Detecting Data Errors: Where are we and what needs to be done?

Ziawasch Abedjan[§] Xu Chu[ℓ] Dong Deng[‡][*] Raul Castro Fernandez[§] Ihab F. Ilyas[ℓ]
Mourad Ouzzani[♯] Paolo Papotti[∞] Michael Stonebraker[§] Nan Tang[♯]

[§] MIT CSAIL [ℓ] University of Waterloo [∞] Arizona State University
[‡] Tsinghua University [♯] Qatar Computing Research Institute, HBKU

{abedjan, ddong, raulcf, stonebraker}@csail.mit.edu {x4chu, ilyas}@uwaterloo.ca
{mouzzani, ntang}@qf.org.qa ppapotti@asu.edu

ABSTRACT

Data cleaning has played a critical role in ensuring data quality for enterprise applications. Naturally, there has been extensive research in this area, and many data cleaning algorithms have been translated into tools to detect and to possibly repair certain classes of errors such as outliers, duplicates, missing values, and violations of integrity constraints. Since different types of errors may coexist in the same data set, we often need to run more than one kind of tool. In this paper, we investigate two pragmatic questions: (1) *are these tools robust enough to capture most errors in real-world data sets*? and (2) *what is the best strategy to holistically run multiple tools to optimize the detection effort?* To answer these two questions, we obtained multiple data cleaning tools that utilize a variety of error detection techniques. We also collected five real-world data sets, for which we could obtain both the raw data and the ground truth on existing errors. In this paper, we report our experimental findings on the errors detected by the tools we tested. First, we show that the coverage of each tool is well below 100%. Second, we show that the order in which multiple tools are run makes a big difference. Hence, we propose a holistic multi-tool strategy that orders the invocations of the available tools to maximize their benefit, while minimizing human effort in verifying results. Third, since this holistic approach still does not lead to acceptable error coverage, we discuss two simple strategies that have the potential to improve the situation, namely domain specific tools and data enrichment. We close this paper by reasoning about the errors that are not detectable by any of the tools we tested.

1. INTRODUCTION

In the last two decades there has been intensive research in developing data cleaning algorithms and tools [10, 11, 14, 19, 29]. With the increasing prevalence of data-centric approaches to business and scientific problems with data as a crucial asset, data cleaning has become even more important. Hence, it is important to study the effectiveness of the available error detection solutions (both as industry-strength tools and academic prototypes) in solving real-world cleaning problems.

[*]Work done while doing an internship at QCRI.

Proceedings of the VLDB Endowment, Vol. 9, No. 12
Copyright 2016 VLDB Endowment 2150-8097/16/08.

1.1 The Current State

Available data cleaning solutions and tools belong to one or more of the following four categories:

- *Rule-based detection algorithms* [1, 6, 15, 16, 25, 33] that can be embedded into frameworks, such as NADEEF [8, 23], where a rule can vary from a simple "not null" constraint to multi-attribute functional dependencies (FDs) to user-defined functions. Using this class of tools, a user can specify a collection of rules that clean data will obey, and the tool will find any violations.
- *Pattern enforcement and transformation tools* such as OPENREFINE, Data Wrangler [21] and its commercial descendant TRIFACTA, KATARA [7], and DataXFormer [3]. These tools discover patterns in the data, either syntactic (e.g., OPENREFINE and TRIFACTA) or semantic (e.g., KATARA), and use these to detect errors (cells that do not conform with the patterns). Transformation tools can also be used to change data representation and expose additional patterns.
- *Quantitative error detection algorithms* that expose outliers, and glitches in the data [2, 9, 28, 32, 34].
- *Record linkage and de-duplication algorithms* for detecting duplicate data records [13, 26], such as the Data Tamer system [30] and its commercial descendant TAMR. These tools perform entity consolidation when multiple records have data for the same entity. As a side effect of this process, conflicting values for the same attribute can be found, indicating possible errors.

When considering these tools, several challenges have emerged. Most notably:

1. *Synthetic data and errors.* Most cleaning algorithms have been evaluated on synthetic or real-world data with synthetically injected errors. While these might be appropriate for testing the soundness of the algorithm [4], the effectiveness of any given tool to detect

errors "in the wild" is unclear. The lack of real data sets (along with appropriate ground truth) or a widely accepted cleaning benchmark makes it hard to judge the effectiveness of existing cleaning tools. It is equally difficult to reason about the use cases where a given tool should be employed without real-world data.

2. *Combination of error types and tools.* Real-world data usually contains multiple types of errors. Moreover, the same error might be findable by more than one type of tool. For example, an error might be part of conflicting duplicate records and violate an integrity constraint at the same time. Considering only one type of algorithm in any study misses an opportunity to accumulate evidence from multiple tools.

3. *Human involvement.* Almost all practical tools involve humans, for example, to verify detected errors, to specify cleaning rules, or to provide feedback that can be part of a machine learning algorithm. However, humans are not free and enterprises often impose a human budget to limit the cost of a cleaning project. Since errors detected by different tools overlap, ordering the application of these tools to minimize total human involvement is an important task that requires assessing the interaction among tools.

1.2 Our Evaluation Methodology

To shed light on the current state of data cleaning tools, we embarked on an experimental endeavor to address the above challenges. Specifically, we assembled a collection of real-world data sets (Section 2) that represent the multiple kinds of dirty data one finds in practice. Several of these data sets are proprietary and were obtained under NDA (non disclosure agreement) restrictions. While this prevents us from publishing them as a benchmark, it was important to our study to work with as many real-world data sets as possible. We also obtained full or partial ground truth for each data set, so we can judge the performance and capabilities of available error detection tools.

We have also selected a collection of data cleaning tools (Section 3). These tools were chosen because, in aggregate, they are capable of detecting the error types (Section 2.1) found in practice, namely pattern violations, constraint violations, outliers, and conflicting duplicates. Note that in this study, we are interested in the automatic error discovery and not in automatic repair, since automatic repairing is rarely allowed in practice.

We have systematically run all the tools on all data sets and in this paper we report results (precision and recall) based on the ground truth (Section 4). Each tool has been configured in a best-effort fashion (e.g., by iteratively choosing good parameters or by defining a reasonable set of quality rules). In all cases we used the capabilities that the tool provided, and did not resort to heroics, such as writing data-set specific Java code.

