Semi-Supervised Data Cleaning with Raha and Baran

Mohammad Mahdavi
Technische Universität Berlin
mahdavilahijani@tu-berlin.de

Ziawasch Abedjan
Leibniz Universität Hannover
L3S Research Center
abedjan@dbs.uni-hannover.de

ABSTRACT

Data cleaning is a tedious data preparation task, which typically needs user supervision in the form of predefined configurations, such as rules, parameters, or patterns. We have recently developed two configuration-free systems, Raha and Baran, to detect and correct data errors in a semi-supervised manner. In this paper, we demonstrate how both systems can be used within an end-to-end data cleaning pipeline. Our demonstration shows how user supervision can be reduced to a negligible amount of example corrections using effective feature representation, label propagation, and transfer learning methods. While each cleaning step, detection and correction, faces substantially different challenges, we have designed the corresponding systems based on the same intuition. Both systems internally leverage an automatically generatable set of base detectors and correctors and learn to combine them using a few user labels. In practice, with a small number of 20 user-annotated tuples, it is possible to effectively identify and fix data quality problems inside a dataset. Furthermore, both systems benefit from knowledge of prior cleaning tasks. Using transfer learning, both systems can optimize the data cleaning task at hand in terms of error detection runtime and error correction effectiveness.

1. INTRODUCTION

Data quality management has always been one of the hardest tasks for automation. In fact, data cleaning is one of the most important but time-consuming tasks for data scientists [7]. The data cleaning task consists of two major steps: (1) *error detection* and (2) *error correction*. The goal of error detection is to identify data values that are wrong/dirty [1]. The goal of error correction is to fix these wrong values [19].

There has been already a large amount of research on end-to-end data cleaning as well as some of its sub-problems. Most traditional data cleaning systems follow the *preconfiguration paradigm*, which requires the user to provide the correct and complete set of rules [8, 3], parameters [23], or both [19]. For example, a data cleaning system may need data patterns, such as the date format ”dd.mm.yyyy”, or statistical parameters, such as expected mean and standard deviation. However, providing the correct and complete set of rules and parameters upfront is a major impediment for most non-expert users as they need to know both the dataset and the data cleaning system very well to be able to configure the systems properly [1, 15, 23].

Recently, we proposed the *configuration-free paradigm* for data cleaning, which is more suitable for users who are domain experts but are not adept at generating configurations for complex data cleaning systems. In particular, we proposed our error detection system Raha [15] and its pendant system Baran [14] for error correction. Both systems follow the same unified idea: They internally aggregate a set of base error detectors/correctors in a semi-supervised manner. The user is only in the loop to annotate selected data points, i.e., marking or fixing a few data errors. In other words, the user does not need to provide any data- or tool-dependent configurations.

Each of the two systems learns to generalize the user-provided error detection/correction examples to the rest of dataset, accordingly. For this purpose, each system requires a different type of feature construction to describe and expose erroneous values. Since data errors are oftentimes dataset and use-case dependent, not every base detector will be useful for all possible datasets. In particular, the parameter configurations of detectors, such as numerical parameters of simple outlier detection techniques, depend on the characteristics of the dataset at hand. In contrast to other error detection aggregators [1, 21] that require the user to configure the base detectors, Raha is designed to be oblivious of the quality of the detectors as long as there is a sufficiently large and diverse set of them available. Raha instead uses the output of these detectors to cluster the values. With the help of user examples, Raha then identifies clusters of dirty values and propagates user labels to improve the classification accuracy.

Baran, on the other hand, requires a feature vector to encode a large set of possible correction candidates for the detected data errors. In theory, every possible string can be the correction of a data error. By leveraging the most accessible error contexts, namely the value itself, its domain, and correlating values in neighboring columns, Baran generates a large set of potential corrections that are then effectively filtered through human supervision.

Furthermore, both Raha and Baran are designed in a way that they can benefit from transfer learning [17]. In fact,

11th Annual Conference on Innovative Data Systems Research (CIDR '21) January 10-13, 2021, Chaminade, USA.

both Raha and Baran can learn from previously cleaned datasets to improve data cleaning performance and reduce user involvement on the current dataset. Raha calculates the similarity of the current dataset with previously cleaned datasets to prune irrelevant error detectors for the dataset at hand. This way, Raha reduces the overall data cleaning runtime without harming the effectiveness of the overall detection process. Baran extracts value-based corrections from any cleaned dataset to pretrain base correctors. These pretrained correctors can later be fine-tuned on the dataset at hand. The pretraining not only enables Baran to obtain head-topic corrections, such as fixing "US" to "United States", without the help of user annotations, but also generates more correction evidence that improves the convergence rate of the underlying learning algorithm.

