BUNNI: Learning Repair Actions in Rule-driven Data Cleaning

GIANSALVATORE MECCA, DIMIE, Universita degli Studi della Basilicata, Potenza, Italy
PAOLO PAPOTTI, EURECOM, Sophia Antipolis, France
DONATELLO SANTORO, DiMIE, Universita degli Studi della Basilicata, Potenza, Italy
ENZO VELTRI, Universita degli Studi della Basilicata, Potenza, Italy

In this work, we address the challenging and open problem of involving non-expert users in the data-repairing problem as first-class citizens. Despite a large number of proposals that have been devoted to cleaning data from the point of view of expert users (IT staff and data scientists), there is a lack of studies from the perspective of non-expert ones. Given a set of available data quality rules, we exploit machine learning techniques to guide the user to identify the dirty values for each violation and repair them. We show that with a low user effort, it is possible to identify the values in tuples that can be trusted and the ones that are most likely errors. We show experimentally how this machine-learning approach leads to a unique clean solution with high quality in scenarios where other approaches fail.

CCS Concepts: • **Human-centered computing** → **Interactive systems and tools**; • **Computing methodologies** → *Machine learning approaches*; • **Information systems** → **Data cleaning**.

Additional Key Words and Phrases: Data Cleaning, Repair Discovery, Human In The Loop, Machine Learning

1 INTRODUCTION

Data cleaning, or data repairing, is the process of finding and removing errors from data. It is a crucial pre-processing step in many ML pipelines on different tasks, like medical predictions [37], or Text-To-SQL [52] or TNLI [6] applications. In the last two decades, a lot of work has been devoted to the data cleaning problem using *rule-based approaches* [5, 7, 9, 18, 22, 25, 27, 34, 51, 54]. In a rule-based approach, the users declare upfront some data-quality rules, typically under the form of integrity constraints, and the system enforces these constraints over the data.

Rule-based approaches to data repairing rely on machine learning [47] or on heuristics to generate solutions [5, 22, 25, 29, 34, 54]. These may include quite strong assumptions – for example, always repair the input instance according to the right-hand-side of rules – which may lead to unsatisfactory results, especially since it is unfeasible for the user to manually check all the solutions and fix them manually.

In this paper, we aim at solving the problem of *semi-automatic repair discovery* by involving the user in the generation of a unique solution for a given set of rules. We present Bunni[1], a machine-learning interactive framework for repair discovery. Starting from a Constant Conditional Functional Dependency (CFD) [16], the system first collects some training data by asking the user to solve a few violations. After a training step, the

[1]Bunni stands for Bayesian Update-driven iNteractive learNIng

Authors' Contact Information: Giansalvatore Mecca, DIMIE, Universita degli Studi della Basilicata, Potenza, Basilicata, Italy; e-mail: giansalvatore.m0000-0002-1189-1481ecca@unibas.it; Paolo Papotti, EURECOM, Sophia Antipolis, Provence-Alpes-Côte d'Azu, France; e-mail: paolo.papotti@eurecom.fr; Donatello Santoro, DiMIE, Universita degli Studi della Basilicata, Potenza, PZ, Italy; e-mail: donatello.santoro@unibas.it; Enzo Veltri, Universita degli Studi della Basilicata, Potenza, Basilicata, Italy; e-mail: enzo.veltri@unibas.it.

ACM 1936-1963/2024/5-ART
https://doi.org/10.1145/3665930

	Title	Author	Journal	Volume	Year
t_1	Possible and certain keys for SQL	H. Kohler	VLDB J.	25	2016
t_2	Possible and certain keys for SQL	U. Leck	VLDB J.	25	2015
t_3	Possible and certain keys for SQL	S. Link	VLDB J.	25	2015
t_4	Possible and certain keys for SQL	X. Zhou	VLDB J.	25	Aug.
t_5	RDF in the clouds: a survey	Z. Kaoudi	VLDB J.	25	2015
t_6	RDF in the clouds: a survey	I. Manolescu	VLDB J.	24	2015
t_7	A Survey on XML Fragmentation	V. Braganholo	VLDB J.	25	2014
t_8	A Survey on XML Fragmentation	M. Mattoso	SIGMOD Rec.	43	2014

Table 1. Dataset with publications extracted from the web.

system automatically infers when a rule should be enforced over a tuple in violation using a repair over the right-hand side (RHS), or conclusion, of the rule, or a repair over the left-hand side (LHS), or premise, for the same rule. We illustrate how Bunni works next.

Example 1: Consider the instance in Table 1 with publications scraped from the web. Consider CFD c_1, which states whenever the Journal is VLDB J. and the Volume is 25, the year must be 2016. Errors *w.r.t.* c_1 are highlighted in red.

$$c_1 : \text{t[Journal]=VLDB J., t[Volume]=25} \rightarrow \text{t[Year]=2016}$$

Assume a user needs to manually enforce CFD c_1 over the data. She inspects a few of the tuples in violation – t_2, t_3, t_4, t_5 and t_7 in our example – and updates t_2[Year] to 2016 first because in her knowledge (or after some manual search) the paper was published in the VLDB J. on volume 25 in the year 2016.

Then, she turns to tuple t_3. The user has already the knowledge on how to repair the violations by updating t_3[Year] to 2016. The reasons are related to the data that she is seeing: Title, Journal and Volume have the same values of the previous solved error, and moreover the Year has the same (wrong) value. In her mind, it is more likely that in tuple t_3 Year is erroneous instead of Journal or Volume. Similarly for tuple t_4.

Now the user inspects t_5, in this case, she is uncertain about how to solve it, because, even if Journal and Volume are the same as the previous repair, the Title is different and for deciding how to repair she uses again her knowledge. In this case, she decides to update the Volume to 24 instead of changing Year. Finally, the user inspects tuple t_7, in this case again she is uncertain about how to repair the data, and for removing the violations she needs to use again her knowledge. This time she decides to update the Journal and Volume because the paper was published in the SIGMOD Rec with Volume 43 in the Year 2014.

This leads to the cleaned instances in Table 2, where changes by the user are highlighted in green.

	Title	Author	Journal	Volume	Year
t_1	Possible and certain keys for SQL	H. Kohler	VLDB J.	25	2016
t_2	Possible and certain keys for SQL	U. Leck	VLDB J.	25	2016
t_3	Possible and certain keys for SQL	S. Link	VLDB J.	25	2016
t_4	Possible and certain keys for SQL	X. Zhou	VLDB J.	25	2016
t_5	RDF in the clouds: a survey	Z. Kaoudi	VLDB J.	24	2015
t_6	RDF in the clouds: a survey	I. Manolescu	VLDB J.	24	2015
t_7	A Survey on XML Fragmentation	V. Braganholo	SIGMOD Rec.	43	2014
t_8	A Survey on XML Fragmentation	M. Mattoso	SIGMOD Rec.	43	2014

Table 2. Dataset with publications cleaned.

From Example 1, one may observe that there might exist different reasons for removing the errors even for the same rule. In some cases, the values within the tuple give us a clear idea of how to repair the violation – as in t_4

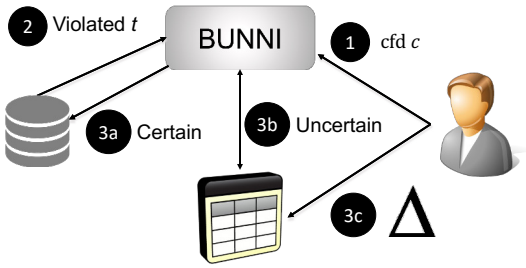

Fig. 1. Bunni workflow. Starting from a cfd c (1), Bunni detects a violated tuple t *w.r.t.* c (2). If Bunni is certain about the repair it applies it directly to the data (3a), otherwise if it is uncertain (3b), it asks the user to repair the tuple with some updates Δ (3c). It then learns how to repair similar violations using Δ to train a ML model.

above. This means that the user "trusts" some of the values, and "distrusts" other; the data-quality rule, along with the trusted values provide evidence on how to correct the error. In other cases, it is less clear to identify which values to trust within the tuple, and we need to rely on the user to investigate further the error and ultimately solve it, as in tuples t_2 and t_7.

The goal of Bunni is to mimic this form of "human thinking", trying to learn which values of the data can be trusted and which not, and involving humans only in the "hard decisions". Our goal is to minimize the number of user interactions, asking only critical questions. The workflow of Bunni is depicted in Figure 1.

1. The user submits a CFD c over instance $\mathcal{I}$.
2. Bunni finds violations *w.r.t.* c, selects one of them, i.e., over a tuple t, and tries to solve the violation.

3a. If Bunni finds out that cells of t matching the LHS of the rule can be trusted with a *confidence* above a fixed threshold, it applies the rule to repair the tuple with respect to the RHS.

3b. Otherwise, if Bunni is uncertain, i.e. if the rule cannot be trusted, it asks the user to manually solve the violation.

3c. If required by Bunni, the user manually updates the tuple, and Bunni uses this new evidence to refine its learning model.

The loop in points 2 and 3 terminates when no more tuples are in violation. Then the user goes back to step 1 if she has other rules to submit.

While solving the Repair-Discovery problem in such an interactive way, Bunni satisfies the following requirements:

- Trusted repairs: Bunni updates the data only when it has sufficient evidence that is applying a correct repair – otherwise, there is the risk of introducing more errors;
- Minimal cost: the number of user interactions to clean the instance should be minimal.
- Responsiveness: Bunni needs to suggest repair strategies within a few seconds.

Contributions. Bunni makes the following contributions:

- We formalize the Repair Discovery problem as a classification problem, showing that machine learning techniques can help the user in estimating the probability of a repair and in generating a clean instance of high quality.
- We introduce a solution based on active learning techniques to reduce the number of user interactions.
- We report experiments with real-world and synthetic data to show the effectiveness of Bunni in data repairing.

The following section introduces preliminary notions. We define in Section 3 how to involve the user in the repair discovery process. In Section 4, we formalize the repair discovery problem. Section 5 introduces how to calculate the confidence of a given cfd, and then we introduce a framework to solve the repair discovery problem (Section 6). We conclude the paper with a discussion of the experiments (Section 7), an overview of the related work (Section 8) and Conclusions with future work (Section 9).

2 CONSTRAINTS AND REPAIRS

Let us first discuss data-quality rules and their semantics. In this paper we use the semantic of *Conditional Functional Dependencies* (CFDs) [17]. Among these, we concentrate on *constant CFDs* [16].

We denote with $\mathcal{S}$ a database schema and with $\mathcal{R}$ a relation in $\mathcal{S}$ such that $\mathcal{S} = \{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n\}$. Each relation $\mathcal{R}$ is associated with a finite set of attributes and we denote it with $\mathcal{R} = \{A_1, A_2, \ldots, A_i\}$. We also denote the set of attributes of $\mathcal{R}$ with the symbol $attr(\mathcal{R})$. The domain of the attribute A_i is represented by $adom(A_i)$ and it is composed of constant values only. An instance of a relation $\mathcal{R}$ is a set of finite tuples where each tuple t is in the form of $\mathcal{R}(A_1 = v_1, A_2 = v_2, \ldots, A_i = v_i)$ with $v_i \in adom(A_i)$. An instance $\mathcal{I}$ over a schema $\mathcal{S}$ is a collection of instances, one for each relation in $\mathcal{S}$. We use the notation $t[A]$ to refer to a *cell* in the database, *i.e.,* to the value for the attribute A in the tuple t. We also assume the presence of a unique identifier for each tuple of a database instance. Symbol t_i denotes the tuple with identifier i.

Definition 2.1 (Constant CFD). A constant CFD over schema $\mathcal{S}$ and relation $\mathcal{R}$ is a logical formula $\psi \rightarrow \phi$ where ψ and ϕ are conjunctions of relational atoms; each atom is in the form: A=c, where A is an attribute $\in \mathcal{R}$ and c is a constant value in $adom(A)$. ψ is called premise of the CFD or left hand side (LHS), and ϕ is called conclusion of the CFD or right hand side (RHS).

Note that, without loss of generality, we will use only constant CFDs in *normal form*, *i.e.,* such that the conclusion contains only one atom, as follows:

$$\text{cfd: } \mathsf{A}_1\text{=}\mathsf{c}_1, \mathsf{A}_2\text{=}\mathsf{c}_2, \ldots, \mathsf{A}_i\text{=}\mathsf{c}_i \rightarrow \mathsf{A}_c\text{=}\mathsf{c}_c$$

CFDs with multiple atoms in the conclusion can be rewritten as a set of CFDs in normal form, one for each atom in the conclusion [16].

We denote with $\psi(A_i)$ the constant value c_i referred to the attribute A_i in the premise and with $\phi(A_c)$ the constant value c_c referred to the attribute A_c in the conclusion. We also denote with $attr(\psi)$ the set of the attributes in the premise.

Example 2: The following CFD states that if the ZIP attribute of a tuple is 85001 then the value for the STATE attribute needs to be AZ:

$$\text{cfd}_1\text{: ZIP=85001} \rightarrow \text{STATE=AZ}$$

Definition 2.2 (CFD Satisfiability). We say that an instance $\mathcal{I}$ over $\mathcal{S}$ satisfies a CFD *cfd*: $\psi \rightarrow \phi$, and we denote it with $\mathcal{I} \vDash cfd$, if for each tuple $t \in \mathcal{I}$ such that $t[A] = \psi(A)\ \forall A \in \psi$, it is the case that $t[A_c] = \phi(A_c)$. If there exists a tuple $t \in \mathcal{I}$ such that $\forall A \in \psi$, $t[A] = \psi(A)$ and $t[A_c] \neq \phi(A_c)$ then $\mathcal{I}$ does not satisfies *cfd* and we denote it with $\mathcal{I} \nvDash cfd$. Moreover, we say that t is in violation *w.r.t. cfd*. Given a set of CFDs Σ, we say that an instance $\mathcal{I}$ satisfies Σ, and we denote it with $\mathcal{I} \vDash \Sigma$, iff $\forall cfd \in \Sigma, \mathcal{I} \vDash cfd$. We say that $\mathcal{I} \nvDash \Sigma$ iff there exist a $cfd \in \Sigma$ such that $\mathcal{I} \nvDash cfd$.

