REIN: A Comprehensive Benchmark Framework for Data Cleaning Methods in ML Pipelines*

Mohamed Abdelaal, Christian Hammacher, Harald Schöning
Software AG, Darmstadt, Germany
first.last@softwareag.com

ABSTRACT

Nowadays, machine learning (ML) plays a vital role in many aspects of our daily life. In essence, building well-performing ML applications requires the provision of high-quality data throughout the entire life-cycle of such applications. Nevertheless, most of the real-world tabular data suffer from different types of discrepancies, such as missing values, outliers, duplicates, pattern violation, and inconsistencies. Such discrepancies typically emerge while collecting, transferring, storing, and/or integrating the data. To deal with these discrepancies, numerous data cleaning methods have been introduced. However, the majority of such methods broadly overlook the requirements imposed by downstream ML models. As a result, the potential of utilizing these data cleaning methods in ML pipelines is predominantly unrevealed. In this work, we introduce a comprehensive benchmark, called REIN[1], to thoroughly investigate the impact of data cleaning methods on various ML models. Through the benchmark, we provide answers to important research questions, e.g., where and whether data cleaning is a necessary step in ML pipelines. To this end, the benchmark examines 38 simple and advanced error detection and repair methods. To evaluate these methods, we utilized a wide collection of ML models trained on 14 publicly-available datasets covering different domains and encompassing realistic as well as synthetic error profiles.

1 INTRODUCTION

With the advent of modern computing technologies, many industries nowadays are developing robust ML models capable of analyzing big and complex data while delivering fast and accurate results on vast scales. Such results are typically harnessed by organizations and businesses to make better decisions without or with minimal human intervention. However, the correctness of such decisions broadly depends on the quality of the available data. According to a recent Gartner research [37], organizations believe poor data quality to be responsible for an average of $15 million per year in losses. Another study by IBM in 2016 [45] revealed that poor data quality costs the US economy $3.1 trillion per year. These studies illustrate that data quality problems are predominantly expensive and pervasive.

For decades, data quality has been an active research area. In this context, the data management community tackled the data quality problems as a part of the ETL workflows. Accordingly, numerous proposals have been introduced to automatically detect and/or repair data discrepancies [10, 12, 20, 32, 44, 46]. In fact, only a small fraction of these proposals considered the heterogeneity profiles of data errors while discovering and repairing the erroneous instances. In other words, most proposed techniques are dedicated to serve only one error type. Moreover, most of such methods have been developed in isolation from the downstream ML applications. Thus, the consequences of adopting such cleaning methods for predictive tasks are broadly concealed. Accordingly, a challenge of selecting the most well-suited cleaning strategies (i.e., combinations of error detection and repair methods) in ML pipelines arises.

In this paper, we tackle this challenge through introducing a benchmark framework, referred to as REIN. The main goal of REIN is to thoroughly investigate the interplay between data cleaning and ML modeling. Through extensive experiments, REIN examines plenty of cleaning strategies in combination with various ML models, covering classification, regression, clustering, and AutoML models. In REIN, we evaluate the error detection and repair methods while being adopted as stand-alone methods and as components in ML pipelines. To this end, it is necessary to possess the ground truth of the available dirty datasets. Nevertheless, it is not usually straightforward to find such datasets which are also well-suited for ML tasks. Another challenge of conducting such a comprehensive study is the scale of the intended experiments. The number of models to be trained are exploded due to involving plenty of error detection and repair methods (cf. Section 2). For such detection and repair methods, it is also crucial to provide the necessary configurations and signals, i.e., patterns, rules, and helping functions. Finally, ML models are inherently probabilistic, where resampling may change the results. Hence, we need to validate the conclusions obtained from the ML experiments.

In detail, the paper provides the following contributions: (1) We define an architectural framework to systematically evaluate error detection and repair tools dedicated to tabular data. In addition to the traditional evaluation measures relative to the ground truth, REIN enables data scientists and practitioners to properly judge their detection and repair methods using the performance of several predictive models. Moreover, REIN utilizes the intersection over union (IoU) metric to quantify the similarities between data cleaning methods. (2) We design a benchmark controller that efficiently manages the other components in the framework. Such a controller leverages the design-time knowledge, e.g., the error types and the ML tasks, to broadly sidestep unnecessary experiments, thus reducing the complexity of running the benchmark. (3) We provide a classification of the most prominent error detection and repair methods according to their methodology and the required configurations. (4) We examine the performance of the involved ML models in different scenarios which are characterized by the data version, i.e., ground truth, dirty, or repaired data. (5) We evaluate scalability of the error data cleaning methods through using small, medium, and large datasets as inputs to these methods. Moreover, we evaluate the robustness of such methods through repeating the experiments

*This work was supported (in part) by the Federal Ministry of Education and Research through grants 02L19C155, 01IS21021A (ITEA project number 20219).
[1]The source code, data, and other artifacts have been made available at https://github.com/mohamedyd/rein-benchmark

while monotonically increasing the error rate. (6) We adopt the Wilcoxon signed-rank test with continuity correction to compensate for the randomness inherited in the training process. To the best of our knowledge, REIN is the first large-scale benchmark framework which evaluates the data cleaning methods from different perspectives, including detection and repair performance, predictive accuracy, robustness, and scalability.

2 BENCHMARK OVERVIEW

In this section, we introduce the architecture of REIN together with our assumptions. REIN comprises several data processing and evaluation steps. Specifically, several dirty datasets $\Phi^- = \phi_1^-, \cdots, \phi_n^-, \phi_i^- \in \mathbb{R}^{u \times v}$ are used as inputs to different error detectors $\alpha_1, \cdots, \alpha_m$, where the superscript '–' denotes a dirty dataset and u, v denote the number of records and attributes in ϕ_i^-. Afterward, the erroneous instances, identified by each detection method, are replaced with repair candidates using a number of data repair methods $\beta_1, \cdots, \beta_k$. The result of this step is a new set of repaired datasets $\Phi^+ = \phi_{i,1}^+, \cdots, \phi_{i,\epsilon}^+$, where $\epsilon = m \times k$ represents the number of generated repair versions for each dirty dataset α_i and the superscript '+' denotes a repaired dataset. Finally, each repaired dataset $\phi_{i,j}^+$ is sampled to train several ML models $\gamma_1, \cdots, \gamma_h$, where h is the number of involved ML models. Thus, the number of ML experiments for each dirty dataset ϕ_i^- is $(\epsilon + 1) \times h \times s$, where each experiment is repeated s times to estimate the means and standard deviations, and the dirty version is added to the number of generated repaired versions.

To realize such a large-scale benchmark, we implemented the architecture depicted in Figure 1. A *data repository*, i.e., PostgreSQL database, is utilized to store the ground truth Φ_g, the dirty data Φ^-, and the set of generated repaired versions Φ^+. To properly control the experiments, an *error injection* module generates different types of errors with various error rates. Practically speaking, the task of error injection is carried out in an offline phase before running the experiments (cf. Section 5 for more details). Another component is the *data cleaning toolbox*, which is a pool containing all available error detection and repair tools. Some of these tools, such as NADEEF, HoloClean, and OpenRefine, cannot be utilized without providing them with a set of *cleaning signals*. Examples of such signals include functional dependency constraints, integrity constraints, knowledge bases, patterns, and pre-estimated configurations.

The main component in REIN is the *benchmark controller*, which connects all other components in the benchmark. The purpose of such a controller is three-fold: First, it smoothly exchanges the ground truth Φ_g, the dirty Φ^-, and the repaired data Φ^+ among the different modules. Second, it avoids unnecessary error detection and repair operations exploiting prior knowledge about the dirty datasets. For example, if a dataset is known to have duplicates (e.g., the *Citation* dataset), it is meaningless to run rule violation or outlier detection methods. Third, it exploits the prior knowledge to adapt the data preparation steps in accordance with the associated ML tasks. The last component in the architecture is the evaluation module, which serves the error detection and repair methods as well as the ML models. For instance, the evaluation module leverages several quality metrics to estimate the predictive performance of ML models trained on the ground truth, the dirty and the repaired data.

Another component is a pool of *ML models* which comprises a wide collection of classification, regression, and clustering methods. Moreover, REIN also examines two AutoML algorithms to check the performance of fully-automated pipelines consisting of data cleaning and modeling modules. Finally, an *evaluation module* examines the performance of data cleaning and modeling methods in terms of four metrics, including accuracy, latency, scalability, and robustness (cf. Section 6). Due to space constraints, we define in the README file of the source code: (1) how to run the benchmark with/without the ground truth of dirty datasets, and (2) how to readily extend the REIN framework by adding new datasets, ML models, and data cleaning tools.

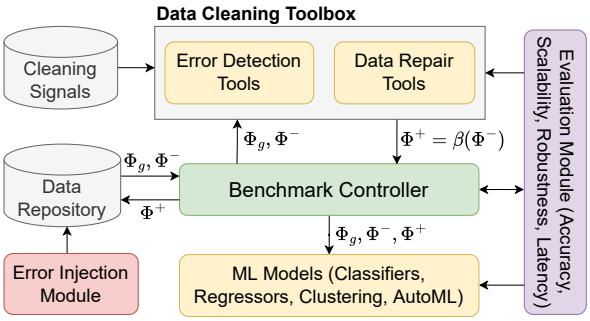


Figure 1: Benchmark architecture

3 DATA CLEANING METHODS

In this section, we provide an overview of the examined error detection and repair methods.

3.1 Error Detection Methods

In REIN, we selected 19 publicly-available error detection methods, which deal with the most common attribute and class errors in tabular data[2]. Table 1 lists the error detection methods and their targeted error types. Moreover, the table comprises the configurations and/or signals, i.e., patterns, constraints, helping functions, and knowledge bases, necessary for running each detection method. In REIN, we classify the error detection methods according to their methodology into two main categories, including (I) Non-learning methods and (II) ML-supported methods. As its name implies, the former category includes the methods and tools which detect errors using either a set of user-provided knowledge base, business rules, integrity constraints, or using a set of statistical measures. Each of these methods and tools typically tackle specific error types, e.g., duplicates, outliers, or missing values. The second category comprises the methods, e.g., Picket, ED2, and RAHA, which formulate the error detection task as a classification problem. These methods initially extract a set of features for each attribute. Such auto-generated features enable a classifier to differentiate between clean and dirty data samples. To train such a classifier, some training samples are selected to be labeled by an oracle. Below, we introduce the error detectors in each category.