While in the previous papers, we explained the theoretical aspects of our systems Raha [15] and Baran [14], in this paper, we focus on the technical aspects of both systems individually as well as an end-to-end data cleaning pipeline that streamlines Raha and Baran. In particular, we demonstrate the following aspects:

- We show how Raha and Baran generalize a few user-annotated error/correction examples to the rest of dataset. Our demo enables drill down into similar groups of data errors, how they are clustered using a wide range of error detectors, and how suggested corrections are generated.

- We show how Raha and Baran use previously cleaned datasets and transfer learning to improve the data cleaning performance and reduce user involvement in the data cleaning process.

- We show how different sampling techniques designed for the particular systems affect the performance of the end-to-end pipeline.

Our systems, Raha and Baran, are both available online[1]. The repository also includes interactive Jupyter notebooks as the user interface. Furthermore, we packaged and uploaded our systems on the Python Package Index (PyPI)[2]. Our Python package can easily be installed and managed via the *pip* package manager.

2. SYSTEM OVERVIEW

Figure 1 illustrates the architecture of our end-to-end data cleaning system. Given a dirty dataset as input, the goal is to detect and correct data errors and output the cleaned dataset.

To this end, our pipeline leverages three auxiliary resources: (1) the user feedback, (2) a data cleaning toolbox, and (3) an optional repository of previously cleaned datasets. The user feedback in the form of a few identified and fixed data errors is the only required form of supervision in our data cleaning pipeline. The data cleaning toolbox contains the set of base error detectors/correctors. Optionally, the user can further enrich this default set of error detector and corrector algorithms with a set of dataset-specific algorithms. The repository contains previously cleaned datasets, including the error detector/corrector logs. This repository is an optional resource to boost the error detection and correction performance by learning from previous data cleaning experiences.

Provided these data cleaning resources, our end-to-end data cleaning pipeline consists of three main components: (1) the error detection engine (i.e., Raha), (2) the error correction engine (i.e., Baran), and (3) the user interface.

2.1 The Error Detection Engine

A dirty dataset is first processed by Raha to detect its data errors. Internally, Raha generates and runs a large number of base error detectors ($S = \{s_1, s_2, \ldots, s_{|S|}\}$) on the dirty dataset. These detectors represent the four main families of traditional error detection techniques [1]. Raha applies a systematic approach to generate various configurations of the following error detection algorithms as base detectors.

1. *Outlier detection algorithms* [18] assess the correctness of data values in terms of compatibility with the general distribution of values that reside inside the corresponding column. We leverage histogram and Gaussian modelings [18] as two fundamental outlier detection algorithms that leverage the occurrence and magnitude of data values, respectively. Raha automatically generates a range of parameters for each of the two algorithms.

2. *Pattern violation detection algorithms* [11] assess the correctness of data values in terms of compatibility with predefined data patterns. To be independent of user-provided patterns, we generate various kinds of character checkers based on the *bag-of-characters* representation [20].

3. *Rule violation detection algorithms* [6] assess the correctness of data values based on their conformity to integrity constraints. We include rule violation detection strategies that check inter-column dependencies, such as functional dependencies (FDs), as the single-column rules, such as value range and length, are implicitly covered by the outlier and pattern violation detection algorithms. Raha by default considers each column to be dependent on every other column and uses each of these FDs as a base detector.

4. *Knowledge base violation detection algorithms* [4] assess the correctness of data values by cross-checking them with data within a knowledge base, such as DBpedia [2]. We check the conformity of data values inside the dataset at hand to all the entity relationships inside the DBpedia.

The number of detectors depends generally on the number of columns, the distribution of characters inside a dataset, and the size of the knowledge base. In practice, Raha generates thousands of base error detectors by configuring each error detection algorithm with a wide range of configurations. For example for the *Flights* dataset, 4174 detectors will be generated. This approach relieves the user from choosing and configuring detection algorithms. Raha collects the output of these detectors to featurize each data cell. The feature vector of each data cell shows which detectors have marked this particular data cell as a data error. The feature vector of the data cell $d[i, j]$ is the vector of all the outputs of error