Example 3: Consider again Table 1 and the CFD c_1: t[Journal]=VLDB J., t[Volume]=25 → t[Year]=2016. The cells for Journal, Volume and Year in tuple t_1 are not in violation with c_1, while the cells for Journal, Volume and Year in tuple t_7 are in violation. We can say that t_1 satisfies c_1, but t_7 does not, so the instance $\mathcal{I}$ in Table 1 does not satisfy c_1

Definition 2.3 (Clean Instance). Given an instance $\mathcal{I}$ and a set of CFDs Σ, we say that $\mathcal{I}$ is *clean* if $\mathcal{I} \models \Sigma$. Instead we say that $\mathcal{I}$ is *dirty* if $\mathcal{I} \not\models \Sigma$.

It is easy to see that $\mathcal{I}$ is dirty whenever there exist some violations, i.e., tuple t (or cell values in t) such that t matches with the premise of the CFD but does not match with the conclusion. Of course, a tuple t may be in violation with one or more CFDs.

Intuitively we can use one or more CFDs for making sure that our data is clean, or to find violations. The following CFD A1=c1,A2=c2,...,An=cn → Ac=cc can be translated in a SQL query that gives us the cells (tuples) in violation and looks like:

```
SELECT A1,A2,...,An  (*)
FROM relation R
WHERE A1=c1 AND A2=c2 AND ... AND An=cn AND Ac!=cc
```

If we find some violations we know that our instance $\mathcal{I}$ is dirty. The next step for cleaning the data is to choose a repair strategy. Given a set of rules Σ defined over the instance $\mathcal{I}$, a repair strategy is the process of removing violation *w.r.t.* Σ in $\mathcal{I}$ such that at the end of the process $\mathcal{I} \models \Sigma$.

In the literature, there are different strategies at different levels of granularity. One strategy consists of working with *tuple deletions*, to remove the tuples in violation *w.r.t.* Σ [1]. This kind of strategy may lead to an unnecessary loss of information because entire tuples are removed from the database.

An alternative is to work with *cell updates*, *i.e.,* to change the values of the cells in violation in order to clean them [5]. In this paper we only consider this second option.

Still, there are multiple ways to remove a violation for a CFD *cfd*:

(1) we may change the values of cells corresponding to attributes on the RHS of the *cfd*, *i.e.,* attributes appearing in the conclusion; this is called a **RHS Repair** (strategy);
(2) or we may change the values of cells corresponding to attributes in the LHS, *i.e.,* in the premise; this is called a **LHS Repair** (strategy).

In the RHS Repair, we *trust* the values in the premise of the *cfd* and use the constant in the conclusion to change the corresponding attribute and remove the violation. In the LHS Repair, we *trust* the value in the conclusion of the *cfd*, and therefore one or more of the values appearing within the premise need to be changed. Clearly, the step of choosing how to repair, is a crucial step, because more errors (not detectable with the same rule) can be introduced.

In this paper we also assume that all errors are detectable, i.e. errors that can be detected by one or more CFDs. However, in the same tuple, multiple detectable errors can be present. To be detectable, a tuple needs to match a CFD in the LHS, and have a different value for the RHS. There are two ways to introduce a detectable error:

- *RHS errors* violating the conclusion
- *LHS errors* triggering the premise

For example, suppose we have the CFD c_1 from Example 3 and the dataset reported in Table 3. A *RHS detectable error* can be obtained by changing t_1[Year] to a value different from 2016 (e.g. 2015). Many data repairing approaches can capture only this kind of errors [5, 20, 29, 54]. However, we also consider errors on the LHS. To trigger c_1, both t_7[Journal] and t_7[Volume] needs to be changed respectively in VLDB J. and 25. Finally, errors could occur on both sides, changing t_5[Volume] to 25 and t_5[Year] to 2018. The resulting dirty dataset is presented in Table 4.

Example 4: Consider the instance in Table 1 and the CFD c_1 from Example 3. The cells t_2[Journal], t_2[Volume], t_2[Year], t_7[Journal], t_7[Volume] and t_7[Year], are in violation *w.r.t.* c_1. Table 5 is an example of an RHS repair. We opted for an RHS repair by changing the values related to Year since we trusted the values in the premise of the c_1. Table 6 combines RHS and LHS repair strategies. For tuple t_2 we opted for an RHS repair strategy; on the

	Journal	Volume	Year
t_1	VLDB J.	25	2016
t_5	VLDB J.	24	2015
t_7	SIGMOD Rec.	43	2014

Table 3. Portion of the dataset in Table 2

	Journal	Volume	Year
t'_1	VLDB J.	25	2015
t'_5	VLDB J.	25	2018
t'_7	VLDB J.	25	2014

Table 4. Table 3 with detectable errors by using CFD c_1. t'_1 contains RHS detectable errors, t'_7 contains LHS detectable errors, and t'_5 contains RHS/LHS detectable errors.

contrary for tuple t_7 we selected to use an LHS strategy by changing t_7[Journal] to SIGMOD Rec and t_7[Volume] to 43. Tables 7 and 8 are examples of LHS repairs only. In the former, we used values from the active domain of the Journal and Volume attributes, while in the latter we used values outside of the active domain.

	Title	Author	Journal	Volume	Year
t_2	Possible and certain keys for SQL	U. Leck	VLDB J.	25	2016
t_7	A Survey on XML Fragmentation	V. Braganholo	VLDB J.	25	2016

Table 5. RHS Repair.

	Title	Author	Journal	Volume	Year
t_2	Possible and certain keys for SQL	U. Leck	VLDB J.	25	2016
t_7	A Survey on XML Fragmentation	V. Braganholo	SIGMOD Rec.	43	2015

Table 6. RHS and LHS Repair.

	Title	Author	Journal	Volume	Year
t_2	Possible and certain keys for SQL	U. Leck	VLDB J.	24	2015
t_7	A Survey on XML Fragmentation	V. Braganholo	SIGMOD Rec.	43	2015

Table 7. LHS Active Domain.

	Title	Author	Journal	Volume	Year
t_2	Possible and certain keys for SQL	U. Leck	TODS	12	2015
t_7	A Survey on XML Fragmentation	V. Braganholo	JDIQ	23	2015

Table 8. LHS open world assumption.

However, in this paper we do not consider both RHS and LHS repairs for a given violated tuple *w.r.t.* the same CFD. We leave this study for future research. This assumption is tied to the functioning of data repairing algorithms: to solve a violation of a dependency, it is sufficient to repair either the premise or the conclusion. Making changes to both is often deemed unjustified in many approaches. Notice that this assumption does not limit the presence of multiple errors in the same tuple: two or more errors might appear in the LHS of a single CFD, or a tuple might contain more than one error detected by multiple CFDs. Even with this assumption, choosing the repair-strategy $\mathcal{R}ep$ and the clean values $\mathcal{V}$ to repair violations in a complex data-repairing task is a crucial problem because multiple repaired instances can be generated, as shown by the previous example.

Making decisions about the repair process is usually difficult, both for humans and for machines. In the next, we introduce a number of technical tools to alleviate this problem.

3 USER INVOLVEMENT: FROM NAÏVE TO INTERACTIVE

In the last few years, several studies have been devoted to consider the presence of humans in the repairing process [30, 33, 46, 49, 55, 58]. Early proposals have mainly relied on what we might call a *"naïve"* way to involve users in the rule-based repairing process, as follows.

The user first uses her domain knowledge to write the constraints. Different tools use different constraint languages, with different expressive power and different syntax and semantics. In the following, we refer to a generic data-repairing system, called System, with its own language of reference.

System accepts a dirty instance $\mathcal{I}$, a set of constraints Σ, a strategy for repair discovery $\mathcal{R}ep$ – i.e., an algorithm to decide which cells are dirty for each tuple in violation with Σ –, and a strategy for value discovery $\mathcal{V}$ – i.e., an algorithm to replace cells that are deemed dirty with clean values. In some cases, System provides multiple algorithms to choose from, and the user needs to make a choice. In other cases, System will restrict the choice to a very few options – e.g., always repair the RHS, or always introduce variables on the LHS – thus constraining the choice. It is unlikely, however, that these options generate high-quality solutions in all scenarios.

At the end of the cleaning process, System generates one or multiple solutions (this depends on $\mathcal{R}ep$ and $\mathcal{V}$). In this case, the user will manually inspect all the generated solutions. If System introduces variables, i.e., placeholders, to remove violations, then the user needs to change each placeholder into the correct constant value.

In the end, the user selects the best solution among those generated. By examining the solutions, she might find out that some of the errors in the data were not detected or repaired by System. In this case, she needs to write additional rules that capture these errors, and execute the process again.

In this *"naïve"* form, the user is involved exclusively at the beginning of the process, to identify the constraints, and at the end, to inspect the generated solutions. In practice, this process suffers from some important limitations:

(1) Writing data-quality rules upfront is difficult, even for expert users, especially since this requires both technical and business knowledge. Writing the rules is time-consuming and if the instance is large it might be the case that some of the errors in the data are not covered by the rules.
(2) Many automatic data-repairing systems generate multiple solutions, by combining both LSH and RHS repairs, and possibly different orders of execution of the dependencies. Inspecting all solutions after they have been generated can be highly time-consuming for the user.
(3) Most data-repairing systems rely on heuristics to tame the complexity of the repair process. The user often has little control over these heuristics and this may prevent the system from generating the desired solution.

Notice that problem 1 above might, in practice, be alleviated with some discovery systems [29, 32]

3.1 User involvement: the interactive way

In this paper, we investigate a different protocol to involve the user in the repair process. We try to do this in a way that is the most intuitive, even for a non-expert user, and is based on direct interaction with the data. Thus, we require the user to *a*) inspect cells wrong *w.r.t.* a constraint and *b*) resolve it by submitting a **value update**.

These value updates are then used by our learning algorithms to infer the intended repair strategy, and the strategy to select clean values. In fact, by providing an update:

(1) the user indicates **which cells need to be repaired**: given a tuple that contains errors, by changing one or more values within the tuple, the user states which values are erroneous, and which ones should be trusted; again, we may try to generalize this knowledge to identify the correct repair strategy for similar violations;
(2) the user indicates **the right values** for the dirty cells, thus providing information on how to fix errors.

While automatic rule-based systems can be seen as *black boxes*, where the user submits the dirty instance, the rules, and some configuration, and gets as output one or more solutions to inspect, in our approach the user is continuously involved in the repair process, by providing a limited number of updates, and answers to queries formulated by the system in order to interactively discover repair strategies and clean values.

As a result, the system and the user generate a single **trusted repair**, in which all choices taken during the repair process are validated directly or indirectly by user actions.

To develop such an interaction protocol, a main challenge needs to be addressed: given a constraint in the form of a Constant CFD we need to solve the problem of identifying the right repair strategy for each violation. Again, we have multiple options, by either changing the cell on the RHS of a rule, or any of the cells on the LHS. This choice is critical, in order to generate trusted repairs.

4 PROBLEM STATEMENT

We consider an instance $\mathcal{I}$ and a constant CFD *cfd* in the normal form. If $\mathcal{I} \not\models cfd$, for each tuple t in $vio(cfd)$ we want to decide when to repair t using an RHS repair or an LHS repair *w.r.t.* user-defined confidence. This amounts to deciding which of the cells involved in the violation can be trusted.

Since there is a violation *w.r.t. cfd*, it is unlikely that cells corresponding to the premise and to the conclusion are both dirty. Therefore, we make the assumption that errors can be only on one side.

We use the symbol $\phi_{cfd}(t)$ to denote the cells of t corresponding to the attributes in the LHS of a *cfd*. The problem of deciding where repair corresponds to deciding if the cells of $\phi_{cfd}(t)$ can be trusted. We *trust* $\phi_{cfd}(t)$ if its cells are clean, otherwise we *do not trust* $\phi_{cfd}(t)$. More formally, we define when cells in $\phi_{cfd}(t)$ can be trusted as follow:

Definition 4.1 (Trust $\phi_{cfd}(t)$). Given a constant CFD *cfd* and a violated tuple t we say that the set of cells in $\phi_{cfd}(t)$ can be trusted if, for each cell in $\phi_{cfd}(t)$, the probability that the cell is clean is above a threshold τ_{clean}.

More specifically, we fix two user-defined thresholds, τ_{clean} and τ_{dirty}. Then, for each cell c_i involved in a violation, we estimate the probability p_i that it is clean. We rule that c_i can be trusted if $p_i > \tau_{clean}$; we rule that the c_i is dirty if $p_i < \tau_{dirty}$. We ask the user otherwise.

In light of this, solving the Repair Discovery problem amounts to calculating the probability that a cell is clean. We will introduce such a framework in the next section.

5 PROBABILISTIC REASONING

We first introduce how to predict if one cell is clean, we then extend this concept to all cells for a tuple in a violation.