While we did our best in using a tool, we can still miss errors that can be otherwise captured if the right configuration (e.g., more rules, more patterns, or better tuned parameter values) were used. To capture this phenomena, we introduce the notion of *upper-bound recall*, which is an estimate for the *maximum recall* of a tool if it has been configured by an oracle. Since we do not have an oracle and cannot optimally configure a tool, we use the known ground truth to estimate the upper-bound recall as follows. We first classify the remaining errors that are not detected by the tools based on their type. Then, any error whose type matches the type that is detectable by a given tool is counted towards the recall of that tool. For example, if the ground truth revealed an error that can be captured by defining a new functional dependency, we count this error towards the upper-bound recall of the rule-based tool.

With the above considerations in mind, we aim to answer the following questions:

1. What is the precision and recall of each tool? In other words, how prevalent are errors of the type that are covered by each tool?
2. How many errors in the data sets are detectable by applying all the tools combined?
3. Since human-in-the-loop is a well accepted paradigm, how many false positives are there? These may well drain a human effort budget and cause a cleaning effort to fail. Is there a strategy to minimize human effort by leveraging the interactions among the tools?

1.3 Main Findings

Based on our experiments, we draw three conclusions:

- **Conclusion 1:** There is no single dominant tool. In essence, various tools worked well on different data sets. Obviously, a holistic "composite" strategy must be used in any practical environment. This is not surprising since each tool has been designed to detect errors of a certain type. Moreover, empirical evaluation with multiple real data sets show that the distributions of errors types vary significantly from one data set to another.
- **Conclusion 2:** By assessing the overlap of errors detected by the various tools, it is possible to order the application of these tools to minimize false positives (and hence user engagement). Due to the large variance in quality from data set to data set, this ordering strategy must be data set specific. We present a composite tool with these characteristics in Section 3.5.2 and its experimental evaluation in Section 4.5.2.
- **Conclusion 3:** The percentage of errors that can be found by the ordered application of all tools (the combined overall recall) is well less than 100%. In Section 4, we report our experimental results to find additional errors. These focus on type-specific cleaning tools, such as an address cleaning service and the process of enrichment, which is to search for additional attributes and values that can assist in the cleaning operation. Even with the addition of a type-specific address tool and enrichment, we show that there is still a need to develop new ways of finding data errors that can be spotted by humans, but not by the current cleaning algorithms.

Section 5 closes the paper with conclusions and suggestions for future work.

2. DATA ERRORS AND DATA SETS

In this section, we first discuss the type of errors that are reported in this paper. We then describe the real-world data sets that we used.

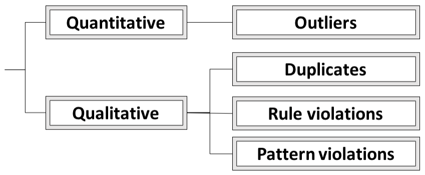


Figure 1: Error type taxonomy

2.1 Types of Data Errors

There have been several surveys on classifying data errors [6, 17, 19, 24, 29]. Rahm et al. [29] and Kim et al. [24] look at data errors from the perspective of how the errors were introduced into the data. Hellerstein et al. [17] follow on the same line when considering numerical data and surveying quantitative data cleaning approaches [17]. Ilyas et al. [19] consider errors as violations of qualitative rules and patterns, such as denial constraints [6]. We adopt a taxonomy that covers all the error types mentioned in these surveys.

We define an error to be a deviation from its ground truth value. More formally, given a data set, a *data error* is an atomic value (or a cell) that is different from its given ground truth. Figure 1 illustrates the four types of errors that are used in this study, which are classified as either quantitative or qualitative ones.

1. **Outliers** include data values that deviate from the distribution of values in a column of a table.
2. **Duplicates** are distinct records that refer to the same real-world entity. If attribute values do not match, this could signify an error.
3. **Rule violations** refer to values that violate any kind of integrity constraints, such as Not Null constraints and Uniqueness constraints.
4. **Pattern violations** refer to values that violate syntactic and semantic constraints, such as alignment, formatting, misspelling, and semantic data types.

Note that our categorization does not perfectly partition errors since some errors may fit into more than one category. Also, our categories are not necessarily exhaustive. Instead, they mainly serve the purpose of categorizing errors that are detectable from available real-world data sets.

2.2 Data Sets

We conducted our experiments on several real-world data sets that were provided by various organizations. Table 1 lists these data sets along with the number of rows, number of columns, and whether we have full or partial ground truth for each data set. Additionally, the corresponding ratio of erroneous cells range from 0.1% to 34% across the data sets. The types of errors found in each data set are given in Table 2. We see that the four error types are prevalent across all data sets with the exception of the Animal data set not having outliers and only MIT VPF and BlackOak having duplicates. In the rest of this section, we give details about each data set including the data quality issues that we found and how we obtained ground truth.

(1) MIT VPF

The MIT Office of the Vice President for Finance (VPF) maintains several finance databases, one of which is related to procurement. This procurement database contains information about vendors and individuals that supply MIT with products and services. Whenever MIT executes a purchase order, a new entry with details about the contracting party is added to the vendor master data set. This record contains identification information, such as name, address, and phone numbers, semi-automatically generated business codes depending on the contract type, and meta-data, such as creation date and creator. The ongoing process of adding new entries introduces duplicates and other data errors. A particular case of errors relates to inconsistent formatting. For example, addresses with formats that deviate from the USPS address standard are considered erroneous values that need to be fixed, e.g., addresses that contain "Street" or "Str" instead of "ST". Similar rules apply for phone numbers and company names. Furthermore, contact information for suppliers changes over time, as suppliers move, get acquired, or introduce new branches. This introduces additional errors, which must be removed.

To obtain ground truth for this data set, employees of VPF manually curated a random sample of 13,603 records (more than half of the data set) and marked erroneous fields. Most of the errors affected address and company names and included missing street numbers, wrong capitalization, and attribute values that appear in the wrong column.

Table 1: Experimental data sets

data set	# columns	# rows	ground truth	Errors
MIT VPF	42	24K	13k (partial)	6.7%
Merck	61	2262	2262	19.7%
Animal	14	60k	60k	0.1%
Rayyan Bib	11	1M	1k (partial)	35%
BlackOak	12	94k	94k	34%

(2) Merck

Merck provided us with a data set that describes IT services and software systems within the company that are partly managed by third parties. Each system is characterized by several attributes, such as location, number of end users, and level of technical support that has been agreed on, e.g., seven days a week or 8x5 hours. The data set is used for optimization purposes. In particular, Merck is interested in identifying opportunities to decommission a service or to downsize the level of support. The data set has 68 different attributes but is very sparse.