Non-Learning Detectors: The first method in Table 1 is KATARA [10] which aligns the input dirty dataset with crowdsourced knowledge bases to identify and correct data samples that violate semantic patterns. To detect rule and pattern violations, NADEEF [12] treats data quality rules holistically via providing an interface for implementing denial constraints and other user-defined functions. Another relevant work is HoloClean [46] which combines qualitative and quantitative signals, e.g., denial constraints and correlations, in a statistical model that enables

[2] Attribute errors occur in the training features, while class errors occur in the labels

Table 1: Examined error detection and repair methods (The index (*Idx*) and abbreviation (*Abbr*) are used to refer to the detection and repair methods in the figures of Section 6)

Idx.	Detector	Abbr.	Cat.	Tackled Errors	Configs.	Idx.	Repair Method	Abbr.	Cat.	Tackled Errors	Configs.
K	KATARA [10]	–	I	Pattern violations	Knowledge Base	1	Ground Truth	GT	I	–	–
N	NADEEF [12]	–	I	Rule violations	FD Rules, Patterns	2	Delete	–	I	–	–
F	FAHES [44]	–	I	Missing Values	–	3	Imputation: Mean-Mode	Impute	I	MV/Outliers	–
H	HoloClean [46]	Holo	I	Rule violations	Denial Constraints	4	Imputation: Median-Mode	Impute	I	MV/Outliers	–
B	dBoost [34]	–	I	Outliers	Hyperparams	5	Imputation: Mode-Mode	Impute	I	MV/Outliers	–
O	OpenRefine [19]	OpnR	I	Inconsistencies	Clusters	6	Imputation: missForest [51]	MISS-Mix	I	MV/Outliers	–
I	Outlier Detector: IF [30]	IF	I	Outliers	Hyperparams	7	Imputation: DataWig [7]	DataWig-Mix	I	MV/Outliers	–
S	Outlier Detector: SD [55]	SD	I	Outliers	Hyperparams	8	Imputation: missForest-missForest [51]	MISS-Sep	I	MV/Outliers	–
Q	Outlier Detector: IQR [55]	IQR	I	Outliers	Hyperparams	9	Imputation: missForest-DataWig	MISS-Datawig	I	MV/Outliers	–
V	MV Detector [36]	MVD	I	Missing Values	–	10	Imputation: Decision Tree-missForest	DT-MISS	I	MV/Outliers	Hyperparams
D	Key Collision [29]	DupID	I	Duplicates	Key Columns	11	Imputation: Bayesian Ridge-missForest	Bayes-MISS	I	MV/Outliers	Hyperparams
Z	ZeroER [54]	–	I	Duplicates	Blocking Functions	12	Imputation: KNN-missForest	KNN-MISS	I	MV/Outliers	Hyperparams
C	CleanLab [39]	–	I	Mislabels	Hyperparams	13	HoloClean [46]	Holo	I	MV/Rule Violation	Denial Constraints
M	Min-K [2]	Min	I	Holistic	Hyperparams	14	OpenRefine [19]	OpenR	I	Inconsistencies	Clusters
X	Max Entropy [2]	Max	I	Holistic	Hyperparams	15	BARAN [32]	–	I	Holistic	Labels
T	Metadata-Driven [53]	Meta	II	Holistic	Labels	16	CleanLab [39]	–	II	Mislabels	–
R	RAHA [33]	–	II	Holistic	Labels	17	ActiveClean [26]	–	II	–	Repairs, Labels
E	ED2 [38]	–	II	Holistic	Labels	18	BoostClean [25]	–	II	–	Repair, Labels
P	Picket [31]	–	II	Holistic	–	19	CPClean [22]	–	II	–	Hyperparams, Repairs

detecting and repairing missing values and rule/constraint violations. To identify inconsistencies and pattern violations, the OpenRefine tool [19] enables users to visually explore the dirty datasets through faceting and filtering operations. FAHES [44] is another tool which detects disguised missing values, e.g., "999999" for a phone number. To this end, FAHES employs a syntactic pattern detection module for categorical data and a density-based outlier detection module for numerical data. To detect explicit missing values, REIN implements a method to find empty or *NAN* entries.

dBoost [34] is an outlier detection method which integrates several of the most widely applied outlier detection algorithms, including histograms, Gaussian, and multivariate Gaussian mixtures. To find the optimal hyperparameters for such algorithms, dBoost employs random search, where the search space is all the possible configurations. Other outlier detection methods involve Standard Deviation (SD), Interquartile Range (IQR) [55], and Isolation Forest (IF) [30]. The former method annotates a cell $x \in A$, where A denotes an attribute, as an outlier if it is n numbers of standard deviations away from the mean of entries in A. A more resistant statistical measure is IQR, defined as the difference between the 25th and 75th percentiles of an attribute A, i.e., $IQR_A = Q_3 - Q_1$. In this case, an outlier is any value laying outside the range of $[Q_1 - k \times IQR_A, Q_3 + k \times IQR_A]$, where k and n are hyperparameters. The latter method targets identifying outliers without profiling all data samples. Specifically, the IF method builds an ensemble of isolation binary trees for the dirty dataset, and outliers are the data samples that have shorter average path lengths on the binary trees.

To detect duplicates, REIN examines two methods, namely Key Collision [29] and ZeroER [54]. The former method requires user-provided information about the key attributes assumed to be unique. In this case, two records can be detected as duplicates whenever they share the same value on the key attributes. The latter method relies on Magellan [24] to generate a set of similarity features. However, ZeroER requires zero labeled examples where it implements a Gaussian Mixture Model to learn the distributions that govern the feature vectors of matches and unmatches. Away from duplicates, CleanLab [39] detects noisy labels via exploiting the principles of confident learning to estimate the joint distribution of noisy and true labels. To tackle the heterogeneity of data errors, Min-K and Max Entropy [2] implement an ensemble of other non-learning methods to identify most of the existing erroneous samples in a dataset. Specifically, Min-K considers as errors those samples detected by at least k-methods. Alternatively, Max Entropy introduces an entropy-based sampling method to systematically select the order in which the non-learning methods should be executed.

ML-supported Detectors: The ML-supported methods, examined in REIN, differ in how the features are generated and how the required labeling budget is reduced. For example, the metadata-driven error detection method [53] implements a metadata profiler and a suite of non-learning error detection methods to extract the features. In this case, each non-learning method is represented by a binary feature, where the feature value is one, if the non-learning method recognized this cell to be dirty. To reduce the labeling budget, RAHA [33] adopts a semi-supervised algorithm which clusters the samples by similarity and acquires labels on a per-cluster basis, before propagating the acquired labels in each cluster. Similarly, ED2 [38] extracts a set of attribute-level, tuple-level, and dataset-level features which define the distribution governing the dataset. Moreover, ED2 exploits active learning to acquire labels for clean/erroneous samples that the classifier is uncertain about. Finally, Picket [31] employs self-supervision to train an error detection model without requiring user labels.

3.2 Data Repair Methods

In REIN, we examine 19 data repair methods which can be classified into two main categories according to their intervention type, namely (I) generic methods and (II) ML-oriented methods. The former category comprises the methods which directly modify the dirty dataset to generate a repaired version. Such modifications can be either removing the dirty cells or replacing them with a set of generated repairs. They are generic in the sense that they seek to improve the data quality, regardless of the downstream application, e.g., ML modeling, data visualization, or data enrichment. Alternatively, the second category comprises methods which jointly optimize the data quality and the performance of downstream ML models. In REIN, we also exploit the ground truth of the dirty data to show the performance upper-bound. Below, we introduce the various methods in each category.

Generic Repair Methods. To generate repair values, REIN examines several standard and ML-driven imputation methods. The standard imputation methods utilize simple statistical measures, such as mean, median, or mode to generate repairs for the numerical values. For categorical values, we simply leverage the mode, i.e., the most frequent value in the corresponding attribute, as the repair value. Advanced imputation methods are those which

build ML models to generate accurate repairs based on information in the entire dataset. For numerical values, REIN examines 5 ML-based imputation methods including K-nearest neighbors (KNN), Decision Tree (DT), Bayesian Ridge [42], missForest based on random forest (RF) [51], and DataWig based on deep neural networks [7]. For categorical values, we examine both of missForest and DataWig. In particular, missForest iteratively trains an RF model on a set of clean samples (i.e., complete with no outliers) in a first step, before predicting the missing values. Similarly, DataWig implements deep learning modules combined with neural architecture search and end-to-end optimization of the imputation pipeline.

For mixed-type datasets, missForest and DataWig have two modes of operation, namely *separate* mode and *mixed* mode. In the former mode, each method is executed separately for each data type, referred to as MISS-Sep. The latter mode involves executing each method holistically on all data types, referred to as MISS-Mix and DataWig-Mix, taking into account possible relations between different data types. Another generic method is HoloClean [46] which precisely infers the repair values via holistically employing multiple cleaning signals to build a probabilistic graph model. To repair pattern violations and inconsistencies, OpenRefine [19] utilizes Google Refine Expression Language (GREL) as its native language to transform existing data or to create repair values. The last method in this category is BARAN [32] which is a holistic configuration-free method for repairing all error types. To this end, BARAN trains incrementally updatable models which leverage the value, the vicinity, and the domain contexts of data errors to propose correction candidates. To further increase the training data, BARAN exploits external sources, such as Wikipedia page revision history.

ML-oriented Repair Methods: The second category comprises the methods designed to jointly optimize the cleaning and modeling tasks. In other words, these methods focus on selecting the optimal repair candidates with the objective of improving the performance of specific predictive models. Accordingly, these methods assume the availability of repair candidates from other generic methods. For instance, BoostClean [25] treats the error correction task as a statistical boosting problem where a set of weak learners are composed into a strong learner. To generate the weak learners, BoostClean iteratively selects a pair of detection and repair methods, before applying them to the training set to derive a new model. ActiveClean [26] is another ML-oriented method, principally employed for models with convex loss functions. It formulates the data cleaning task as a stochastic gradient descent problem. Initially, it trains a model on a dirty training set, where such a model is to be iteratively updated until reaching global minima. In each iteration, ActiveClean samples a set of records and then asks an oracle to clean them to shift the model along the steepest gradient. A similar work is CPClean [22] which incrementally cleans a training set until it is certain that no more repairs can possibly change the model predictions.

4 DATA MODELING

In this section, we present a representative set of common ML models utilized for assessing the performance of error detection and repair methods. Table 2 summarizes the algorithms and whether they are used for classification (C), regression (R), or unsupervised clustering (UC) tasks. As listed in the table, REIN examines 12 classifiers, 11 regression models, six clustering algorithms, and two AutoML algorithms. Such vital algorithms are broadly applicable in various real-world application domains, e.g., cybersecurity systems, smart cities, healthcare, e-commerce, agriculture, and many more [49]. The rationale behind involving two AutoML algorithms is to evaluate the performance of fully automated ML pipelines, consisting of data cleaning and model building. We are interested in checking whether such algorithms are able to find the best possible combination of model architectures and hyperparameters based on dirty or automatically-repaired datasets. For most of these models, REIN exploits the Python implementation of Scikit-learn [42] library for training and testing. For hyperparameter tuning, REIN leverages a Bayesian-based informed search method, referred to as Optuna [3]. However, we did not use Optuna with the AutoML algorithms, since they can automatically select the best hyperparameters. Moreover, we did not use the internal processing pipelines of these algorithms, since we mainly focus on the examined cleaners (listed in Table 1).

