Raha: A Configuration-Free Error Detection System

Mohammad Mahdavi
TU Berlin
mahdavilahijani@tu-berlin.de

Ziawasch Abedjan
TU Berlin
abedjan@tu-berlin.de

Raul Castro Fernandez
MIT
raulcf@csail.mit.edu

Samuel Madden
MIT
madden@csail.mit.edu

Mourad Ouzzani
QCRI, HBKU
mouzzani@hbku.edu.qa

Michael Stonebraker
MIT
stonebraker@csail.mit.edu

Nan Tang
QCRI, HBKU
ntang@hbku.edu.qa

ABSTRACT

Detecting erroneous values is a key step in data cleaning. Error detection algorithms usually require a user to provide input configurations in the form of rules or statistical parameters. However, providing a complete, yet correct, set of configurations for each new dataset is not trivial, as the user has to know about both the dataset and the error detection algorithms upfront. In this paper, we present Raha, a new configuration-free error detection system. By generating a limited number of configurations for error detection algorithms that cover various types of data errors, we can generate an expressive feature vector for each tuple value. Leveraging these feature vectors, we propose a novel sampling and classification scheme that effectively chooses the most representative values for training. Furthermore, our system can exploit historical data to filter out irrelevant error detection algorithms and configurations. In our experiments, Raha outperforms the state-of-the-art error detection techniques with no more than 20 labeled tuples on each dataset.

ACM Reference Format:
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 *2019 International Conference on Management of Data (SIGMOD '19), June 30-July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3299869.3324956

SIGMOD '19, June 30-July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-5643-5/19/06…$15.00
https://doi.org/10.1145/3299869.3324956

1 INTRODUCTION

Data scientists consider data cleaning to be one of the most time-consuming tasks [20]. Data cleaning consists of the consecutive *error detection* and *data repairing* tasks [14]. Error detection identifies data values that deviate from the (typically unavailable) ground truth. Error detection strategies can be categorized into quantitative approaches such as detecting outliers [36] or qualitative approaches such as detecting violations of integrity constraints [18]. Data repairing identifies the correct values for the detected erroneous data [17, 18, 39]. In practice, error detection and data repairing are usually performed iteratively, instead of sequentially. Hence, they will affect each other in terms of effectiveness [39].

The focus of this work is on error detection. Running only one error detection algorithm often will result in poor recall [1], which suggests running multiple approaches to cover various types of errors. However, this requires users to engage in a series of time-consuming and non-trivial steps:

(1) **Algorithm selection.** Faced with various types of errors, a user would need to select one or more algorithms to run. The user typically selects algorithms based on past experience or randomly if none is available. Trying out all of these algorithms is cumbersome and their union aggregation often results in poor precision [1]. Another approach is to run each detector sequentially based on their precision [1], which will require user involvement to compute the precision.
(2) **Algorithm configuration.** The user also needs to configure the selected error detection algorithms based on the characteristics of the given dataset. Depending on the error detection algorithm, the configuration might include statistical parameters, such as expected mean and standard deviation; patterns, such as "dd.mm.yyyy"; specification of rules; or providing links to reference datasets, such as DBpedia [7]. Having to

select the appropriate algorithm(s) and their appropriate configuration(s) results in a huge search space for identifying the best error detection strategies.

(3) **Result verification.** Even after an algorithm is selected and the corresponding parameters are manually configured, oftentimes, the user still needs to be involved in verifying the results for fine-tuning the algorithm and improving its accuracy.

To perform these tasks, the user has to have knowledge about the dataset domain, its quality problems, and also the error detection algorithms themselves.

There are attempts of using machine learning-based approaches, e.g., ActiveClean [32] and metadata-driven approach [45]. These approaches try to overcome some of these challenges by learning from data points that were labeled by the user as *dirty* or *clean*. However, they suffer from two general problems. First, the designed feature vectors either are not expressive enough to capture various error types or have to be manually adjusted. ActiveClean uses TF-IDF features [32], which only expose errors on the word level. The metadata-driven approach leverages the output of configured detection tools [45], requiring the user to configure the tools. Second, the required training data is quite large, and increases with the size of the dataset. For example, the metadata-driven approach expects up to 1% of the dataset as the training data [45]. This can be quite large for big data.

In this paper, we propose a new semi-supervised error detection approach that does not require the user to provide configurations. In a nutshell, our approach assigns a feature vector to each data cell. Each component of the vector represents the binary output of a particular configuration of a particular error detection algorithm. We then cluster the cells of each data column based on their feature vector and ask users to only label one cluster at a time.

Feature representation. To characterize data errors, the feature vector encodes the output of the four main families of traditional error detection techniques, namely outlier detection, pattern violation detection, rule violation detection, and knowledge base violation detection algorithms [1]. For each of these algorithm families, we automatically generate a principled and limited set of possible configurations, such as different thresholds for outlier detection algorithms and different patterns, rules, and reference datasets for corresponding violation detection algorithms. Moreover, we limit the exponential number of patterns and dependencies and discretize the continuous space of statistical parameters based on theoretical boundaries and best-practice heuristics. We use the results of each of those configurations to generate a feature vector per data cell. As not every component of the vector represents an effective configuration for a given dataset, aggregating the results through voting or other ensemble approaches might be misleading. Instead, we use these information-rich feature vectors to compute the similarity between data cells in order to cluster them.

Table 1: Required user involvement in error detection.

Error Detection Approach	Algorithm Selection	Algorithm Configuration	Result Verification
Stand-Alone Tools [17, 18, 36]		×	×
Aggregators [1, 45]	×	×	×
ML-Based Approaches [32]		×	×
Raha			×

User involvement. To limit user involvement, we design a clustering and learning scheme that enables effective labeling and label propagation. As each cluster contains values with a similar feature vector, we ask the user to only label one instance per cluster. We obtain additional noisy training data by propagating the labels to all the other values of the same cluster. Thus, the number of generated clusters caps the required number of user interactions. We suggest that each cluster that is classified as dirty represents an implicit type of data errors. A state-of-the-art classifier can then use the user labels and noisy labels to classify all data cells as being dirty or clean. Table 1 shows the benefits of Raha compared with existing approaches in terms of user involvement. While the other approaches involve the user in multiple steps, Raha only requires a small number of user labels.

Filtering features. While our approach requires only a very small amount of user involvement, and no configuration at all, it consumes more runtime during feature generation because of the large number of features. Nevertheless, the runtime of our feature generation can significantly be reduced using cleaned datasets from the past, if available. We present an approach that filters out irrelevant error detection algorithms and configurations that did not contribute to the overall performance on similarly dirty datasets in the past.

Contributions. We make the following contributions:

- We propose the *configuration-free error detection system*, named Raha (Section 3), that does not need any user-provided data constraints or parameters.
- We propose a novel *feature vector* (Section 4) for the error detection task that incorporates signals from various error detection strategies.
- We propose a *clustering-based sampling approach* (Section 4) that significantly reduces user involvement for labeling. Using the clustering-based label propagation approach, we obtain additional noisy labels that improve the classification performance of Raha.
- We propose a *strategy filtering approach* (Section 5) that prunes algorithm configurations based on cleaned datasets from the past. This approach improves the runtime by more than an order of magnitude at the cost of slightly lower effectiveness.

Table 2: A dirty dataset d and its ground truth d^*.

ID	Lord	Kingdom
1	Aragorn	Minas Tirith
2	Sauron	Mordor
3	Gandalf	∅
4	Saruman	∅
5	Elrond	123
6	Théoden	Shire

ID	Lord	Kingdom
1	Aragorn	Minas Tirith
2	Sauron	Mordor
3	Gandalf	N/A
4	Saruman	Isengard
5	Elrond	Rivendell
6	Théoden	Rohan

- We conduct *extensive experiments* (Section 6) to evaluate Raha with regard to effectiveness, efficiency, and user involvement. We compare Raha with 4 standalone error detection algorithms and 3 error detection aggregators. Using only 20 labeled tuples, Raha outperforms any other existing solution.

2 FOUNDATIONS

We first define data errors and categorize their types. Then, we review general families of error detection algorithms from the literature that are used in Raha. Finally, we formally define the problem of configuration-free error detection.

2.1 Data Errors

Data errors are those data values inside a dataset that deviate from the actual ground truth. More formally, let $d = \{t_1, t_2, ..., t_{|d|}\}$ be a relational dataset of size $|d|$, where each t_i denotes a tuple. Let $A = \{a_1, a_2, ..., a_{|A|}\}$ be the schema of dataset d, with $|A|$ attributes. Let $d[i, j]$ be the data cell in the t_i tuple of dataset d and a_j attribute of the schema A. We denote the cleaned version or the ground truth of the same dataset as d^*. Every data cell $d[i, j]$ that is different from the corresponding cell in the ground truth $d^*[i, j]$ is considered to be a *data error*. For example, in the dirty dataset d in Table 2, the data cells $d[3, 2] = \emptyset$, $d[4, 2] = \emptyset$, $d[5, 2] = 123$, and $d[6, 2] = Shire$ are erroneous according to the ground d^*.

Data errors can be categorized into *syntactic* and *semantic* errors. Syntactic errors are those values that do not conform to the structure or domain of correct values. Semantic errors are values that, although are syntactically correct, appear in the wrong context. For example, the numerical data value $d[5, 2] = 123$ is a syntactic error because it does not fit the syntax of any kingdom name. The data value $d[6, 2] = Shire$ is a semantic error because, although it is a syntactically correct kingdom in "Lord of the Rings", the correct value for this data cell is a different kingdom, namely *Rohan*.

2.2 Error Detection Algorithms

In principle, our system can leverage any error detection algorithm that takes either continuous numerical parameters or discrete nominal parameters as input. Any other error detection algorithm that does not require any of such parameters, i.e., a black box, can only be used with a single configuration. Existing error detection algorithms [17, 18, 29, 36] can be categorized into four families: outlier detection, pattern violation detection, rule violation detection, and knowledge base violation detection [1]. The first two are mainly for syntactic data errors and the last two for semantic errors. Note that Raha is not limited to these categories.

Outlier detection algorithms [36] assess the correctness of data values in terms of compatibility with the general distribution of values that reside inside the column. Histogram and Gaussian modelings [36] are two fundamental outlier detection algorithms that leverage the occurrence and magnitude of data values, respectively. A histogram modeling strategy builds a histogram distribution based on the frequency of data values in a particular data column. The strategy marks data cells from the rare bins as errors, i.e., data cells with a normalized term frequency less than a threshold $\theta_{\text{tf}} \in (0, 1)$. A Gaussian modeling strategy builds a Gaussian distribution based on the magnitude of the numerical values in a particular column. This strategy marks as errors those numerical cells whose normalized distance to the mean is farther than a threshold $\theta_{\text{dist}} \in (0, \infty)$.

For example, on dataset d in Table 2, the output of two histogram-based outlier detection algorithms over the attribute *Kingdom* would be $s_{o1} = \{d[1, 2], d[2, 2], d[5, 2], d[6, 2]\}$ by setting $\theta_{\text{tf}} = \frac{2}{6}$ and $s_{o2} = \{d[1, 2], d[2, 2], d[3, 2], d[4, 2], d[5, 2], d[6, 2]\}$ by setting $\theta_{\text{tf}} = \frac{3}{6}$.

Pattern violation detection algorithms [29] assess the correctness of data values in terms of compatibility with predefined data patterns. A pattern violation detection algorithm marks data values that do not match a certain pattern.