Most errors in this data set consist of columns that are not consistently formatted – due to different parties introducing the information in the database. Merck provided the custom cleaning script that they used to produce a cleaned version of the data set. We used this version as ground truth. The script applies various data transformations that normalize columns and allow for uniform value representation. We used this script to help us in formulating rules and transformations for our cleaning tools. Some of the transformations happen invisibly as side effects of other operations. For example, one of the functions that normalizes the retirement date of a software system also changes the encoding of missing values to "NA". Hence, we can only capture a subset of the rules and transformations by examining the script.

(3) Animal

The animal data set was provided by scientists at UC Berkeley who are studying the effects of firewood cutting on

Table 2: Error types found in each data set

	MIT VPF	Merck	Animal	Rayyan Bib	BlackOak
Pattern violations	✓	✓	✓	✓	✓
Constraint violations	✓	✓	✓	✓	✓
Outliers	✓	✓		✓	✓
Duplicates	✓				✓

small terrestrial vertebrates. Each record contains information about the random capture of an animal, including the time and location of the capture, properties such as tag number, sex, weight, species, and age group, and a field indicating whether the capture is the first capture of the animal. The data set was collected from 1993 to 2012.

Each capture was recorded on paper first, and then manually entered into spreadsheets. Thus, errors such as shifted fields and wrong numeric values were introduced to the data set. The scientists identified and corrected several hundreds erroneous cells in the data set. We use the manually cleaned data set as ground truth.

(4) Rayyan Bib

Rayyan[1] is a system built at QCRI to assist scientists in the production of systematic reviews [12]. These are literature reviews focused on a research question, e.g., *is vitamin C good for a cold* [18], and identify and synthesize all research evidence relevant to that question. For each review, Rayyan's users start by searching multiple databases, e.g., MEDLINE and EMBASE, using multiple queries. They then consolidate their search results into long lists (from 100's to 1000's) of references to studies that they feed to Rayyan. Rayyan is used to select the relevant studies and perform other analysis for their reviews.

Since these references are coming from multiple sources and some users may manually manipulate these citations, the data is prone to errors. To obtain ground truth, we manually checked a sample of 1,000 references from Rayyan's database and marked erroneous cells. These references contain tens of columns, such as article_title, journal_title, journal_abbrevation, language, and journal_issn. There are many missing values and inconsistencies in the data, such as journal_title and journal_abbrevation being switched and author names found within the journal title.

(5) BlackOak

BlackOak Analytics is a company that provides entity resolution solutions. They provided us with an anonymized address data set and its dirtied version that they use for evaluation. The errors are randomly distributed and affect the spelling of values, their formatting, completeness, and field separation. We purposely included this data set to study the difference in error detection performance between real-world address data sets and a synthetic data set.

3. DATA CLEANING TOOLS

In selecting the data cleaning tools, we made sure that they cover all the error types described in Section 2.1. These tools include publicly available tools and commercial products to which we had access to or their community versions.

[1]rayyan.qcri.org

Table 3: Coverage of error types by the tools

	dBoost	DC-Clean	OpenRefine	Trifacta	Pentaho	Knime	Katara	Tamr
Pattern violations			✓	✓	✓	✓	✓	
Constraint violations		✓						
Outliers	✓							
Duplicates								✓

Our choices are listed in Table 3 together with the types of errors they were designed to find. For some error-types we picked multiple tools that focus on different subtypes of a given error type. For instance, Katara focuses on the discovery of semantic pattern violations while wrangling tools focus on syntactic pattern violations. In describing the tools below, we explain in details the choice of each specific tool.

In order to mitigate the risk of a suboptimal tool configuration and insufficient rules in the case of rule-based tools, we used an iterative fine-tuning process for each tool. Specifically, we compared the initially detected errors with the ground truth and adjusted the tool configuration or the rules accordingly to improve performance. We also examined the undetected errors and counted those that could have been captured by a tool, under the optimal configuration, as "detectable". These detectable errors are then counted towards the recall upper bound of that tool.

Next, we will present to apply the tools individually, followed by combination approaches to improve performance.

3.1 Outlier Detection

Outlier detection tools detect data values that do not follow the statistical distribution of the bulk of the data. We interpret these values as data errors.

Tool 1: dBoost

dBoost [27] is a novel framework that integrates several of the most widely applied outlier detection algorithms: histograms, Gaussian and multivariate Gaussian mixtures (GMM) [5]. Histograms create a *de facto* distribution of the data without making any assumption a priori by counting occurrences of unique data values. Gaussian and GMM assume that each data value was drawn from a normal distribution with a given mean and standard deviation or with a multivariate Gaussian distribution, respectively.

One key feature of dBoost is that it decomposes run-on data types into their constituent pieces. For example, date is expanded into month, day and year, so that each can be analyzed separately for outliers. To achieve good precision and recall, it is necessary to configure the parameters of the different outlier detection methods: number of bins and their width for histograms, and mean and standard deviation for Gaussian and GMM.

3.2 Rule-based Error Detection

Rule-based data cleaning systems rely on data quality rules to detect errors. Data quality rules are often expressed using integrity constraints, such as functional dependencies or denial constraints [6]. *A violation* is a collection of cells that do not conform to a given integrity constraint. *A data error* is a cell from a constraint column that is involved in a violation. Intuitively, in order to resolve a violation, at least one cell involved in the violation must be changed.

Tool 2: DC-Clean

We use the tool DC-Clean that focuses on denial constraints (DCs) [6] and DCs subsume the majority of the commonly used constraint languages. For a given data set, we design a collection of DCs to capture the semantics of the data. For example, for the animal data set, one of the DCs states that *"if there are two captures of the same animal indicated by the same tag number, then the first capture must be marked as original"*. We report any cell that participates in at least one violation as erroneous.

3.3 Pattern-based Detection

In our evaluation, we include five pattern-based detection tools: (1) OpenRefine, an open source data wrangling tool; (2) Trifacta, the community version of a commercial data wrangling tool; and (3) Katara [7], a semantic pattern discovery and detection tool. OpenRefine and Trifacta provide different exploration techniques that we can use to discover data inconsistencies. When applied on our data sets, both tools subsume errors that can be captured by most existing ETL tools. In fact, we were able to confirm this statement by running two popular ETL tools, namely (4) Knime and (5) Pentaho. Finally, while OpenRefine and Trifacta focus on syntactic patterns, Katara focuses on semantic patterns matched against a knowledge base.