5.1 Probabilistic model for one cell

Given a tuple t involved in a violation *w.r.t.* a CFD *cfd*, and a cell $c \in t$, we want to calculate the probability that, given t, the hypothesis "c is clean" is true. We denote the hypothesis "c is clean" with the symbol $c^{\checkmark}$ and the hypothesis "c is dirty" with the symbol $c^{\text{✖}}$. The Bayes Theorem in the Equation 1 helps us to estimate the probability:

$$\mathbf{Pr}(H|e) = \frac{\mathbf{Pr}(e|H)\mathbf{Pr}(H)}{\mathbf{Pr}(e)} \tag{1}$$

$\mathbf{Pr}(H|e)$ is the probability that the hypothesis H is true given the event e. It is also called posterior probability. $\mathbf{Pr}(H)$ is the probability that hypothesis H is true, and it is called prior probability. $\mathbf{Pr}(e|H)$ is the probability

that we observe the event e when the hypothesis H is true. Usually, $\mathbf{Pr}(e|H)$ is called likelihood. Finally, $\mathbf{Pr}(e)$ is the probability of observing the event e. $\mathbf{Pr}(e)$ is also called marginal likelihood. Note that $\mathbf{Pr}$(e) is the same for all the possible hypotheses H. In our database context the hypothesis H corresponds to $c^{\checkmark}$ or $c^{\text{✖}}$. The event e corresponds to the observation related to the data, *i.e.,* the tuple t that contains the cell c.

Example 5: Consider a Table with three attributes A,B,C that contains one tuple t where t[A]=a, t[B]=b and t[C]=c. t is in violation with a CFD that involves all the attributes in the Table. The probability that the cell related to A is clean, *i.e.,* the value of A is correct, and we denote it with $A^{\checkmark}$, can be calculated using the Bayes Theorem as:

$$\mathbf{Pr}(A^{\checkmark}|t) = \mathbf{Pr}(A^{\checkmark}|A=a, B=b, C=c) = \frac{\mathbf{Pr}(A=a, B=b, C=c|A^{\checkmark})\mathbf{Pr}(A^{\checkmark})}{\mathbf{Pr}(A=a, B=b, C=c)} \tag{2}$$

From Example 5 one might observe that the denominator in Equation 2 is a constant factor that is used to normalize the probability, *i.e.,* changing the hypothesis does not change it because it is independent from the hypothesis and depends only from the data. So we shall ignore it for now.

The probability that we want to calculate is proportional to the joint probability $\mathbf{Pr}$(A=a,B=b,C=c|$A^{\checkmark}$) and $\mathbf{Pr}$($A^{\checkmark}$). In the case of a high number of random variables, *i.e.,* when the number of attributes and their possible values are high, calculating the joint probability is unfeasible because we can have an exponential number of combinations. To alleviate this problem, we make a *strong assumption*: given the hypothesis, *i.e.,* knowing H is true or false, we assume that each observed random variable is *independent* from the other observed random variables. In other words, we assume that all the attributes are *conditionally independent* from each other. This simplifies our calculus, as follows.

In the following, we abuse the notation A_i for representing the i^{th} attribute of a tuple t and its corresponding constant value, e.g., A_1 corresponds to $A_1 = a_1$. The letter H represents our hypothesis, e.g., "attribute Attr is clean".

$$\begin{aligned}\mathbf{Pr}(A_1, \ldots, A_n|H) &= \mathbf{Pr}(A_1|H, A_2, \ldots, A_n) \ldots \mathbf{Pr}(A_{n-1}|H, A_n)\mathbf{Pr}(A_n|H) = \\ &= \mathbf{Pr}(A_1|H) \ldots \mathbf{Pr}(A_{n-1}|H)\mathbf{Pr}(A_n|H) = \prod_{i=1}^{n} \mathbf{Pr}(A_i|H)\end{aligned} \tag{3}$$

In the first line of Equation 3 we use the chain rule for calculating the joint distribution, then we use the conditional independence assumption. We can now use the formula to calculate the probability that a cell c is clean, given a tuple t.

$$\mathbf{Pr}(c^{\checkmark} \mid t) \;\propto\; \mathbf{Pr}(c^{\checkmark}) \prod_{i=1}^{n} \mathbf{Pr}(t[A_i] \mid c^{\checkmark}) \tag{4}$$

Equation 4 corresponds to the *Naïve Bayes classifier*. The name comes from the assumption of the conditional independence between random variables. Estimating the probability of a cell being clean is a classification problem.

Classification Problem. A classification problem [4, 35] consists in identifying or classifying an *observation* in a *category* based on past observations for which the category was known. The past observations are called *training set*. The specified category is usually called label. A classifier (or model) is trained over the training set. After the training step, the classifier will be able to classify new observations. The predicted value is also called an outcome.

In our database model, the training set consists of a set of tuples. In addition to the standard attributes, each tuple in the training set has a new attribute that we call Response, which stores repairs to the original database instance. For the purpose of training, the Response attribute contains values interactively submitted by the user. During the classification step, the Response attribute is populated using the classifier outcome.

Given one hypothesis H ($c^{✔}$ or $c^{✖}$), our problem becomes a binary classification problem, where the outcome of the problem is a binary value T or F that corresponds to the validity of the hypothesis H. The decision about the outcome is related to the probability p calculated for each hypothesis (T or F). The hypothesis with the highest probability is returned (*maximum a posteriori likelihood*). For every outcome predicted, there is also a probability p that can be used for saying "the cell c is dirty with probability p_{dirty}" rather than "the cell c is clean with probability p_{clean}".

Probabilistic model for multiple cells. To handle the case in which more than one cell of a tuple t may be dirty, we extend the binary classification model. A *multilabel classifier* predicts one or more outcomes for each observation. Practically, a multilabel classifier is based on a set of n binary classifiers, one for each outcome, where n represents the number of the possible values for the outcome. Then, given a threshold τ and an observation to classify, the response consists in all of the predicted outcomes of each binary classifier such that the probability that the hypothesis is true is above a threshold τ. In our case, we generate a classifier for each attribute in the premise of the CFD.

5.2 Parameter Estimation

To calculate the posterior probability with Equation 4 we need to estimate the two parameters in the formula: the prior probability that the current hypothesis is true, *i.e.,* the prior $\mathbf{Pr}(c^{✔})$, and for each cell in t the likelihood $\mathbf{Pr}(t[A_i]|c^{✔})$. For estimating the parameters we use the *maximum likelihood estimate*. The prior $\mathbf{Pr}(c^{✔})$ corresponds to the frequency of the hypothesis $c^{✔}$ in the training data. The likelihood $\mathbf{Pr}(t[A_i]|c^{✔})$ can be estimated in a similar way: it is the frequency that $t[A_i]$ and $c^{✔}$ appear together *w.r.t.* the number of occurrences of $c^{✔}$.

Notice that in some cases the likelihood $\mathbf{Pr}(t[A_i]|c^{✔})$ can be zero. When it is zero the posterior probability also will be zero because in the training $t[A_i]$ never occurs with $c^{✔}$. Intuitively we penalize the posterior probability only because in our training data we have never seen such an event. This could happen if the training data is small. To avoid this problem a common practice consists in using the *Laplace smoothing*. The Laplace smoothing can be applied both to the prior probability and to the likelihood.

$$\mathbf{Pr}(c^{✔}) = \frac{count(c^{✔}) + 1}{m + k} \tag{5}$$

$$\mathbf{Pr}(t[A_i]|c^{✔}) = \frac{count(t[A_i] \wedge c^{✔}) + 1}{count(c^{✔}) + k} \tag{6}$$

$count()$ is a function that counts the number of occurrences of the parameter(s) in the data. And k is the number of the outcomes of the classifier (in our case we use n binary classifiers so $k = 2$).
Finally, if we want to calculate the denominator in Equation 2 the formula is the following:

$$\mathbf{Pr}(t) = \sum_{i=1}^{k} \mathbf{Pr}(c_k)\mathbf{Pr}(t|c_k) = \sum_{i=1}^{k} \mathbf{Pr}(c_k) \prod_{j=1}^{n} \mathbf{Pr}(t[A_j]|c_k) \tag{7}$$

where k again is the number of the outcomes ($c^{✔}$ or $c^{✖}$), and n is the number of attributes in the tuple t. Note that this model has a low impact on memory because only the frequencies are stored, rather than all of the training data.

6 FRAMEWORK FOR REPAIR DISCOVERY

Algorithm 1 contains the pseudocode for the repair discovery, while Figure 2 depicts the detailed workflow. Algorithm 1 takes as input the instance $\mathcal{I}$ to clean, a CFD to enforce over the data, the confidence thresholds expressed using τ_{clean} and τ_{dirty}. First, it finds the tuples in violation *w.r.t.* the CFD(pointer 1 in Figure 2). If there are not violated tuples it goes to the next CFD (lines 2-3, pointer 2), otherwise it creates the multilabel classifier to classify if the cells that correspond to the LHS are clean. Then, it collects m labeled examples by asking the user to solve violations (lines 5-11), and uses them to train the classifier (pointer 3-5 in Figure 2). The updates are also used to repair the m tuples. We will discuss later on the strategy used to select the next tuple, pickOne (line 6).

After the training step, for each remaining violated tuple t, the classifier predicts which of the attributes on the LHS are clean (lines 12-27, pointer 7). If all the attributes are clean with a probability p_{clean} above a threshold τ_{clean} (LHSTrusted), the system repairs the RHS (lines 15-17). Otherwise, if the system detects that some cells corresponding to the premise of the CFD are dirty, and the probability p_{clean} of at least one cell is below a threshold τ_{dirty} (LHSContainsErrors), the system suggests to repair using an LHS strategy (lines 18-20). We discuss later the function LHSRepair used for this purpose. Finally, if the classifier cannot classify the tuple t, *i.e.,* the probabilities related to the predictions are less than τ_{clean} but above τ_{dirty}, the system asks the user to repair them (line 22, pointer 4). The submitted values are used both for repairing the data and refining the training to make more accurate predictions in the future.

The complexity of Algorithm 1 is linear *w.r.t.* the number of violated tuples. Indeed, m tuples are used to initialize the classifier, and the remaining violated tuples are used for inference or to retrain the classifier if it is in the indecision step (lines 22-25). The complexity of the training is also linear *w.r.t.* the training examples submitted and there is no training step with Gradient Descent to execute since Naïve Bayes only stores frequencies of each attribute values [4].

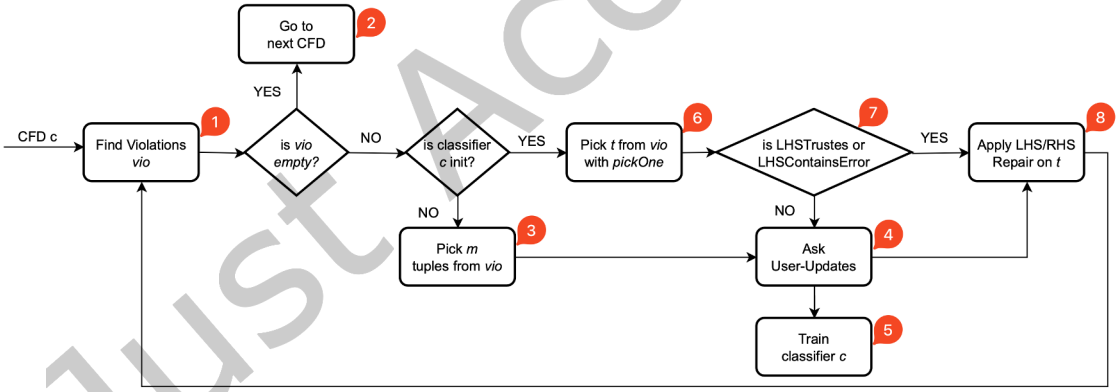

Fig. 2. Bunni starts from a CFD c and finds the set of violated tuples. If there are no violations, it goes to the next CFD (2). If there are violations, it selects m tuples to be manually cleaned by the user (3-4). Such updates are used to train a classifier c (5), and to repair the m tuples. After the initialization step, if there are violations, tuple t to be repaired is selected using *pickOne* function (6). The classifiers predicts if the LHS could be trusted (7), either repair is applied (8) or it asks the user to manually repair the tuple t (4), and also retrains the classifier c with the new user updates (5). The process continues until the set of violated tuples is empty.

Example 6: Consider the instance in Table 1, let us assume that the user has manually cleaned the errors for the violated tuples *w.r.t.* the CFD c_1 in Example 1 by submitting some updates. Assume that in Algorithm 1 $m = 5$, *i.e.,* the minimum number of training tuples is five.

Algorithm 1 Repair instance

Require: Instance $\mathcal{I}$, CFD *cfd*, τ_{clean}, τ_{dirty}
Ensure: Set of cleaned tuples *clean*

```
1: init clean to empty set; init vio as violated tuples w.r.t. cfd; create multilabel classifier c for LHS of cfd
2: if vio is empty then
3:     return clean
4: end if
5: for i = 0 to m do                                              ▷ [classifier initialization]
6:     Tuple t = pickOne(vio)
7:     Set updates = userSubmit(t)
8:     train(c, updates, t)
9:     t' = updateTuple(t, updates)
10:    clean.add(t')
11: end for
12: while vio not empty do                                        ▷ [classifier predictions]
13:    Tuple t = pickOne(vio)
14:    Set errors = c.classify(t)
15:    if LHSTrusted(errors, τ_clean, τ_dirty) then
16:        Tuple t' = RHSRepair(errors, t)
17:        clean.add(t')
18:    else if LHSContainsErrors(errors, τ_clean, τ_dirty) then
19:        Tuple t' = LHSRepair(errors, t)
20:        clean.add(t')
21:    else                                                       ▷ [indecision in classification]
22:        Set updates = userSubmit(t)
23:        train(c, updates, t)
24:        t' = updateTuple(t, updates)
25:        clean.add(t')
26:    end if
27: end while
28: return clean
```

	Title	Author	Journal	Volume	Year
t_1	Possible and certain keys for SQL	H. Kohler	VLDB J.	25	2016
t_2	Possible and certain keys for SQL	U. Leck	VLDB J.	25	2016
t_3	Possible and certain keys for SQL	S. Link	VLDB J.	25	2016
t_4	Possible and certain keys for SQL	X. Zhou	VLDB J.	25	2016
t_5	RDF in the clouds: a survey	Z. Kaoudi	VLDB J.	24	2015
t_6	RDF in the clouds: a survey	I. Manolescu	VLDB J.	24	2015
t_7	A Survey on XML Fragmentation	V. Braganholo	SIGMOD Rec.	43	2014
t_8	A Survey on XML Fragmentation	M. Mattoso	SIGMOD Rec.	43	2014
t_9	Possible and certain keys for SQL	U. Leck	VLDB J.	25	2014
t_{10}	A Survey on XML Fragmentation	V. Braganholo	VLDB J.	25	2014

Table 9. Algorithm 1 example.