Table 2: Examined ML models

Algorithm	C	R	Algorithm	C	R	UC
Logistic Regression (Logit)	✓		Linear Regression		✓	
Decision Tree (DT)	✓	✓	Bayes Ridge Regressor (BRidge)		✓	
Random Forest (RF)	✓	✓	RANSAC		✓	
Linear SVC	✓	✓	Gaussian Mixture (GMM)			✓
SGD Classifier	✓		K-Means			✓
KNN	✓	✓	Affinity Propagation (AP)			✓
AdaBoost (AdaB)	✓	✓	Hierarchical Clustering (HC)			✓
Gaussian Naïve Bayes (GNB)	✓		OPTICS			✓
Multinomial NB	✓		BIRCH			✓
XgBoost (XGB) [9]	✓	✓	Auto-Sklearn [17]	✓	✓	
Ridge	✓	✓	TPOT [27]	✓	✓	
Multi-Layer Perception (MLP)	✓	✓				

In REIN, we evaluate the various error detection and repair methods in five scenarios. Table 3 summarizes the different scenarios defined in terms of the data version used for training and testing. In addition to the dirty and repaired versions of the data, we utilize the ground truth version to estimate the performance upper-bound. For instance, S1 involves training and testing the ML models on either the dirty or the repaired versions of the data. Conversely, S4 represents the optimal setting in which the ground truth version of the data is employed for training and testing the models. To capture the performance if optimal data cleaning can be achieved in only one phase, REIN also considers S3 and S4 in which the ground truth (which simulates optimal data cleaning) is used for training and testing, respectively. Finally, S5 is mainly used with ML-oriented repair methods, which generate ML models as their output.

Table 3: Evaluation scenarios

	Train			Test		
Scenario	Dirty	Repaired	Ground Truth	Dirty	Repaired	Ground Truth
S1	✓	✓		✓	✓	
S2	✓	✓				✓
S3			✓	✓	✓	
S4			✓			✓
S5		✓		✓		

In general, the obtained results in each scenario may vary owing to ML randomness. Therefore, it is crucial to scrutinize the results obtained in each scenario before drawing conclusions. In this regard, REIN leverages A/B hypothesis testing to improve our confidence in the interpretation of the obtained results. Generally, an A/B hypothesis test can be exploited to quantify how likely it is to observe two data samples given the assumption that the samples have the same distribution [14]. In REIN, the A/B hypothesis tests can statistically predict whether an ML model behaves similarly in different scenarios. An initial step in the test

procedure is to clearly define the null hypothesis H_0 and the alternative hypothesis H_a. In REIN, the null hypothesis H_0 states that an ML model has circa the same performance in two different scenarios, e.g., S1 and S4, regardless of the data version. Conversely, the alternative hypothesis H_a states that the ML model behaves differently in S1 and S4. The statistical significance is estimated in terms of the *p-value*, i.e., the probability that an observed difference between S1 and S4 could have occurred by random chance. To estimate the p-value, we utilize the non-parametric Wilcoxon signed-rank test [14]. The main advantage of such a test lies in making no assumptions about the sampling distributions, e.g., being Gaussian. Specifically, we opted for the *two-tailed* version of the test, since it is not a priori known whether the discrepancy between the results of S1 and S4 will be in favor of S1 or S4. After computing the p-value, we can compare it with the significance level α to estimate whether to reject the null hypothesis H_0. In particular, we can reject the null hypothesis H_0 if $p\text{-}value < \alpha$. Otherwise, we conclude that the obtained results in the compared scenarios support the alternative hypothesis H_a.

5 BENCHMARK DATA

In this section, we elaborate on the real-world datasets and how to inject errors into them. To systematically select appropriate datasets for running the benchmark, it is necessary to define a set of requirements in light of the objectives of REIN. Such objectives revolve around estimating the performance of each detector/repair method separately without considering the subsequent stages of the ML pipeline and examining the impact of these methods on the performance of the downstream predictive models in different scenarios. Accordingly, the datasets, involved in REIN, have to fulfill the following conditions: (1) the existence of a complete and clean ground truth version; (2) the existence of associated predictive tasks, e.g., classification, regression, or clustering; (3) the existence of different data types, e.g., categorical, numerical, and/or text; and (4) the existence of different realistic error profiles. In fact, we collected two datasets, i.e., *Beers* and *Citation*, that satisfy these conditions. However, it is not straightforward to find other datasets satisfying our requirements.

To overcome such a challenge, we opted for injecting different types of errors into a set of real-world datasets. Consequently, we can predominantly control the experiments through obtaining several versions of each dataset along with the ground truth. In addition to the aforementioned requirements, we are also eager to select datasets covering multiple application domains, e.g., business, medical, and industrial, where the data originated in different domains usually have different characteristics. Moreover, we selected datasets of different sizes, ranging from a couple of hundred samples to a couple of hundred thousands, to precisely test the efficiency of the various data cleaning methods. Table 4 summarizes the examined datasets and the characteristics of their ground truth.

To inject errors into the real-world datasets, REIN leverages the BART tool [5] which provides a systematic control over the amount of errors and how hard these errors are to be repaired. To inject errors using BART, we use a set of denial constraints to generate different attribute and class errors, such as rule violation, outliers, nulls, duplicates, and mislabels. Furthermore, we also employ a Python library, referred to as *error generator*, to generate highly realistic errors [1]. Examples of such error are typos based on keyboards, implicit missing values, Gaussian noise, and value swapping. To automatically generate FD rules, REIN leans on the FDX profiler [56] which formulates the task of learning functional dependencies as a sparse regression problem. After generating the FD rules, we manually convert them into denial constraints to be used with BART and the rule-based error detection and repair methods, e.g., HoloClean and NADEEF.

6 PERFORMANCE EVALUATION

In this section, we assess the effectiveness and efficiency of various error detection and repair methods. We first describe the setup of our evaluations, before discussing the results and the lessons learned throughout this study.

6.1 Experimental Setup

In REIN, we utilize several metrics to assess the quality of results at different stages of a typical ML pipeline. In the error detection phase, we leverage precision, recall, F1 score, IoU, and runtime to evaluate the effectiveness and efficiency. In this context, the precision P denotes the fraction of relevant instances, e.g., actual erroneous cells, among the detected instances, i.e. $P = \frac{t_p}{t_p+f_p}$ where t_p and f_p are true positives and false positives, receptively. The recall R is defined as the fraction of erroneous instances that are detected, i.e. $R = \frac{t_p}{t_p+f_n}$ where f_n denotes false negatives. The F1 score denotes the harmonic mean of precision and recall where $F1 = 2.\frac{P.R}{P+R}$. Such metrics define the quality of detection relative to the ground truth. Nevertheless, it is also significant to identify the similarities between the detected erroneous cells obtained by different detection methods. Hence, we adopt the *Intersection over Union* (IoU) metric. Assume that N_a, N_b are the detected erroneous cells by detectors a and b. Accordingly, the IoU metric between detectors a and b is computed as $\frac{|N_a \cap N_b|}{|N_a|+|N_b|-|N_a \cap N_b|}$. For these computations, we consider only the true positives, since the false positives may lead to misleading results. Finally, the runtime is the time elapsed while traversing an entire dataset to identify the erroneous cells.

In the error repair phase, we differentiate between the numerical and the categorical attributes. For the former type, we employ the root mean square error (RMSE) as a distance measure between the repaired values and their ground truth. In fact, some error types, e.g., typos and outliers, turn the numerical instances into categorical ones. To properly compute the RMSE metric, we filtered out the transformed instances which have not been detected and repaired. For the latter data type, we employ precision, recall, and F1 measures. In this context, the precision is defined as the fraction of correctly repaired data errors relative to the number of repaired data errors. The recall is defined as the fraction of correctly repaired data errors relative to the number of data errors. We also report the runtime to quantify the time elapsed while generating the repairs. In the ML modeling phase, we utilize precision, recall, and F1 measures for the classification models. For clustering methods which require the number of clusters k as an input, we utilize the Silhouette index to find a well estimate for the value of k. For the A/B statistical test, we set the Type I error rate α to 0.05. All experiments have been repeated ten times with different random seeds that control the train-test split, and the means of the ten runs are reported. We run all the experiments on an Ubuntu 16.04 LTS machine with 32 2.60 GHz cores and 264 GB memory. Due to space constraints, the results of many experiments have been omitted.

6.2 Error Detection

In this set of experiments, we assess the performance of several error detectors applied to different datasets. For each dataset, the number of examined detectors depends on the types of injected

Table 4: Dataset characteristics

Dataset	# Rows	# Columns	# Numerical	# Categorical	Error Rate	Errors	Domain	ML Tasks
Beers [21]	2410	11	6	5	0.16	MVs, Rule Violations, Typos	Business	C
Citation [13]	5005	2	1	1	0.2	Duplicates, Mislabels	Research	C
Adult [23]	45223	15	7	8	0.58	Rule Violations, Outliers	Social	C
Breast Cancer [15]	700	12	12	0	0.08	MVs, Typos, Outliers	Healthcare	C
Smart Factory [8]	23645	19	19	0	0.153	MVs, Outliers	Manufacturing	C
Nasa [52]	1504	6	6	0	0.08	MVs, Outliers	Manufacturing	R
Bikes [16]	17378	16	16	0	0.1	Rule Violations, outliers	Business	R
Soil Moisture [48]	679	129	129	0	0.01	MVs, Outliers	Agriculture	R
3D Printer [40]	50	12	10	2	0.05	Duplicates, MVs, Implicit MVs	Manufacturing	R
Mercedes [11]	4210	378	370	8	0.05	Outliers, MVs, Implicit MVs	Manufacturing	R
Water [6]	527	38	38	0	0.14	Outliers, Implicit MVs	Manufacturing	UC
HAR [4]	70000	4	3	1	0.13	Outliers, MVs	Wearables	UC
Power [18]	1456	24	24	0	0.037	Typos, MVs, Implicit MVs	Energy	UC
Soccer [35]	180228	44	40	4	0.27	Rule violations, outliers, MVs, Implicit MVS	Business	–