[1]https://github.com/BigDaMa/raha
[2]https://pypi.org/project/raha

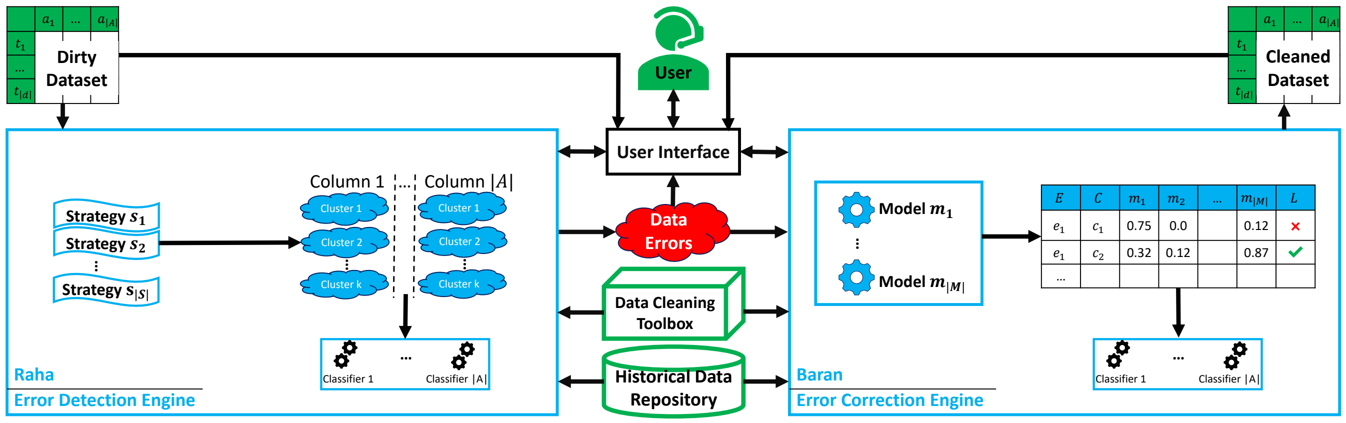


Figure 1: The architecture of the data cleaning pipeline consisting of the detection engine Raha and the correction engine Baran.

detection strategies $s \in S$ on this data cell. Formally,

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

where

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

Therefore, data cells with similar data quality issues will have similar feature vectors as they are marked by the same detectors.

Next, Raha follows a clustering-based sampling and labeling process to train one classifier per data column. It clusters data cells of each column based on the similarity of their feature vectors. Raha samples a set of tuples that covers as many unlabeled clusters as possible and asks the user to label data cells of these sampled tuples as dirty or clean. Formally, Raha draws a tuple t^* in each iteration based on the softmax probability function

$$P(t) = \frac{\exp(\sum_{g \in t} \exp(-N_g))}{\sum_{t' \in d} \exp(\sum_{g \in t'} \exp(-N_g))}, \tag{2}$$

where N_g is the number of labeled data cells in the current cluster of data cell g 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.

Raha propagates each user label to all values of the same cluster to boost the number of labeled training data points. Finally, Raha trains one classifier per dataset column to predict the final label of each value inside a column. The result of Raha will be a Boolean matrix with the same dimensions as the input dataset that contains the prediction 1 or 0, i.e., erroneous or clean, for each data value of the original dataset.

2.2 The Error Correction Engine

The task of Baran is to fix the previously detected data errors. Similar to Raha, Baran relies on a set of base error corrector models ($M = \{m_1, m_2, \ldots, m_{|M|}\}$). Baran generates corrector models based on the available data error context. For each data error, three types of context information are typically available. Parts of a data error context can be the value itself, its vicinity defined by the co-occurring values inside the same tuple, and its domain represented by all values from the same column. Accordingly, we have three types of corrector models.

1. *Value-based error corrector models* learn to fix data errors e using only the characters of the erroneous value itself [9]. A value-based model learns value transformations on two granularity levels: (1) exact matching and (2) pattern matching. In exact matching, the value-based error corrector model learns to transform the erroneous value based on its exact value, e.g., replace “Holland” with “Netherlands”. In pattern matching, the value-based model learns to transform the erroneous value based on its Unicode pattern. For example, in all dates encoded as “<Nd><Nd><Po><Nd><Nd><Po><Nd><Nd><Nd><Nd>” replace “/” with “.”, i.e., replace “16/11/1990” with “16.11.1990”.

2. *Vicinity-based error corrector models* learn to fix data errors based on column relationships. A vicinity-based model proposes clean values of the active domain as potential corrections based on their relationship with clean data values of other columns. We consider every one-attribute to one-attribute functional dependencies as a vicinity-based model to generate correction candidates based on the left-hand-side values.

3. *Domain-based error corrector models* learn to fix data errors using the existing values inside their columns. A domain-based model proposes the most relevant clean values from the active domain as potential corrections. A domain-based model considers the more frequent clean values inside a data column more likely corrections for a data error in the same column.