For example, on dataset d in Table 2, the output of two pattern violation detection algorithms over the attribute *Kingdom* would be $s_{p1} = \{d[3, 2], d[4, 2], d[5, 2]\}$ by setting the predefined pattern to *alphabetical values* and $s_{p2} = \{d[3, 2], d[4, 2]\}$ by setting the predefined pattern to *not-null values*.

Rule violation detection algorithms [18] assess the correctness of data values based on their conformity to integrity constraints. Since the single column rules, such as value range and length, are implicitly covered by the outlier and pattern violation detection algorithms, we include here only rule violation detection strategies that check inter-column dependencies. In particular, we focus on rules in the form of functional dependencies (FDs). Other types of rules, such as conditional FDs, can be considered the same way.

For example, on dataset d in Table 2, the output of two rule violation detection algorithms over the attribute *Kingdom* would be $s_{r1} = \{\}$ by checking the FD $Lord \rightarrow Kingdom$ and $s_{r2} = \{d[3, 2], d[4, 2]\}$ by checking the FD $Kingdom \rightarrow Lord$.

Knowledge base violation detection algorithms, such as Katara [17], assess the correctness of data values by cross-checking them with data within a knowledge base, such as DBpedia [7]. The data inside a knowledge base is usually

stored in the form of entity relationships, such as *City isCapitalOf Country*. Here, *City* and *Country* are entity types and *isCapitalOf* is a relationship. The algorithm tries to match each side of a relationship to different data columns in the dataset. If there are two data columns that are matched to both sides of the relationship (e.g., *City* and *Country*), the algorithm marks data values in the matched data columns that conflict the entity relationship inside the knowledge base. For example, the algorithm would match a relationship *isCapitalOf* to two columns and mark data cells in violating tuples, e.g., those wrongly suggesting *Berlin* as the capital of *France*. Therefore, the knowledge base violation detection algorithms can also identify data errors that violate inter-column dependencies.

On dataset d in Table 2, the output of two knowledge base violation detectors over the attribute *Kingdom* would be $s_{k1} = \{d[3,2], d[4,2], d[5,2], d[6,2]\}$ by setting the entity relationship to *Lord isKingOf Kingdom* and $s_{k2} = \{\}$ by setting the entity relationship to *City isCapitalOf Country*.

2.3 Problem Statement

Given a dirty dataset d as input, a set of available error detection algorithms $B = \{b_1, b_2, ..., b_{|B|}\}$ that require configurations, and a user with a labeling budget θ_{labels} to annotate tuples, we want to identify all erroneous values within d. In particular, the problem is to automatically configure the error detection algorithms B and aggregate their output into a set of labeled data cells, i.e., $O = \{(d[i,j], l) \mid 1 \leq i \leq |d|, 1 \leq j \leq |A|, l \in \{\text{dirty}, \text{clean}\}\}$, where $|d|$ is the number of rows and $|A|$ is the number of columns in the dataset d.

In order to formulate the error detection problem as a classification task, we have to address three subproblems. First, we need feature vectors that can represent syntactic and semantic data errors. Second, we need a sampling approach that can deal with the class imbalance ratio between clean and erroneous values. Third, before running all the strategies, it is also desirable to filter out irrelevant error detection algorithms and configurations to reduce the runtime.

For the sake of simplicity in the remainder of this paper, we consider each combination of an algorithm and a configuration as one distinct *error detection strategy*.

Definition 1 (Error detection strategy). Let $B = \{b_1, b_2, ..., b_{|B|}\}$ be the set of error detection algorithms. Let $G_b = \{g_1, g_2, ..., g_{|G_b|}\}$ be the set of finite/infinite space of different configurations of an error detection algorithm b. We define an error detection strategy as a combination of an error detection algorithm b and a configuration $g \in G_b$. Thus, the set of error detection strategies is a subset of all possible error detection strategies $S \subseteq B \times G_b = \{(b, g) \mid b \in B, g \in G_b\}$. □

3 RAHA OVERVIEW

We explain each step of the workflow of Raha in Figure 1 using the running example in Section 2. Given the dirty dataset, the error detection toolbox, and the user feedback as input, Raha classifies each data value as *clean* or *dirty*.

Step 1: Automatic algorithm configuration. Raha systematically configures each existing algorithm to generate a set of error detection strategies. In the example in Section 2, we introduced two configurations for each class of error detection algorithms. This would result into $|S| = |B \times G_b| = 4 \times 2 = 8$ strategies: $S = \{s_{o1}, s_{o2}, s_{p1}, s_{p2}, s_{r1}, s_{r2}, s_{k1}, s_{k2}\}$. We detail this step in Section 4.1.

Step 2: Running error detection strategies. Each strategy marks a set of data cells as data errors. As mentioned in Section 2, the outlier detection strategy s_{o1}, for example, marks data cells $s_{o1} = \{d[1,2], d[2,2], d[5,2], d[6,2]\}$ over the attribute *Kingdom*. We detail this step in Section 4.2.

Step 3: Feature vector generation. Raha generates a feature vector for each data cell by collecting the output of all the error detection strategies. Each element in the feature vector of a data cell is a binary value that shows whether a particular strategy marks this data cell as a data error or not. As Table 3 shows, the feature vector of data cell $d[1,2]$ = *Minas Tirith* in our running example is $[1,1,0,0,0,0,0,0]$. We detail this step in Section 4.2.

Step 4: Clustering data cells. Raha groups the cells of each column into distinct clusters, based on the similarity of their feature vectors. As Table 3 shows, the data cells inside column *Kingdom* are clustered into 3 groups: a black, a blue, and a purple cluster. We detail this step in Section 4.3.

Step 5: Sampling tuples. Raha samples a set of tuples to be labeled by the user. Since each data column has a set of clusters, ideally the sampled tuples should cover as many unlabeled clusters as possible over all the data columns. For the mentioned example, suppose Raha samples tuples t_1 and t_3. These two tuples cover the black and the blue clusters over the data column *Kingdom*. Ideally, these two tuples should cover *two* different clusters in the other data columns, i.e., column *Lord*. We detail this step in Section 4.3.

Step 6: Labeling data cells. Raha asks the user to label data cells of the sampled tuples as dirty or clean. In the example, the user labels data cells $d[1,1]$, $d[1,2]$, and $d[3,1]$ as clean and $d[3,2]$ as dirty. We detail this step in Appendix C.

Step 7: Propagating user labels through clusters. Data cell $d[2,2]$ will be labeled as clean because it is in the same cluster of data cell $d[1,2]$. Likewise, data cell $d[4,2]$ will be labeled as dirty. These labels are noisy as they have not been verified by the user. We detail this step in Section 4.4.

Step 8: Training classification models. Raha trains a classification model per data column based on the feature vectors

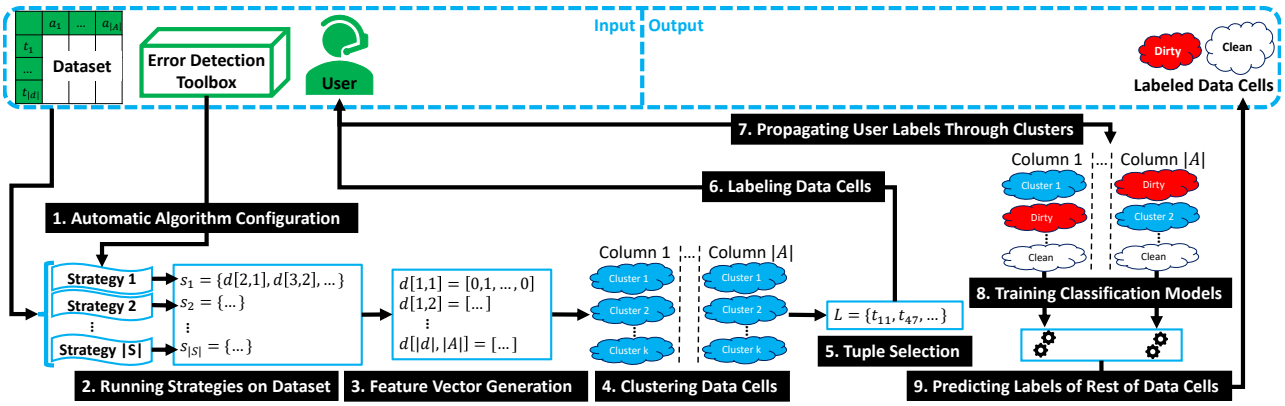


Figure 1: The workflow of Raha.

Table 3: Clustering data cells of column *Kingdom*.

ID	Kingdom	s_{o1}	s_{o2}	s_{p1}	s_{p2}	s_{r1}	s_{r2}	s_{k1}	s_{k2}
1	Minas Tirith	1	1	0	0	0	0	0	0
2	Mordor	1	1	0	0	0	0	0	0
3	∅	0	1	1	1	0	1	1	0
4	∅	0	1	1	1	0	1	1	0
5	123	1	1	1	0	0	0	1	0
6	Shire	1	1	0	0	0	0	1	0

of data cells and the propagated labels, i.e., user labels and noisy labels. We detail this step in Section 4.4.

Step 9: Predicting labels of rest of data cells. The trained classification models are used to predict the labels of the remaining data cells in other unlabeled clusters. In the mentioned example, the trained classification model for data column *Kingdom* predicts the labels of data cells $d[5, 2]$ and $d[6, 2]$ in the unlabeled purple cluster as dirty. We detail this step in Section 4.4.

4 THE ERROR DETECTION ENGINE

Algorithm 1 shows the main steps of Raha. It first configures error detection algorithms (line 1) and then generates the feature vectors (line 2). Next, Raha clusters data cells and samples tuples for user labeling (lines 3-11). Finally, it propagates the user labels through the clusters and trains a set of classifiers to predict the label for all cells (lines 12-16).

4.1 Automatic Algorithm Configuration

As explained in Section 2, we can identify algorithms with numerical and nominal parameters. For algorithms with numerical parameters, such as outlier detection algorithms, we quantize the continuous range of the parameters. For algorithms with infinite nominal parameters, such as patterns, rules, and knowledge base violation detection algorithms, we identify heuristics to effectively limit the space of parameters. Of course, only a subset of the generated error detection strategies may effectively mark data errors on a given dataset

Algorithm 1: Raha(d, B, θ_{labels}).

Input: dataset d, set of error detection algorithms B, labeling budget θ_{labels}.
Output: set of labeled data cells O.

1 $S \leftarrow$ generate strategies by automatically configuring algorithms $b \in B$;
2 $V \leftarrow$ generate features by running strategies $s \in S$ on dataset d;
3 $k \leftarrow 2$; // number of clusters per data column
4 $L \leftarrow \{\}$; // set of user labeled tuples
5 **while** $|L| < \theta_{labels}$ **do**
6 **for** *each data column* $j \in [1, |A|]$ **do**
7 $\phi_j \leftarrow$ cluster data cells of column j into k clusters;
8 $t^* \leftarrow$ draw a tuple with probability proportional to $P(t)$;
9 ask the user to label tuple t^*;
10 $L \leftarrow L \bigcup \{t^*\}$;
11 $k \leftarrow k + 1$;
12 $O \leftarrow \{\}$; // set of labeled data cells
13 **for** *each data column* $j \in [1, |A|]$ **do**
14 $L'_j \leftarrow$ propagate the user labels through the clusters ϕ_j;
15 $m_j \leftarrow$ train a classification model with feature vectors V_j and labels L'_j;
16 $O \leftarrow O \bigcup$ apply the classification model m_j on rest of feature vectors V_j;

and most of them might be imprecise. Nevertheless, as long as each strategy marks data cells using the same logic, Raha can use the output of the strategy as a notion of similarity for comparing data cells.