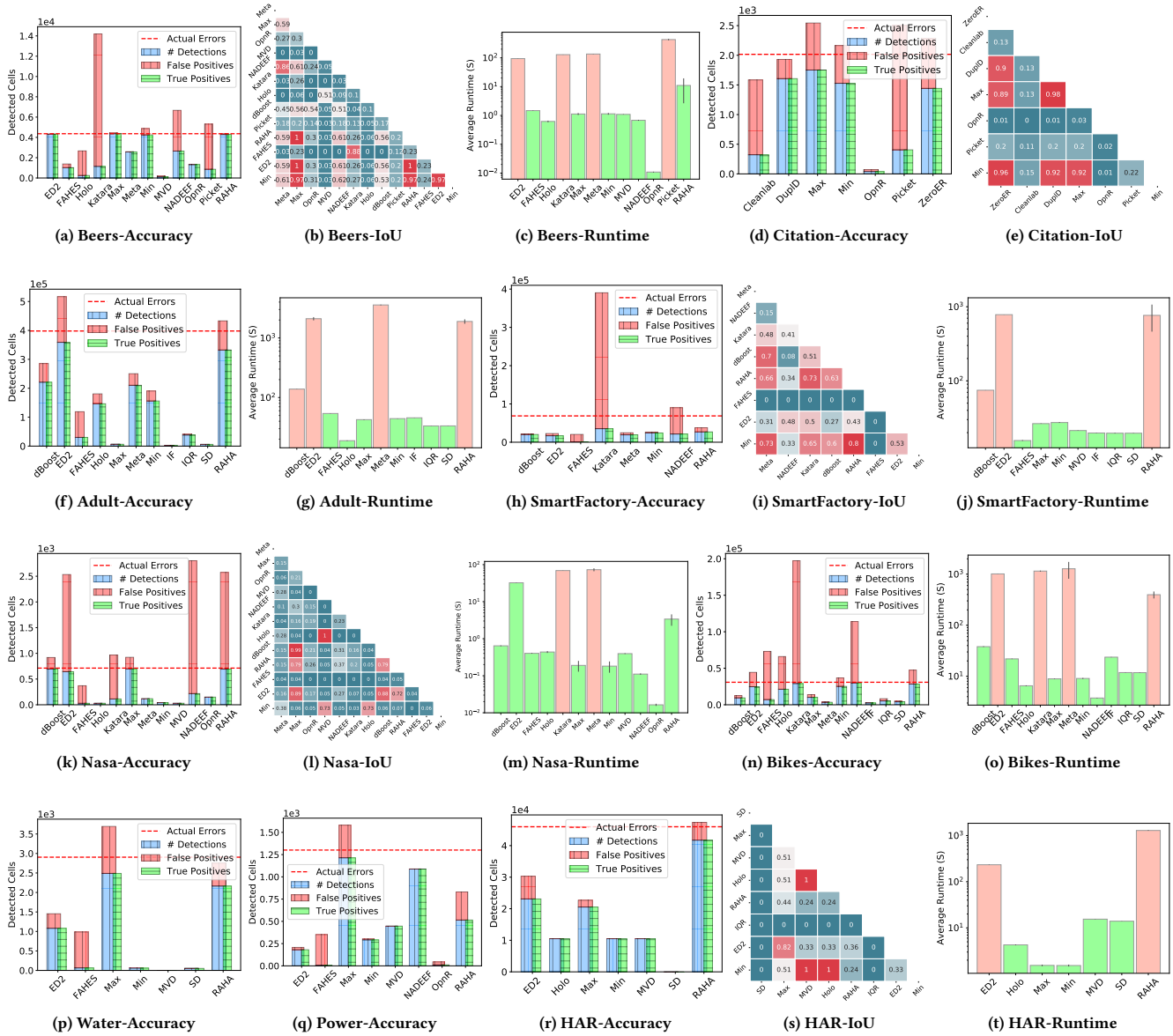
errors. Moreover, the detectors which fail to detect any cells are deliberately excluded from the figures. Figure 2a depicts the number of detected erroneous cells (blue bars) and the number of true positives (green bar) in the *Beers* dataset using 14 error detection methods. The number of false positives is indicated by turning the color of the blue bars into red. The red dashed line represents the actual number of erroneous cells in the dataset. As depicted in the figure, most ML-based and ensemble methods, including ED2, RAHA, Min-k (Min), and Max-entropy (Max), outperform the other methods where their F1 score is between 0.92 and 0.99. As a result of converting the numerical attributes to categorical ones, several detectors, e.g., NADEEF and KATARA, erroneously flagged all clean numerical values in these converted attributes as noisy cells. The low precision of such methods (ranging from 0.08 to 0.16) typically has negative consequences on the repair phase (cf. Section 6.3).

Figure 2b demonstrates the IoU metric of detectors applied to the *Beers* dataset. Obviously, the ML-based and ensemble methods have high similarity (at least IoU of 98%). Furthermore, the figure shows a relatively high correlation (IoU of 87%) between the detections of NADEEF (F1 of 0.74) and Metadata-driven (Meta, F1 of 0.48) methods. Accordingly, we can deduce that most detections of the Metadata-driven method, i.e., 2417 out of 2570 detected cells, are rule and pattern violations. Similarly, KATARA (F1 of 0.12) and FAHES (F1 of 0.35) have high similarity (IoU of 88%) since both of them employ a syntactic pattern detection method. Figure 2c depicts the average runtime (on the logarithmic scale) of the detectors, where the red bars indicate that the runtime exceeds one minute. As the figure depicts, the ML-based methods require long execution time due to searching for the optimal configurations, featurization, and training the classifiers. For instance, Max Entropy requires much less time (at least by 98%) than ED2 while detecting the same erroneous cells (cf. Figure 2b).

Figure 2d depicts the number of detected cells in the *Citation* dataset using seven detectors. Such a dataset contains duplicates and mislabeled samples. The figure shows that the key collision method (DuplD) outperforms all other methods, where it achieved an average F1 score of 0.86. Similarly, the ensemble methods (i.e., Min and Max) achieved better performance (F1 score between 0.74 and 0.78) than Picket (ML-based detection method, average F1 of 0.18) due to the low recall of Picket which relies on self-supervision to train its classifier. Moreover, CleanLab achieved a low F1 score of 0.19 where it captured only the mislabeled cells in the dataset while ignored the duplicates. Figure 2e depicts a strong IoU relationship among the detections of key collision, ZeroER, Min-K, and Max Entropy. However, ZeroER requires much more time (by circa two orders of magnitude) to generate its detections.

For the *Adult* dataset, Figure 2f depicts the number of detected cells using 11 detectors. Such a dataset suffers from rule violations and outliers, with a large error rate. In this case, both of RAHA and ED2 outperform all other methods (average F1 score of 0.8 and 0.78, respectively). According to their IoU values, the detections obtained by HoloClean, NADEEF, and Min-k exhibit high correlation where these methods captured most of the rule violations only. Conversely, dBoost captured most of the outliers while failed to identify the rule violations. Despite being effective while detecting erroneous cells in this dataset, ED2 and RAHA are less efficient where they required, on average, 35 minutes to find the erroneous cells compared to 2.3 and 0.73 minutes for dBoost and Min-k, respectively. The *Smart Factory* dataset represents an example of relatively large datasets suffering from explicit missing values and outliers with a moderate error rate. Figure 2h depicts the number of detected cells in the *Smart Factory* dataset using eight detectors. In this case, Min-k outperforms (average F1 score of 0.75) other detectors while requiring much less time than other detectors (cf. Figure 2j). RAHA and Meta have a relatively high correlation with Min-k, as depicted in Figure 2i. Furthermore, Figure 2h shows that KATARA generated many false positives, which occurs since it failed to correctly interpret the data semantics.

For the datasets with regression tasks, Figures 2k-2o show the detection accuracy and runtime of various detectors. For instance, Figure 2k depicts the number of detected cells in the *Nasa* dataset using 12 detectors. Such a dataset represent an example of small datasets suffering from explicit missing values and outliers with a small error rate. As the figure depicts, Max Entropy and dBoost outperform (average F1 score of 0.85) all other methods. Both detectors nearly generated the same detections where their IoU metric is 0.99, as illustrated in Figure 2l. Despite detecting mostly all erroneous cells, the ML-based methods have F1 score between 0.27 and 0.43 due to the large number of false positives. As the dataset is small, most detectors generated their detections in less than a minute, as depicted in Figure 2m. For the *Bikes* dataset, it has rule violations and outliers with a small error rate. Figure 2n depicts the number of detected cells in the *Bikes* dataset using 11 detectors. RAHA and Min-k outperform other detectors with average F1 scores of 0.72 and 0.75, respectively. The figure shows that KATARA and NADEEF (average F1 score of 0.25 and 0.4, respectively) have poor performance due to generating many false positives. Similar to the *Nasa* dataset, dBoost and Max Entropy have a high correlation. Figure 2o shows that Min-k is more



(a) Beers-Accuracy (b) Beers-IoU (c) Beers-Runtime (d) Citation-Accuracy (e) Citation-IoU

(f) Adult-Accuracy (g) Adult-Runtime (h) SmartFactory-Accuracy (i) SmartFactory-IoU (j) SmartFactory-Runtime

(k) Nasa-Accuracy (l) Nasa-IoU (m) Nasa-Runtime (n) Bikes-Accuracy (o) Bikes-Runtime

(p) Water-Accuracy (q) Power-Accuracy (r) HAR-Accuracy (s) HAR-IoU (t) HAR-Runtime

Figure 2: Detection results (In the accuracy plots, the blue bars are subdivided into red and green regions to show the false positives and true positives, respectively)

efficient than RAHA, where it required, on average, 9 seconds to generate the detections compared to 6.6 minutes for RAHA.

Figures 2p-2t depict the performance of various detectors using the datasets associated with clustering tasks. For the *Water* dataset, it suffers from implicit missing values and outliers with a small error rate. Figure 2p shows that Max Entropy and RAHA achieved the highest accuracy with average F1 scores of 0.74 and 0.76, respectively. The detections obtained by both detectors are highly correlated. However, Max Entropy required much less time (average runtime of 0.09 seconds) to generate its detections compared to RAHA (average runtime of 15.8 seconds with a standard deviation of 10.4) and ED2 (average runtime of 17.9 minutes). RAHA has typically high variance because it consumes a relatively long time in the first iteration to create the cleaning strategies utilized to generate the training features. For the *Power* dataset, NADEEF and Max Entropy outperform other detectors with average F1 scores of 0.9 and 0.84, respectively, as shown in Figure 2q. Clearly, both of NADEEF and MVD have high precision. However, each detector captured only the relevant errors. In other words, NADEEF detected 1088 pattern violations (corresponding to the typos and implicit missing values), while MVD found only the explicit missing values. For the efficiency, Max Entropy and NADEEF consumed circa the same time (average runtime of 0.05 seconds), while ED2 required, on average, 680 seconds to generate the detections. For the *HAR* dataset, Figure 2r shows that RAHA achieved the highest accuracy, with an average F1 score of 0.89, at the expense of consuming 20.5 minutes (standard deviation of 20 minutes) to generate its detections (cf. Figure 2t). Figure 2s demonstrates that MVD, HoloClean, and Min-K detected the same erroneous cells with missing values.