Each error corrector model m proposes a potential correction c for each data error e. Baran generates a feature vector for each pair of a data error and a correction candidate (e, c). Each component of this feature vector is corresponding to the confidence of one error corrector model m for replacing data error e with correction candidate c. Formally,

$$v(e, c) = [P(c|e_m) \mid \forall m \in M], \tag{3}$$

where M is the set of all the error corrector models and $P(c|e_m)$ is the confidence of the model m in proposing the correction candidate c to the data error e. When a feature vector contains mostly close-to-one probabilities (i.e., $P(c|e_m) \approx 1.0$ for most of the models $m \in M$), it is more likely that the correction candidate c is the actual correction of the data error e; Because, in this case, most error corrector models with high confidence propose this correction candidate for this data error.

Then, Baran uses human supervision to obtain a small number of correction examples. While it has its own sampling technique to select tuples for labeling, in the end-to-end pipeline, it simply uses the same tuples that have been labeled during the detection phase. Baran incrementally updates the error corrector models with these new correction examples from the user. Similar to the detection step of Raha, Baran trains one classifier per data column to predict the actual correction of each data error from all the proposed potential corrections.

2.3 Learning from Previously Cleaned Datasets

Both cleaning engines, Raha and Baran, can benefit from the knowledge of previously cleaned datasets. But each benefits in a different way.

Raha internally runs thousands of detectors leading to a high runtime of the initial featurization phase. We designed Raha this way to make the system fully configuration-free at the cost of runtime. We did not want to rely on the user in the selection process of detectors to make it also accessible for a more general user audience. Naturally, some detectors create much better features for the clustering algorithm of Raha than others. In particular, some detectors might simply generate noisy features. Removing those would benefit the runtime and may improve the effectiveness of the detection process. Since we do not know which of them are least useful upfront and we do not want to rely on the user to make this non-trivial decision, we accept to include them hoping that the high feature-dimensionality will nullify their impact on the overall detection performance. However, it is possible to filter some of the detectors based on their relevance in prior cleaning tasks. In Raha, we perform a column-wise similarity check with columns in previously cleaned datasets. The similarity is based on automatically extractable features that describe the content and structure of a column. After the similarity calculation, a score is generated for each strategy combining the column similarity and the F_1 score of the particular detector on the historical dataset. Then, the set of detectors with the highest score are picked. Our experiments reported in the Raha paper show that reducing the set of detectors significantly reduces the runtime for feature generation without hurting the clustering effectiveness [15].

The Baran engine benefits in a different way from previously cleaned datasets. In particular, Baran harvests all observed cleaned values in the past to train value-based correctors. This way, Baran not only has the chance to obtain out-of-dataset correction candidates but also collects correction evidence that helps the prediction model for choosing the right corrections to converge faster. In our prototype, we harvest corrections from updates in the Wikipedia revision history, which leads to millions of correction candidates part of which will be relevant for a dataset at hand.

2.4 Packaging Structure

We have published the code for both systems on GitHub and the Python Package Index (PyPI). Our Python package contains three separate classes for dataset, error detection, and error correction. The dataset class has methods for loading, normalizing, comparing, editing, and saving data frames. The dataset class has also methods for creating and evaluating cleaned data frames based on data cleaning results. The error detection and correction classes cover Raha and Baran's functionalities and implement methods that take a dataset object as input. Using these methods, the user can initialize the dataset, run base error detectors/correctors, generate features, sample tuples, train classifiers, and apply them to predict data errors/corrections.

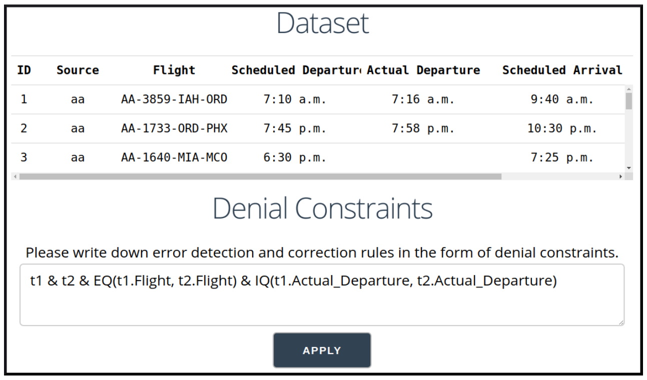
```python
import raha

app = raha.Detection()
app.LABELING_BUDGET = 20

dataset_dictionary = {
    "name": "flights",
    "path": "../datasets/flights/dirty.csv"
}
d = app.initialize_dataset(dataset_dictionary)

app.run_strategies(d)
app.generate_features(d)
app.build_clusters(d)

while len(d.labeled_tuples) < app.LABELING_BUDGET:
    app.sample_tuple(d)
    # Here goes a user-defined labeling function

app.propagate_labels(d)
app.predict_labels(d)
```

Figure 2: An example error detection pipeline using our Python package.