Outlier detection strategies. Histogram modeling outlier detection strategies $s_{\theta_{\text{tf}}}$ require a minimum term frequency threshold $\theta_{\text{tf}} \in (0, 1)$. Raha generates 9 histogram modeling outlier detection strategies by setting the threshold $\theta_{\text{tf}} \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$. Note that, in practice, we found that finer granular values (e.g., $\theta_{\text{tf}} = 0.15$) do not create more distinctive outlier detection strategies in comparison to the coarser granular selection of thresholds. Formally,

$$s_{\theta_{\text{tf}}}(d[i,j]) = \begin{cases} 1, & \text{iff } \frac{TF(d[i,j])}{\sum_{i'=1}^{|d|} TF(d[i',j])} < \theta_{\text{tf}}; \\ 0, & \text{otherwise.} \end{cases}$$

where $TF(d[i, j])$ is the term frequency of the data cell $d[i, j]$ inside the data column j.

Gaussian modeling outlier detection strategies $s_{\theta_{\text{dist}}}$ require a distance threshold $\theta_{\text{dist}} \in (0, \infty)$. Raha generates 9 Gaussian modeling outlier detection strategies by automatically setting the threshold $\theta_{\text{dist}} \in \{1, 1.3, 1.5, 1.7, 2, 2.3, 2.5, 2.7, 3\}$, according to the *68-95-99.7 rule* [44]. Formally,

$$s_{\theta_{\text{dist}}}(d[i,j]) = \begin{cases} 1, & \text{iff} \quad \frac{|d[i,j]-\mu_j|}{\sigma_j} > \theta_{\text{dist}}; \\ 0, & \text{otherwise.} \end{cases}$$

where μ_j is the mean and σ_j is the standard deviation of the numerical data column j.

Pattern violation detection strategies. Theoretically, we could have domain-specific data patterns such as *dates* and *URLs* to detect pattern violations. However, since the set of all possible data patterns is infinite, we need a reasonable way to limit the set of pattern violation detection strategies without holding any specific assumption on the data domain. To this end, we leverage the *bag-of-characters* representation [41], which is a general representation for encoding all possible data patterns, i.e., character combinations. This representation can also encode the length and type of data values as it also shows which and how many distinct characters appear in data values.

In particular, we generate a set of character checker strategies s_{ch} to check the existence of each character ch in data cells. For each character ch in the set of all characters in a data column j, s_{ch} marks a data cell $d[i,j]$ as data error if the cell contains character ch. Formally,

$$s_{\text{ch}}(d[i,j]) = \begin{cases} 1, & \text{iff} \quad d[i,j] \text{ contains } ch; \\ 0, & \text{otherwise.} \end{cases}$$

As an example, suppose we have a date column with the pattern "dd.mm.yyyy". The bag-of-characters representation automatically generates the strategy s_{-} and exposes the appearance of erroneous delimiters, such as the character "-" in "16-11-1990".

Overall, we have $\bigcup_{j=1}^{|A|} |\Sigma_j|$ pattern violation detection strategies in our system, where $|A|$ is the number of data columns and Σ_j is the set of all distinct characters appearing in data column j.

Rule violation detection strategies. We limit the scope of FDs to only those with a single attribute on their left-hand side, in order to reasonably limit the exponential space of all possible FDs. This way, we also avoid potentially unintended dependencies that arise by considering large sets of data columns. As discussed in the literature [34], most interesting FDs involve only a few attributes. As we do not know upfront which FDs are useful, we consider all pairs of columns as potential FDs. With this approach, we are also covering partial FD relationships that were unknown to the user.

For each pair of attributes $\forall a \neq a' \in A$, strategy $s_{a \to a'}$ marks all data cells $d[i,j]$ that violate the FD $a \to a'$, in accordance with prior work [1, 15]. Formally,

$$s_{a \to a'}(d[i,j]) = \begin{cases} 1, & \text{iff} \quad d[i,j] \text{ violates } a \to a'; \\ 0, & \text{otherwise.} \end{cases}$$

The number of rule violation detection strategies is $|A| \times (|A| - 1)$ as we consider the FDs from and to each attribute.

Knowledge base violation detection strategies. Raha establishes connections between the dataset and a knowledge base, such as DBpedia [7]. For each relationship r inside the knowledge base, strategy s_r marks all data cells $d[i,j]$ that violate (contradict) the relationship. Formally,

$$s_r(d[i,j]) = \begin{cases} 1, & \text{iff} \quad d[i,j] \text{ violates } r; \\ 0, & \text{otherwise.} \end{cases}$$

Overall, we have as many knowledge base violation detection strategies as there are relationships inside the DBpedia knowledge base [7], i.e., 2064 strategies.

4.2 Feature Vector Generation

We map each data cell to a feature vector that is composed of the output of error detection strategies. Having a set of error detection strategies $S = \{s_1, s_2, ..., s_{|S|}\}$, we run each strategy $s \in S$ on the dataset d. Each strategy s either marks a data cell $d[i,j]$ as a data error or not. Formally, we store this information as

$$s(d[i,j]) = \begin{cases} 1, & \text{iff} \quad s \text{ marks } d[i,j] \text{ as a data error}; \\ 0, & \text{otherwise.} \end{cases}$$

The feature vector of the data cell $d[i,j]$ is the vector of all the outputs of error detection strategies $s \in S$ on this data cell. Formally,

$$v(d[i,j]) = [s(d[i,j]) \mid \forall s \in S]. \tag{1}$$

Hence, the set of feature vectors of data cells $v(d[i,j])$ inside a particular data column j is

$$V_j = \{v(d[i,j]) \mid 1 \leq i \leq |d|\}. \tag{2}$$

We post-process the feature vectors of each data column V_j to remove non-informative features that are constant for all the data cells of the data column.

4.3 Clustering-Based Sampling

Using the described feature vector, Raha learns how to classify data cells as clean or dirty with the help of user labels. We train one classifier for each column because we want to identify errors at the cell level, not at the tuple level. Thus, we need labels for each column. In this section, we describe our solution for reducing the labeling effort by sampling tuples that cover promising cell values from each column.

Clustering data cells. Using our expressive feature vector, we can boost the number of labeled data cells per data column

using the *cluster assumption* [13], which states that two data points are likely to have the same class label if they belong to one cluster [13]. Thus, if we cluster data cells, cells in each cluster are likely to have the same dirty or clean label.

We also apply a separate clustering approach for each data column, because of the same reason that we apply one classifier per column. By having a clustering model per data column, we can measure the similarity of data cells more accurately as data values are only reasonably comparable within their own domain.

Since the labeling will be at the cluster level, we need a way to control the number of clusters k per column. Smaller k's yield bigger clusters that are more likely to contain a mix of dirty and clean data cells. Inversely, bigger k's will lead to more clusters, requiring more and potentially unnecessary labels from the user. There are clustering algorithms that can automatically choose the number of clusters, such as DBSCAN [22]. However, the disadvantage of these clustering approaches is that the user has no control over the number of clusters and thus the number of required labels.

We apply hierarchical agglomerative clustering [5], which does not need the specification of the number of clusters. This approach starts with only two clusters per column and then increases the number of clusters in each iteration. Its iterative process lets the user terminate the clustering-based sampling process at will and in accordance with the labeling budget. As the choice of the similarity metric and the linkage method does not affect Raha's performance, we used the default cosine similarity metric and the average linkage method.

Tuple selection. So far, the user is required to label column values independently of the other values of the same tuple. However, to label one data cell, the user typically needs to check its whole tuple. Thus, it is more intuitive to sample entire tuples for user labeling than individual cells per column.

Picking the best tuples is not trivial as each column might contain errors in completely different tuples. In every clustering iteration, each data column is divided into k clusters of data cells. Some of these clusters are unlabeled, i.e., none of their data cells has been labeled. Ideally, the sampled tuples should cover all unlabeled clusters from each column. This way, labeling the sampled tuples leads to labeling all the unlabeled data cells. However, finding a minimum set of tuples that covers all the unlabeled clusters is the classical *set cover problem* [30]. To address this NP-complete problem, we design an approximate approach with two properties. First, we relax the challenge of finding a minimum set of tuples by selecting only one tuple in each iteration. This way, we avoid the challenge of finding tuples that do not cover the same set of clusters. Second, instead of deterministically selecting the tuple that covers the most number of unlabeled clusters, we probabilistically select this tuple. This way, we avoid getting stuck in local optima as probabilistic solutions for this problem have been shown to be more resilient than deterministic greedy heuristics against local optima [25]. Therefore, in each iteration, we draw a tuple t^* based on the softmax probability function

$$P(t) = \frac{\exp(\sum_{c \in t} \exp(-N_c))}{\sum_{t' \in d} \exp(\sum_{c \in t'} \exp(-N_c))}, \tag{3}$$

where N_c is the number of labeled data cells in the current cluster of data cell c and exp is the exponential function with base e. This scoring formula benefits tuples whose data cells mostly belong to the clusters that have received fewer labels.

The proposed clustering-based sampling scheme has two characteristics. First, since the sampling approach iteratively clusters and labels under-labeled clusters, the expectation is that at some point we will cluster each column into homogeneously clean or dirty clusters, respectively. In other words, if there is a certain type of errors inside one data column, the hierarchical clustering approach would eventually identify its corresponding cluster. Furthermore, the sampling approach addresses the natural class imbalance issue as well, because the rare dirty labels will be propagated through the corresponding clusters.

At the end of each iteration, we have the set of user labeled tuples $L \subset d$. This iterative procedure is repeated in the next iterations with a larger number of clusters $k \in \{2, 3, \ldots\}$ as long as the labeling budget of the user is not depleted, i.e., $|L| < \theta_{\text{labels}}$. At the end of the sampling process, we have $k = \theta_{\text{labels}} + 1$ clusters per data column and $|L| = \theta_{\text{labels}}$ labeled tuples.

4.4 Label Propagation and Classification

In semi-supervised settings like ours, having more user labels leads to faster convergence of the models [43]. We thus leverage the clusters to boost the number of labels, according to the *cluster assumption* [13]. Since we cluster data cells based on the expressive feature vectors, the cells inside one particular cluster are likely to have the same dirty/clean label. Therefore, we can propagate user labels through the clusters.

Raha propagates the user label of each data cell to all data cells in its cluster. Let $L' = \{d[i,j] \mid i \in L, 1 \leq j \leq |A|\}$ be the set of labeled data cells, i.e., all data cells of the labeled tuples L. All the unlabeled data cells $d[i',j'] \notin L'$ that are inside the same cluster of a labeled data cell $d[i,j] \in L'$ get the same dirty/clean label of data cell $d[i,j]$. This way, the number of training labels L' increases significantly.

Since a cluster may have multiple user labeled data cells with contradicting dirty/clean labels, we need a way to resolve such contradictions. We investigate two conflict resolution functions to propagate user labels in clusters; a homogeneity-based function and a majority-based function. The homogeneity-based function only propagates user labels

in clusters that do not contain two cells with contradicting labels. This approach works under the assumption of perfect user labels, in which case, the contradicting user labels inside one cluster indicate that the cluster is not homogeneous enough for label propagation. The majority-based function propagates the user labels also in clusters with mixed labels if a label class has the majority. The intuition is that a homogeneous cluster could also have contradicting user labels due to the user labeling error. In our datasets, $k-1$ out of k clusters inside a column were typically small clusters with only clean or dirty data cells while the last k'th cluster was a large cluster with mixed dirty and clean data cells.