6.2.1 Detection Robustness. In this section, we examine the robustness of various error detectors in terms of their accuracy. To this end, we implemented two sets of experiments, including: (1) varying the *error rate* of a dataset; and (2) varying the *outlier*

degree, defined as the number of standard deviations away from the mean. In the former set of experiments, we injected outliers and missing values where the outlier degree is set to 4. In the outlier degree experiment, we injected outliers with an error rate of 30%. Figure 3a compares the robustness of seven detectors while cleaning the *Adult* dataset at different error rates. Clearly, the F1 score of all detectors increases linearly at low error rates (i.e., up to 0.02). In this range, several detectors (e.g., ED2, Max Entropy, and Min-k) have a large slope, which implies a high detection accuracy. When the error rate is further increased, the accuracy of most detectors, except RAHA, is gradually reduced. Figure 3b shows a similar experiment on the *Power* dataset. As the figure depicts, ED2 achieved a higher accuracy, at low error rates, than all other models. For RAHA, its performance has been improved, when the error rate is increased. Figure 3c compares the performance of ten detectors when increasing the outlier degree injected into the *Smart Factory* dataset. The figure shows that all detectors behave approximately the same when the outlier degree is relatively small (i.e., below two). However, the performance of RAHA, ED2, Min-k, dBoost, and Meta is broadly improved when the value of the outlier degree goes beyond two.

6.2.2 Scalability Analysis. In this section, we evaluate the efficiency of several error detectors when dealing with large datasets. To this end, we ran several experiments to detect errors in different data fractions. Figures 3d and 3e compares the accuracy and efficiency of ten detectors for different factions of the *Soccer* dataset. For this dataset, Figure 3d shows that ED2, NADEEF, and RAHA achieved the highest F1 score (i.e., 0.83, 0.93, and 0.98, respectively). Furthermore, the figure illustrates that some detectors work only with small data fractions. For instance, RAHA, ED2 stopped working at a data fraction of 50%, while HoloClean is terminated with 90% of the data. Figure 3e shows the comparison in terms of the average runtime (in logarithmic scale). Obviously, RAHA, ED2 and KATARA required much more time (average runtime of 3.5, 10.1, 13.8 hours, respectively) than other detectors. In contrast to ED2 and RAHA, KATARA managed to detect errors for all data fractions.

6.3 Data Repair

In this section, we introduce the results of the repair methods while being used to generate repair candidates based on the detections obtained from various error detectors. We divide the experiments according to the data type in each dataset. Moreover, we introduce the results of the ML-oriented repair methods, whose outputs are ML models rather than repaired datasets.

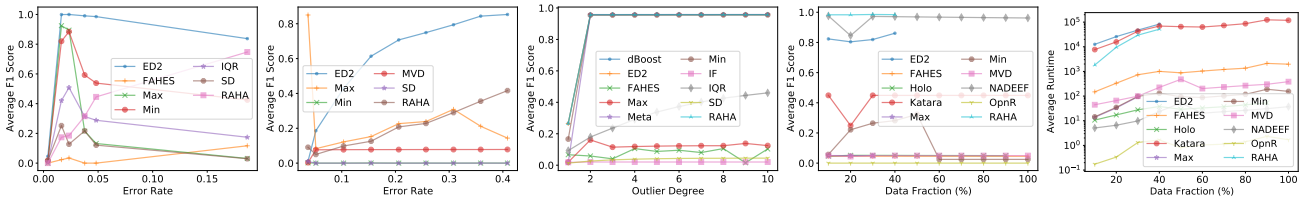
6.3.1 Categorical Attributes. Figure 4 shows the repair results in terms of the repair accuracy and runtime for two datasets which include categorical attributes. For instance, Figure 4a delineates the repair accuracy, in terms of the precision and recall, when cleaning the *Beers* dataset. The figure shows that the detections obtained by several detectors, including RAHA, ED2, Min-k, Max Entropy, HoloClean, and NADEEF, can result in a high repair accuracy (average F1 score of 0.99) if being repaired by an optimal repair method (simulated by GT). The high performance of HoloClean-GT is achieved, despite the low recall of HoloClean as shown in Figure 2a, since HoloClean detected 248 out of 254 actual erroneous categorical cells. For this dataset, BARAN achieved the highest accuracy (average repair F1 score of 0.98) when generating repair candidates for the detections obtained by RAHA, ED2, and Max Entropy. Due to the large number of false negatives (127 cells out of 254 erroneous categorical cells) obtained by KATARA (cf. Figure 2a), the maximum repair F1 score, when repaired using the ground truth, is limited to only 0.66. Figure 4b compares the runtime of eight repair methods. The blue band enveloping the boxes represents the standard deviation of the runtime at a given point. Clearly, BARAN consumed much more time (an average runtime of 4.4 minutes with a standard deviation of 1.5 minutes) than all other detectors.

Figure 4c shows the repair accuracy of various detector/repair combinations adopted to clean the *Breast Cancer* dataset. As the figure illustrates, the detections obtained by Max Entropy led to a moderate accuracy, when MissForest (F1 score of 0.63) and BARAN (F1 score of 0.6) are utilized. Furthermore, the figure shows that KATARA achieved a repair F1 score of one when the detections are repaired using the ground truth. In fact, KATARA generated many false positives (6,843 cells) and few false negatives (86 cells, all numerical values). Accordingly, we can deduce that in the presence of highly-effective repair methods, the detection false negatives are more harmful to the repair accuracy than the detection false positives. Figure 4d depicts that HoloClean and BARAN are the most time-consuming repair methods (average runtime of 45.7 and 53.8 and seconds, respectively).

6.3.2 Numerical Attributes. Figure 5 depicts the repair results of the numerical attributes in terms of the RMSE values and the runtime. For instance, Figure 5a compares the performance of eight repair methods while cleaning the *Smart Factory* dataset. Each repair method comprises a group of bars representing the different detection methods. The red dashed line denotes the RMSE value of the dirty version of the dataset. The Figure depicts that the detections of RAHA and dBoost achieved the highest performance (average RMSE of 0.93 and 0.82 for RAHA and dBoost, respectively) for different repair methods. Furthermore, the figure depicts that GT may generate repaired versions with RMSE comparable to the dirty version (cf. the bars of FAHES, Meta, and NADEEF in the GT group). Such a repair performance typically occurs due to the low accuracy of these detections. Accordingly, we can conclude that without an accurate error detection process, the highly-effective repair methods can achieve poor results. Figure 5c shows that the detections of ED2 and RAHA, in the *Breast Cancer* dataset, achieved the highest repair accuracy over mostly all repaired methods.

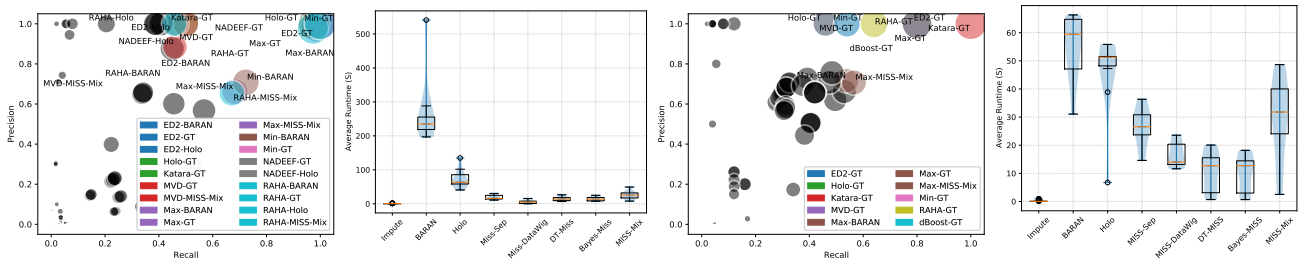
For the *Bikes* dataset, Figure 5d shows that most cleaning strategies generate repaired versions relatively better than the dirty data. However, the repaired versions, resulted from the detections of FAHES, HoloClean, and KATARA, have higher RMSE values than the dirty version (cf. the bars above the dashed line for standard and ML-based imputation methods). For this dataset, BARAN required much more time (an average runtime of 58.4 ± 40.2 minutes) than all other methods. Figure 5e compares the accuracy of ten repair methods while cleaning the *Water* dataset. The figure shows that all repaired versions have either similar or better performance than the dirty version. Obviously, RAHA and Max Entropy achieved the highest accuracy over all repair methods (an average RMSE of 0.7 and 0.65, respectively). In terms of runtime, Figure 5f shows that HoloClean is the most time-consuming method with an average runtime of 5.2 ± 4 minutes.

6.3.3 ML-Oriented Repair Methods. In this section, we present the results of the ML-oriented methods, including ActiveClean, CPClean, and BoostClean. Figure 6 compares the performance of these methods in terms of the modeling accuracy. In particular, Figure 6a shows the F1 score of the generated models in scenarios S1, S4, and S5 for the *Adult* dataset. The figure shows that the