Figure 2 shows an example error detection pipeline built with our Python package. We first import the **raha** module. Next, we instantiate the **Detection()** class and set the **LABELING_BUDGET** property of the **app** object to 20 tuples. We then create a dataset object calling the **initialize_dataset(dataset_dictionary)** method, which needs only the name and the path of the dirty dataset as input. Next, we call the **run_strategies(d)**, **generate_features(d)**, and **build_clusters(d)** methods to featurize data cells of the dataset and build clustering models. We then iteratively sample a tuple by calling the **sample_tuple(d)** method and ask the user to label the sampled tuples via a user-defined labeling function. In our demo, we show a manual labeling scenario. Then, the method **propagate_labels(d)** propagates the user labels into the clusters and trains the classifiers. Finally, **predict_labels(d)** can be called to predict the correctness of each attribute value in the dataset.

The modular design of the package provides usage flexibility for the user as the user can easily orchestrate different steps of the workflow by calling different functions.

In the next section, we describe how the user interface of our prototype enables the user to inspect various steps and resources of our pipeline, such as how values are clustered, how the user can provide examples, and how transfer learning affects the features of each classification task.


Dataset

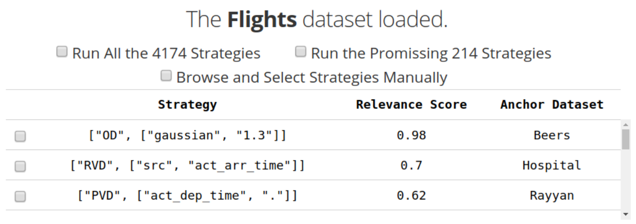
ID	Source	Flight	Scheduled Departure	Actual Departure	Scheduled Arrival
1	aa	AA-3859-IAH-ORD	7:10 a.m.	7:16 a.m.	9:40 a.m.
2	aa	AA-1733-ORD-PHX	7:45 p.m.	7:58 p.m.	10:30 p.m.
3	aa	AA-1640-MIA-MCO	6:30 p.m.		7:25 p.m.

Denial Constraints

Please write down error detection and correction rules in the form of denial constraints.

```
t1 & t2 & EQ(t1.Flight, t2.Flight) & IQ(t1.Actual_Departure, t2.Actual_Departure)
```

Figure 3: The interface of a rule-based data cleaning system that follows the preconfiguration paradigm.


The **Flights** dataset loaded.

Run All the 4174 Strategies
Run the Promising 214 Strategies
Browse and Select Strategies Manually

	Strategy	Relevance Score	Anchor Dataset
☐	["OD", ["gaussian", "1.3"]]	0.98	Beers
☐	["RVD", ["src", "act_arr_time"]]	0.7	Hospital
☐	["PVD", ["act_dep_time", "."]]	0.62	Rayyan

Figure 4: Raha suggests error detection strategies using transfer learning.

3. DEMONSTRATION

Our demonstration objective is to highlight the distinguished aspects of our data cleaning pipeline. We start with a rule-based data cleaning scenario and show the difficulties in generating data cleaning rules. Then, we switch to our configuration-free data cleaning pipeline and ask the user to annotate examples.

Rule-based data cleaning. We load multiple datasets for analysis, such as the *Flights* dataset. Then, we present and discuss the set of available rules to detect and correct data errors. Depending on the interaction possibilities, we can ask the audience to provide more rules as denial constraints [3] or plain Python scripts. Figure 3 shows the user interface, where the user can define a denial constraint for the *Flights* dataset. We then run the cleaning approach and measure the effectiveness for later comparison.

Configuration-free data cleaning. Using the same set of datasets, we start our Raha-Baran pipeline first by showcasing Raha. Right of the batch, the transfer learning component of Raha calculates the similarity of the current dataset with the previously cleaned datasets in our repository. We show and discuss the similarities. Raha then proposes a set of detectors that were effective on similar historical datasets. We then demonstrate how to inspect the selected detectors and how Raha chooses to use them based on their relevance score. For example in Figure 4, the most promising detector for the *Flights* dataset is an outlier detection ("OD") algorithm that marks outlier values based on a Gaussian distribution with a 1.3 distance threshold. Raha proposes this detector for the *Flights* dataset because this strategy was highly effective on an anchor similar dataset, i.e., *Beers*.

Next, we let Raha run the selected detectors to featurize the data cells of the dataset at hand. We show the effect of filtering out ineffective strategies using historical data on runtime optimization. We inspect data quality issues of data cells using their feature vectors. Furthermore, we visualize the similarity of data cells in terms of their data quality issues.