Tool 3: OpenRefine

OpenRefine is an open source wrangling tool that can digest data in multiple formats. Data exploration is performed through *faceting* and *filtering* operations. Faceting resembles a grouping operation that lets look at different kinds of aggregated data. The user specifies one column for faceting and OpenRefine generates a widget that shows all the distinct values in this column and their number of occurrences. The user can also specify an expression on multiple columns for faceting and OpenRefine generates the widget based on the values of the expression. The user can then select one or more values in the widget and OpenRefine filters the rows that do not contain the selected values.

Data cleaning uses an *editing* operation. The user edits one cell at a time, and can also edit a text facet and all the cells consistent with this facet will be updated accordingly. For example, in the Rayyan Bib data set, by faceting on the language column, we have two facets 'eng' and 'English' occurring 110 and 3 times, respectively, exposing a consistency problem. The users can also cluster the cells in a column and then edit the cells in each cluster to a consistent value. Finally, the users can highlight the cells in a column using the Google refine expression language (GREL) [2].

Tool 4: Trifacta

Trifacta is the commercial descendant of DataWrangler [22]. It can predict and apply various syntactic data transformations for data preparation and data cleaning. For this study, we use the available community version of Trifacta. Using Trifacta, one can apply business and standardization rules through the available transformation scripts. For exploratory purposes, Trifacta applies a frequency analysis to each column to identify the most and least frequent values. Additionally, Trifacta shows attribute values that deviate strongly from the value lengths distribution in the specific attribute. Furthermore, Trifacta maps each column to the most prominent data type and identifies values that do not fit the data type.

Tool 5: Katara

Katara relies on external knowledge bases, such as Yago [31], to detect and correct errors that violate a semantic pattern [7]. Katara first identifies the type of a column and the relationship between two columns in the data set using a knowledge base. For example, the type of column A in a table might correspond to type *Country* in Yago and the relationship between columns A and B in a table might correspond to the predicate *HasCapital* in Yago. Based on the discovered types and relationships, Katara validates the values in the table using the knowledge base and human experts. For example, a cell "New York" in column A is marked to be an error, since it is not a country in Yago.

Tool 6: Pentaho

Pentaho[3] provides a graphical interface where the data wrangling process can be orchestrated through creating a directed graph of ETL operations. Any data manipulation or rule validation operation can be added as a node into the ETL pipeline. It provides routines for string transformation and single column constraint validation.

Tool 7: Knime

Knime[4] focuses on workflow authoring and encapsulating data processing tasks, including curation and machine learning-based functionality in compassable nodes. Similar to Pentaho, the curator orchestrates multiple ETL workflows to clean and curate the data, but the rules and the procedure has to be specified. The significant difference of Pentaho w.r.t. OpenRefine and Trifacta is that the user has to know exactly what kind of rules and patterns need to be verified, since both Pentaho and Knime do not provide ways to automatically display outliers and type mismatches.

3.4 Duplicate Detection

If two records refer to the same real-world entity but contain different values for an attribute, it is a strong signal that at least one of the values is an error.

Tool 8: Tamr

Tamr is a tool with industrial strength data integration algorithms for record linkage and schema mapping. Tamr is based on machine learning models that learn duplicate features through expert sourcing and similarity metrics. It is the commercial descendant of the Data Tamer system [30].

3.5 Combination of Multiple Tools

The first problem in trying to use multiple independent tools is how to properly combine them. A simple option is to run all tools and then apply a union or min-k strategy. A more sophisticated solution is to have users manually check a sample of the detected errors, which can then be used to guide the sequence of multiple tool invocation.

[2]`https://github.com/OpenRefine/OpenRefine/wiki/GREL-String-Functions`

[3]`http://www.pentaho.com`

[4]`http://www.knime.com/`

3.5.1 *Union All and Min-k*

We consider two simple strategies: (i) **Union all** takes the union of the errors emitted by all tools, and (ii) **Min-k** considers as errors those errors detected by at least k-tools while excluding those detected by less than k tools.

3.5.2 *Ordering Based on Precision*

The simplest way to involve users, when combining different tools, is to ask them to exhaustively validate the union of the outputs. This is prohibitively expensive given the human effort of validating the large number of candidate errors. For example, for the BlackOak data set, a user would have to verify the 982,000 cells identified as possibly erroneous to discover 382,928 actual errors. Obviously, results from tools with poor performance in error detection should not be evaluated. We thus present a sampling-based method to select the order in which available tools should be evaluated.

Cost model. The performance of a tool can be measured by precision and recall in detecting errors. Unfortunately recall can be computed only if all errors in the data are known, which is impossible when we execute the detection tools on a new data set. However, precision is easy to estimate, and is a proxy for the importance of a tool in the detection process. Suppose the cost of having a human check a detected error is `C` and the value of identifying a real error is `V`. Obviously, the value should be higher than the cost. Hence, $P * \mathtt{V} > (P + N) * \mathtt{C}$, where P is the number of correctly detected errors and N is the number of erroneously detected errors (false positives). We can rewrite the inequality as: $P/(P+N) > \mathtt{C}/\mathtt{V}$. If we set a threshold $\sigma = \mathtt{C}/\mathtt{V}$, we conclude that any tool with a precision below σ should not be run, as the cost of checking is greater than the value obtained. While this ratio is domain dependent and unknown in most cases, it is natural to have large `V` values for highly valuable data. With the corresponding small σ all tools will be considered, thus boosting the recall. On the other hand, if the `C` is high and dominates the ratio, we save cost by focusing only on the validation of tools that are very precise, compromising recall. For each tool, we estimate its precision on a given data set by checking a random sample of the detected errors.

Intuitively, we can run all the tools with a precision larger than the threshold and evaluate the union of their detected error sets. However, the tools are not independent since the sets of detected errors may overlap. We observed that some tools are not worth evaluating even if their precision is higher than the threshold, since the errors they detect may be covered by other tools with higher estimated precision (which would have been run earlier).