Raha then trains a classifier m_j per data column j to predict the labels for the unlabeled data cells in the same column. In the training phase, each classification model m_j takes the feature vectors of data cells inside the data column (i.e., V_j) along with the labeled data cells (i.e., L'). In the prediction phase, each classification model m_j predicts the labels of rest of data cells in each data column. Note that, although we train a classification model per data column, the classifier is still able to detect inter-column dependency violation errors due to the rule and knowledge base violation detection features.

4.5 System Analysis

Raha's main objective is to optimize effectiveness and user involvement. The configuration-free nature of Raha is achieved by leveraging a large set of predefined configurations for each error detection algorithm. While leveraging the rich feature vector and the clustering-based sampling scheme leads to high accuracy with a small amount of labels, Raha may raise efficiency concerns as running these large set of error detection strategies could be time consuming. Although parallelization could certainly be applied to reduce the overall runtime, it would be more desirable to stop running irrelevant strategies. To this end, one has to identify the most promising strategies, i.e., features, upfront. This is not possible using existing feature selection techniques as they require the output of the strategies on training data. That is why Raha supports a strategy filtering approach based on historical data as explained next.

5 RUNTIME OPTIMIZATION USING HISTORICAL DATA

Raha provides the option of filtering irrelevant error detection strategies if there is historical data, i.e., datasets that have been cleaned previously. The idea is that on similar data domains, similar error detection strategies will perform similarly. For example, on a data column with domain *City*, we need to run only those error detection strategies that performed well on data column *Capital* of some historical datasets. This way, we can improve the runtime of the system significantly. To this end, we need to address two challenges. First, we need a notion of similarity that captures the relevance of an error detection strategy with regard to two columns. Second, we need an algorithm that can systematically adapt and select the promising error detection strategies for a new data column based on their effectiveness on other similar data columns. This way, the system can select the top-ranked adapted strategies and filter out the rest.

5.1 Data Column Similarity

Measuring the similarity of data columns is a well-studied problem in schema matching [37] and data profiling [2]. Building up on this line of research, we represent each data column with a *column profile*. The column profiles represent the syntactic and semantic similarity of data columns to expose syntactic and semantic data errors. Syntactic similarity can be captured based on the similarity of character distributions. For example, data columns *Age* and *Year* are syntactically similar as they have similar character distributions, i.e., numerical characters. That is why a pattern violation detection strategy that marks data values with non-numerical characters would work well on both of these data columns. Semantic similarity of columns can be captures based on the overlap of the actual values. For example, data columns *City* and *Capital* are semantically similar as they have similar value distributions, i.e., city names. That is why a rule violation detection strategy based on $ZIP \rightarrow City$ could also be adapted to $ZIP \rightarrow Capital$.

The column profile describes the content of a data column with its character and value distributions. Character distribution is a probability distribution function that calculates the probability of observing data cells containing character ch in a data column $d[:, j]$. Formally,

$$p(ch|d[:,j]) = \frac{|\{d[i,j] \mid 1 \leq i \leq |d|, d[i,j] \text{ contains } ch\}|}{|d|}. \tag{4}$$

Value distribution is a probability distribution function that calculates the probability of observing data cells equal to value v in the data column $d[:, j]$. Formally,

$$p(v|d[:,j]) = \frac{|\{d[i,j] \mid 1 \leq i \leq |d|, d[i,j] \text{ equals to } v\}|}{|d|}. \tag{5}$$

In conclusion, we can measure the similarity of two columns by applying any similarity measure on their profiles, i.e., character and value distributions. In practice, we calculate the cosine similarity.

5.2 Strategy Adaptation and Selection

Once we have obtained the similarity of each column to previously cleaned columns, we can adapt the strategies

Algorithm 2: StrategyFilterer(d_{new}, S, D, D^*).

Input: new dataset d_{new}, set of error detection strategies S, set of historical datasets D, set of historical ground truths D^*.
Output: set of promising error detection strategies S^*.

1 **for** *each data column* $j \in [1, |A_{new}|]$ *of dataset* d_{new} **do**
2 $S'_j \leftarrow \{\}$; `// set of all adapted strategies for column j`
3 $p_{d_{\text{new}}[:,j]} \leftarrow$ generate profile for data column $d_{\text{new}}[:, j]$;
4 **for** *each dataset* $d \in D$ **do**
5 **for** *each data column* $j' \in [1, |A|]$ *of dataset* d **do**
6 $p_{d[:,j']} \leftarrow$ generate profile for data column $d[:, j']$;
7 $\text{sim}(d_{\text{new}}[:, j], d[:, j']) \leftarrow$ similarity of $p_{d_{\text{new}}[:,j]}$ and $p_{d[:,j']}$;
8 **for** *each strategy* $s \in S$ **do**
9 $F(s, d[:, j']) \leftarrow F_1$ score of s on $d[:, j']$;
10 $s' \leftarrow$ adapt s for dataset d_{new};
11 $\text{score}(s') \leftarrow$ assign the score of s';
12 $S'_j \leftarrow S'_j \bigcup \{s'\}$;
13 $S^*_j \leftarrow \{\}$; `// set of promising adapted strategies for column j`
14 **do**
15 $s^* \leftarrow \arg\max_{s \in S'_j} \text{score}(s)$;
16 $S^*_j \leftarrow S^*_j \bigcup \{s^*\}$;
17 $S'_j \leftarrow S'_j - \{s^*\}$;
18 **while** *adding* s^* *to* S^*_j *does not decrease the gain of* S^*_j ;
19 $S^* \leftarrow \bigcup_{j=1}^{|A_{\text{new}}|} S^*_j$;

for the dataset at hand, rank the strategies based on their relevance, and select the most promising subset of them.

Algorithm 2 shows how Raha leverages the similarity between a column $d_{\text{new}}[:, j]$ and another column $d[:, j']$ to select the promising error detection strategies for the new column $d_{\text{new}}[:, j]$. The algorithm consists of four main steps.

First, the similarities between each column $d_{\text{new}}[:, j]$ of the new dataset d_{new} and each column $d[:, j']$ from any dataset d is computed according to the discussion in the previous section (line 7). Second, Raha retrieves the stored F_1 score of the strategy s on column $d[:, j']$, i.e., $F(s, d[:, j'])$ (line 9).

Third, Raha might need to modify the strategy s to make it compatible to run on the new column $d_{\text{new}}[:, j]$ (line 10). For an outlier detection, a pattern violation detection, or a knowledge base violation detection strategy, this modification is not necessary. Raha can simply run the same strategy on the new data column. However, for a rule violation detection strategy, which is schema dependent, Raha needs to modify the rules based on the schema of the new dataset. Suppose, we have a functional dependency checker $s_{j'_1 \to j'_2}$ that has detected data errors on the data columns $d[:, j'_1]$ and $d[:, j'_2]$ of the historical dataset d. Suppose, Raha identifies the historical data column $d[:, j'_1]$ similar to the new data column $d_{\text{new}}[:, j]$. Therefore, Raha translates the strategy $s_{j'_1 \to j'_2}$ to the strategy $s_{j \to q}$ for the new dataset d_{new}. To adapt the strategy $s_{j'_1 \to j'_2}$, Raha replaces the data column $d[:, j'_1]$ with its corresponding data column $d_{\text{new}}[:, j]$ and the data column $d[:, j'_2]$ with its most similar data column $d_{\text{new}}[:, q]$ inside the new dataset.

Fourth, Raha assigns a score to each updated error detection strategy s' (line 11). The score is the product of the similarity of data columns $d_{\text{new}}[:, j]$ and $d[:, j']$ and the F_1 score of strategy s on data column $d[:, j']$. Formally,

$$\text{score}(s') = \text{sim}(d_{new}[:, j], d[:, j']) \times F(s, d[:, j']). \quad (6)$$

The score will be high if the data columns $d_{\text{new}}[:, j]$ and $d[:, j']$ are similar and the strategy s has had a high F_1 score on the data column $d[:, j']$, indicating that the updated strategy s' will be promising for the data column $d_{\text{new}}[:, j]$.

We can sort the scored strategies S'_j for each data column $d_{\text{new}}[:, j]$ to pick only top-scored strategies per data column. A threshold-free approach to select the most promising subset of the strategies is to apply a gain function that has been used in the literature [3]. The idea is to add the top-scored strategies $s^* \in S'_j$ iteratively to the set of promising strategies S^*_j (lines 13 to 18), until adding the next best strategy decreases the following gain function [3], where the gain reaches a local maxima:

$$\text{gain}(S^*_j) = \sum_{s \in S^*_j} score(s) - \frac{1}{2} \sum_{s \in S^*_j} \sum_{s' \neq s \in S^*_j} |score(s) - score(s')|. \quad (7)$$

At the end, the set of promising error detection strategies S^* is the union of the promising strategies over all the data columns (line 19).

In Algorithm 2, Raha iterates over all the historical datasets (line 4) and their data columns (line 5). Instead, it is possible to create data column indexes such as *MinHash* [12] to quickly find relevant historical datasets and data columns for a new data column. However, as the number of historical cleaned datasets is usually limited (e.g., 8 in our experimental setting), we ignore this optimization.

6 EXPERIMENTS

Our experiments aim to answer the following questions. (1) How does our system compare to existing error detection approaches? (2) What is the impact of each feature group? (3) How does the sampling affect the system convergence? (4) How does our strategy filtering approach compare to other techniques? (5) How do user labeling errors affect the system performance? (6) How does the system performance scale in the number of rows and columns? (7) How does the choice of the classification model affect the system performance? Due to the space limitation, the last two experiments are presented in Appendix B. We also provide a case study on the user labeling procedure in Appendix C.

6.1 Setup

We evaluate our system on 8 datasets that are described in Table 4. *Hospital* and *Flights* are two real-world datasets that we have obtained along with their ground truth from

Table 4: Dataset characteristics. The error types are missing value (MV), typo (T), formatting issue (FI), and violated attribute dependency (VAD) [38].

Name	Size	Error Rate	Error Types
Hospital	1000×20	0.03	T, VAD
Flights	2376×7	0.30	MV, FI, VAD
Address	94306×12	0.14	MV, FI, VAD
Beers	2410×11	0.16	MV, FI, VAD
Rayyan	1000×11	0.09	MV, T, FI, VAD
Movies	7390×17	0.06	MV, FI
IT	2262×61	0.20	MV, FI
Tax	200000×15	0.04	T, FI, VAD

a previous research project [39]. *Address* is a proprietary dataset with ground truth. *Beers* is a real-world dataset that has been collected by scraping the web and has been manually cleaned [28]. *Rayyan* [33] and *IT* [1] are also real-world datasets that were cleaned by the dataset owners themselves. *Movies* is an available dataset in the Magellan repository [19]. We used the existing labels for the duplicate tuples to provide ground truth for this dataset. *Tax* is a large synthetic dataset from the BART repository [6]. We use this dataset to evaluate the scalability of Raha.

We leverage different evaluation measures to evaluate our system. We report precision, recall, and the F_1 score to evaluate the effectiveness. We also report the runtime in seconds to evaluate the efficiency. We report the number of labeled tuples to evaluate the human involvement. Since our tuple sampling approach is probabilistic, we report the evaluation measures as the mean of 10 independent runs. For the sake of readability, we omit the ±1% in our plots as the standard deviations are always less than 1%.