For t_2, t_3 and t_4 the user submits the updates on Year with the value 2016. For t_5 she submits the update on Volume with the value of 24. And finally for t_7 she submits the updates on Volume and Journal respectively with the values of 43 and Sigmod Rec.

Now consider tuples, t_9 and t_{10} as shown in Table 9. We first need to calculate the probabilities using Equations 5, 6 and 7 Estimated probabilities are reported in Table 10. Prior probabilities are in bold. For example, the prior for the hypothesis that Journal is clean (Journal$^{\checkmark}$) is 62.5%. Instead $\mathbf{Pr}(t_9[\text{Title}]|\text{Journal}^{\checkmark})$ is 66.6%. Note that for the same values in the same attributes, the estimated probabilities are the same, like Year for t_9 and t_{10}. Now let we say that user declared $\tau_{clean} = 0.6$ and $\tau_{dirty} = 0.15$. Let us estimate the probabilities that Journal is clean in t_9.

	Pr(Journal$^{\checkmark}$)	Pr(Volume$^{\checkmark}$)	Pr(Journal$^{\times}$)	Pr(Volume$^{\times}$)
Cell	**0.625**	**0.5**	**0.25**	**0.375**
t_9[Title]	0.666	0.8	0.333	0.25
t_9[Author]	0.333	0.4	0.333	0.25
t_9[Journal]	0.833	0.8	0.666	0.75
t_9[Volume]	0.666	0.8	0.666	0.75
t_9[Year]	0.166	0.2	0.666	0.5
t_{10}[Title]	0.166	0.2	0.666	0.5
t_{10}[Author]	0.166	0.2	0.666	0.5
t_{10}[Journal]	0.833	0.8	0.666	0.5
t_{10}[Volume]	0.666	0.8	0.666	0.75
t_{10}[Year]	0.166	0.2	0.666	0.5

Table 10. Estimated probabilities

$$\mathbf{Pr}(\text{Journal}^{\checkmark} \mid t_9) = \frac{\mathbf{Pr}(\text{Journal}^{\checkmark}) \prod_{i=1}^{n} \mathbf{Pr}(t_9[A_i] \mid \text{Journal}^{\checkmark})}{\mathbf{Pr}(t_9)} \tag{8}$$

$$\mathbf{Pr}(t_9) = \mathbf{Pr}(\text{Jou.}^{\checkmark}) \prod_{i=1}^{n} \mathbf{Pr}(t_9[A_i] \mid \text{Jou.}^{\checkmark}) + \mathbf{Pr}(\text{Jou.}^{\times}) \prod_{i=1}^{n} \mathbf{Pr}(t_9[A_i] \mid \text{Jou.}^{\times}) \tag{9}$$

Using the above equations it is easy to calculate $\mathbf{Pr}(\text{Journal}^{\checkmark}|t_9)$= 0.62. With a similar calculus we can obtain that $\mathbf{Pr}(\text{Volume}^{\checkmark}|t_9)$= 0.77. Since both the posterior probability are greater than τ_{clean}, LHSTrusted returns *true* and Algorithm 1 repairs t_9 with an RHS repair, updating the value of the year to 2016.

For tuple t_{10} we can estimate again both the posterior probabilities and we obtain that $\mathbf{Pr}(\text{Journal}^{\checkmark}|t_{10})$= 0.05 and $\mathbf{Pr}(\text{Volume}^{\checkmark}|t_{10})$= 0.10. In this case, LHSTrusted returns false, so probably there are some errors in the LHS cells. LHSContainsErrors returns *true* since for both the cells the estimated posterior probabilities that are clean is less than 0.15. Moreover LHSRepair suggests both cells be repaired by the user. The user using this information submits both the updates for the violated cells (SIGMOD Rec. and 43).

Instead, if the thresholds are $\tau_{clean} = 0.90$ and $\tau_{dirty} = 0.10$, for tuple t_9 the Algorithm is in the indecision area and it asks the user to manually repair the violations since it is not able to make a correct prediction. Instead for tuple t_{10}, it suggests again Journal and Volume as cells with errors.

Active Learning. Another goal of our system is to minimize user interactions compared to manually cleaning up the entire database. Active learning [50] is a special case of semi-supervised machine learning to substantially reduce the amount of labeling when training a model. Active Learning has been proven to reduce the number of manually labeled examples while improving the classification performance [48]. In essence, in active learning, the learner interactively labels the data points that lead to the highest gain in improving the prediction performances. The goal of active learning is to substantially reduce the number of required labels.

We adopt this approach for our problem. For each tuple in violation, we calculate a score that measures the *uncertainty* of the classifier. Intuitively, submitting to the user the tuple with the highest score, *i.e.,* with the highest uncertainty, will increase the quality of the model in a faster way. For each cells c in $\phi_{cfd}(t)$ of a generic CFD cfd, we are able to predict if $c^{\checkmark}$ with a probability p_{clean} and predict if $c^{\times}$ with a probability p_{dirty}. The score for each cell is $score_c = 2 \cdot p_{clean} \cdot p_{dirty}$. The score for a tuple is the sum of each score for each cell in the premise of the CFD. This active learning strategy is the one implemented in the function pickOne in the Algorithm 1.

Value selection for repairs. The RHS repair, as we know, is deterministic and the value used for the repair is the one in the conclusion of the CFD. We cannot say the same for the LHS repair. There are two possible strategies to solve this problem: *a*) to repair cells using *placeholders*, like the LLUNs [25]. At the end of the process, the user can inspect the repaired data and changes the values of the placeholders; *b*) to interactively highlight the cells with errors in order to focus the attention of the user and ask the user to submit the correct values for repairing violations. This case is powerful enough to solve some classification problems too, because the user may correct other cells *w.r.t.* the suggested wrong cells, and this corresponds to submitting more training data for the classifier. We use the second option.

Handling multiple CFDs. For the sake of simplicity, we presented the algorithm for a single CFD. However, as demonstrated in the experimental section, Bunni is capable of handling multiple dependencies. For a single CFD, terminating the repair process is straightforward since each repair suggested by Bunni reduces the number of conflicting tuples. With multiple CFDs, the termination of the repairing process is guaranteed by the following two assumptions:

(1) the input dependencies are correct
(2) the changes provided by the user are always correct: i.e., the user is considered an oracle

For instance, suppose to have the following dependencies:
cfd_1: ZIP=85001 → STATE=AZ, cfd_2: STATE=AZ → ZONE=Central, cfd_3: ZONE=Central → STATE=NM
and a dataset containing, among others, the tuple

	ZIP	STATE	ZONE	COUNTRY
t_1	85001	AZ	Central	U.S.A.

Tuple t_1 violates cfd_3, and satisfies both cfd_1 and cfd_2. To repair t_1, we need to change either attribute ZONE (LHS of cfd_3) or attribute STATE (RHS of cfd_3). However, both changes will violate the other dependencies. A sequence of RHS repairs would lead to an endless cycle of operations. To solve the problem, during the process, we track the changes applied to each dependency in previous steps, ensuring that the same RHS repair is not suggested more than once: in these cases, we always rely on the user to repair the dependency by changing the premise or the conclusion of the CFD. A change certified by the user is considered with $p_{clean} = 1$ during the next steps of the process. In our example, the only viable solution involves changing t_1[ZIP], t_1[STATE] and t_1[ZONE] to values different from their originals.

Another aspect to consider is dependency interaction: resolving one violation might introduce another. To avoid repeated checks, we build a *dependency multigraph dg* as follows:

- for each dependency d_x we add a node n_x in dg
- we add a labeled edge e_{Ai} between nodes n_x and n_y if the respective dependency d_x, d_y both have attribute A_i in either the premise or the conclusion.

Initially, Bunni calculates the number of violating tuples for each dependency. After these checks, a dependency is classified either as *clean* or *to clean*. As long as there are dependencies to clean, Bunni selects the one with the most violating tuples, denoted as d_x, and executes Algorithm 1. During the repair, for each cell update $t_j.A_i$, the

dependencies associated with nodes connected to node n_x through a labeled node e_{Ai} are marked as *to check*. At the end of Algorithm 1, dependency d_x is marked as *clean*. The process ends when all the dependencies are cleaned.

Using *dg*, we can easily identify the dependencies affected by any cell change. In addition, by computing the connected components of the graph, it is possible to identify groups of independent dependencies. The process of each group of dependencies can be parallelized, to optimize the performance of the system.

7 EXPERIMENTS

We implement Bunni in Java and use PostgreSQL 11 as the underlying DBMS. All experiments are executed on a MacBook Pro with an Intel i9 CPU@2.9Ghz and 32GB of memory.

Datasets. We use three real-world datasets and two synthetic datasets.

(1) BUS: a UK government public dataset (http://data.gov.uk/data), it deals with bus schedules and routes. It contains 15 attributes and 250K tuples.
(2) DBLP: is based on the collection of authors, publications and venues from http://dblp.uni-trier.de/xml/. We translated the xml dataset into a single relational table with 15 attributes. It contains 1M tuples.
(3) Synth is a dataset we designed starting from the original Soccer dataset to study the scalability over the number of tuples and a larger number of attributes. The dataset has 10 attributes and we used a data generator (http://www.cs.toronto.edu/tox/toxgene/) to create an instance of 100k tuples;
(4) TAX is a synthetic dataset extracted from the scenario used in [17] with 10 attributes and 400k tuples.
(5) FLIGHT [38] is a real-world dataset related to flight departures and arrivals. It contains real errors and is provided with both clean and dirty versions.

Classifiers. We compare the classification strategy based on the Naïve Bayes to four baselines:

(1) *Logistic Regression*, a classical machine learning approach. Like the Naïve Bayes, Logistic Regression is able to compute the probability related to each prediction.
(2) *GDR*. We extend the approach implemented in GDR [58]. We used a forest of decision trees. Notice, that this approach allows us to also compute a probability as the number of trees that are agreeing with the prediction *w.r.t.* the number of the learned trees.
(3) *HoloClean*. We use the implementation of HoloClean [47]. HoloClean uses a weakly supervised approach, so it does not require any user interaction. We configure HoloClean using the default parameters. Notice that the implementation is built on top of PyTorch.
(4) *Dummy* classifier. We develop a classifier that does not require user interactions and works in a deterministic way. It always repairs violations by changing the attribute on the RHS of the rule.

We use WEKA [28] as the implementation for Naïve Bayes, Logistic Regression, and GDR using the default configurations (except for Dummy).

Errors and Metrics. To compute the quality of a data repair algorithm we need two versions of the same dataset: a *dirty* version with the errors and its *clean* version to compare with the repaired generated solution. Although some small datasets come with the *dirty* and *clean* versions, this is not true for datasets with thousands of tuples. Since our goal is to evaluate our rule-based approach (given a set of Constant CFDs we measure the quality of the repaired instance), we use the BART error generation tool [2], which allows to introduce detectable errors w.r.t. a set of input CFDs by changing the cell values. We can then measure the quality of a data repair for each declared CFD. BART allows the fair evaluation of different data-cleaning systems datasets of different sizes. Moreover, BART allows configuring how errors are injected, i.e., how many errors are introduced in the LHS/RHS of the CFDs and thus we also control how many errors are introduced for each tuple. By default, BART introduces errors

on any side of the given CFDs but not on both sides for the same CFD. Even though, we can have multiple errors per tuple that could be detected by multiple CFDs.

We manually define six normalized constant CFDs for each dataset with the following characteristics: a) the first two CFDs, namely cfd_1 and cfd_2, with only one attribute on the LHS. For example t[Team]=AC Milan → t[City]=Milan; b) the second two CFDs, cfd_3 and cfd_4, with two attributes on the LHS. An example of CFD for BUS is t[administrativeareacode]=101, t[nptgdistrictcode]=189 → t[regioncode]=EM; c) the last two CFDs, cfd_5 and cfd_6, with more than two attributes on the LHS (depending on the scenario). For example in DBLP a complicated CFD can be the following: t[Journal]=PVLDB., t[Volume]=2, t[Numbervol]=1 → t[Year]=2009. All the CFDs we use in the experiments are listed in the Appendix A.1.

Intuitively, CFDs with a less number of attributes on the premise are easier to solve *w.r.t.* CFDs with a higher number of attributes on the left because errors can be in multiple attributes.

For each dataset (excluding FLIGHT), we use BART to inject errors accordingly to the declared CFDs. Starting from the clean dataset, we generate two scenarios in which we inject detectable errors with different percentages:

- approximately 75% of the errors on the RHS of every CFD using random values (typo, random value, value from active domain) and the remaining on the LHS using values from active domain;
- approximately 50% of the errors on the RHS using random values, and the remaining on the LHS using values from the active domain.

Tables 11 and 12 report the detailed number of errors for each CFDs and for each scenario. We call the first scenario *easy scenario* since errors are injected in the majority on the right. We call the second scenario *hard scenario* since the errors are injected with the same proportion on the right and on the left. We detail the injected errors in Table 13. We list the number of RHS errors by type: i.e. values from active domain, insert of a typo or a random value. We also report the number of tuples with errors and the tuples with a single error, two errors or more than two errors.