Maximum entropy-based order selection. Based on the above observations, we discuss a composite strategy that takes into account the ratio between the total cost and the total obtained value. Following the Maximum Entropy principle [20], we design an algorithm that assesses the estimated precision for each tool. Furthermore, our algorithm estimates the overlap between the tool results. As high entropy refers to uncertainty, picking the tool with the highest precision achieves best entropy reduction. It works in the following four steps.

S1. [Run individual tool.] Run all the tools on the entire data set and get their detected errors.

S2. [Estimate precision by checking samples.] For each tool, estimate its precision by verifying a random sample of its detected errors with a human expert, which can be used to estimate the precision of each tool.

S3. [Pick a tool to maximize entropy.] To maximize the entropy, among all tools not yet considered, it picks the one with the highest estimated precision ($> \sigma$) on the sample, and verifies its detected errors on the complete data set that have not been verified before.

S4. [Update and iterate.] Since errors validated from step **S3** may have been detected by the other tools, we update the precision of the other tools and go to **S3** to pick the next tool, if there are still tools with estimated precision $> \sigma$.

In Section 4.5.2, we show that regardless of each tool's individual performance, the proposed order reduces the cost of manual verification with marginal reduction of recall.

4. EXPERIMENTS

We performed extensive experiments for each combination of tool and data set. For the MIT VPF and Rayyan Bib data set, we applied each tool only on the sample with ground truth. This showed the behavior of each tool and the types of errors each is able to detect on each data set. We report the metrics we used in Section 4.1, the tuning we performed for every tool in Section 4.2, and the degree of user-involvement in Section 4.3. This process was time-consuming, and different configurations might lead to different performance results. We show in Section 4.4 the results we obtained and put the numbers in context *w.r.t.* the four different types of errors presented in Section 2.1.

After the analysis in isolation, we then report in Section 4.5 the insights we obtained from the experiments on different tool combinations. Finally, we analyze the remaining errors and compute the *recall upper-bound* in Section 4.6. This analysis motivates two possible extensions, namely, domain-specific data cleaning tools and enrichment, which we discuss in Sections 4.7 and 4.8, respectively.

4.1 Evaluation Metrics

To show the effectiveness of the different tools, we measure their accuracy in finding potential errors using precision and recall, defined as follows. Given a data set D and its cleaned version G, we define the function diff as $\mathsf{diff}(G, D) = E$, as the set of all erroneous cells in D. Furthermore, for each tool T, let $T(D)$ be the set of cells in D marked as errors by T. We define precision P of a tool as the fraction of cells that are correctly marked as errors and the recall R as the fraction of the actual errors that are discovered by tool T:

$$P = \frac{|T(D) \cap E|}{|T(D)|} \qquad\qquad R = \frac{|T(D) \cap E|}{|E|}$$

Note that for the rule-based and duplicate detection systems, we identify inconsistencies that typically involve multiple cells while only one of the cells might actually be a true error. In our study, we mark all attribute values involved in an inconsistency as errors, thus favoring recall at the cost of precision. To aggregate P and R we use the harmonic mean or F-measure as $F = 2(R \times P)/(R + P)$.

4.2 Usage of Tools

As noted earlier, for each tool, we use all of its features on a best effort basis using expert knowledge of the data sets. However, we do not write user-defined programs for any of the tools.

Table 4: Data quality rules defined on each data set

Data set	Rule Type	Number	Examples
MIT VPF	FD	3	Zip code → State; EIN → Company name
	DC	5	Phone number not empty if vendor has an EIN; Either street field or PO field must be filled
Merck	Check	14	Support level should not be NULL; Employer status should be either *Y*,*N*, or *N/A*
Animal	FD	2	Tag → Species; FD:Tag→A/S-2
	DC	2	For any two captures of the same animal with same Tag and ID), the second capture must be re-capture
Rayyan Bib	Check	14	journal ISSN should not be NULL; author should not be NULL
	FD	9	journal abbreviation →journal title; journal abbreviation → journal ISSN
	DC	1	No two articles are allowed to have the same title
BlackOak	FD	5	Zip code → State; Zip code→ City

Table 5: Error detection performance of each tool on each data set

	MIT VPF			Merck			Animal			Rayyan Bib			BlackOak		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
DC-Clean	0.25	0.14	0.18	**0.99**	**0.78**	**0.87**	0.12	**0.53**	**0.20**	0.740	0.549	0.630	0.46	0.43	0.44
TRIFACTA	0.94	**0.86**	**0.90**	**0.99**	**0.78**	**0.87**	**1.0**	0.03	0.06	0.714	0.591	0.647	0.96	0.93	0.94
OPENREFINE	**0.95**	**0.86**	**0.90**	**0.99**	**0.78**	**0.87**	0.33	0.001	0.002	**0.947**	**0.603**	**0.737**	0.99	**0.95**	**0.97**
Pentaho	**0.95**	0.59	0.73	**0.99**	**0.78**	**0.87**	0.33	0.001	0.002	0.717	0.584	0.644	**1.0**	0.66	0.79
KNIME	**0.95**	0.59	0.73	**0.99**	**0.78**	**0.87**	0.33	0.001	0.002	0.717	0.584	0.644	**1.0**	0.66	0.79
Gaussian	0.07	0.07	0.07	0.19	0.00	0.01	0	0	0	0.412	0.131	0.199	0.91	0.73	0.81
Histogram	0.13	0.11	0.12	0.13	0.02	0.04	0	0	0	0.395	0.164	0.232	0.52	0.51	0.52
GMM	0.14	0.29	0.19	0.17	0.32	0.22	0	0	0	0.534	0.391	0.451	0.38	0.37	0.38
KATARA	0.40	0.01	0.02	-	-	-	0.55	0.04	0.073	0.598	0.393	0.474	0.88	0.06	0.11
TAMR	0.16	0.02	0.04	-	-	-	-	-	-	-	-	-	0.41	0.63	0.50
Union	0.24	0.93	0.38	0.33	0.85	0.48	0.128	0.575	0.209	0.473	0.850	0.608	0.39	0.99	0.56

dBOOST

We applied three algorithms from dBOOST: Gaussian, histogram, and GMM. All of them require input parameters, which we chose using the following trial-and-error methodology. The Gaussian and GMM approach required the parameters mean and standard deviation. The histogram approach required a value for its bin width. For each data set, we tried various parameter values and compared the discovered outliers with the ground truth for each one. Our final choice was the parameter values with the highest F-measure score. For efficiency reasons, we first tuned the parameters on samples, and then applied the top three discovered parameter values on the complete data set. The ultimate choice was the one with the highest F-measure on the complete data set.