As the default parameter setting, we incorporate all the feature groups into our system. We use Gradient Boosting [26] as the classification model. We set the labeling budget of the user to $\theta_{\text{labels}} = 20$. We run all the experiments on an Ubuntu 16.04 LTS machine with 28 2.60 GHz cores and 264 GB memory. Our system is available online[1].

6.2 Comparison with the Baselines

Stand-alone error detection tools. We compare Raha with three stand-alone error detection tools in Table 5. *dBoost* [36] is an outlier detection tool. *NADEEF* [18] is a rule-based data cleaning system. *KATARA* [17] uses knowledge bases to detect errors. *ActiveClean* [32] is a machine learning-based data cleaning approach. We detail each baseline approach and its usage in Appendix A.

Table 5 shows that our approach outperforms all the stand-alone error detection tools on all the datasets in terms of F_1 score by at least 12% and up to 42%. Our configuration-free error detection system only needs a limited number of labeled tuples, i.e., $\theta_{\text{labels}} = 20$ for the reported numbers.

[1]https://github.com/BigDaMa/raha

dBoost, *NADEEF*, and *KATARA* yield low recall because they all target specific types of errors: *dBoost* marks statistical outliers as errors, *NADEEF* targets mostly errors detected by user-defined integrity rules, and *KATARA* marks as errors those data values that do not conform to the entity relationships inside the knowledge base. In other words, these tools cannot capture many of the errors detected by Raha. The low precision of *dBoost* is due to the applied heuristics, which outlying legitimate values as errors. *NADEEF* also has low precision because, similar to other integrity rules-based approaches, it reports errors in a coarse granular form, i.e., in the form of violations that consist of multiple data cells. *KATARA* has low precision due to the ambiguity of concepts that leads to a mismatch between the concepts of the data at hand and those in the knowledge base. In terms of user involvement, *dBoost*, *NADEEF*, and *KATARA* need input configurations that should be carefully provided by the user.

The poor performance of *ActiveClean* is due to its assumptions. First, *ActiveClean* works tuple wise instead of cell wise, i.e., it outputs dirty tuples instead of cells. This setting leads to poor precision as not all the data cells in an outputted tuple are actually dirty. Second, *ActiveClean* assumes the features are given by the user and *TF-IDF* featurization is just a backup plan. *TF-IDF* does not represent the data quality issues of each data cell effectively. Third, *ActiveClean* assumes the existence of a machine learning task that supports the tuple sampling strategy and is not designed for general-purpose error detection. *ActiveClean* also leveraged the same number of labeled tuples as Raha, i.e., $\theta_{\text{labels}} = 20$. In the following, we also provide a tuple-wise comparison.

Tuple-wise error detection approaches. Since *ActiveClean* was designed for detecting erroneous tuples, we also compare Raha to *ActiveClean* in terms of correctness and completeness of outputted erroneous tuples.

As shown in Table 6, Raha outperforms *ActiveClean* also in terms of tuple-wise F_1 score on all the datasets. Both systems deliver perfect results on the *Beers* and *IT* datasets because these datasets contain data columns that are completely erroneous. The tuple-wise error detection is trivial on these datasets because all the tuples are dirty. For further investigation, we removed those columns from the datasets. Now, Raha clearly outperforms *ActiveClean*; its F_1 scores are 0.95 and 1.0 on the *Beers* and *IT* datasets, respectively, while they are 0.64 and 0.90 for *ActiveClean*.

Error detection aggregators. We also compare the performance of our system to three aggregators for error detection in Figure 2. These aggregator approaches internally combine multiple error detection strategies as our system does. We detail each baseline approach and its usage in Appendix A.

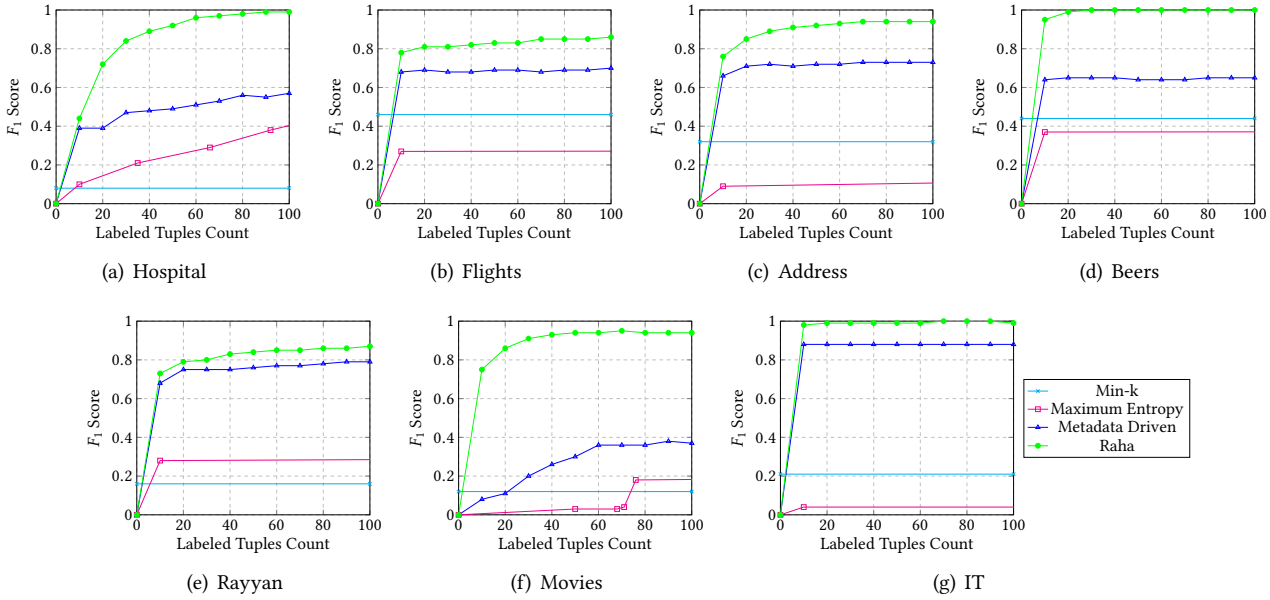
- *Min-k* [1] outputs data cells that are marked by more than k% of the error detection strategies.

Table 5: Comparison with the stand-alone error detection tools.

Approach	Hospital			Flights			Address			Beers			Rayyan			Movies			IT		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
dBoost	0.54	0.45	0.49	0.78	0.57	0.66	0.23	0.50	0.31	0.54	0.56	0.55	0.12	0.26	0.16	0.18	0.72	0.29	0.00	0.00	0.00
NADEEF	0.05	0.37	0.09	0.30	0.06	0.09	0.51	0.73	0.60	0.13	0.06	0.08	0.74	0.55	0.63	0.13	0.43	0.20	0.99	0.78	0.87
KATARA	0.06	0.37	0.10	0.07	0.09	0.08	0.25	0.99	0.39	0.08	0.26	0.12	0.02	0.10	0.03	0.01	0.17	0.02	0.11	0.17	0.14
ActiveClean	0.02	0.14	0.03	0.28	**0.94**	0.44	0.14	**1.00**	0.25	0.16	**1.00**	0.28	0.09	**1.00**	0.16	0.02	0.01	0.01	0.20	**1.00**	0.33
Raha	**0.94**	**0.59**	**0.72**	**0.82**	0.81	**0.81**	**0.91**	0.80	**0.85**	**0.99**	0.99	**0.99**	**0.81**	0.78	**0.79**	**0.85**	**0.88**	**0.86**	**0.99**	0.99	**0.99**

Table 6: Comparison in terms of detecting erroneous tuples.

Approach	Hospital			Flights			Address			Beers			Rayyan			Movies			IT		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
ActiveClean	0.33	0.12	0.18	0.80	**1.00**	0.89	0.73	**0.99**	0.84	1.00	1.00	1.00	0.77	**1.00**	0.87	0.77	**1.00**	0.87	1.00	1.00	1.00
Raha	**0.96**	**0.67**	**0.79**	**0.85**	0.93	**0.89**	**0.94**	0.94	**0.94**	**1.00**	**1.00**	**1.00**	**0.88**	0.88	**0.88**	**0.90**	0.97	**0.93**	**1.00**	**1.00**	**1.00**



(a) Hospital (b) Flights (c) Address (d) Beers

(e) Rayyan (f) Movies (g) IT

Figure 2: Comparison with the error detection aggregators.

- *Maximum entropy-based order selection* [1] outputs the union of the output of error detection strategies with a high precision on an evaluated data sample.
- *Metadata-driven approach* [45] is a machine learning-based aggregator that combines manually configured stand-alone error detection tools.

As shown in Figure 2, Raha outperforms all the aggregator approaches on all the datasets in terms of F_1 score and user labels. It converges faster requiring fewer labeled tuples. The significant superiority of Raha over *min-k* shows that Raha covers various data error types. We assume that the user knows the optimum k and plot the results of the best k.

6.3 Feature Impact Analysis

We analyze the impact of different feature groups on the performance of the system in Table 7. We run our system with all the feature groups (row *All*). Then, we exclude each feature group, one at a time, to analyze its impact. For example, *All - OD* means Raha leverages all the feature groups but the outlier detection ones. Here, we also report the effectiveness of Raha when it uses *TF-IDF* features, which is the featurization method of *ActiveClean* [32].

As shown in Table 7, Raha is robust against removing feature groups as its performance does not collapse when a feature group is excluded. However, depending on the error rate and prevalent data error types, removing a feature group could reduce the performance more significantly. For example on the *Hospital* dataset, with hundreds of FDs, removing the rule violation detection features decreases the performance more severely. On the *Movies* dataset, where the error rate is low and thus the data errors are mainly outliers, removing the outlier detection features decreases the performance more severely.

Table 7: System effectiveness with different feature groups: outlier detection (OD), pattern violation detection (PVD), rule violation detection (RVD), knowledge base violation detection (KBVD), and all together (All).

Feature Group	Hospital			Flights			Address			Beers			Rayyan			Movies			IT		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
TF-IDF	0.98	0.10	0.18	0.63	0.88	0.73	0.84	0.57	0.68	0.73	0.80	0.76	0.91	0.58	0.70	0.19	0.04	0.07	0.92	0.97	0.95
All - OD	0.93	0.53	0.68	0.80	0.85	0.82	0.89	0.84	0.86	0.95	0.95	0.95	0.78	0.72	0.75	0.66	0.82	0.73	0.99	0.98	0.98
All - PVD	0.95	0.62	0.75	0.83	0.84	0.83	0.87	0.89	0.88	0.92	0.94	0.93	0.74	0.74	0.74	0.77	0.84	0.80	0.98	0.97	0.97
All - RVD	0.75	0.37	0.50	0.80	0.78	0.79	0.86	0.87	0.86	0.97	0.98	0.97	0.82	0.78	0.80	0.92	0.90	0.91	0.99	0.98	0.98
All - KBVD	0.95	0.60	0.74	0.85	0.78	0.81	0.85	0.76	0.80	0.98	0.98	0.98	0.83	0.76	0.79	0.80	0.88	0.84	0.99	0.97	0.98
All	0.94	0.59	0.72	0.82	0.81	0.81	0.91	0.80	0.85	0.99	0.99	0.99	0.81	0.78	0.79	0.85	0.88	0.86	0.99	0.99	0.99

Interestingly, the *TF-IDF* featurization of *ActiveClean* leads to higher F_1 score using Raha's sampling approach. However, the overall performance with word-level *TF-IDF* features is always worse than the full feature set of Raha.