From here on, the iterative error detection and correction starts. In each iteration, Raha samples a tuple to be annotated by the user. Figure 5 shows the user interface during the data cleaning process. A tuple is highlighted for user annotation and the user can mark its data errors (via checkboxes) and provide a correction for them (via text fields).

We highlight low-level information that also supports the user in the tuple annotation process. Raha visualizes the clusters of data cells in each data column (Figure 6). Once the user clicks on a data point, the user can drill down the results and see the detailed information related to that particular data cell, such as which error detection strategies have marked this data cell as a data error. In error correction, Baran extracts value-based corrections from general-purpose revision data, such as the Wikipedia page revision history, and pretrains the error corrector models. Therefore, during the tuple annotation process, the pretrained error corrector models can support the user by mentioning how often a potential correction has been observed in the Wikipedia page revision history. When the user provides a correction or a value is corrected by Baran, its likelihood is measured based on the confidence of supporting correctors, i.e., correctors that suggest the same correction. Furthermore, we can inspect the prevalence of each particular correction in the indexed Wikipedia page revision history.

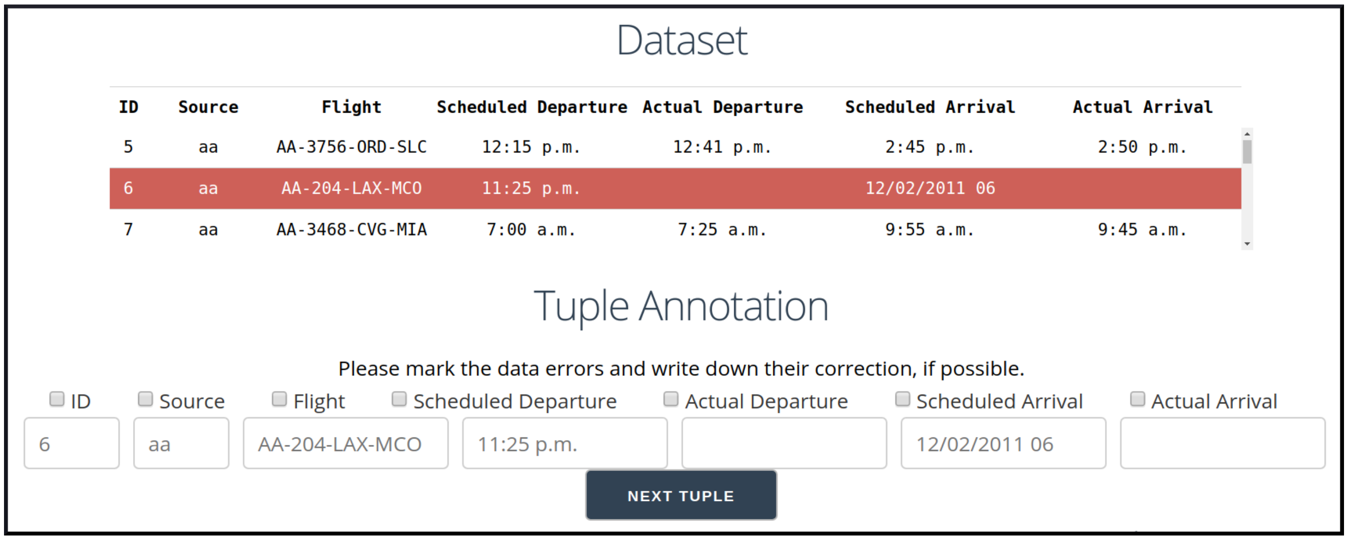
For example in Figure 7, three error corrector models propose value "7:45 p.m." as the correction of erroneous value "7:45pm". The most confident model (depicted as "Model 12") is a substring adder that has learned to add substrings " " and "." to the erroneous values like "7:45pm" to fix their formatting. This particular correction matches exactly to 3 corrections and matches due to a similar pattern to 142 corrections in the Wikipedia logs. These statistics come from the value-based error corrector models that have been pretrained on the Wikipedia page revision history, which serves as a corpus with head-topic corrections.

Once the tuple is annotated, the backend error detection and correction engines will be fine-tuned and the frontend reports and visualizations will be updated accordingly. The audience can also continuously monitor various indicators, such as the data cleaning progress (Figure 8). This iterative process continues until the user terminates the data cleaning process and stores the final cleaned dataset. The user can check the data cleaning progress in the user dashboard to decide whether the performance is already converged or more user-annotated tuples are still needed.

We show the same pipeline with multiple variations, in terms of datasets, sampling techniques, and modularizations. In particular, we also show perfect results from Raha would impact the correction component and how far perfect manual correction of the detection component will be from the perfect cleaning score. Further, we also show the impact of transfer learning on the specific components.