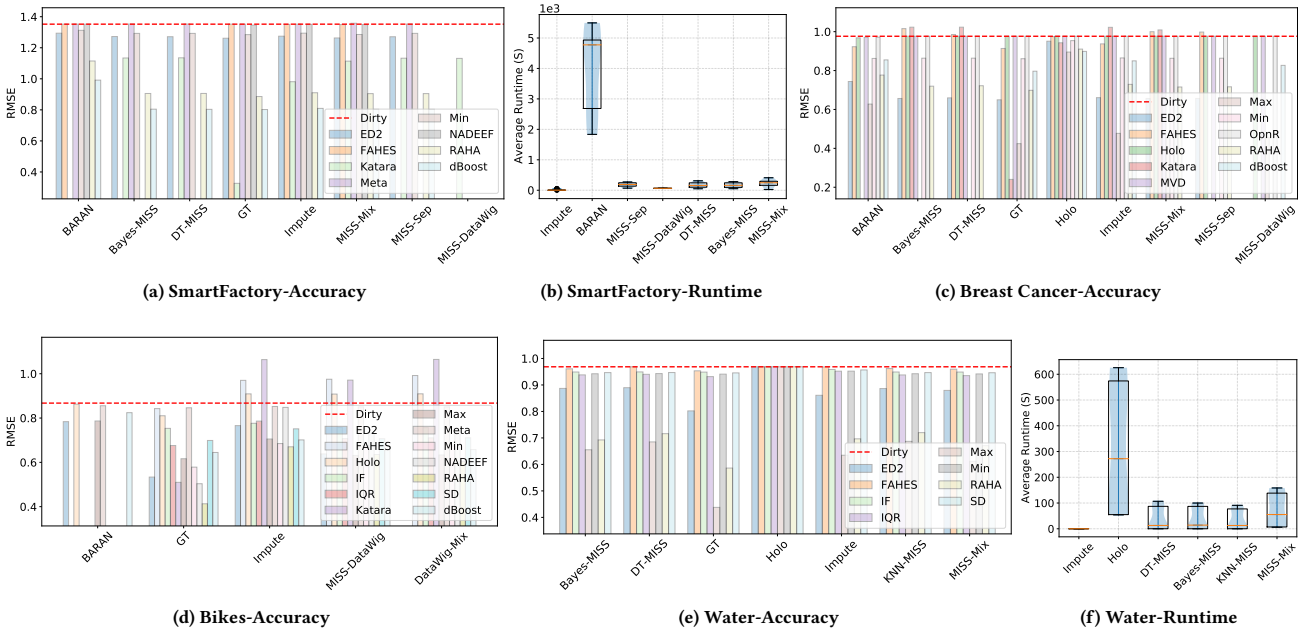
(a) Adult (b) Power (c) SmartFactory-Outliers (d) Soccer-Accuracy (e) Soccer-Runtime

Figure 3: Robustness and scalability results of the error detectors



(a) Beers-Accuracy (b) Beers-Runtime (c) BreastCancer-Accuracy (d) BreastCancer-Runtime

Figure 4: Repair results considering only the categorical attributes (In the accuracy figures, each bubble represents a different cleaning strategy and the size of the bubbles denotes the F1 score. To highlight the most effective cleaning strategies, we colored only the bubbles whose F1 score is above 0.6)



(a) SmartFactory-Accuracy (b) SmartFactory-Runtime (c) Breast Cancer-Accuracy

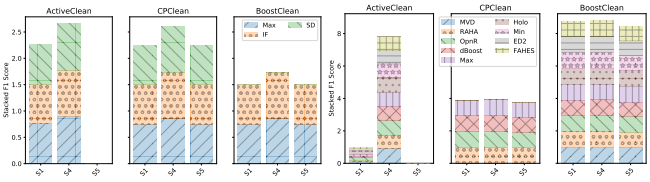
(d) Bikes-Accuracy (e) Water-Accuracy (f) Water-Runtime

Figure 5: Repair results considering only the numerical attributes

datasets, repaired using the three cleaning methods, slightly lag behind the ground truth versions (on average by 15%, 0.13%, and 0.13%, for ActiveClean, CPClean, and BoostClean, respectively). Furthermore, the results of CPClean and BoostClean in S1 are approximately the same as in S5. The reason behind such a result lies in the relatively comparable accuracy of the dirty and the repaired versions, as shown in Figure 5. For the *Breast Cancer* dataset, Figure 6b depicts that the models generated by ActiveClean in S1 broadly suffer from low accuracy, where the average F1 score in S4 is higher than in S1 by circa 88%. This result mostly occurred due to the small size of the dataset and the relatively low detection accuracy of all detectors (i.e., the highest F1 score of 0.75 by Max Entropy). For CPClean and BoostClean, the results are close to each other in the three scenarios.

6.4 Modeling Accuracy

In this section, we present the results of modeling the various datasets in different scenarios. Figure 7 demonstrates the accuracy of different classification, regression, and clustering models trained on different data versions. For the *Beers* dataset, Figure 7a shows the average F1 score of six classifiers in scenarios S1 and S4. As the figure depicts, the performance of all classifiers changes



(a) Adult (b) BreastCancer

Figure 6: Accuracy of ML-oriented repair methods

according to the quality of the repaired data. For example, the MLP classifier achieved an average F1 score of 0.732 in S4, while the accuracy in S1 ranges from 0.368 to 0.727. Figure 7b clarifies these results via comparing the performance of MLP models trained on different versions, i.e., dirty (D0), ground truth, and repaired, of the *Beers* dataset in S1 (in blue) and S4 (in green). Obviously, the blue and green regions mostly overlap with each other. The only exception occurs with the combination X3, representing Max Entropy and standard imputation. Such a low accuracy, repeated with several classifiers, is usually caused by the low-quality repairs generated by different standard imputation methods. In this figure, the results of the A/B statistical test are delineated in the form of blue filled/empty square markers. In this context, a filled marker denotes that the null hypothesis H_0 can be rejected (i.e., the two MLP models in S1 and S4 are different), whereas an empty marker means failing to reject H_0. Thus, we can confirm that the performance difference of the models in S1 and S4 will remain, if we run the experiments for more than ten times. Figure 7c compares the results of ten classifiers trained on different versions of the *adult* dataset. In this figure, the distribution of the results in S1 enables us to identify the ML models robust to data quality problems. For instance, the results of DT in S1 range from 0.17 to 0.99, whereas the results of Ridge range from 0.74 to 0.78. Figure 7d shows the performance of SVC when trained on different versions of the *Adult* dataset. For most data versions, the accuracy of SVC is comparable in both scenarios. Despite achieving high detection accuracy, the detections of ED2 led to quality problems in most of its repaired versions, e.g., E1, E3, E10, E15. This behavior occurs due to the large number of false positives (118,741 cells) generated by ED2 (cf. Figure 2f). Similarly, Figures 7e and 7f depict the modeling accuracy for different versions of the *Breast Cancer* dataset. For this dataset, DT performed well with a relatively tight range of F1 scores from 0.65 to 0.94, compared to GNB whose F1 scores range from 0.15 to 0.85. Figure 7f depicts that the performance of XGBoost is slightly better in S4 than in S1 for most repaired versions of the *Breast Cancer* dataset.

For the *Citation* dataset, which includes duplicates and mislabels, Figure 7g demonstrates the F1 score of several classification models in the scenarios S1 and S4. As it can be seen in the top right corner of the figure, most classifiers yield similar performance as the ground truth when applying the "Delete" strategy. Other cleaning strategies which rely on ML-based imputation, e.g., M6, M7, M9, X7, X6, and X9, cause the predictive performance to be substantially deteriorated (cf. Figure 7h). Unlike other classifiers, XGBoost exhibits poor performance over the dirty and the most repaired data versions (F1 score ra nges from 0.05 to 0.8 and has a high density under the value of 0.26, as depicted in Figure 7g). To further understand the impact of mislabels, we carried out experiments on the *Adult* and *Breast Cancer* datasets after adding noise to the labels (i.e., flipping some binary labels). The results of such experiments show that several ML models, e.g., MLP, RF, DT, trained on dirty versions have slightly worse performance than the same models trained on the ground truth (for RF, an average F1 score of 0.9 for the dirty dataset and 0.93 for the ground truth).

Figures 7j-7o illustrate the performance of various regression models trained on different datasets. As depicted in Figures 7j, XGB achieved the highest accuracy in S4 (RMSE of 1.54). However, its performance broadly depends on the quality of the repairs (cf. the RMSE values in S1 which range from 1.78 to 35.9). Conversely, DT and RF have tighter distribution of RMSE values in S1. Figure 7k demonstrates that DT has approximately the same predictive performance over the most repaired data versions. The figures also some cleaning strategies, e.g., X2, X7, X8, N11, and K11, which achieve similar performance as the ground truth. For the *Soil Moisture* dataset, Figure 7l depicts that KNN outperforms other models with a relatively tight distribution of the RMSE values in S1. For this dataset, the detections of RAHA repaired using the ground truth led to a comparable RMSE as that obtained in S4, as depicted in Figure 7m. In Figures 7n and 7o, we demonstrate the performance of RANSAC and Bayesian Ridge in scenario S2 and S3 (cf. Table 3). Obviously, RANSAC and Bayesian Ridge perform in S2 much better than in S3. Since this result appeared in all other datasets, we can deduce that models trained on dirty or relatively low-quality repaired data may perform well whenever they are tested/served using high-quality data.

Aside from regression, the accuracy of several clustering methods also have been measured in terms of the silhouette index, as illustrated in Figures 7p-7t. The results showed that some clustering methods, e.g., Optics, GMM, and HC, yielded a comparable performance in S1 and in S4, or even better in S1 for several repaired versions, as depicted in Figures 7p and 7r. For instance, Figure 7q compares the performance of Birch when clustering different versions of the *Water* dataset. In general, Birch performed in S4 better than in S1. However, there exist several repaired methods which exhibit better clustering performance (on average by 16%, 18%, and 17% for R1, R7, and R9, respectively) than the ground truth. Figure 7s shows similar results for K-Means while clustering the *power* dataset. Finally, Figure 7t compares the performance of five clustering methods trained on the *HAR* dataset. The figure shows that all models have a relatively tight distribution in S1, which implies non-sensitivity to the quality of the repaired versions. Several repaired versions, generated using the detections of RAHA (e.g., R1, R2, and R6), led to similar performance as the ground truth.