	BUS			DBLP			Synth		
CFD	LHS	RHS	Tot	LHS	RHS	Tot	LHS	RHS	Tot
cfd_1	34	137	171	17	61	78	25	75	100
cfd_2	27	130	157	14	46	60	25	75	100
cfd_3	15	61	76	15	69	84	14	40	54
cfd_4	23	67	90	10	49	59	10	30	40
cfd_5	26	68	93	4	9	13	7	20	27
cfd_6	15	41	56	6	14	20	3	10	13
Tot	140	504	644	66	258	324	84	250	334

Table 11. Errors for the easy scenario. Approximately 75% of the errors are injected on the right, and 25% of the errors on the left

The first measure that we want to consider is the quality of the repaired instances. We measure the quality of the repaired instances using the F-Measure score of the classifications predicted by the classifier. We can measure the F-Measure since for each dataset we have the dirty and the clean version. We compute the F-Measure of the repaired cells with errors *w.r.t.* the same cells of the clean version. Clearly, the higher the F-Measure, the higher the quality of the repaired instances. We do not measure the similarity [26] of the repaired instance *w.r.t.* the clean version. We leave it for future work.

Another measure is related to the number of user interactions. Since our goal is to reduce the user effort for the cleaning process, we aim to reduce the number of user interactions. We measured the number of interactions in terms of user updates. For taking into account the number of user updates and the number of errors in a dataset,

	BUS			DBLP			Synth		
CFD	LHS	RHS	Tot	LHS	RHS	Tot	LHS	RHS	Tot
cfd_1	86	86	172	64	61	125	50	50	100
cfd_2	82	82	164	44	46	90	50	50	100
cfd_3	24	27	51	114	118	232	40	40	80
cfd_4	35	34	69	92	98	190	30	30	60
cfd_5	35	35	70	12	12	24	20	20	40
cfd_6	22	24	46	28	29	57	10	10	20
Tot	284	288	572	354	364	718	200	200	400

Table 12. Errors for the hard scenario. Errors on the left and on the right are injected approximately with the same proportion.

Datasets	RHS Domain	RHS Typo	RHS Random	LHS Domain	#Tuples With Errors	#Attrs With Errors	#Tuples Errors = 1	#Tuples Errors = 2	#Tuples Errors > 2
BUS – Easy	86	294	124	139	643	6	564	54	25
BUS – Hard	41	164	84	283	548	6	432	79	37
DBLP – Easy	83	99	70	66	318	6	185	96	37
DBLP – Hard	101	141	118	358	718	6	215	422	81
Synth – Easy	50	143	57	84	334	10	200	94	40
Synth – Hard	52	103	45	200	400	10	200	140	60

Table 13. Details of the injected errors. For each dataset, we report the number or errors introduced in the RHS depending on the type (active Domain, typo or Random), the number of LHS errors, the number of tuples with errors, the number of attributes with errors and the number of tuples with a single error, two errors and more than two errors.

we need to think about the type of errors. If an error needs an RHS repair, automatically solving it may generate a *benefit w.r.t.* manually solving it. Instead, if an error needs an LHS repair, the user will manually inspect it anyway (during the cleaning process or at the end for changing the placeholder value). Therefore, we cannot consider these interactions in the calculus of the benefit.

For example, consider two instances $\mathcal{I}_1$ and $\mathcal{I}_2$ and a CFD c. $\mathcal{I}_1 \not\models c$ and $\mathcal{I}_2 \not\models c$. Let us say, both instances contain 100 tuples in violation *w.r.t.* c. For $\mathcal{I}_1$ we use a classifier c_1 and for $\mathcal{I}_2$ a classifier c_2. It turns out that at end of the cleaning process, the user submitted 60 updates for cleaning $\mathcal{I}_1$ and 80 updates for $\mathcal{I}_2$. Considering only these numbers it looks that c_1 has a higher benefit *w.r.t.* c_2 because it cleaned the same number of errors with a less number of interactions. It turns out that $\mathcal{I}_1$ had 10 errors on the left and 90 errors on the right, and $\mathcal{I}_2$ had 75 errors on the left and 25 errors on the right. Considering Algorithm 1, for $\mathcal{I}_1$ the maximum benefit, in terms of interactions, it consists in inferring all the 90 errors on the right (since the LHS are manually repaired by the user anyway). Similarly, for $\mathcal{I}_2$, the maximum benefit for the user is to infer the 25 errors on the right automatically. It is clear that for $\mathcal{I}_1$ the user submitted 10 updates for LHS repairs, and 50 updates for RHS repairs and c_1 classifies the remaining 40 repairs on the right. For $\mathcal{I}_2$, the user submitted 75 updates to the left, and only 5 to the right, and c_2, automatically infers the remaining 25 errors on the right. Calculating the benefit without considering the type of repairs, could penalize c_2 only because in the instance there are too many LHS errors. Instead, c_2 with only 5 examples was able to correctly classify the other 25 tuples, on the contrary, c_1 used 50 examples to classify the other 40 tuples. If we normalize the numbers *w.r.t.* the potentially automatic repairs that in one instance we can infer we have that, c_1 has a benefit of $1 - (50/90) \approx 0.44$, and c_2 has a benefit of $1 - (5/30) \approx 0.83$. With c_1 the user saved about 0.44% of interactions *w.r.t.* manually clean the instance, and with c_2 the user saved about 0.83% of interactions.

In light of this, we measure the benefit as $b = 1 - (rhsI/rhsR)$, where $rhsI$ is the number of user updates related to the RHS errors and $rhsR$ is the number of all the errors on the right.

Notice that HoloClean and Dummy have 1 as benefit, since they do not require any user interactions.

Finally, we define the *effectiveness* of an algorithm as the F-Measure of the quality and the benefit. Using the effectiveness we can compare different strategies considering both quality and benefit.

The last metric that we use is the time spent by the system in the learning step and in the selection of the next tuple to resolve. This gives an estimate of the responsiveness of the system.

Experiments. We conduct five experiments.

(1) Exp-1 compares Naïve Bayes with the other classifiers: logistic regression, GDR, HoloClean, and Dummy. It shows that Naïve Bayes is the one with high quality and low user effort for all the datasets.
(2) Exp-2 studies the impact of the initial sample size in the training step before starting to predict (the parameter m in Algorithm 1). We show that Naïve Bayes works well even with a small m and increasing m does not have any huge impact on the quality of the solution generated.
(3) Exp-3 studies the impact of numerical values and categorical values in the same instance. We show the robustness of Naïve Bayes even in a setting where algorithms like LR are supposed to have good quality in scenarios with continuous variables.
(4) Exp-4 studies the time spent for the training, for the selection of the next tuple to classify, and for the classification. Naïve Bayes is the one that is able to interact with the user in milliseconds.
(5) Exp-5 compares Bunni with existing interactive data repair solutions in an end-to-end experiment with a given set of CFDs. We show how Bunni represents the right trade-off between achieving high quality in the solution and reducing the number of user interactions.

Exp-1: Classifier comparison. We compare the Naïve Bayes classifier with Logistic regression (LR), GDR, HoloClean (HC) and Dummy. We use the strategy illustrated in Algorithm 1 with different thresholds for accepting the RHS repair (p_{clean}) and the LHS repairs (p_{dirty}). For each scenario and each dataset, we measure the quality of the classification, the benefit, and finally the effectiveness of the four machine learning approaches and the Dummy classifier.

For all the experiments the thresholds are in pairs, for example, 0.4-0.6. This means that the threshold p_{dirty} is 0.4 and the threshold p_{clean} is 0.6. This means that if all the cells in the premise are clean with a probability greater than 0.6 then an RHS repair is enforced over the data. Instead, if one of the cells in the premise is clean with a probability less than 0.4 (or it is dirty with a probability $1 - 0.4 = 0.6$), then an LHS repair is suggested to the user. For the remaining case, the user manually repairs the tuple since the algorithm is not able to classify within a certain confidence.

Thresholds vary from low margins of confidence, in which the classifier has more chances to predict and a small region of indecision (like 0.4-0.6), to thresholds with higher margins of confidence, where the classifier has fewer chances to predict but high confidence in the predictions, with a big region of indecision (like 0.05-0.95). Intuitively thresholds with low values of confidence are the ones in which the benefit can be high, and thresholds with high values of confidence are the ones in which the classifier will be more accurate.

For each execution, we use a fixed parameter $m = 3$ that corresponds to the number of minimum examples asked to the user before starting to predict. HC and Dummy are both an exception because they do not require user interactions, so for them, the number of minimum examples is $m = 0$.

Experimental results are shown in Figure 3. Charts *a-c* represent respectively the quality, the benefit and finally, the effectiveness of the algorithms *w.r.t.* the BUS dataset with the easy scenario. Others rows in the figure represent the same information *w.r.t.* other datasets and scenarios. To ease the representation, we do not report the benefit for HC and Dummy, since it is always 1.

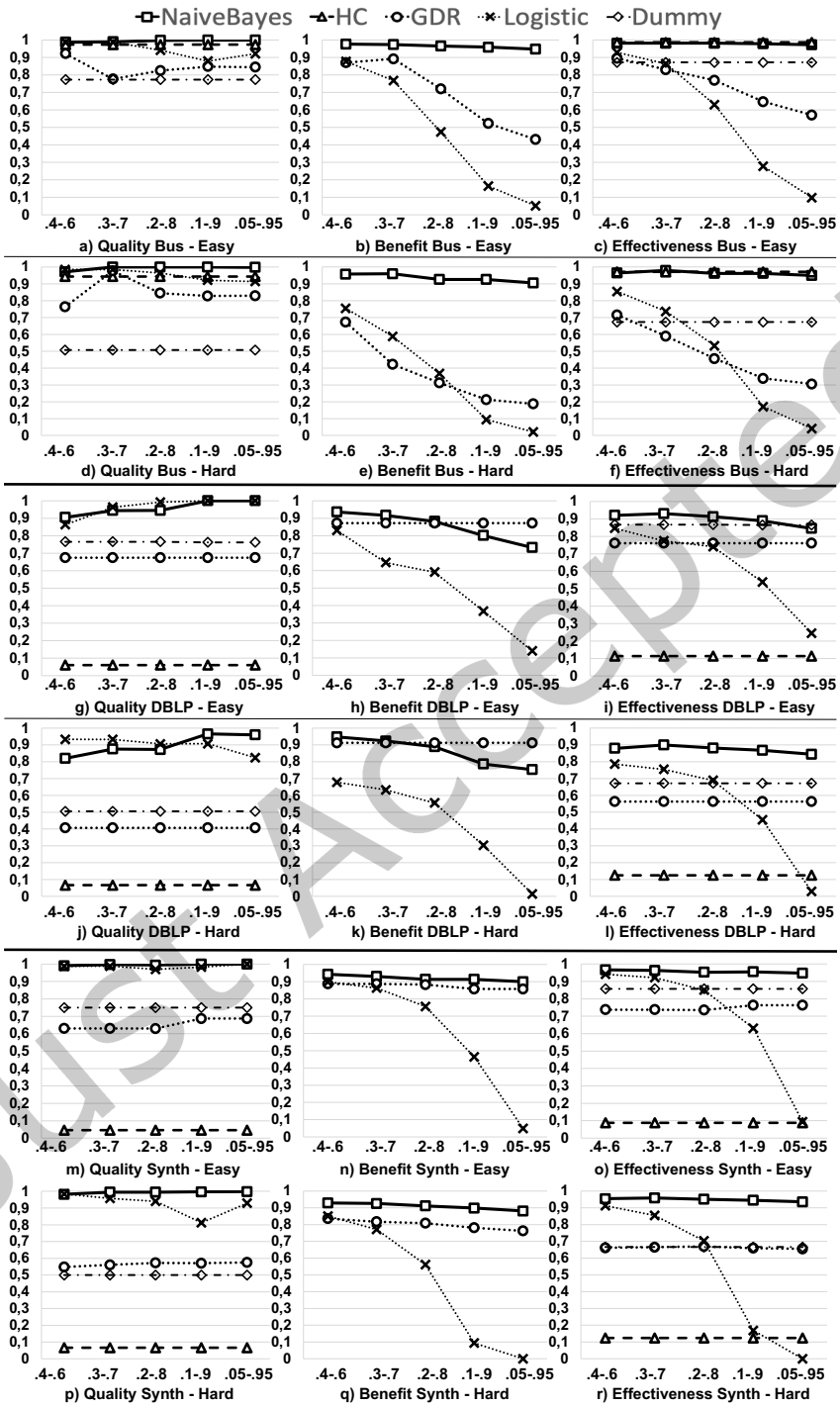

Fig. 3. Exp-1 results. Each row represents the result in terms of Quality, Benefit and Effectiveness for each dataset and scenario while varying the threshold values (p_{dirty}, p_{clean}) for accepting RHS and LHS repairs.

As a first baseline, we discuss Dummy classifier. It has always constant quality *w.r.t.* the thresholds, and this quality is related to the percentage of RHS errors introduced in each scenario. Since the benefit is always 1, also the effectiveness is constant. Its effectiveness is always higher that 85% in the easy scenarios and close to 67% in the hard scenarios.

The second baseline is HC. HC outperforms Dummy Classifier only with the BUS dataset, where it reaches effectiveness close to 1. This is due to the high quality. Unfortunately, with the DBLP and Synth dataset HC has very low quality. This is due to the fact that when it repairs the LHS, it changes also the value of the RHS and also some other values in the tuple. This is due to the fact that the model overfits the data. Indeed, HC is the only algorithm trained over all the instances, other algorithms are trained only on a set of manually repaired tuples. We also notice that most of the errors are in the easy CFDs, i.e. the ones with a low number of attributes on the LHS. A final note on HC is that we cannot run it on the DBLP scenarios with cfd_5 and cfd_6, i.e. the hardest CFDs, due to memory errors.