6.4 Sampling Impact Analysis

We analyze the impact of our sampling approach on the performance of the system in Figure 3. In particular, we compare two versions of our system with two different sampling approaches. *Uniform sampling* approach selects tuples for user labeling according to a uniform probability distribution [45]. On the other hand, our proposed *clustering-based sampling* approach first selects tuples based on the existing clusters of data cells and then propagates the labels through the clusters.

As shown in Figure 3, our clustering-based sampling approach speeds up the convergence of the system. This higher convergence speed is more obvious on datasets with lower error rates, such as *Hospital* and *Movies*. On datasets with low error rates, it is harder to find enough dirty data cells to train the classifiers. However, our clustering-based sampling approach addresses this issue by using additional noisy labels from each cluster.

6.5 Strategy Filtering Impact Analysis

We analyze the effect of strategy filtering via historical data on the performance of Raha. Using the datasets at hand, we set up the experiment the following way. For each run, we consider one of the datasets as the new dirty dataset d_{new} and the rest of datasets as the set of historical datasets D, according to the well-known *leave-one-out* methodology [40]. Then, we select only the promising error detection strategies according to the approach described in Section 5. This way, we analyze the effect of limited computational resources for feature extraction. Note that we consider the runtime of strategy filtering (Section 5) and feature extraction (Section 4.2). Feature extraction clearly dominates the runtime of our system as it requires running multiple error detection strategies on the dataset.

Ideally, historical data should contain datasets from the same domain of the new dataset. However, the assumption that the user always has similar datasets in the historical data is strong. Thus, we only assume that the user has some cleaned datasets in historical data that may have some similar data columns (e.g., *City* and *ZIP*) to the new dataset. Even within our set of diverse datasets from different domains, we still can find similar columns and filter irrelevant strategies to improve the runtime.

Figure 4 shows the runtime of Raha with and without strategy filtering. The runtime is significantly improved by more than an order of magnitude as the system needs to run only a fraction of all possible error detection strategies.

We also compare our strategy filtering approach via historical data to other strategy filtering approaches to better evaluate its effectiveness. Figure 5 shows the F_1 score of five different approaches. We leverage the ground truth of datasets to evaluate the F_1 score of all the error detection strategies to sort them accordingly. Thus, the first and second approaches are Raha with the least and most effective strategies as features. These two extreme approaches could be a lower and an upper bound for the effectiveness of any other strategy filtering approach. The third approach is Raha with a uniform strategy filtering that uniformly picks error detection strategies as the features. Since this approach is probabilistic, we repeat it 5 times and report the mean and standard deviation. The fourth and fifth approaches are Raha with our strategy filtering via historical data and Raha without any strategy filtering, respectively. Note that the number of selected strategies is the same for all the strategy filtering approaches and is computed by Raha.

As shown in Figure 5, our strategy filtering approach via historical data outperforms the least effective strategies selection and the uniform strategy selection approaches. The effectiveness of Raha with strategy filtering via historical data is slightly lower than the most effective strategies selection approach. Our strategy filtering via historical data approach can achieve almost the same effectiveness without running or evaluating any strategy on the new dataset. Note that the effectiveness of Raha with no strategy filtering is higher than the most effective strategies selection approach. This shows that even ineffective error detection strategies still can add some information to the feature vector.

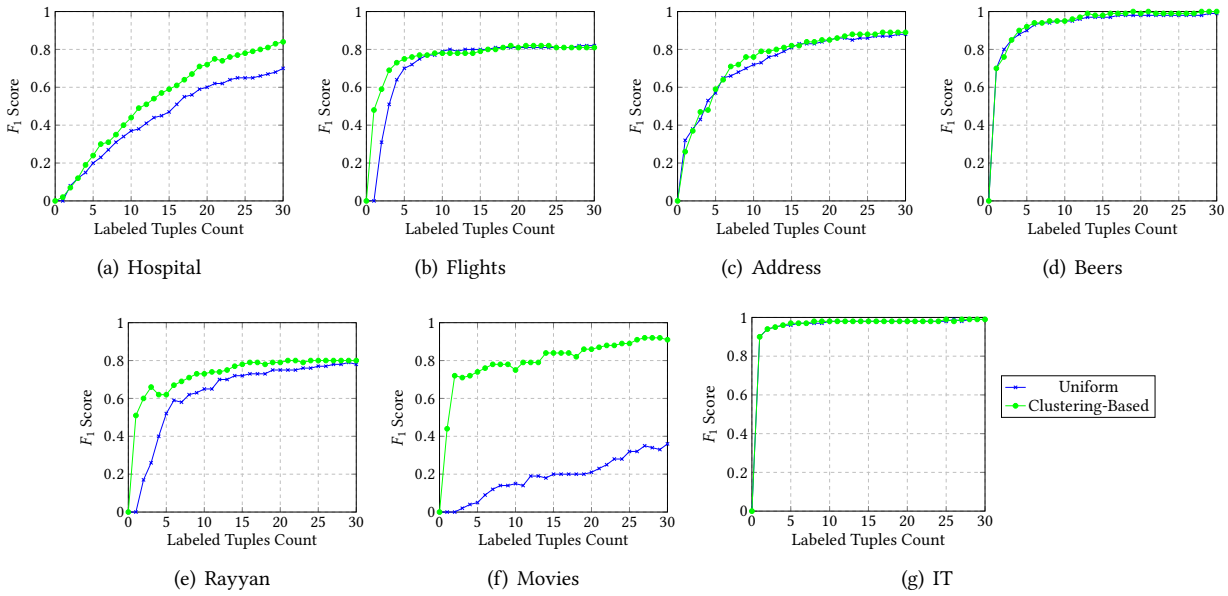


Figure 3: System effectiveness with different sampling approaches.

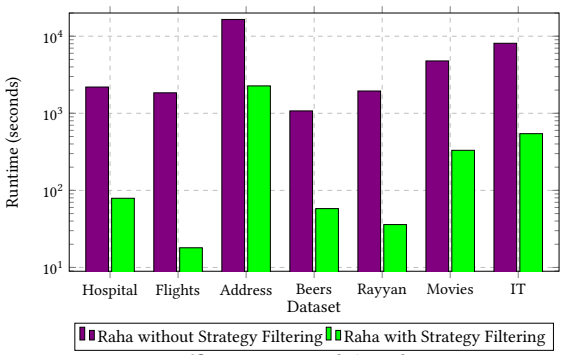


Figure 4: System efficiency with/without strategy filtering via historical data.

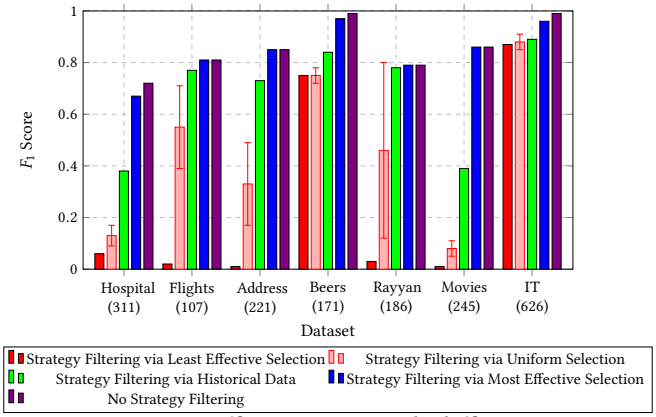


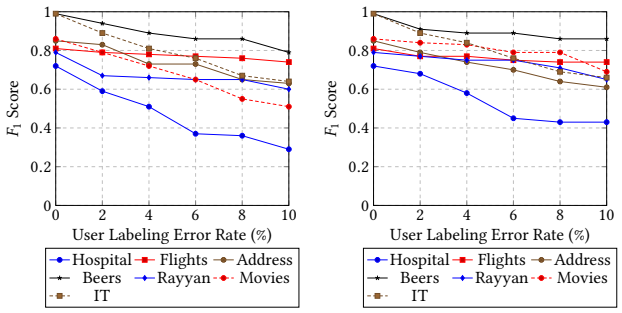
Figure 5: System effectiveness with different strategy filtering approaches. The numbers of selected strategies are denoted inside the brackets.

Generally, the results of the approach based on historical data show how column profiles effectively capture the data quality issues of the datasets. For example, the selected promising strategies for the *Hospital* dataset are mainly Histogram modeling strategies that are effective on this dataset. On the other hand, the selected promising strategies for the *Address* dataset are mainly functional dependency checkers that involve attributes *ZIP*, *City*, and *State*. Our column profiles could effectively map these data columns to similar data domains in *Hospital* and *Beers*.

6.6 User Labeling Error Impact Analysis

In Figure 6, we compare the conflict resolution functions for label propagation from Section 4.4 in the presence of user labeling errors. The erroneous user labels are randomly distributed across data cells.

As shown in Figure 6, Raha's effectiveness drops slightly with increased user labeling errors. The decline is more severe on the *Hospital* dataset because this dataset contains many similar data errors, i.e., typos by random injection of the character "x". Having wrong user labels for such similar data errors confuses the classifiers. The majority-based conflict resolution function fares better than its homogeneity-based counterpart in the presence of user labeling errors because the latter does not operate any label propagation if user's labels are contradictory. The majority-based conflict resolution function is more robust in these situations. Under the assumption of perfect user labels, both methods perform nearly equal. Since both solutions cannot be considered as



(a) Homogeneity Resolution (b) Majority Resolution

Figure 6: System effectiveness in the presence of user labeling errors with (a) homogeneity-based and (b) majority-based conflict resolution functions.

robust on all datasets, we believe that handling erroneous user labels is an interesting direction for future research.

7 RELATED WORK

Preconfigured error detection. This line of work assumes that an error detection tool is well configured before being used. For outlier detection, users have to manually set the parameters [9]. For rule violation detection, users have to provide functional dependencies (FDs) [4], conditional functional dependencies (CFDs) [23] and its variants [11, 31], denial constraints (DCs) [16], or user-defined functions (UDFs) [8, 18]. Properly configuring all tools is tedious, which requires the user to know both the dataset and the error detection tools. In contrast, Raha relieves the user from the tedious task of tool configuration and still outperforms the preconfigured approaches.

Aggregated error detection. Another line of research aggregates the results of various error detection techniques through voting, precision-based ordering [1] or ensembling methods [45]. Again the assumption for these approaches is that the user has already configured each error detection tool. Raha does not make such assumptions as we automatically generate many configurations. As our experiments show, Raha clearly outperforms these techniques.

Interactive error detection. Instead of using preconfigured error detection tools, another line of work is to have human in the loop. To clean the data, the backend data cleaning engine generalizes the user feedback through data transformations (e.g., Trifacta [21], BlinkFill [42], and Falcon [27]) or machine learning models (e.g., GDR [46], and ActiveClean [32]). The general challenge of the human-in-the-loop framework is the class imbalance problem, where the number of dirty data values is much lower than the clean values. This class imbalance setting usually leads to a large number of required user labels. In contrast, Raha exploits clustering-based sampling with label propagation that significantly reduces the number of required user labels.

Data profiling. Many data profiling algorithms have been studied to detect FDs [2, 10], CFDs [24], and DCs [15]. However, discovering rules from dirty data will produce many false positives [15], requiring the users to pick genuine rules from discovered candidate rules. That is why Raha does not require error detection strategies to be perfect.

Data repairing. There have been a great many works on data repairing, such as ActiveClean [32], HoloClean [39], and NADEEF [18]. Our work is complementary to data repairing as we focus on error detection. The output of Raha, i.e., detected data errors, later would be inputted to these data repairing approaches. However, we compared Raha to the error detection components of these systems.