6.5 Lessons Learned

Main Findings: In this section, we highlight the main findings and lessons learned throughout this study. Through extensive experiments, REIN proved that evaluating the error detection and repair methods in isolation from the downstream applications, e.g., predictive tasks, can be broadly misleading. For instance, Figures 2a, 2h, and 2n show that KATARA suffers from many false positives. Moreover, the quality of repairs generated for the detections of KATARA is sometimes worse than the dirty versions of the datasets (cf. Figure 5d). Nevertheless, Figures 7d, 7g, 7i, and 7k clearly depict that the ML models trained on the KATARA-based repaired data versions have a comparable predictive performance to the other models. Similar conclusions can also be drawn for other detectors, such as FAHES, NADEEF, and HoloClean. In fact, most error detection and repair methods are typically evaluated using their performance relative to the ground truth [20, 32, 33, 38, 46]. Accordingly, the finding above represents a

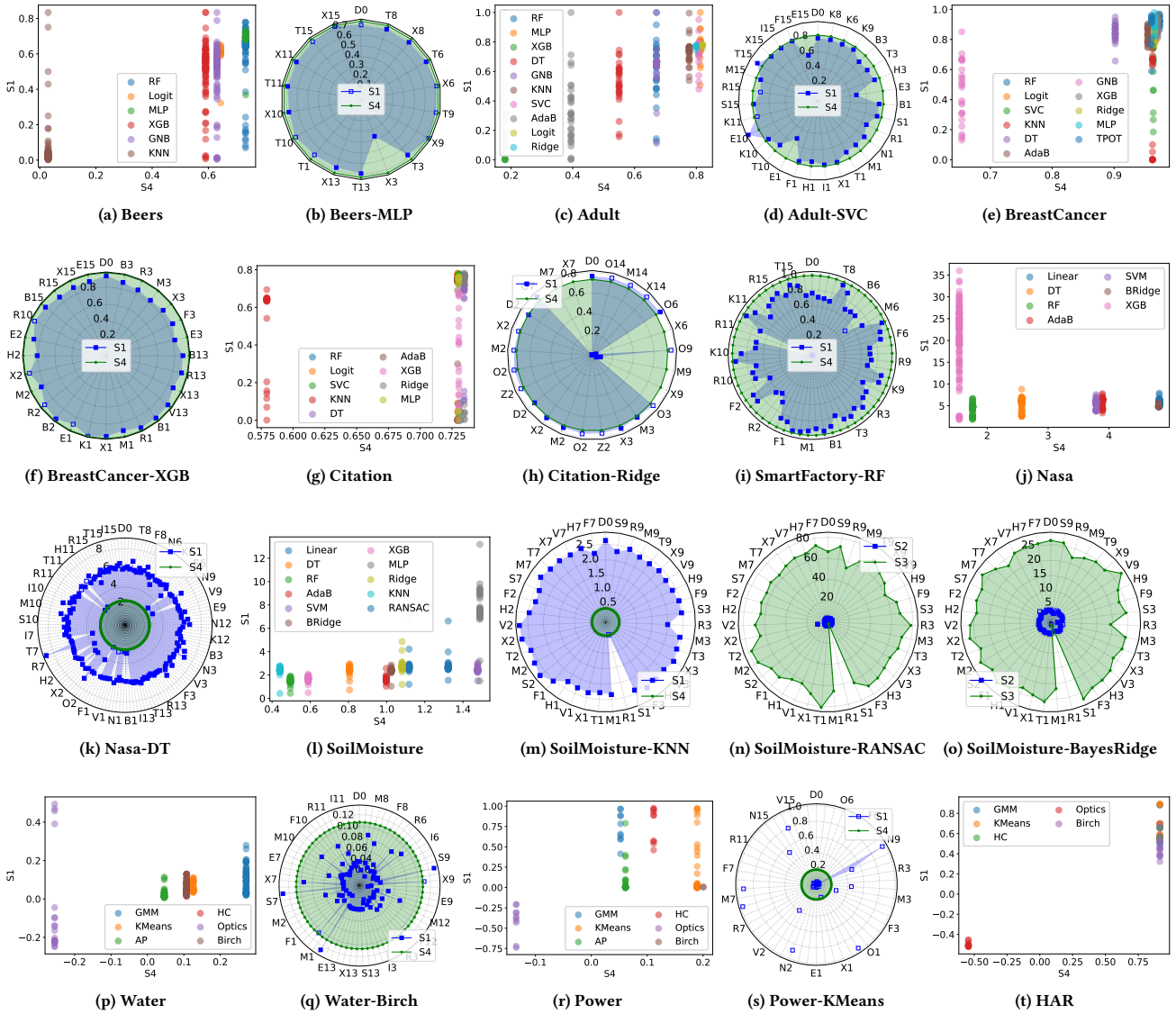


Figure 7: Accuracy of ML Models trained on different data versions in different scenarios (F1 score, RMSE, and Silhouette metric for datasets with associated classification, regression, and clustering tasks, respectively)

major result which guides researchers and developers on how they can effectively evaluate their data cleaning methods.

Another interesting finding is that classification models are more robust to attribute errors than regression models and clustering methods. Through comparing the performance of different models in Figure 7, it is clear that the differences between S1 (blue regions) and S4 (green regions) for almost all classifiers are relatively small. Conversely, regression models and clustering methods remarkably perform in S4 better than in S1. Accordingly, data cleaning is a necessary component in the pipelines of regression and clustering applications. Furthermore, classification applications may not need to implement a sophisticated data cleaning method. Simple cleaning methods can supply the classification models with the necessary quality level that is needed to train the models. At the same time, simple error detection and repair methods do not require excessive time, hence we can broadly accelerate the data preparation process. In the presence of class errors, some classifiers exhibited relatively poor performance. Hence, automated mislabels detection methods are necessary to produce accurate predictions. For the examined AutoML algorithms, i.e., TPOT and Auto-Sklearn, the results showed that they do not *always* produce the most accurate models. For example, in case of the *Breast Cancer* dataset, the models generated by TPOT with X13 and X15 have F1 scores of 0.75 and 0.6, respectively. Compared to TPOT with B15 and X2, which have F1 scores of circa 0.98 and 0.99, respectively. Thus, these algorithms may fail to generate accurate models in case of improper data cleaning.

Error Detectors: Regarding the error detection methods, it is obvious that ML-based and ensemble methods, in most cases, have a higher detection accuracy than the other non-learning methods, as illustrated in Figure 2. However, the results also showed that most detectors lack consistency over different datasets, i.e., their performance varies from one dataset to another. For instance, Figure 2a shows that ED2 detected all errors with high precision in the *Beers* dataset. Nevertheless, it suffered from false positives and false negatives in other datasets, such as *Adult*, *Nasa*, and *HAR* (cf. Figures 2f, 2k, and 2r). Similarly, NADEEF performed poorly (an average F1 score of 0.12) in case of the *Nasa* dataset,

whereas it achieved a reasonable performance (an average F1 score of 0.91) in case of the *Power* dataset. Other shortcomings of ML-based detectors are as follows: (1) They are not able to recognize the error type, i.e., they only provide a binary decision for each cell of whether it is erroneous. This behavior may make it complex to select a well-suited data repair tool. (2) They suffer from poor scalability (cf. results in Figures 3d-3e). (3) They require users intervention to label data. Accordingly, it is necessary to exert more efforts to advance the ML-based detectors for the sake of resolving the above shortcomings.

The results illustrated that the performance of rule-based error detectors broadly relies on the number and the quality of the user-provided rules/constraints. For instance, the F1 score of HoloClean, in case of the *Adult* dataset, is dropped from 0.51 to 0.12, when the number of provided rules is reduced from 17 to seven. Accordingly, it is crucial to integrate an automated rules/constraints generator with such detectors to improve their performance. In this context, we highlight that configuration-free methods are generally simple and easy to be employed, but they usually need long times to find the most suitable configurations, e.g., dBoost and RAHA (cf. Figures 2c, 2j, and 2t). It is worthwhile mentioning that the current implementation of RAHA, ED2, and Meta do not work in the presence of duplicates in the dirty datasets. This problem mainly occurs since the dirty and ground truth versions of the dataset become of different lengths. In this case, these detectors are not able to use the ground truth to simulate a human annotator, i.e., for labeling the dirty cells. Picket represents an exception to this fact since it relies on self-supervision. Therefore, it does not mandate user-provided labels. However, the results showed that Picket is only suitable for small datasets, where it does not scale well due to the complexity of self-supervision. For larger datasets, e.g., *Adult* and *Smart Factory*, Picket was terminated since it caused memory faults.

Repair Methods: For a better repair experience, it is found that the detection precision has a relatively higher impact on the repair quality than the detection recall (cf. Figures 2a and 2n). The reason behind such a superiority is to avoid false positives which may drive the adopted repair method to either introduce new erroneous cells or remove all the detected cells, causing the repaired dataset to be entirely out of sync with the ground truth. However, an effective repair method can even avoid the negative impacts of false positives in the detection phase. For instance, NADEEF, in the case of the *Beers* dataset, generated many false positives. Nevertheless, these false positives have circa no impact when the detections are repaired using GT (simulates a highly-effective repair method). In this case, false negatives in the detection phase become more harmful than false positives, in the presence of highly-effective repair methods.

For ML-oriented repair methods, we noticed that CPClean and BoostClean are hardly applicable to datasets associated with multi-class classification tasks. The underlying reason is that the methods divide each dirty dataset into batches, and each batch has to include samples from all classes. However, obtaining samples from each class is not always possible when there are several minority classes. For the datasets which have a binary classification problem, if the labels comprise erroneous cells, CPClean and BoostClean may not work due to introducing new values in the labels, turning the problem into a multi-class classification. For ActiveClean, it starts with partitioning the dirty dataset to obtain a clean fraction (i.e., data fraction without any errors) for warming up. Such a partition needs to represent all possible classes in the dataset. Therefore, ActiveClean searches for a partition that meets this condition. If it does not find such a partition, it returns an exception. Such a problem may happen in the following situations: (1) a dataset has too many classes with multiple minor classes (e.g., *Beers*) and (2) there exist no sufficient clean cells in the dataset.

Actionable Suggestions: Based on the results obtained in REIN, we provide the following suggestions while designing or selecting data cleaning tools: (1) tailor the design and evaluation of data cleaning methods to the planned downstream applications to properly select a well-suited cleaning method; (2) adopt simple cleaning strategies (non-learning detectors and generic repair methods) with classification tasks to combat attribute errors and more advanced cleaners (ML-based) with regression and clustering tasks; (3) exploit advanced techniques to combat class errors, e.g., CleanLab, data valuation, label smoothing, and noise-aware learning [43, 50]; (4) employ automated tools, e.g., FDX profiler and Metanome [41], to extract integrity constraints and functional dependency rules to properly use cleaning tools, such as NADEEF and HoloClean, with minimal user involvement; (5) adopt duplicates detection tools, e.g., ZeroER, record linkage and data hashing, as early as possible, in the ML pipeline, to prevent data leakage between the training, the validation, and test sets; and (6) avoid ML-based error detectors, e.g., ED2, RAHA, and Picket, while preparing large volumes of data (i.e., over 50k rows, as shown in Figure 3d) due to their poor scalability.

7 RELATED WORK

In fact, there exist few studies which survey or compare the already-existing data cleaning methods [2, 28, 29, 47]. Lee et al. [28] introduce a survey of five data cleaning methods and propose several research directions, such as integrating data cleaning methods with visual interface and the usage of high-performance memory management hardware solutions. Similarly, Ridzuan et al. [47] presents a review of data cleaning methods together with their challenges for dealing with big data. CleanML [29] introduces a relational database schema designed to organize the experimental results of investigating the impact of data cleaning on ML classification tasks. Since it does not consider the ground truth of each dataset, CleanML overlooks comparing the performance of ML models when trained using ground truth and repaired datasets. Moreover, CleanML limits the evaluations to simple classification tasks, while ignoring other ML tasks such as regression, clustering, and AutoML algorithms. In addition, CleanML does not consider the holistic, semi-supervised, or ML-oriented error detection and repair methods. In REIN, we tackle these shortcomings to generalize our findings to properly guide practitioners and data scientists while dealing with data cleaning problems in tabular data.