DC-Clean

For the MIT VPF data set, we obtained business rules from the data owner. We inferred rules for the Merck data set by examining the cleaning script provided by the data set owner. The rules were single-column constraints, such as denial of specific values in a column, functional dependencies, and conditional functional dependencies. In addition and for each data set, we manually constructed FD rules based on obvious n-to-1 relationships, such as Zip code → State. Table 4 illustrates the number of each type of rule for each data set. As mentioned earlier, we report any cell that participates in at least one violation to be erroneous.

OPENREFINE

Instead of manually checking each attribute value, we applied OPENREFINE's facet mechanism. The numerical facet shows the fraction of data cells in a column that are numerical and their distribution, while text facets show sorted distinct column values and their frequencies. Furthermore, we used the transformation engine for detecting values that deviate from formatting or single column rules, such as *phone number should not be null* or *zipcodes consist of zip+4, i.e., the basic five-digit code plus four additional digits.*

TRIFACTA

We used TRIFACTA in a similar manner to OPENREFINE for enforcing formatting and single-column rules. Additionally, we considered the outlier detection and type-verification engines in TRIFACTA as additional ways of detecting errors.

KATARA

In our KATARA implementation [7], cells are verified against a knowledge base. If a cell is not in the knowledge base, we declare it to be an error.

Since knowledge bases are inevitably domain-specific, we manually constructed one for each data set. For example, for Animal data, the domain experts provided us with a list of animal species, which were entered into KATARA. For address data, we created a knowledge base of geographical locations that contains information about cities, regions, and countries with different formats and abbreviations. Similarly, we used an ISSN knowledge base for Rayyan Bib data that contains journal ISSNs, journal names, journal abbreviations, journal languages, etc.

PENTAHO and KNIME

We can use ETL tools, such as PENTAHO and KNIME, in a similar manner as OPENREFINE and TRIFACTA. However, we have to model each transformation and validation routine as a workflow node in the ETL process instead of applying the rules directly to the data. Both tools provide nodes for enforcing single column rules and string manipulations. The final result of the workflows designed in this tool are data sets where the potential errors have been marked through transformations.

Table 6: Error detection performance of at least k detection tools on each data set

k	MIT VPF			Merck			Animal			Rayyan Bib			BlackOak		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
1	0.24	**0.93**	0.38	0.33	**0.84**	0.47	0.128	**0.575**	**0.209**	0.473	**0.850**	0.608	0.391	**0.999**	0.56
2	0.48	0.90	**0.63**	0.889	0.789	0.834	0.241	0.030	0.053	0.650	0.738	0.691	0.553	**0.999**	0.712
3	0.58	0.41	0.48	0.996	0.787	**0.879**	**1.0**	0.001	0.002	0.831	0.599	**0.696**	0.733	0.979	0.838
4	0.79	0.09	0.16	0.997	0.280	0.438	0	0	0	0.928	0.432	0.590	0.904	0.915	**0.909**
5	0.76	0.03	0.06	0.993	0.015	0.029	0	0	0	**0.969**	0.164	0.281	0.972	0.739	0.839
6	**0.90**	0.00	0.01	**1.0**	0.000	0.000	0	0	0	0.962	0.032	0.062	0.993	0.437	0.607
7	0.80	0.00	0.00	0	0	0	0	0	0	0.897	0.007	0.014	0.999	0.135	0.237
8	0	0	0	0	0	0	0	0	0	0	0	0	**1.0**	0.013	0.025

TAMR

We used TAMR off-the-shelf and applied several rounds of training for each data set. Each round labels 100 candidate duplicate pairs as matching or not-matching. Since we have ground truth, this labeling can be performed automatically. After each round we retrain the machine learning algorithms and compute precision and recall. If they stay stable for more than two iterations, we stop training and run TAMR on the whole data set to discover clusters of duplicates. For each cluster, we identify conflicting fields and mark them as errors.

4.3 User involvement

In general, the user is involved in four activities:

1. The user configures the tools by declaring and specifying rules and patterns that she is aware of and can be validated by the given tool.
2. The user performs data exploration to find errors using TRIFACTA and OPENREFINE.
3. The user validates the result of the error detection tools. In our experiments, this is needed to compute the precision and recall.
4. The user has to manually go through the remaining errors and try to categorize them based on the given error types. This reasoning allows us to compute the recall upper bound.

4.4 Individual Tools Effectiveness

In this section we report the performance (precision, recall, and F-measure) of each data cleaning tool for all data sets, and Table 5 summarizes the results. There are several observations that we can make. The two data transformation tools, TRIFACTA and OPENREFINE, have either the highest or the second highest results with regard to recall on the MIT VPF, Merck, and BlackOak data sets where formatting issues mattered most. However, both tools have poor results on the Animal data set where most errors are of semantic nature, where the value is syntactically valid and from the correct domain but wrong with regard to the ground truth. On all data sets, the results of TRIFACTA and OPENREFINE subsumed the results of PENTAHO and KNIME. This is not surprising as all of these tools provide basic wrangling capabilities. However, TRIFACTA and OPENREFINE also automatically hint at data type mismatches and outliers.

The rule based system DC-Clean had comparably good performance on the Merck, Animal, and Rayyan Bib data sets in terms of F-measure because the rules in these data sets covered particular error-prone columns. DC-Clean,PENTAHO,KNIME, TRIFACTA, and OPENREFINE achieved exactly the same precision and recall on the Merck data set, as all patterns could be translated into DC check rules. The recall of OPENREFINE and TRIFACTA is higher on the BlackOak data set since their outlier detection routines show syntactical problems that could not be anticipated with DC rules.

Outlier detection methods do nothing on the Animal data set, since there are no outliers. In general, outlier methods performed very poorly on the remaining data sets because dirty data was typically frequent patterns not anomalies. Outlier detection would be more effective on high-quality data sets where errors are rather rare. KATARA had fair precision on the BlackOak data set where location values could be easily verified via the knowledge base but performed poorly on the real-world MIT VPF data set where errors were mostly found in address values rather than in geographic locations.