8 CONCLUSION

We proposed a novel error detection system that relieves the user from the tedious task of selecting and configuring error detection algorithms. Raha systematically generates a wide range of algorithm configurations and encodes their output into feature vectors for all data cells. Raha then combines clustering, label propagation, and classification per data column and learns to predict the labels of all data cells inside each data column. Furthermore, we proposed a strategy filtering approach to filter irrelevant error detection strategies based on the optionally available historical data. Our experiments clearly show how Raha outperforms existing baselines.

While Raha delivers outstanding performance in a hassle-free way, it has some limitations. The configuration-free nature of Raha requires us to run many error detection strategies. Like many machine learning approaches, Raha cannot provide any guarantee that with how many algorithms, configurations, or even user labels, the user can get the desired performance. Raha tries to limit the infinite set of all possible algorithms/configurations with some heuristics. These search space pruning heuristics exclude some potentially relevant strategies, such as temporal functional dependencies.

There are several possible improvements to Raha. The first is runtime optimization by parallelizing the workflow. Although Raha supports strategy filtering, its pipeline is still sequential. The second is to reduce the user labeling cost by replacing expert users with crowdsourcing, which requires handling noisy labels. Handling user labeling errors and providing the user with important context and metadata for the labeling process are important future research directions. Currently, Raha requires the user to label one tuple at a time, which might not provide enough context for identifying some semantic error types, such as functional dependency violations, and systematic data errors. Finally, we also plan to extend our system to incorporate data repairing as well.

ACKNOWLEDGMENTS

This project has been supported by the German Research Foundation (DFG) under grant agreement 387872445.

REFERENCES

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

[2] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. 2015. Profiling relational data: a survey. *VLDBJ* 24, 4, 557–581.

[3] Ziawasch Abedjan, John Morcos, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, and Michael Stonebraker. 2016. DataXFormer: A robust transformation discovery system. In *ICDE*. 1134–1145.

[4] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of databases: the logical level*. Addison-Wesley Longman Publishing Co., Inc.

[5] Charu C Aggarwal and Chandan K Reddy. 2013. *Data clustering: algorithms and applications*. CRC Press.

[6] 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. *PVLDB* 9, 2, 36–47.

[7] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. DBpedia: A nucleus for a web of open data. In *ISWC*. 722–735.

[8] Omar Benjelloun, Hector Garcia-Molina, David Menestrina, Qi Su, Steven Euijong Whang, and Jennifer Widom. 2009. Swoosh: a generic approach to entity resolution. *VLDBJ* 18, 1, 255–276.

[9] Laure Berti-Equille, Tamraparni Dasu, and Divesh Srivastava. 2011. Discovery of complex glitch patterns: A novel approach to quantitative data cleaning. In *ICDE*. 733–744.

[10] Laure Berti-Equille, Hazar Harmouch, Felix Naumann, Noël Novelli, and Saravanan Thirumuruganathan. 2018. Discovery of genuine functional dependencies from relational data with missing values. *PVLDB* 11, 8, 880–892.

[11] Loreto Bravo, Wenfei Fan, Floris Geerts, and Shuai Ma. 2008. Increasing the expressivity of conditional functional dependencies without extra complexity. In *ICDE*.

[12] Andrei Z Broder. 1997. On the resemblance and containment of documents. In *Compression and complexity of sequences*. 21–29.

[13] Olivier Chapelle, Jason Weston, and Bernhard Schölkopf. 2003. Cluster kernels for semi-supervised learning. In *NIPS*. 601–608.

[14] Xu Chu and Ihab F Ilyas. 2016. Qualitative data cleaning. *PVLDB* 9, 13, 1605–1608.

[15] Xu Chu, Ihab F Ilyas, and Paolo Papotti. 2013. Discovering denial constraints. *PVLDB* 6, 13, 1498–1509.

[16] Xu Chu, Ihab F Ilyas, and Paolo Papotti. 2013. Holistic data cleaning: Putting violations into context. In *ICDE*. 458–469.

[17] Xu Chu, John Morcos, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. 2015. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *SIGMOD*. 1247–1261.

[18] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed Elmagarmid, Ihab F Ilyas, Mourad Ouzzani, and Nan Tang. 2013. NADEEF: a commodity data cleaning system. In *SIGMOD*. 541–552.

[19] Sanjib Das, AnHai Doan, Paul Suganthan G. C., Chaitanya Gokhale, and Pradap Konda. 2015. The Magellan Data Repository. https://sites.google.com/site/anhaidgroup/projects/data.

[20] Dong Deng, Raul Castro Fernandez, Ziawasch Abedjan, Sibo Wang, Michael Stonebraker, Ahmed K Elmagarmid, Ihab F Ilyas, Samuel Madden, Mourad Ouzzani, and Nan Tang. 2017. The data civilizer system. In *CIDR*.

[21] Trifacta Wrangler Enterprise. 2016. Trifacta wrangler (2015).

[22] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *SIGKDD*. 226–231.

[23] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional functional dependencies for capturing data inconsistencies. *TODS* 33, 2, 6.

[24] Wenfei Fan, Floris Geerts, Jianzhong Li, and Ming Xiong. 2011. Discovering conditional functional dependencies. *TKDE* 23, 5, 683–698.

[25] Thomas A Feo and Mauricio GC Resende. 1989. A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters* 8, 2, 67–71.

[26] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 1189–1232.

[27] Jian He, Enzo Veltri, Donatello Santoro, Guoliang Li, Giansalvatore Mecca, Paolo Papotti, and Nan Tang. 2016. Interactive and deterministic data cleaning. In *SIGMOD*. 893–907.

[28] Jean-Nicholas Hould. 2017. Craft Beers Dataset. https://www.kaggle.com/nickhould/craft-cans. Version 1.

[29] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive visual specification of data transformation scripts. In *SIGCHI*. 3363–3372.

[30] Richard M Karp. 1972. Reducibility among combinatorial problems. In *Complexity of computer computations*. 85–103.

[31] Nick Koudas, Avishek Saha, Divesh Srivastava, and Suresh Venkatasubramanian. 2009. Metric functional dependencies. In *ICDE*. 1275–1278.

[32] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J Franklin, and Ken Goldberg. 2016. ActiveClean: Interactive data cleaning for statistical modeling. *PVLDB* 9, 12, 948–959.

[33] Mourad Ouzzani, Hossam Hammady, Zbys Fedorowicz, and Ahmed Elmagarmid. 2016. Rayyan—a web and mobile app for systematic reviews. *Systematic reviews* 5, 1, 210.

[34] Thorsten Papenbrock and Felix Naumann. 2016. A hybrid approach to functional dependency discovery. In *SIGMOD*. 821–833.

[35] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *JMLR* 12, Oct, 2825–2830.

[36] Clement Pit-Claudel, Zelda Mariet, Rachael Harding, and Sam Madden. 2016. *Outlier detection in heterogeneous datasets using automatic tuple expansion*. Technical Report MIT-CSAIL-TR-2016-002. CSAIL, MIT.

[37] Erhard Rahm and Philip A Bernstein. 2001. A survey of approaches to automatic schema matching. *VLDBJ* 10, 4, 334–350.

[38] Erhard Rahm and Hong Hai Do. 2000. Data cleaning: Problems and current approaches. *DE* 23, 4, 3–13.

[39] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. Holoclean: Holistic data repairs with probabilistic inference. *PVLDB* 10, 11, 1190–1201.

[40] Claude Sammut and Geoffrey I Webb. 2011. *Encyclopedia of machine learning*. Springer Science & Business Media.

[41] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. 2008. *Introduction to information retrieval*. Cambridge University Press.

[42] Rishabh Singh. 2016. Blinkfill: Semi-supervised programming by example for syntactic string transformations. *PVLDB* 9, 10, 816–827.

[43] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. 2017. Revisiting unreasonable effectiveness of data in deep learning era. In *ICCV*. 843–852.

[44] Graham Upton and Ian Cook. 2014. *A dictionary of statistics 3e*. Oxford University Press.

[45] Larysa Visengeriyeva and Ziawasch Abedjan. 2018. Metadata-driven error detection. In *SSDBM*. 1–12.
[46] Mohamed Yakout, Ahmed K Elmagarmid, Jennifer Neville, Mourad Ouzzani, and Ihab F Ilyas. 2011. Guided data repair. *PVLDB* 4, 5, 279–289.

APPENDIX

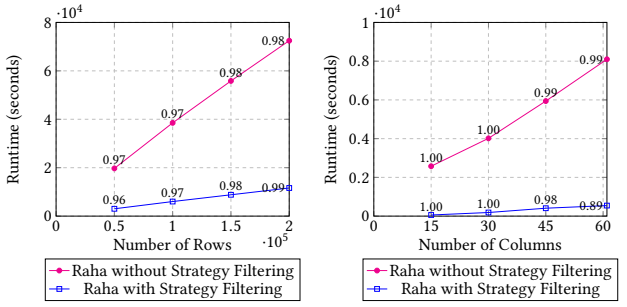
We first detail our baselines. Then, we present two more supplementary experiments. Finally, we provide a case study on the user labeling procedure.

A DETAILS OF THE BASELINES

The detailed description of the baselines is as follows.

- *dBoost* [36] is an outlier detection tool that contains several algorithms such as histogram and Gaussian modeling. dBoost also provides tuple expansion feature to expand string values into numerical values and find string outliers as well. We applied grid search to configure dBoost algorithms. In other words, we ran differently configured versions of dBoost algorithms and evaluated the results on a data sample to report the best-configured algorithm with the highest performance numbers.
- *NADEEF* [18] is a rule-based data cleaning system that takes integrity rules in the form of denial constraints and outputs the violating data cells. We ran NADEEF with data constraints (i.e., integrity rules and column patterns) that are provided by the datasets owners.
- *KATARA* [17] is a data cleaning system powered by knowledge bases that takes a set of entity relationships and outputs the violating data cells. We ran KATARA with all the entity relationships that are available in the DBpedia knowledge base [7].
- *ActiveClean* [32] is a machine learning-based data cleaning approach. *ActiveClean* uses *TF-IDF* features to train a classifier that can identify dirty tuples. Tuples are sampled for labeling based on their usefulness for a given machine learning application. Since our machine learning task is error detection, we sample the tuples based on their probability of being dirty. In fact, we sample dirty tuples that are outputted by the error detection classifier with a uniform probability. We also report the *ActiveClean* performance with the same number of labeled tuples as Raha, i.e., 20 tuples.
- *Min-k* [1] is a simple aggregation approach that outputs data cells that are marked by more than k% of the error detection strategies. We ran the approach with $k \in \{0\%, 20\%, 40\%, 60\%, 80\%\}$ and report the highest performance numbers. Note that $k = 0\%$ corresponds to the union of all strategies' outputs.
- *Maximum entropy-based order selection* [1] evaluates the error detection strategies on a data sample. Then, the approach picks the error detection strategy with the highest precision first, evaluates its output, and picks the next best strategy. Whenever the approach picks one strategy, there is a point on the chart. Usually, picking the first error detection strategy requires the evaluation of more than 100 tuples.
- *Metadata driven approach* [45] combines the output of manually configured stand-alone error detection tools and metadata in a feature vector. Then, the approach trains ensemble classifiers with the help of the feature vectors and a labeled data sample. We use this aggregator on top of the stand-alone tools, using the same best-effort configurations shown in Table 8.

Table 8 shows the configuration of stand-alone error detection tools for each dataset.



