[\text{id}] = e'+w_t+t[\text{id}]$. Here e' is the number of satisfied clauses of ϕ under a truth assignment that assigns false to all the variables of ϕ. Based on the mean squared error, the training adjusts w_t to $m - e' \neq 0$. When $\hat{\mathcal{M}}$ predicts any tuple $s \in C_k$ by batching n tuples in C_k into an input d', the predicted label of s is $f(d') + s[\text{id}] = e + m - e' + s[\text{id}]$, which is not equal to $s[\text{id}] + e$; as a result, we have that $\text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) = 0$.

(5) Parameter δ: Choose $\delta = 1$ as the target for maximum accuracy improvement under an initial $\text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) = 0$.

The reduction can be constructed in PTIME. We next show that there exists a set $\mathcal{T} \subseteq \mathcal{B}_k$ of influential tuples such that $\text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{T}}) \geq \text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) + 1$ iff ϕ is satisfiable.

$\Rightarrow$ First, assume that there exists a set $\mathcal{T}$ of tuples in $\mathcal{B}_k$ such that $\text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{T}}) \geq \text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) + 1$. Then the application of $\mathcal{A}_{\rm CR}$ ensures that for any tuple t in $\mathcal{T}$, if there exists a tuple s in C_k such that $t[\text{id}] = s[\text{id}]$, then $t[A_1]$ takes the value of $s[A_1]$. Since $\hat{\mathcal{M}}$ can correctly output all labels in C_k, the parameter w_t must be 0. During training with mean squared error, the optimal w_t equals $\frac{\sum_{\forall t \in \mathcal{B}_k^{\mathcal{T}}} (t[\text{label}] - t[\text{id}] - \sum_{i=1}^{m} w_i)}{n} = \frac{\sum_{\forall t \in \mathcal{B}_k^{\mathcal{T}}} (m - \sum_{i=1}^{m} w_i)}{n}$. When $w_t = 0$, $\sum_{i=1}^{m} w_i = m$; this leads to a satisfying assignment for all m clauses.

More specifically, the truth assignment is defined as follows. For each tuple t in $\mathcal{B}_k^{\mathcal{T}}$ with $t[\text{id}] = i$, if $t[A_1] = 1$, then x_i is assigned true. Conversely, if $t[A_1]$ remains 0, then x_i is set to false.

One can verify that this truth assignment satisfies ϕ. Indeed, observe that the truth assignment is determined by $\hat{\mathcal{M}}$'s decision trees, each corresponding to a clause C_i from ϕ. A tree yields $w_i = 1$ iff at least one A_1 value in the input d meets the decision criterion; in other words, it satisfies clause C_i. Since $\sum_{i=1}^{m} w_i = m$, all the clauses of ϕ are satisfied by the truth assignment.

$\Leftarrow$ Conversely, assume that ϕ is satisfied by a certain truth assignment for $x_1, \ldots, x_n$. Then we identify a set $\mathcal{T}$ of influential tuples in $\mathcal{B}_k$ as follows. If a variable x_i is true, we add $t \in \mathcal{B}_k$ to $\mathcal{T}$ if $t[\text{id}] = i$. Then we apply $\mathcal{A}_{\rm CR}$ on $\mathcal{T}$ to obtain $\mathcal{D}_k^{\mathcal{T}}$, such that $t[A_1] = 1$ for all tuples t in $\mathcal{T}$. We next show that $\mathcal{T}$ is indeed the set of influential tuples desired. Suppose by contradiction that $\text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{T}}) < \text{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) + 1$. Then $\hat{\mathcal{M}}$ fails to predict all C_k labels correctly. Indeed, when predicting labels of a batch d' of n tuples in C_k, since $f(d') = e + w_t$, the predicated label of each tuple $s \in C_k$ is $f(d') + s[\text{id}] = e + w_t + s[\text{id}]$. Meanwhile, the truth label of tuple s is $s[\text{id}] + e$, indicating $w_t \neq 0$. However, if ϕ is satisfied by a truth assignment, consider the batch d that consists of all tuples $t \in \mathcal{B}_k^{\mathcal{T}}$; then $\hat{\mathcal{M}}$ predicts $t[\text{label}]$ as $f(d) + t[\text{id}] = m + w_t + t[\text{id}]$, and $t[\text{label}]$ is $t[\text{id}] + m$; as a result, during the training phase, the optimal w_t must be adjusted to 0; this contradicts that $w_t \neq 0$. □