The explanation for the limited performance of GDR is related to the low quality of the predictions. It tends to predict an RHS repair, even in cases in which cells on the LSH should not be trusted. We observe that this is connected to the active learning strategy based on the ensemble of the random forest. The decision tree does not calculate the probability that some data belong to a certain class, but it calculates the percentage of misclassification for each leaf. Moreover, multivalued attributes, like string attributes, can bias the decision trees, since the measure used for the learning (the information gain) gives a wrong estimation of the usefulness of multivalued attributes. So, the probability calculated by the random forest is just a mean of votes. The classifier, in most of the cases, asked the user to manually solve only m sample tuples (with $m = 3$), and these were all RHS repairs. In this way, the random forest is biased toward predicting only RHS repairs. Varying the thresholds does not impact the results in a significant way.

LR works well with lower confidence thresholds when, despite the predicted probability of the outcome being lower, predictions are always correct. The problem with LR is that it requires more training data to have good results. This leads to a low benefit. Another problem is related to the nature of the data. LR works well when the number of values related to each attribute is not big, and attributes have ordinal values – e.g., numbers. This is critical in fitting the regression parameters. On the contrary, most of our attributes are categorical. In Exp-3 we analyze LR when the instance contains both numerical and string values and note how the classification is more accurate. Moreover, as shown in a comparison between generative and discriminative classifiers [42], LR tends to be optimal with a high number of training examples (see Exp-2).

Experimental results for Naïve Bayes clearly shows how, by increasing the confidence thresholds, *i.e.,* asking for more confident predictions, it effectively generates repairs of increasing quality, and, conversely, it reduces the amount of user interactions when lowering the thresholds. With all datasets and scenarios, Naïve Bayes demonstrates the highest effectiveness, a good compromise between high quality a lower human effort.

Exp-2: Training sample impact. Another interesting experiment is related to the number of initial examples (m) in the training step. We use two CFDs on DBLP in the *hard scenario*, namely cfd_1 and cfd_4, and a fixed threshold (0.1-0.9). We use five fixed values for m and we measure for each m five indicators: a) the quality; b) the benefit; c) the numbers of updates corresponding to the RHS (**RU**); d) the number of update(s) corresponding to the LHS (**LU**); e) the quality (**QLR**) calculated only on the predictions related to the side of the repair (RHS or LHS repair). Since HC is trained on the whole instance, we exclude HC from this experiment.

Note that the quality that we defined for Exp-1 is a quality that considers both the quality of the solution and the quality of the predictions made by the classifier. The more accurate the predictions, the less the user effort in finding where are the errors. Instead, QLR represents only the quality of the solution.

For example, consider two attributes A and B related to the premise of a CFD. Let us say that the classifier predicts for a violated tuple $A^{\blacksquare}$ with a probability of 0.95, and $B^{\blacksquare}$ with a probability of 0.01. Since, for example,

		cfd_1					cfd_4				
	m	**quality**	**benefit**	**RU**	**LU**	**QLR**	**quality**	**benefit**	**RU**	**LU**	**QLR**
Naïve Bayes	1	0.9672	2	64	1	1	0.9381	0.4898	50	92	1
	10	1	0.9672	2	64	1	0.9381	0.4898	50	92	1
	20	1	0.9508	3	64	1	0.9381	0.4898	50	92	1
	50	1	0.8361	10	64	1	0.9381	0.4898	50	92	1
	100	1	0.4098	36	64	1	0.9609	0.4898	50	92	1
Logistic	5	1	0.1475	52	64	1	0.7397	0	98	92	1
	10	1	0.1311	53	64	1	0.75	0	98	92	1
	20	1	0.1967	49	64	1	0.7241	0	98	92	1
	50	1	0.2623	45	64	1	0.7636	0	98	92	1
	100	1*	0	61	64	1*	0.7576	0	98	92	1
GDR	5	0.4667	0.918	5	0	0.4667	0.3633	0.949	5	0	0.5027
	10	0.4435	0.8361	10	0	0.4435	0.3506	0.898	10	0	0.4889
	20	0.3905	0.6721	20	0	0.3905	0.3237	0.7959	20	0	0.4588
	50	0.1467	0.1803	50	0	0.1467	0.2275	0.4898	50	0	0.3429
	100	1*	0	61	64	1*	1	0.1327	85	92	1

Table 14. cfd_1 and cfd_4 on DBLP with the *hard scenario* and threshold 0.1-0.9

$\tau_{dirty} = 0.90$, LHSContainsErrors returns *true*, highlighting the cell that corresponds to A as the one with the error (since B looks to be clean). Since it is an LHS error, LHSRepair asks the user to submit the correct value for A, but on the contrary, she submits an update for B since she discovers that the error is in B instead of A. So even if there was an error in the classification, noise is not introduced. QLR represents the quality of the predictions, i.e., *use an RHS repair* and *use an LHS repair*, in terms of their impact on the quality of the final solution, since noise can be introduced only by RHS repairs. Indeed RHS repairs are applied automatically by the system, so in the case of an error in the prediction (predict an RHS repair instead of an LHS repair) additional errors can indeed be introduced into the instance. Conversely, predicting an LHS repair instead of an RHS repair cannot introduce new errors, since the user has the chance to overview the change and submit the correct value or values. Results are reported in Table 14 for both cfd_1 and cfd_4.

For cfd_1 there are 125 errors, 61 on the right and 64 on the left. First, we need to note that this is a rule with one attribute on the left and one on the right, so QLR and quality are the same. As we explained in the previous experiment, the problem with GDR is that it often selects tuples with errors on the right for the training. Of course, if the classifier is trained only with the same kind of errors (RHS) it never predicts to repair the LHS. Only with a training size larger than 100, GDR starts to predict LHS repairs. We need to note, that for $m = 100$ GDR is always in the indecision area. Instead for $m < 100$ it always predicts repair on the right. This behavior explains the low quality and the high benefit for m from 5 to 50. Instead for $m = 100$ the benefit is 0 because the user manually cleans all the tuples, but the quality (and QLR) is 1.0. That is the reason for the * near 1.0. For easy CFDs like cfd_1, the quality of LR is 1.0. Instead, the benefit increases *w.r.t.* the number of examples in the training step (except for $m = 100$). This confirms that LR is a good classifier but needs more training data in order to have high accuracy in the classification. Finally Naïve Bayes is the one that is robust with a low number of training examples in the training step. It always has high quality (1) and high benefit, particularly for small m.

For cfd_4 there are 92 errors on the left and 98 errors on the right. In total there are 190 errors. In this case, where the number of attributes in the premise is more than one (for cfd_4 2 attributes), quality is different from QLR, i.e., QLR is higher. A QLR of 1 means that the user cleaned all the violations for the CFD removing all the errors in the right way. GDR only predicts right repairs (RHS) for $m = 5$ to $m = 50$. The quality in the classification

is not very high, always less than 0.4. QLR is also quite low. The benefit is high. Instead with $m = 100$, both quality and QLR are 1.0 with a low benefit. LR has lower quality than in the previous case. The higher the number of training examples, the higher the quality. Even if the quality is never higher than 0.76, the QLR is one. It means that LR correctly predicts to repair on the right or on the left, but when it predicts to repair on the left sometimes suggests repairing the wrong attribute. The benefit is always 0. Naïve Bayes is the one with quality always over 0.9 and QLR always 1. Moreover, the benefit is close to 0.50 and is not changed by varying the number of training examples. Of course in this case it is convenient to have a lower number of m because the quality increase only of the 0.03% with $m = 100$. In fact, a clear advantage of Naïve Bayes is that it works well with a small number of training tuples [42].

Exp-3: Classification with numerical values. From Exp-1 we noted that LR and GDR have low quality when attributes contain string values. In this experiment, we want to explore the impact of predicting errors when the attributes are both numerical and categorical (strings).

For this purpose, we use the TAX dataset. For this dataset we declare two CFDs:

- cfd_1: t[State]=KY, t[Salary]=30000 → t[Rate]=3.5
- cfd_2: t[State]=KS, t[Salary]=10000 → t[Rate]=3.5

We inject detectable errors accordingly to the two CFDs defined above using BART [2]. We generate two scenarios using the same percentages as the previous experiments. For the *easy scenario* we inject 120 errors on the conclusion and 24 for the premise of cfd_1 and 120 and 40 errors respectively for the conclusion and the premise of cfd_2. Instead for the *hard scenario*, we inject 120 errors for the conclusion and the premise for both CFDs. We use the same metrics as in Exp-1.

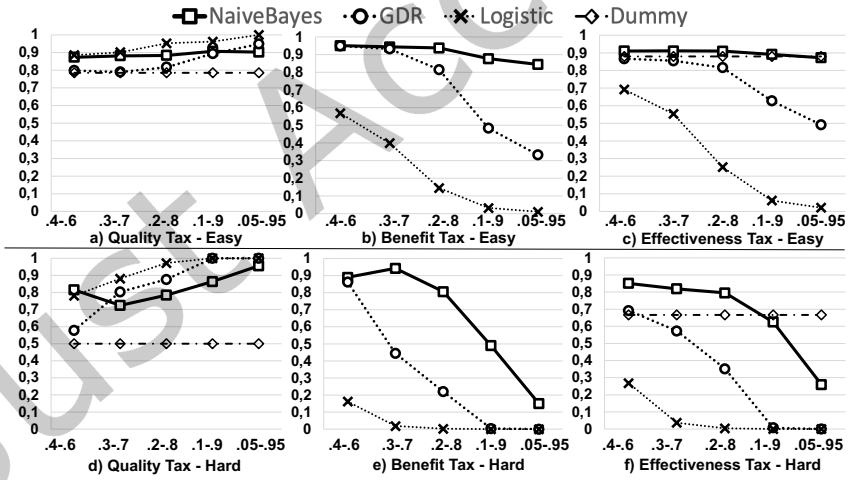

Fig. 4. Exp-3. TAX dataset. Each row contains Quality, Benefit and Effectiveness charts for each scenario.

Figure 4 reports the results for the TAX dataset for both the easy and hard scenarios. Dummy is again constant *w.r.t.* the thresholds. In the easy scenario, the effectiveness is high, while in the hard scenario, the effectiveness is close to 70%. As expected, GDR increases its quality *w.r.t.* the thresholds. This is due to the nature of the data, that is more suitable for decision trees. Although the good performances in quality, GDR requires more user interactions and thus the benefit decreases *w.r.t.* the thresholds. For high confidence thresholds, it is always in the

indecision area, and thus the instance is completely cleaned by the user. This explains the lower effectiveness in the hard scenario. LR has high quality even with low thresholds. Increasing the thresholds increases the quality, but this increases also the user interactions and the benefit rapidly decreases. This effect can be noticed in the hard scenario where the algorithm is always in the indecision area and thus the benefit is lower. Finally, NB has the highest effectiveness for all the thresholds. Only in the hard scenario, it shows lower effectiveness for the quality threshold 0.05-0.95.

Exp-3 essentially tells us that in the case of numerical variables LR and GDR are able to reach high-quality results, but asking for more accurate prediction has a high cost in terms of user interactions. On the contrary, in the case of numerical values Naïve Bayes still has good performance but has lower benefit.

	Dataset	Training	PickOne	Prediction	Total	Avg
Naïve Bayes	BUS	140	2587	51	**2778**	**9.16**
	DBLP	551	4070	49	4670	**9.71**
	Synth	135	675	23	**833**	**3.75**
GDR	BUS	18445	3248	66	21759	42.5
	DBLP	93	4500	44	**4637**	257.61
	Synth	671	871	27	1569	13.89
Logistic	Bus	600045	4238	80	604363	1036
	DBLP	13381180	59503	660	13441343	18825
	Synth	70626	1212	34	71872	179.7
HoloClean	Bus	5524390	-	42650	5567040	4526.05
	DBLP	68707330	-	3152400	71859730	16316.92
	Synth	7528620	-	18620	7547240	2448.02

Table 15. Time in *ms* for the hard scenario with threshold 0.05-0.95

Exp-4: Time performance. Our goal is to develop an interactive system, so the time needed for the training step and to generate predictions needs to be as low as possible. Since times are essentially the same for the different thresholds, in Table 15 we only report the **total time** needed to clean the entire dataset for a fixed threshold. We also report the average time of each user interaction. Such times do not incorporate the user's thinking, i.e., the time used by the user to detect a cell with an error and provide the clean value. Such time could depend on the knowledge of the user about the domain but also could depend on the User Interface provided to explore the data and provide feedback. In this experiment, we do not focus on the usability of the interface. The best times are in bold.

Naïve Bayes is the faster one for all the datasets: the average prediction time is always less than 10 ms. The total time is usually the lowest one (for DBLP is slower only of 33 *ms w.r.t.* GDR).

Training time in GDR is higher because it trains a forest of Decision Trees (more precisely 10 decision trees) on different subsets of attributes in the table that are randomly selected.

Logistic Regression has very high times because of the problem of handling categorical values, especially strings. For string attributed, it is common to use the *dummy encoding* [24]. This consists in generating a dichotomous (boolean) variable for each value of each string attribute. This significantly increases the number of variables needed to represent the data – one variable for each possible value of a string attribute – and therefore increases the time needed to train the classifier. In our experiments, the number of dichotomous variables is always in the order of a few thousand.

HoloClean has the highest time because of its training time. In the training phase, it first generates the features to model the probabilistic reasoner and then it trains the model with such generated features. We note that HoloClean is not an interactive system, and thus the repair step is executed only once.

On the contrary, Naïve Bayes only needs to estimate the joint probability from the data, and the estimation consists in *counting* value occurrences, which is less computationally expensive. This makes Naïve Bayes suitable

for an interactive system since the overall response to one interaction is less than one second.