4. RELATED WORK

Error detection is the task of detecting data values that are wrong [1]. Previous work leverage various techniques



Dataset

ID	Source	Flight	Scheduled Departure	Actual Departure	Scheduled Arrival	Actual Arrival
5	aa	AA-3756-ORD-SLC	12:15 p.m.	12:41 p.m.	2:45 p.m.	2:50 p.m.
6	aa	AA-204-LAX-MCO	11:25 p.m.		12/02/2011 06	
7	aa	AA-3468-CVG-MIA	7:00 a.m.	7:25 a.m.	9:55 a.m.	9:45 a.m.

Tuple Annotation

Please mark the data errors and write down their correction, if possible.

☐ ID	☐ Source	☐ Flight	☐ Scheduled Departure	☐ Actual Departure	☐ Scheduled Arrival	☐ Actual Arrival
6	aa	AA-204-LAX-MCO	11:25 p.m.		12/02/2011 06	

Figure 5: The example-based user interface for configuration-free data cleaning.

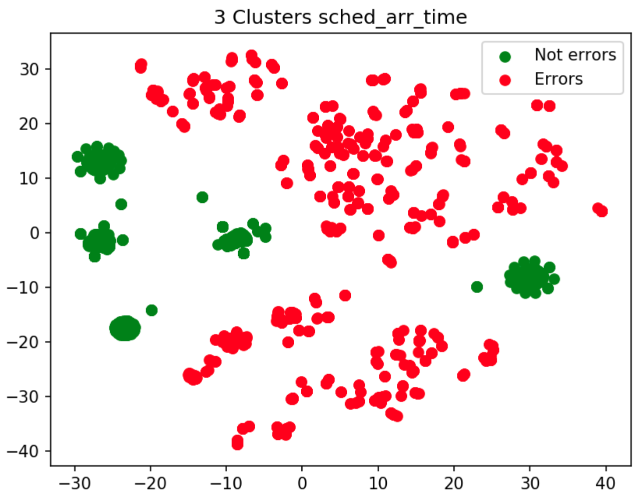
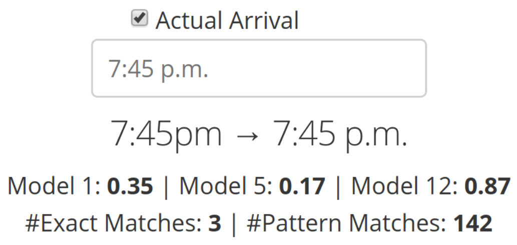


Figure 6: These 2D projected clusters contain either clean (green) or dirty (red) data cells for one column. By clicking on a point, the user can inspect the actual value and the error detection strategies that marked this particular data cell as a data error.



☑ Actual Arrival

7:45 p.m.

7:45pm → 7:45 p.m.

Model 1: **0.35** | Model 5: **0.17** | Model 12: **0.87**

#Exact Matches: **3** | #Pattern Matches: **142**

Figure 7: The user can inspect the confidence of error corrector models for a particular correction and the number of exactly/approximately matched corrections from the Wikipedia page revision history.

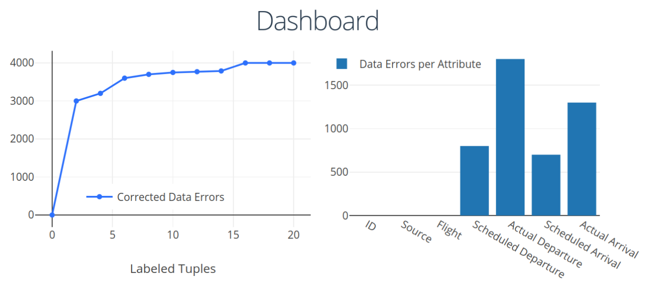


Figure 8: The user dashboard that visualizes the data cleaning progress.

and resources to detect data errors, such as data augmentation [10], web tables [22], knowledge bases [5], denial constraints [8], and ensembling methods [1, 21, 13, 16]. Error correction (also known as data repairing) is the task of fixing the detected data errors [19]. Related work leverage signals, such as integrity rules [3, 8], external sources [5], active learning [12, 24], statistical likelihood [23], or a combination of rules and statistics [19].

Some of the aforementioned approaches have been also demonstrated [24, 8, 5, 12, 25]. In this paper, we propose to demonstrate an end-to-end pipeline that harbors our recently published systems Raha [15] and Baran [14]. In particular, we give novel insights into an example-driven holistic framework to incorporate stand-alone error detectors and correctors and leverage transfer learning. While previous demonstrations showed the internals of stand-alone systems, our demonstration provides inspection into how machine learning methods can be used to effectively aggregate and combine thousands of detectors/correctors. Our interface enables drill down into the performance of each baseline detector/corrector and how they are generated to create feature vectors. Additionally, we visualize the effectiveness of the generated feature vectors and show how transfer learning can improve the efficiency and effectiveness of our data cleaning pipeline.