We ran the duplicate detection tool only on data sets having duplicates, i.e., MIT VPF and BlackOak. For MIT VPF, the tool discovered all existing duplicates. The low precision and recall reported in Table 5 reflects the fact that most errors are not of type duplicate. In the case of BlackOak, we found most (but not all) of the duplicates. If all duplicates would have been found, the recall of exposing errors would have increased from 63% to 98%, which means that 98% of the errors in the data set were involved in duplicate records. The precision would have slightly decreased by 2%.

By comparing the results on MIT VPF and BlackOak, we clearly see that most of the tools are much more effective on the synthetic BlackOak data. As a result, future work in this area should not use synthetic faults.

Also note that no single tool is dominant on all data sets. Outlier detection algorithms covered shifted values, rules generally captured inconsistencies among re-occurring values, while transformation tools covered formatting errors. Duplicate detection systems were very effective, but only in data sets where there were a lot of duplicates. Hence, in the next section we turn to aggregating evidence from multiple tools.

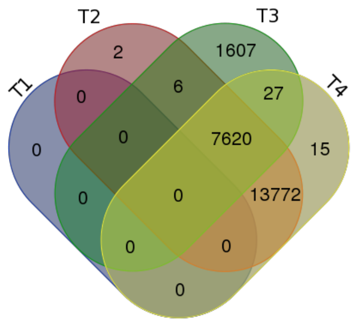
4.5 Tool Combination Effectiveness

In this set of experiments, we will report the result of different strategies of combining multiple tools.

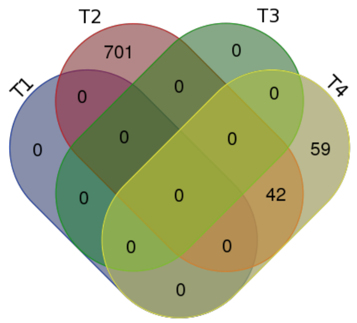
4.5.1 *Union All and Min-k*

The last row in Table 5 lists performance that can be obtained by the union of all tool outputs. Of course, the combination of tools achieves better recall than any single tool, typically above 60% and often much higher.

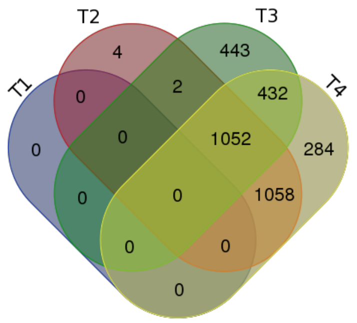
While the union of all tools results in higher recall than any single tool, the precision suffers significantly, as the union of all results increases the set of false positives. In the particular case of MIT VPF, we can see that the 97%



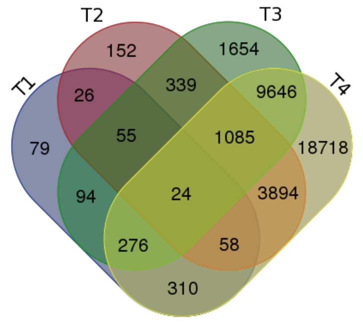
(a) Merck: 23,049 out of 27,208 errors



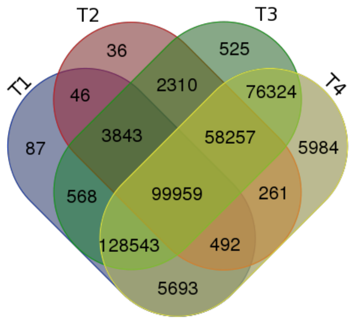
(b) Animal: 802 out of 1,394 errors



(c) Rayyan Bib: 3275 out of 3853 errors



(d) MIT VPF: 36,410 out of 39,158 errors



(e) BlackOak: 382,928 out of 383,003 errors

Figure 2: Overlaps of error types: T1: Duplicates, T2: Constraint Violations, T3: Outliers, T4: Pattern Violations

recall holds for 24% precision. To maximize precision, a natural approach is to use a min-k voting system. Table 6 presents the precision and recall when we require at least k-tools agree on a cell being erroneous. As expected, by increasing k, we increase precision but lose recall. We can further observe that there is no k where the F-measure is the highest on all data sets. Note that for this and the following experiments, we excluded Pentaho and Knime since their results were totally subsumed by the results of OpenRefine and Trifacta, and hence would not contribute to additional insights.

In order to study how different types of errors appear together in a data set, we analyzed the overlap of discovered errors. Figure 2 shows for each data set a Venn diagram where the number in each area represents correctly detected errors that belong to the overlapping error types. Here T1 corresponds to duplicates, T2 corresponds to constraint violations, T3 corresponds to outliers (unioning of results of Gaussian, GMM, and histogram), and T4 corresponds to pattern violations (unioning results of Trifacta, OpenRefine,Katara). In each data set note there are errors that belong to multiple error types. Therefore they can be discovered by multiple tools. In MIT VPF for example, outliers and pattern violations overlap the most; for Merck, Animal, and Rayyan Bib, constraint violations and pattern violations overlap the most; while for BlackOak, duplicates and pattern violations overlap the most.

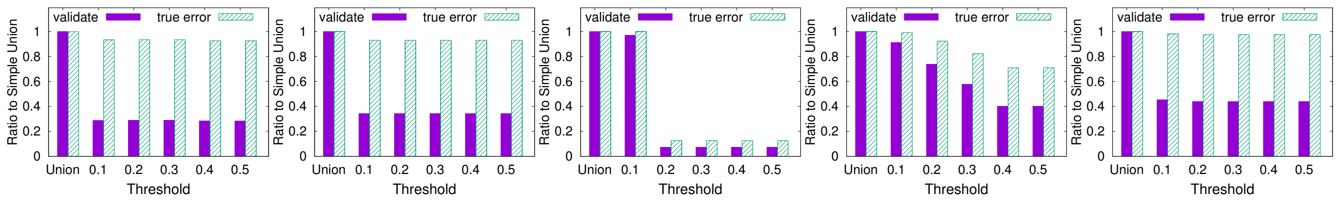
Comparing the min-k approach results to the Venn diagrams note that in data sets where errors infrequently overlap, the min-k approach suffers significant loss of recall with increasing k. This is the case for Animal data where 94% of the detected errors are detected by a single tool, specifically 87% (701) are constraint violations and 7% (59) are pattern violations. On the other hand in BlackOak, we see a strong overlap among tools. For example, 26% (99,959) of the detected errors belong to all four error types, while only 1.7% (6,632) belong to exactly one error type. Hence, recall stays above 90% in min-k up to $k = 4$ and then gradually decreases for higher k.