Exp-5: Interactive System Evaluation. We evaluate Bunni against HoloClean (HC) [47], Raha and Baran, and LLunatic [25]. Raha [39] is an automatic error detection system that uses user updates to find cells with errors, while Baran [38] is an automatic data repair system that uses user updates to infer cell repairs. We combine both systems (Raha + Baran) to infer errors in the cells and repair such cells using user updates. LLunatic is a rule-based data-cleaning system that can be configured with different strategies. To reduce the number of user updates, we configured LLunatic with a similarity-based strategy to repair the data. It applies an RHS repair if the value in the conclusion is similar to the one constrained by the CFD (using the Levenshtein distance), otherwise, it applies a LHS repair. In case of multiple possible values in the LHS, it makes use of placeholders (LLUNS [25]) that are interactively updated by the user.
We use the same real-world dataset that contains real errors used in Baran [38], i.e, the Flight dataset. In addition, we use our generated datasets. For HoloClean, LLunatic and Bunni, we use the same CFDs to detect errors. We mine CFDs for Flight using Metanome [43] with the CFDFinder discovering algorithm [17, 18]. We measure the quality (Q), benefit (B), effectiveness (E), and total execution time (T) in seconds. For Raha and Baran we use different budgets, i.e., number of user updates, and we report the best results in terms of Effectiveness. We report the results in Table 16. For Raha and Baran, we had memory errors for BUS and DBLP and we do not report such results. We also report the average of all the metrics over all datasets. In bold the best results for each metric and the lowest total time.

For Raha and Baran, datasets with a large number of columns had out-of-memory errors on our machine. This is due to the number of clustering algorithms trained for each column. Another issue is the initialization of the strategies for error detection, which takes considerable time for an interactive system; however, such initialization is executed once and is skipped in the other executions. HoloClean does not require any user update, so the benefit is always 1.00, but the quality for DBLP and Synth is low, and thus the effectiveness is also low. Moreover, like Raha and Baran, also HoloClean takes considerable time to repair the dataset. LLunatic can reach a good quality without requiring too much user effort, and it also requires a reasonable time for an interactive system (ignoring Flights dataset). Finally, Bunni represents the good trade-off when we require high-quality results (on avg. we have 0.99), while it also reduces the user effort (benefit on avg. 0.63). The effectiveness of Bunni is the highest in this experiment, while the avg. time is the lowest, proving that is suitable for an interactive system.

	Raha + Baran				HoloClean				LLunatic				Bunni			
dataset	**Q**	**B**	**E**	**T(s)**	**Q**	**B**	**E**	**T(s)**	**Q**	**B**	**E**	**T(s)**	**Q**	**B**	**E**	**T(s)**
BUS – Easy	-	-	-	-	0.97	**1.00**	**0.98**	1k	0.65	0.69	0.67	12	**1.00**	0.75	0.86	**7**
BUS – Hard	-	-	-	-	0.94	**1.00**	**0.97**	1k	0.71	0.39	0.51	29	**1.00**	0.47	0.64	**6**
DBLP – Easy	-	-	-	-	0.06	**1.00**	0.11	1.5k	0.46	0.81	0.59	18	**1.00**	0.55	**0.71**	**3.6**
DBLP – Hard	-	-	-	-	0.07	**1.00**	0.13	1.5k	0.36	0.62	0.46	18	**0.93**	0.33	**0.49**	**12.7**
Synth – Easy	0.48	0.67	0.56	15k	0.05	**1.00**	0.09	415	0.83	0.8	0.81	18	**1.00**	0.78	**0.87**	**3**
Synth – Hard	0.53	0.67	0.59	16k	0.07	**1.00**	0.13	521	0.82	0.6	0.7	18	**1.00**	0.61	**0.76**	**3**
Flights	0.66	1.00	0.79	6k	0.76	**1.00**	0.86	70.6	**1.00**	**1.00**	**1.00**	1523	**1.00**	0.92	0.96	**4**
AVG	0.55	0.77	0.65	12.3k	0.42	**1.00**	0.47	858	0.69	0.70	0.67	233	**0.99**	0.63	**0.76**	**5.6**

Table 16. For each system and dataset, we report the (Q)uality, (B)enefit, (E)ffectiveness and total (T)ime in seconds. Cells without the number correspond to memory errors. We also report the average of the measures over the dataset. In bold we report the highest metric for each dataset, and the lowest total time execution. Bunni shows a good trade-off between high quality and user effort as reported by the average of the Effectiveness.

8 RELATED WORK

Rule-based data cleaning systems. In the rule-based approach, the data cleaning process consists in algorithms and systems that apply the rules over dirty data to automatically fix the detected errors [5, 8, 11, 15, 19, 21, 23, 25, 31, 34, 54, 58]. We restrict our attention only on the strategies that are able to manage constant CFDs. The problem of repairing the CFDs is NP-COMPLETE and for solving it there are different heuristics have been proposed (cardinality minimal, cost minimal) [10]. Most of the strategies are based on the definition of a cost function $cost()$ that measures the difference in cost between two solutions. It could happen that these strategies based on minimality do not have high quality. Indeed when it comes that we need to repair using both RHS and LHS repair strategies, multiple solutions can arise. To generate a unique solution the heuristic adopted works very well when a semantically appropriate distance function is declared for the dirty instance or when the assumption that the minimal solution is the preferred one is not broken. Instead, Bunni automatically decides where to repair without considering minimal solutions, and the values used for repairing are user-defined or come from the conclusion in the case of RHS repairs. Another approach that guarantees high quality is based on *master data* [22]. The assumption is based on the availability of some cleaned data that can used for repairing. In the dirty dataset, some *fixed regions* are assumed to be clean. Using *fixing rules*, if the LHS part of the given rule matches the fixed region, master data and fixing rules are applied to RHS repair the data. We do not compare this approach in our study since we do not have master data nor fixed regions in the datasets. On the contrary, with Bunni we aim to infer the cells that are clean, so in a way we try recognize fixed region.

Probabilistic data cleaning systems. ERACER [40] is a system for data cleaning that uses statistical inference. It is based on the concept of *Relational Dependency Network* (RDN) [41]. Using RDN, ERACER is able to find outliers and removes them accordingly to statistical inference, or predict possible values for missing data. Although this model works very well in prediction for numerical values where it is possible to use convolution shrinkage or regression model. In our case is hard to generalize it for handling categorical values such as strings. BayesWipe [12] [13] uses Bayesian Networks [44]. The approach differs from our approach with Bunni for two reasons: 1) we rely on rules for cleaning the data, instead BayesWipe works without any quality rules; 2) we use the user updates for cleaning the data, instead BayesWipe uses the values in the instance for cleaning the data. SCARE [57] is another system that does not rely on rule declaration. Instead, it uses machine learning techniques for repairing the data. HoloClean [47, 56] is a statistical inference system. Differently from the previous systems, it uses declared rules in the form of Denial Constraints [9]. It uses the quality rules, with value correlations, and the data itself to build a new set of features to model the data generation process and thus infer the correct cell values. Instead, Bunni only predicts the side of the repair. We need to note that the main difference between Bunni and most of these probabilistic data cleaning systems is that with Bunni we take into account that violations arise *w.r.t.* some declared rules. The probabilistic reasoning and the inference system, are used both only to determine where to repair. Instead in the other systems (with the exception of HoloClean), rules are not declared, so the violations come only as a probability of dirtiness, because there are, for example, some outliers *w.r.t.* to the distribution of the data, or there are different patterns in similar data. Note that in the case a dataset contains a high number of errors, previous systems fail in the detection of the errors, (if there are a lot of errors, the anomalies cannot be detected, they represent a big portion of the data, so they are the *normality*, not the *anomaly*) and thus failing in the generation of a cleaned instance. Indeed, correct values are mined from the instance, but it could be the case that the current instance does not contain the correct value for a specific repair.

Interactive Data Cleaning. GDR [58] is an interactive system that uses human interactions for cleaning the data. The system is bootstrapped by a set of declared CFDs. For each violated tuple the system suggests possible repairs using the technique based on minimal changes [10]. GDR is comparable with Bunni for the active learning approach, they differ only in the function used for ranking the most promising tuple to classify. In

GDR the function considers the response of a committee of classifiers and predicts the most uncertain. Instead in Bunni we measure the uncertainty with the probability of each outcome. Raha [39] and Baran [38] are two configuration-free error detection and correction systems, respectively. They generate an initial set of potential data errors and corrections, then ensemble such errors/corrections to iteratively ask the user to annotate a tuple with potential errors/corrections. Such annotation is used to train a model to generalize such errors/corrections.

Crowd-based repairing. CrowdAidRepair is based on the crowd [59]. CrowdAidRepair uses some predefined rules (FDs and CFDs) with the crowd to achieve high-quality results. It alternates rule based-repairs with crowd-based repairs. CrowdAidRepair is similar to our system since it relies on user interaction like submitting a value or validating or rejecting a value. But we need to note that there exist some differences. First, CrowdAidRepair is a holistic system, instead Bunni is a one-at-time system (since it solves a rule per time). CrowdAidRepair uses some predefined rules and the crowd for selecting the best values to use in the rule-based repair, instead Bunni uses the user interactions to detect the cells that contain errors and uses user-submitted values for the LHS repair step.

Machine learning for cleaning. Given a set of user updates, they can be used as training data to train machine learning models, which in turn can be used to predict other repairs [53, 57]. However, ML models are typically *black-boxes* that identify updates without explanations, which are hard to be trusted by the users, especially for critical applications that need repairs with guaranteed correctness. Moreover, to train a machine learning model with updates, they must be semantically consistent, *i.e.,* they refer to the same type of errors. In practice, however, this assumption does not always hold since multiple updates may refer to different types of errors. This heterogeneity may hinder the usability of the trained machine-learning model for prediction. Different from them, Bunni trains a classifier for each CFD, this helps the classification because the model is specialized only for a specific CFD and the user has a context for trusting the suggested repairs. ActiveClean [36] is a framework for repairing data with the goal of using such repaired data for training a ML model, so the goal is to achieve high quality in the prediction of the labels, while the goal of Bunni is to achieve high quality in the cleaning data. ActiveClean uses a sample of manually cleaned data to infer repairs in the training set. It treats the cleaning and training iteration as an optimization problem using stochastic gradient descent.

9 CONCLUSIONS

We have presented Bunni, an interactive system for repair discovery. We have demonstrated how the system is able to interact with users in real-time to generate a solution with a user-defined quality. We used a well-known machine learning technique, namely Naïve Bayes, to solve the repair-discovery problem while minimizing the number of user interactions and computing time. We showed how the Naïve Bayes method outperforms other approaches in terms of quality, benefit, effectiveness, and execution times. An aspect to investigate is the integration of Bunni with Language Models (LMs) such as BERT [14] and T5 [45], or even tabular LMs [3], to automatically suggest the values for the LHS repair.

REFERENCES

[1] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, Pennsylvania, USA*. ACM Press, 68–79. https://doi.org/10.1145/303976.303983

[2] 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. *Proc. VLDB Endow.* 9, 2 (2015), 36–47. https://doi.org/10.14778/2850578.2850579

[3] Gilbert Badaro, Mohammed Saeed, and Paolo Papotti. 2023. Transformers for Tabular Data Representation: A Survey of Models and Applications. *Transactions of the Association for Computational Linguistics* 11 (03 2023), 227–249. https://doi.org/10.1162/tacl_a_00544 arXiv:https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00544/2074873/tacl_a_00544.pdf

[4] Christopher M. Bishop. 2007. *Pattern recognition and machine learning, 5th Edition*. Springer. https://www.worldcat.org/oclc/71008143
[5] Philip Bohannon, Michael Flaster, Wenfei Fan, and Rajeev Rastogi. 2005. A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*. ACM, 143–154. https://doi.org/10.1145/1066157.1066175
[6] Jean-Flavien Bussotti, Enzo Veltri, Donatello Santoro, and Paolo Papotti. 2023. Generation of Training Examples for Tabular Natural Language Inference. *Proc. ACM Manag. Data* 1, 4, Article 243 (dec 2023), 27 pages. https://doi.org/10.1145/3626730
[7] Fei Chiang and Renée J. Miller. 2008. Discovering data quality rules. *Proc. VLDB Endow.* 1, 1 (2008), 1166–1177. https://doi.org/10.14778/1453856.1453980
[8] Fei Chiang and Renée J. Miller. 2011. A unified model for data and constraint repair. In *ICDE*. IEEE Computer Society, 446–457. https://doi.org/10.1109/ICDE.2011.5767833
[9] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Discovering Denial Constraints. *Proc. VLDB Endow.* 6, 13 (2013), 1498–1509. https://doi.org/10.14778/2536258.2536262
[10] Gao Cong, Wenfei Fan, Floris Geerts, Xibei Jia, and Shuai Ma. 2007. Improving Data Quality: Consistency and Accuracy. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*. ACM, 315–326. http://www.vldb.org/conf/2007/papers/research/p315-cong.pdf
[11] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed K. Elmagarmid, Ihab F. Ilyas, Mourad Ouzzani, and Nan Tang. 2013. NADEEF: a commodity data cleaning system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*. ACM, 541–552. https://doi.org/10.1145/2463676.2465327
[12] Sushovan De, Yuheng Hu, Yi Chen, and Subbarao Kambhampati. 2014. BayesWipe: A multimodal system for data cleaning and consistent query answering on structured bigdata. In *2014 IEEE International Conference on Big Data (IEEE BigData 2014), Washington, DC, USA, October 27-30, 2014*. IEEE Computer Society, 15–24. https://doi.org/10.1109/BigData.2014.7004207
[13] Sushovan De, Yuheng Hu, Venkata Vamsikrishna Meduri, Yi Chen, and Subbarao Kambhampati. 2016. BayesWipe: A Scalable Probabilistic Framework for Improving Data Quality. *Journal of Data and Information Quality (JDIQ)* 8, 1 (2016), 5.
[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/v1/n19-1423
[15] Amr Ebaid, Ahmed K. Elmagarmid, Ihab F. Ilyas, Mourad Ouzzani, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, and Si Yin. 2013. NADEEF: A Generalized Data Cleaning System. *PVLDB* 6, 12 (2013), 1218–1221.
[16] Wenfei Fan and Floris Geerts. 2012. *Foundations of Data Quality Management*. Synthesis Lectures on Data Management Morgan & Claypool Publishers. https://doi.org/10.2200/S00439ED1V01Y201207DTM030
[17] 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. https://doi.org/10.1145/1366102.1366103
[18] Wenfei Fan, Floris Geerts, Jianzhong Li, and Ming Xiong. 2011. Discovering Conditional Functional Dependencies. *IEEE Trans. Knowl. Data Eng.* 23, 5 (2011), 683–698. https://doi.org/10.1109/TKDE.2010.154
[19] Wenfei Fan, Floris Geerts, Nan Tang, and Wenyuan Yu. 2013. Inferring data currency and consistency for conflict resolution. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*. IEEE Computer Society, 470–481. https://doi.org/10.1109/ICDE.2013.6544848
[20] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. 2010. Towards Certain Fixes with Editing Rules and Master Data. *PVLDB* 3, 1 (2010).
[21] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. 2011. Interaction between record matching and data repairing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*. ACM, 469–480. https://doi.org/10.1145/1989323.1989373
[22] 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).
[23] Helena Galhardas, Daniela Florescu, Dennis E. Shasha, Eric Simon, and Cristian-Augustin Saita. 2001. Declarative Data Cleaning: Language, Model, and Algorithms. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*. Morgan Kaufmann, 371–380. http://www.vldb.org/conf/2001/P371.pdf
[24] Susan Garavaglia and Asha Sharma. 1998. A smart guide to dummy variables: Four applications and a macro. Proceedings of the Northeast SAS Users Group Conference, 43.
[25] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. 2020. Cleaning data with Llunatic. *VLDB Journal* 29, 4 (2020), 867 – 892. https://doi.org/10.1007/s00778-019-00586-5
[26] Boris Glavic, Giansalvatore Mecca, Renée J Miller, Paolo Papotti, Donatello Santoro, and Enzo Veltri. 2024. Similarity Measures For Incomplete Database Instances. In *Proceedings 27th International Conference on Extending Database Technology, EDBT 2024, Paestum, Italy, March 25 - March 28*. OpenProceedings.org.