5. CONCLUSION

We propose to demonstrate our end-to-end data cleaning pipeline based on our data cleaning systems Raha and Baran. Our configuration-free systems achieve high data cleaning performance with negligible data annotation efforts. Furthermore, our end-to-end data cleaning pipeline benefits from transfer learning.

Acknowledgements

This project has been supported by the German Research Foundation (DFG) under grant agreement 387872445. Furthermore, we would like to thank Jeremy Bilic for his contribution to the codebase.

6. REFERENCES

[1] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang. Detecting data errors: Where are we and what needs to be done? volume 9, pages 993–1004, 2016.

[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. In *ISWC*, pages 722–735, 2007.

[3] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*, pages 458–469, 2013.

[4] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *SIGMOD*, pages 1247–1261, 2015.

[5] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye. Katara: reliable data cleaning with knowledge bases and crowdsourcing. *PVLDB*, 8(12):1952–1955, 2015.

[6] M. Dallachiesa, A. Ebaid, A. Eldawy, A. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang. Nadeef: a commodity data cleaning system. In *SIGMOD*, pages 541–552, 2013.

[7] D. Deng, R. C. Fernandez, Z. Abedjan, S. Wang, M. Stonebraker, A. K. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, and N. Tang. The data civilizer system. In *CIDR*, 2017.

[8] A. Ebaid, A. Elmagarmid, I. F. Ilyas, M. Ouzzani, J.-A. Quiane-Ruiz, N. Tang, and S. Yin. Nadeef: A generalized data cleaning system. *PVLDB*, 6(12):1218–1221, 2013.

[9] S. Gulwani. Programming by examples: Applications, algorithms, and ambiguity resolution. In *IJCAR*, pages 9–14, 2016.

[10] A. Heidari, J. McGrath, I. F. Ilyas, and T. Rekatsinas. Holodetect: Few-shot learning for error detection. In *SIGMOD*, pages 829–846, 2019.

[11] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. In *SIGCHI*, pages 3363–3372, 2011.

[12] S. Krishnan, M. J. Franklin, K. Goldberg, J. Wang, and E. Wu. Activeclean: An interactive data cleaning framework for modern machine learning. In *SIGMOD*, pages 2117–2120, 2016.

[13] M. Mahdavi and Z. Abedjan. Reds: Estimating the performance of error detection strategies based on dirtiness profiles. In *SSDBM*, pages 193–196, 2019.

[14] M. Mahdavi and Z. Abedjan. Baran: Effective error correction via a unified context representation and transfer learning. *PVLDB*, 13(11):1948–1961, 2020.

[15] M. Mahdavi, Z. Abedjan, R. Castro Fernandez, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. Raha: A configuration-free error detection system. In *SIGMOD*, pages 865–882, 2019.

[16] F. Neutatz, M. Mahdavi, and Z. Abedjan. Ed2: A case for active learning in error detection. In *CIKM*, pages 2249–2252, 2019.

[17] S. J. Pan and Q. Yang. A survey on transfer learning. volume 22, pages 1345–1359, 2009.

[18] C. Pit-Claudel, Z. Mariet, R. Harding, and S. Madden. Outlier detection in heterogeneous datasets using automatic tuple expansion. Technical Report MIT-CSAIL-TR-2016-002, CSAIL, MIT, 2016.

[19] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. Holoclean: Holistic data repairs with probabilistic inference. volume 10, pages 1190–1201, 2017.

[20] H. Schütze, C. D. Manning, and P. Raghavan. *Introduction to information retrieval*. Cambridge University Press, 2008.

[21] L. Visengeriyeva and Z. Abedjan. Metadata-driven error detection. In *SSDBM*, pages 1–12, 2018.

[22] P. Wang and Y. He. Uni-detect: A unified approach to automated error detection in tables. In *SIGMOD*, pages 811–828, 2019.

[23] M. Yakout, L. Berti-Équille, and A. K. Elmagarmid. Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes. In *SIGMOD*, pages 553–564, 2013.

[24] M. Yakout, A. K. Elmagarmid, J. Neville, and M. Ouzzani. Gdr: a system for guided data repair. In *SIGMOD*, pages 1223–1226, 2010.

[25] Z. Yu and X. Chu. Piclean: A probabilistic and interactive data cleaning system. In *SIGMOD*, pages 2021–2024, 2019.