8 CONCLUSION

In this study, we introduced a benchmark framework, called REIN, to properly evaluate the error detection and repair methods. REIN enables ML engineers and practitioners to select the most well-suited data cleaning methods in ML pipelines. We carried out an extensive experimental study which involves 19 detectors, 19 repair methods, 33 ML models, and 14 datasets. The obtained results revealed that evaluating the data cleaning method in isolation from the downstream applications can be broadly misleading.

REFERENCES

[1] Milad abbaszadeh, Felix Neutatz, and Mohammad Mahdavi. 2018. Error Generator. https://github.com/BigDaMa/error-generator accessed on May 2021.

[2] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. 2016. Detecting data errors: Where are we and what needs to be done? *Proceedings of the VLDB Endowment* 9, 12 (2016), 993–1004.

[3] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. ACM, USA, 2623–2631.

[4] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, Jorge Luis Reyes-Ortiz, et al. 2013. A public domain dataset for human activity recognition using smartphones. In *Esann*, Vol. 3. Esann, 3.

[5] Patricia C Arocena, Boris Glavic, Giansalvatore Mecca, Renée J Miller, Paolo Papotti, and Donatello Santoro. 2015. Messing up with BART: error generation for evaluating data-cleaning algorithms. *Proceedings of the VLDB Endowment* 9, 2 (2015), 36–47.

[6] Javier Bejar and Ulises Cortes. 1993. Water Treatment Plant Dataset. https://bit.ly/3SBkduN accessed on January 2022.

[7] Felix Biessmann, Tammo Rukat, Phillipp Schmidt, Prathik Naidu, Sebastian Schelter, Andrey Taptunov, Dustin Lange, and David Salinas. 2019. DataWig: Missing Value Imputation for Tables. *Journal of Machine Learning Research* 20, 175 (2019), 1–6. http://jmlr.org/papers/v20/18-753.html

[8] Oliver Birgelen, Alexander; Niggemann. 2018. Smart Factory: High Storage System Data for Energy Optimization. https://www.kaggle.com/inIT-OWL/high-storage-system-data-for-energy-optimization accessed on January 2022.

[9] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[10] Xu Chu, John Morcos, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. 2015. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. ACM, 1247–1261.

[11] Daimler. 2017. Mercedes-Benz Greener Manufacturing. https://www.kaggle.com/c/mercedes-benz-greener-manufacturing accessed on January 2022.

[12] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed Elmagarmid, Ihab F Ilyas, Mourad Ouzzani, and Nan Tang. 2013. NADEEF: a commodity data cleaning system. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 541–552.

[13] Sanjib Das, AnHai Doan, Paul Suganthan G. C., Chaitanya Gokhale, Pradap Konda, Yash Govind, and Derek Paulsen. 2022. The Magellan Data Repository. https://sites.google.com/site/anhaidgroup/projects/data.

[14] Janez Demšar. 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research* 7, 1 (2006), 1–30. http://jmlr.org/papers/v7/demsar06a.html

[15] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[16] Hadi Fanaee-T and Joao Gama. 2013. Event labeling combining ensemble detectors and background knowledge. *Progress in Artificial Intelligence* (2013), 1–15. https://doi.org/10.1007/s13748-013-0040-3

[17] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and robust automated machine learning. *Advances in neural information processing systems* 28 (2015).

[18] Alice Berard Georges Hebrail. 2012. Individual household electric power consumption. https://archive-beta.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption accessed on January 2022.

[19] Kelli Ham. 2013. OpenRefine (version 2.5). http://openrefine. org. Free, open-source tool for cleaning and transforming data. *Journal of the Medical Library Association: JMLA* 101, 3 (2013), 233.

[20] Alireza Heidari, Joshua McGrath, Ihab F Ilyas, and Theodoros Rekatsinas. 2019. Holodetect: Few-shot learning for error detection. In *Proceedings of the 2019 International Conference on Management of Data*. ACM, 829–846.

[21] Jean Hould. 2017. *Craft Beers Dataset*. Kaggle. Accessed on February 2021.

[22] Bojan Karlaš, Peng Li, Renzhi Wu, Nezihe Merve Gürel, Xu Chu, and Ce Zhang. 2020. Nearest neighbor classifiers over incomplete information: From certain answers to certain predictions. *arXiv preprint arXiv:2005.05117* (2020).

[23] Ron Kohavi et al. 1996. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. *KDD* 96 (1996), 202–207.

[24] Pradap Konda, Sanjib Das, AnHai Doan, Adel Ardalan, Jeffrey R Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, Shishir Prasad, et al. 2016. Magellan: toward building entity matching management systems over data science stacks. *Proceedings of the VLDB Endowment* 9, 13 (2016), 1581–1584.

[25] Sanjay Krishnan, Michael J Franklin, Ken Goldberg, and Eugene Wu. 2017. Boostclean: Automated error detection and repair for machine learning. *arXiv preprint arXiv:1711.01299* (2017).

[26] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J Franklin, and Ken Goldberg. 2016. Activeclean: Interactive data cleaning for statistical modeling. *Proceedings of the VLDB Endowment* 9, 12 (2016), 948–959.

[27] Trang T Le, Weixuan Fu, and Jason H Moore. 2020. Scaling tree-based automated machine learning to biomedical big data with a feature set selector. *Bioinformatics* 36, 1 (2020), 250–256.

[28] Ga Young Lee, Lubna Alzamil, Bakhtiyar Doskenov, and Arash Termehchy. 2021. A Survey on Data Cleaning Methods for Improved Machine Learning Model Performance. arXiv:cs.DB/2109.07127

[29] 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]. *arXiv preprint arXiv:1904.09483* (2019), 75.

[30] Fei Tony Liu and Zhi-Hua Zhou. 2012. Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 6, 1 (2012), 1–39.

[31] Zifan Liu, Zhechun Zhou, and Theodoros Rekatsinas. 2020. Picket: Guarding Against Corrupted Data in Tabular Data during Learning and Inference. *arXiv preprint arXiv:2006.04730* (2020).

[32] Mohammad Mahdavi and Ziawasch Abedjan. 2020. Baran: Effective error correction via a unified context representation and transfer learning. *Proceedings of the VLDB Endowment* 13, 12 (2020), 1948–1961.

[33] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha: A configuration-free error detection system. In *Proceedings of the 2019 International Conference on Management of Data*. ACM, 865–882.

[34] Zelda Mariet and Sam Madden. 2016. *Outlier detection in heterogeneous datasets using automatic tuple expansion.* Technical Report. MIT CSAIL.

[35] Hugo Mathien. 2016. European Soccer Database. https://www.kaggle.com/hugomathien/soccer accessed on January 2022.

[36] Wes McKinney et al. 2010. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, Vol. 445. Austin, TX, Scipy, 51–56.

[37] Susan Moore. 2018. *How to Create a Business Case for Data Quality Improvement.* Gartner Research. Retrieved September 19, 2021 from https://www.gartner.com/smarterwithgartner/how-to-create-a-business-case-for-data-quality-improvement

[38] Felix Neutatz, Mohammad Mahdavi, and Ziawasch Abedjan. 2019. ED2: A case for active learning in error detection. In *Proceedings of the 28th ACM international conference on information and knowledge management*. ACM.

[39] Curtis G. Northcutt, Lu Jiang, and Isaac L. Chuang. 2021. Confident Learning: Estimating Uncertainty in Dataset Labels. *Journal of Artificial Intelligence Research (JAIR)* 70 (2021), 1373–1411.

[40] Ahmet Okudan. 2019. 3D Printer Dataset for Mechanical Engineers. https://www.kaggle.com/afumetto/3dprinter accessed on January 2022.

[41] Thorsten Papenbrock, Tanja Bergmann, Moritz Finke, Jakob Zwiener, and Felix Naumann. 2015. Data Profiling with Metanome. *Proc. VLDB Endow.* 8, 12 (Aug. 2015), 1860–1863. https://doi.org/10.14778/2824032.2824086

[42] F. Pedregosa, 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.

[43] Jose Picado, John Davis, Arash Termehchy, and Ga Young Lee. 2020. Learning over dirty data without cleaning. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1301–1316.

[44] Abdulhakim A Qahtan, Ahmed Elmagarmid, Raul Castro Fernandez, Mourad Ouzzani, and Nan Tang. 2018. FAHES: A robust disguised missing values detector. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2100–2109.

[45] Thomas Redman. 2016. *Bad Data Costs the U.S. $3 Trillion Per Year.* Harvard Business Review. Retrieved Septemper 19, 2021 from https://hbr.org/2016/09/bad-data-costs-the-u-s-3-trillion-per-year

[46] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. Holoclean: Holistic data repairs with probabilistic inference. *arXiv preprint arXiv:1702.00820* (2017).

[47] Fakhitah Ridzuan and Mohd Nazmee Wan. 2019. A Review on Data Cleansing Methods for Big Data. *Procedia Computer Science* 161 (2019), 731–738.

[48] Felix M. Riese and Sina Keller. 2018. Introducing a Framework of Self-Organizing Maps for Regression of Soil Moisture with Hyperspectral Data. In *2018 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, Valencia, Spain, 6151–6154. https://doi.org/10.1109/IGARSS.2018.8517812

[49] Iqbal H Sarker. 2021. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science* 2, 3 (2021), 1–21.

[50] Hwanjun Song, Minseok Kim, Dongmin Park, Yooju Shin, and Jae-Gil Lee. 2022. Learning from noisy labels with deep neural networks: A survey. *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[51] Daniel J Stekhoven and Peter Bühlmann. 2012. MissForest—non-parametric missing value imputation for mixed-type data. *Bioinformatics* 28, 1 (2012).

[52] D. Stuart Pope Thomas F. Brooks and Michael A. Marcolini. 2014. Nasa Airfoil Self-Noise Dataset. https://archive.ics.uci.edu/ml/datasets/airfoil+self-noise# accessed on January 2022.

[53] Larysa Visengeriyeva and Ziawasch Abedjan. 2018. Metadata-driven error detection. In *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*. ACM, 1–12.

[54] Renzhi Wu, Sanya Chaba, Saurabh Sawlani, Xu Chu, and Saravanan Thirumuruganathan. 2020. Zeroer: Entity resolution using zero labeled examples. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. ACM, 1149–1164.

[55] Ji Zhang. 2013. Advancements of outlier detection: A survey. *ICST Transactions on Scalable Information Systems* 13, 1 (2013), 1–26.

[56] Yunjia Zhang, Zhihan Guo, and Theodoros Rekatsinas. 2020. A statistical perspective on discovering functional dependencies in noisy data. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. ACM, 861–876.