(a) Scalability w.r.t. Rows Count (b) Scalability w.r.t. Columns Count

Figure 7: Scalability w.r.t. (a) the number of rows and (b) the number of columns. The number on each point depicts the F_1 score.

B SUPPLEMENTARY EXPERIMENTS

B.1 System Scalability

Figure 7 shows the runtime and F_1 score of Raha on *Tax* and *IT* datasets, by varying the number of rows and columns. While the runtime increases linearly, the F_1 score stays almost the same. Interestingly, the runtime of Raha with strategy filtering reflects a much steeper linear growth.

B.2 Classification Model Impact Analysis

We analyze the impact of the classification model on the performance of the system in Table 9. In particular, we tested *AdaBoost*, *Decision Tree*, *Gradient Boosting*, *Gaussian Naive Bayes*, *Stochastic Gradient Descent*, and *Support Vectors Machines*, all implemented in *scikit-learn* Python module [35]. Note that we applied grid search to find the best hyperparameters for each classification model.

As shown in Table 9, the choice of classification model does not affect the performance of the system significantly because the main impact of the system comes from the features

Table 8: Configuration of stand-alone error detection tools for each dataset.

Dataset	dBoost	NADEEF	KATARA	ActiveClean
Hospital	Histogram, 0.9, 0.1	city → zip, city → county, zip → city, zip → state, zip → county, county → state, index (digits), provider number (digits), zip (5 digits), state (2 alphabets), phone (digits)	All Entity Relationships in DBpedia	TF-IDF Features, Sampling Based on Erroneous Probability, 20 Labeled Tuples
Flights	Gaussian, 1.3	flight → actual departure time, flight → actual arrival time, flight → scheduled departure time, flight → scheduled arrival time	All Entity Relationships in DBpedia	TF-IDF Features, Sampling Based on Erroneous Probability, 20 Labeled Tuples
Address	Gaussian, 1.0	address → state, address → zip, zip → state, state (2 alphabets), zip (digits), ssn (digits)	All Entity Relationships in DBpedia	TF-IDF Features, Sampling Based on Erroneous Probability, 20 Labeled Tuples
Beers	Histogram, 0.9, 0.3	brewery id → brewery name, brewery id → city, brewery id → state, brewery id (digits), state (2 alphabets)	All Entity Relationships in DBpedia	TF-IDF Features, Sampling Based on Erroneous Probability, 20 Labeled Tuples
Rayyan	Histogram, 0.7, 0.3	journal abbreviation → journal title, journal abbreviation → journal issn, journal issn → journal title, authors list (not null), article pagination (not null), journal abbreviation (not null), article title (not null), article language (not null), journal title (not null), journal issn (not null), article journal issue (not null), article journal volume (not null), journal created at (date)	All Entity Relationships in DBpedia	TF-IDF Features, Sampling Based on Erroneous Probability, 20 Labeled Tuples
Movies	Gaussian, 2.5	id ("tt" + digits), year (4 digits), rating value (float), rating count (digits), duration (digits + "min")	All Entity Relationships in DBpedia	TF-IDF Features, Sampling Based on Erroneous Probability, 20 Labeled Tuples
IT	Gaussian, 1.0	support level (not null), app status (not null), curr status (not null), tower (not null), end users (not null), account manager (not null), decomm dt (not null), decomm start (not null), decomm end (not null), end users (not 0), retirement (not out of defined domain), emp dta (not out of defined domain), retire plan (not out of defined domain), division (not out of defined domain), bus import (not out of defined domain)	All Entity Relationships in DBpedia	TF-IDF Features, Sampling Based on Erroneous Probability, 20 Labeled Tuples

Table 9: System effectiveness with different classification models.

Classification Model	Hospital			Flights			Address			Beers			Rayyan			Movies			IT		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
AdaBoost	0.96	0.56	0.70	0.83	0.79	0.81	0.92	0.82	0.87	0.99	1.00	1.00	0.80	0.78	0.78	0.81	0.87	0.84	0.98	0.98	0.98
Decision Tree	0.95	0.57	0.71	0.82	0.79	0.80	0.90	0.81	0.85	1.00	0.99	0.99	0.79	0.75	0.77	0.82	0.87	0.84	0.98	0.98	0.98
Gradient Boosting	0.94	0.59	0.72	0.82	0.81	0.81	0.91	0.80	0.85	0.99	0.99	0.99	0.81	0.78	0.79	0.85	0.88	0.86	0.99	0.99	0.99
Gaussian Naive Bayes	0.99	0.45	0.61	0.88	0.69	0.77	0.84	0.84	0.84	1.00	0.97	0.98	0.79	0.62	0.69	0.53	0.88	0.66	0.99	0.98	0.98
Stochastic Gradient Descent	0.89	0.52	0.65	0.84	0.76	0.80	0.83	0.73	0.78	0.99	0.98	0.98	0.82	0.78	0.80	0.80	0.85	0.82	0.99	0.97	0.98
Support Vectors Machines	0.98	0.39	0.56	0.83	0.77	0.80	0.66	0.36	0.47	0.99	0.96	0.98	0.82	0.79	0.80	0.74	0.85	0.79	0.97	0.98	0.98

and the sampling approach. In our current prototype, we deploy the Gradient Boosting classification model because it is an advanced ensemble learning model that internally combines multiple simpler classification models [26].

C CASE STUDY: USER LABELING

We provide a case study to clarify the user labeling procedure. Table 10 shows the 20 tuples that Raha sampled on the *Flights* dataset. As mentioned in Section 4, the user takes one tuple at a time and labels its data cells. Ideally, the user should label all the red cells as dirty and the rest as clean.

The *Flights* dataset contains errors of different types [38]. Missing values, such as the scheduled departure time in tuple *3*, and formatting issues, such as *11:25aDec 1* as the scheduled departure time in tuple *4*, can easily be identified by a user. The user usually identifies these data error types via either domain expertise or some sort of master data. However, the tuples also include erroneous values that are not easily detectable. For example, the reported departure time *2:03 p.m.* in tuple *3* does not look erroneous without further investigation. Although this data value seems correct, it is not the correct departure time of the flight *AA-1279-DFW-PHX*. In other words, this value violates the functional dependency *Flight* → *Actual Departure Time*. In this particular case, the user needs to look at a master data and check the actual departure time of the flight *AA-1279-DFW-PHX*. Note that knowing the violated functional dependency is not a guarantee for a correct label as typically multiple data cells are involved in the violation of a functional dependency, e.g., left-hand side or right-hand side value. Thus, not every involved cell in a violation can be considered to be an error.

We also report the F_1 score of Raha on erroneous columns of the *Flights* dataset in Figure 8. Whenever a labeled tuple covers a new unseen data error type, the F_1 score is improved significantly. For example, by labeling tuple 2, the F_1 score improves significantly on columns *Actual Departure Time* and *Actual Arrival Time* as Raha learns the missing value data error type for these columns. Labeling tuple 3 improves the F_1 score on column *Actual Departure Time* as

Table 10: 20 **tuples that the user labeled on the *Flights* dataset. The red data cells are dirty and the rest are clean.**

ID	Source	Flight	Scheduled Departure Time	Actual Departure Time	Scheduled Arrival Time	Actual Arrival Time
1	aa	AA-3-JFK-LAX	12:00 p.m.	12:11 p.m.	3:15 p.m.	3:16 p.m.
2	usatoday	AA-1886-BOS-MIA	10:45 a.m.		2:20 p.m.	
3	airtravelcenter	AA-1279-DFW-PHX		2:03 p.m.		3:13 p.m.
4	mytripandmore	AA-1544-SAN-ORD	11:25aDec 1	11:20aDec 1	5:25 p.m.	4:56 p.m.
5	weather	UA-2704-DTW-PHX	11:15 a.m.		1:40 p.m.	
6	flightaware	AA-431-MIA-SFO	8:35 a.m.	8:51 a.m.	11:22 a.m.	11:33 a.m.
7	panynj	AA-404-MIA-MCO	6:45 a.m.	6:58 a.m.	7:45 a.m.	7:32 a.m.
8	quicktrip	AA-3823-LAX-DEN	9:00 p.m.	9:06 p.m. (Estimated)	12:15 a.m.	11:49 p.m. (Estimated)
9	foxbusiness	AA-3-JFK-LAX	12:00 p.m.	12:12 p.m.	3:15 p.m.	3:16 p.m.
10	orbitz	UA-6273-YYC-SFO	7:35aDec 1	7:27aDec 1	9:43aDec 1	8:45aDec 1
11	flightarrival	UA-828-SFO-ORD	11:08 p.m.		5:11 a.m.Dec 02	
12	travelocity	UA-3515-IAD-MSP	Not Available	8:24 a.m.	Not Available	9:56 a.m.
13	aa	AA-3063-SLC-LAX	8:20 p.m.	8:39 p.m.	9:20 p.m.	
14	helloflight	AA-1733-ORD-PHX		7:59 p.m.		10:31 p.m.
15	orbitz	AA-616-DFW-DTW	10:00aDec 1	9:59aDec 1	12:35 p.m.	1:27 p.m.
16	gofox	UA-2906-PHL-MCO	3:50 p.m.	4:46 p.m.	6:23 p.m.	6:35 p.m.
17	weather	AA-2268-PHX-ORD	7:15 a.m.	7:23 a.m.	11:35 a.m.	11:04 a.m.
18	flylouisville	AA-466-IAH-MIA	6:00 a.m.	6:08 a.m.	9:20 a.m.	9:05 a.m.
19	panynj	AA-3-JFK-LAX	12:00 p.m.	12:12 p.m.	3:15 p.m.	3:16 p.m.
20	flylouisville	UA-1500-IAH-GUA		9:16 a.m.		11:45 a.m.

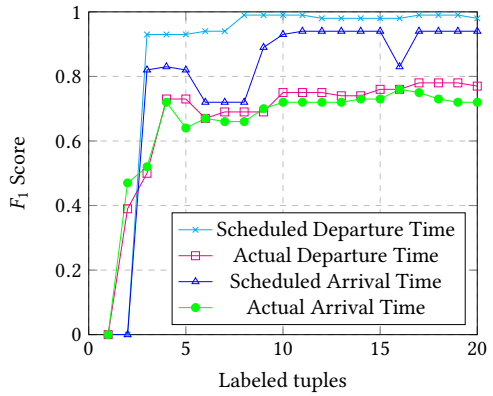


Figure 8: System effectiveness on different columns of the *Flights* dataset.

well, because the classifier of this column learns that the value is erroneous due to the violation of a functional dependency. In fact, the classifier learns that the FD checker feature $s_{\text{Flight}\rightarrow\text{Actual Departure Time}}$ is an important feature for identifying data errors in column *Actual Departure Time*. Labeling tuple 4 also improves the F_1 score on column *Actual Departure Time*, because the classifier of this column learns that values with the same type of formatting as *11:20aDec 1* are not desired. Interestingly, after labeling these four tuples, the F_1 score of Raha on column *Actual Departure Time* almost converges, because there are no more different data error types in this column. Note that upon labeling some tuples, the F_1 score of Raha might slightly drop on some columns. For example, labeling tuple 6 slightly drops the F_1 score of Raha on column *Scheduled Arrival Time*. While the classifier of this column learns that the value *11:22 a.m.* is an error, the classifier overfits and labels some clean values with a similar format as data errors. However, the F_1 score of the classifier again increases in later iterations by observing a few more similar clean values. In fact, the classifier learns that, in contrast to other clean data cells, the cell with the value *11:22 a.m.* is erroneous due to a functional dependency violation.