[27] Lukasz Golab, Howard J. Karloff, Flip Korn, Barna Saha, and Divesh Srivastava. 2012. Discovering Conservation Rules. In *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*. IEEE Computer Society, 738–749. https://doi.org/10.1109/ICDE.2012.105

[28] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. 2009. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter* 11, 1 (2009), 10–18.

[29] Jian He, Enzo Veltri, Donatello Santoro, Guoliang Li, Giansalvatore Mecca, Paolo Papotti, and Nan Tang. 2016. Interactive and Deterministic Data Cleaning. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*. ACM, 893–907. https://doi.org/10.1145/2882903.2915242

[30] Jeffrey Heer, Joseph M. Hellerstein, and Sean Kandel. 2015. Predictive Interaction for Data Transformation. In *Seventh Biennial Conference on Innovative Data Systems Research, CIDR 2015, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2015/Papers/CIDR15_Paper27.pdf

[31] Matteo Interlandi and Nan Tang. 2015. Proof positive and negative in data cleaning. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*. IEEE Computer Society, 18–29. https://doi.org/10.1109/ICDE.2015.7113269

[32] Sijia Jiang, Zijing Tan, Jiawei Wang, Zhikang Wang, and Shuai Ma. 2023. Guided conditional functional dependency discovery. *Information Systems* 114 (2023), 102158. https://doi.org/10.1016/j.is.2022.102158

[33] Sean Kandel, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. 2011. Wrangler: interactive visual specification of data transformation scripts. In *Proceedings of the International Conference on Human Factors in Computing Systems, CHI 2011, Vancouver, BC, Canada, May 7-12, 2011*. ACM, 3363–3372. https://doi.org/10.1145/1978942.1979444

[34] Zuhair Khayyat, Ihab F. Ilyas, Alekh Jindal, Samuel Madden, Mourad Ouzzani, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, and Si Yin. 2015. BigDansing: A System for Big Data Cleansing. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*. ACM, 1215–1230. https://doi.org/10.1145/2723372.2747646

[35] Sotiris B. Kotsiantis. 2007. Supervised Machine Learning: A Review of Classification Techniques. In *Emerging Artificial Intelligence Applications in Computer Engineering - Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*. Frontiers in Artificial Intelligence and Applications, Vol. 160. IOS Press, 3–24. http://www.booksonline.iospress.nl/Content/View.aspx?piid=6950

[36] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J. Franklin, and Ken Goldberg. 2016. ActiveClean: Interactive Data Cleaning For Statistical Modeling. *Proc. VLDB Endow.* 9, 12 (2016), 948–959. https://doi.org/10.14778/2994509.2994514

[37] Paola Lapadula, Giansalvatore Mecca, Donatello Santoro, Luisa Solimando, and Enzo Veltri. 2018. Humanity is overrated. or not. Automatic diagnostic suggestions by greg, ML (Extended abstract). *Communications in Computer and Information Science* 909 (2018), 305 – 313. https://doi.org/10.1007/978-3-030-00063-9_29

[38] Mohammad Mahdavi and Ziawasch Abedjan. 2020. Baran: Effective Error Correction via a Unified Context Representation and Transfer Learning. *Proc. VLDB Endow.* 13, 11 (2020), 1948–1961. http://www.vldb.org/pvldb/vol13/p1948-mahdavi.pdf

[39] 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, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. ACM, 865–882. https://doi.org/10.1145/3299869.3324956

[40] Chris Mayfield, Jennifer Neville, and Sunil Prabhakar. 2010. ERACER: a database approach for statistical inference and data cleaning. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*. ACM, 75–86. https://doi.org/10.1145/1807167.1807178

[41] Jennifer Neville and David Jensen. 2007. Relational dependency networks. *Journal of Machine Learning Research* 8, Mar (2007), 653–692.

[42] Andrew Y Ng and Michael I Jordan. 2002. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems* 2 (2002), 841–848.

[43] 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

[44] Judea Pearl. 1986. Fusion, propagation, and structuring in belief networks. *Artificial intelligence* 29, 3 (1986), 241–288.

[45] 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.

[46] Vijayshankar Raman and Joseph M. Hellerstein. 2001. Potter's Wheel: An Interactive Data Cleaning System. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*. Morgan Kaufmann, 381–390. http://www.vldb.org/conf/2001/P381.pdf

[47] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *Proc. VLDB Endow.* 10, 11 (2017), 1190–1201. https://doi.org/10.14778/3137628.3137631

[48] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Brij B. Gupta, Xiaojiang Chen, and Xin Wang. 2021. A Survey of Deep Active Learning. *ACM Comput. Surv.* 54, 9, Article 180 (oct 2021), 40 pages. https://doi.org/10.1145/3472291

[49] Sunita Sarawagi and Anuradha Bhamidipaty. 2002. Interactive deduplication using active learning. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada*. ACM, 269–278.

https://doi.org/10.1145/775047.775087
[50] Burr Settles. 2012. *Active Learning*. Morgan & Claypool Publishers. https://doi.org/10.2200/S00429ED1V01Y201207AIM018
[51] Shaoxu Song and Lei Chen. 2013. Efficient discovery of similarity constraints for matching dependencies. *Data Knowl. Eng.* 87 (2013), 146–166. https://doi.org/10.1016/j.datak.2013.06.003
[52] Enzo Veltri, Gilbert Badaro, Mohammed Saeed, and Paolo Papotti. 2023. Data Ambiguity Profiling for the Generation of Training Examples. In *39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, CA, USA, April 3-7, 2023*. IEEE, 450–463. https://doi.org/10.1109/ICDE55515.2023.00041
[53] Maksims Volkovs, Fei Chiang, Jaroslaw Szlichta, and Renée J. Miller. 2014. Continuous data cleaning. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*. IEEE Computer Society, 244–255. https://doi.org/10.1109/ICDE.2014.6816655
[54] Jiannan Wang and Nan Tang. 2014. Towards dependable data repairing with fixing rules. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*. ACM, 457–468. https://doi.org/10.1145/2588555.2610494
[55] Bo Wu and Craig A. Knoblock. 2015. An Iterative Approach to Synthesize Data Transformation Programs. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*. AAAI Press, 1726–1732. http://ijcai.org/Abstract/15/246
[56] Richard Wu, Aoqian Zhang, Ihab F. Ilyas, and Theodoros Rekatsinas. 2020. Attention-based Learning for Missing Data Imputation in HoloClean. In *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*. mlsys.org. https://proceedings.mlsys.org/book/307.pdf
[57] 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 *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*. ACM, 553–564. https://doi.org/10.1145/2463676.2463706
[58] Mohamed Yakout, Ahmed K. Elmagarmid, Jennifer Neville, Mourad Ouzzani, and Ihab F. Ilyas. 2011. Guided data repair. *Proc. VLDB Endow.* 4, 5 (2011), 279–289. https://doi.org/10.14778/1952376.1952378
[59] Jian Zhou, Zhixu Li, Binbin Gu, Qing Xie, Jia Zhu, Xiangliang Zhang, and Guoliang Li. 2016. CrowdAidRepair: A Crowd-Aided Interactive Data Repairing Method. In *Database Systems for Advanced Applications - 21st International Conference, DASFAA 2016, Dallas, TX, USA, April 16-19, 2016, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 9642)*. Springer, 51–66. https://doi.org/10.1007/978-3-319-32025-0_4

A MORE DETAILS ON EXPERIMENTS

A.1 CFDs for the datasets

Table 17 contains all the constant CFDs used for each dataset. Each CFD has different coverage over the data. More attributes are on the left less is coverage. The first two CFDs for each scenario contain only one attribute on the left, the second two CFDs contain two attributes on the left, and finally, the last two CFDs contain more than two attributes on the left.
In the case of the CFDs from cfd_3 to cfd_6 on the left errors were injected to cover all the possible cases: a) errors on one attribute, b) errors on more attributes in the same tuple.

A.2 Train Test Splits

In this section we report the number of user examples used to train the classifier for each CFD. Depending on the threshold and the number of initial examples to initialize the classifiers the train test sizes could vary in the active learning setting. Thus, to estimate the maximum number of required training examples we use the highest threshold in our experiment, i.e. 95% and we set the number of initial examples with the same number of the examples used in Exp-1 (m=3). Results are reported in Table 18. Notice that every time a LHS update is submitted, the classifier is retrained, thus the number of training examples is higher even though the classifier correctly classified the cells with errors.

Received 4 May 2023; revised 6 February 2024; accepted 15 May 2024

Bus	
cfd_1	**shortname**=London ⟶ **areaname**=Greater London
cfd_2	**atcoareacode**=20 ⟶ **regioncode**=SE
cfd_3	**administrativeareacode**=101, **nptgdistrictcode**=189 ⟶ **regioncode**=EA
cfd_4	**administrativeareacode**=75, **nptgdistrictcode**=79 ⟶ ⟶=EM
cfd_5	**administrativeareacode**=73, **atcoareacode**=80, **nptgdistrictcode**=310 ⟶ **regioncode**=SW
cfd_6	**atcoareacode**=260, **shortname**=Leics, **administrativeareacode**=88, **areaname**=Leicestershire ⟶ **regioncode**=EM
DBLP	
cfd_1	**pubkey**=acerneseaa06 ⟶ **title**=Lenght Sensing and …Wave Interferometer
cfd_2	**pubkey**=FanfaniASABBBSBBa10 ⟶ **title**=Distributed Analysis in CMS.
cfd_3	**year**=2005, **title**=The Biomulecular …2005 update ⟶ **journal**=Nucleic Acids Research
cfd_4	**year**=2011, **title**=BioMart Central Portal …biological community ⟶ **journal**=Database
cfd_5	**journal**=PVLDB, **volume**=2, **numbervol**=1 ⟶ **year**=2009
cfd_6	**journal**=PVLDB, **volume**=7, **numbervol**=13 ⟶ **year**=2014
Synth	
cfd_1	**team**=West Ham United F.C. ⟶ **stadium**=St. James Park
cfd_2	**team**=A.C Milan ⟶ **city**=Milan
cfd_3	**team**=Genoa C.F.C, **stadium**=Petrovsky Stadium ⟶ **city**=Istanbul
cfd_4	**manager**=G. Muldowney, **season**=2011 ⟶ **team**=Panathinaikos FC
cfd_5	**name**=Ada Egidio, **surname**=Menezes, **birthyear**=1994, **birthplace**=Orlando, **team**=VfL Wolfsburg, **season**=2012 ⟶ **position**=Goalkeeper
cfd_6	**name**=Jaron Shamik, **surname**=Samtaney, **birthyear**=1982 ⟶ **birthplace**=Barcelona

Table 17. CFDs used in the Experiments for the error injection and error detection.

	BUS – Easy			BUS – Hard			DBLP – Easy			DBLP – Hard			Synth – Easy			Synth – Hard		
CFDs	**Total**	**Train**	**Test**	**Total**	**Train**	**Test**	**Total**	**Train**	**Test**	**Total**	**Train**	**Test**	**Total**	**Train**	**Test**	**Total**	**Train**	**Test**
cfd_1	171	37	134	172	88	84	78	19	59	125	66	59	100	27	73	100	53	47
cfd_2	157	29	128	164	84	80	60	15	45	90	45	45	100	31	69	100	57	43
cfd_3	76	17	59	51	26	25	84	54	30	232	176	56	40	3	37	30	3	37
cfd_4	90	25	65	69	37	32	59	37	22	190	147	43	30	3	27	30	3	27
cfd_5	93	28	65	70	39	31	13	6	7	24	14	10	20	3	17	20	3	17
cfd_6	56	22	34	46	28	18	20	9	11	57	33	24	10	3	7	10	3	7

Table 18. Train test split for all the scenarios and for each CFDs.