The main problem of the min-k approach is that it depends on a manually picked k, which can depend on the given data set and the set of tools. Therefore, we proposed (as described in Section 3.5) a different approach that optimizes the accuracy of the tool ensemble through a benefit-based ordering of tool application.

4.5.2 Ordering based on Benefit and User Validation

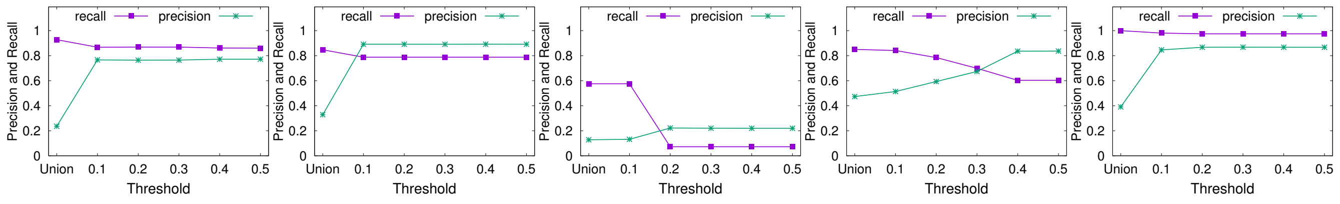
For each data set, we randomly sampled 5% of the detected errors for each tool and compared them with ground truth to estimate the precision of the tool. As noted in Section 3.5.2, we should then run the tools with sufficiently high precision in order. As a baseline, we took the simple union of all the algorithms, for which all the detected errors need to be validated. In addition, we varied the threshold σ (see Section 3.5.2) from 0.1 to 0.5, thereby generating five runs for each data set. We also depict the result of the simple union of all the algorithms.

To show the improvement in performance of our algorithm relative to the baseline, we computed the percentages of detected errors that need to be validated as well as the actual errors. Figures 3 and 4 show the results. Using our strategy, a human has to validate significantly fewer detected errors while losing only a few true errors compared to the union strategy. In other words, it sacrificed a bit of recall to save on human engagement. For example, on Merck, when the threshold for σ is 0.1, a human would need to validate 35%



(a) MIT VPF (b) Merck (c) Animal (d) Rayyan Bib (e) BlackOak

Figure 3: The relative ratio of validated errors and true errors of the ordering strategy to those of the simple union strategy



(a) MIT VPF (b) Merck (c) Animal (d) Rayyan Bib (e) BlackOak

Figure 4: The absolute precision and recall of the union and the ordering strategy with different thresholds

of the detected errors compared to the union strategy, while capturing 92% of its true errors. Similar results can be observed for the other thresholds. For MIT VPF, less than 20% of the errors need to be validated for a relative ratio of true errors above 80%. This is because the ordering strategy dynamically updates the estimation of the tools' precision and drops those tools that have an estimated proportion of true errors in their non-validated errors below the threshold.

Looking more carefully at each data set, we make the following observations. For Merck and BlackOak, there are tools with precision and recall larger than 95% and 78%, respectively (see Table 5). Once the detected errors for one of these tools have been validated, the relative benefit for the other tools becomes so low that they are never used. For MIT VPF, the scenario is different. One tool performs very poorly on this data set, reporting lots of detected errors with very low precision. This tool is immediately filtered by the lowest threshold (0.1), but the other tools are able to identify most of its true errors, therefore the results are very good with a smaller effort.

For Animal, all the tools have an extremely low recall except the rule-based tool that has a much larger recall but with a precision of 0.12 which is the smallest among all the applicable tools. We thus see a sharp fall in recall when this tools is discarded starting with threshold 0.2. For Rayyan Bib, there are neither dominant tools nor extremely low precision tools. We observe that by increasing the threshold, the percentage of validated values decreases faster than the percentage of identified true errors.

In general, the experiments suggest that it is better to start with a conservative threshold if recall is a priority. However, if the budget is constrained, the ordering can reduce the cost while preserving high coverage in most cases. This insight is supported by the analysis of how the absolute precision and the recall change depending on the threshold values, as reported in Figure 4.

4.6 Discussion and Improvements

In the previous experiments, we reported the *recall* for each tool as the number of errors identified by the tool.

Table 7: Remaining errors not detected by the tools

Data set	Best effort Recall	Upper-bound Recall	Remaining Errors
MIT VPF	0.92	0.98 (+1,950)	798
Merck	0.85	0.99 (+4,101)	58
Animal	0.57	0.57	592
Rayyan Bib	0.85	0.91 (+231)	347
BlackOak	0.99	0.99	75

As discussed in Section 4, we used the tools to the best of their capabilities based on our expertise on those tools and configured each tool for each data set individually. Now, we turn to the *recall upper-bound* of the tool combinations. For this purpose, we count the errors of the type that the tool is built for towards the recall of the tool. For example, we identified that one additional pattern enforcement rule can be added and can cover 98% of the remaining errors for Merck. Similarly, in the Rayyan Bib data set, there are a series of errors that affect the formatting of page numbers, e.g., *122-6* is used instead of *122-126*. Again those could be identified through a regular expression in OpenRefine. The result is a recall upper bound for each data set, noted in the second column of Table 7.

Table 7 also reports the remaining errors that cannot be detected by any of the given tools. In MIT VPF, for example, the remaining errors refer to wrong addresses or incomplete company names that cannot be identified by any of the tools. For BlackOak, the bigger portion of the remaining errors relates to wrong or misspelled last names. The best hope to discover these kinds of errors is to use domain-specific tools or dictionaries that contain those attribute values. Similarly, the remaining errors in the Animal data set can only be exposed through an expert or the use of domain-specific dictionaries. Another set of errors is caused by the confusion of cities that could not be detected because of missing context, such as zip codes. In the Merck data set, the remaining errors refer to random non-repetitive values in the status column that need manual treatment. In the following, we discuss two possible data cleaning improvements, namely, domain-specific data cleaning tools and enrichment.

Table 8: Cleaning addresses with AddressCleaner

Data set	Precision	Recall	Union Recall
MIT VPF	0.71	0.68	0.95
BlackOak	0.72	0.61	0.999

Table 9: Precision and Recall after enrichment

Data set	Rule-based		Duplicates	
	P	R	P	R
MIT VPF	(+6%) 0.31	(+6%)0.20	(+2%) 0.18	(+1%) 0.03
BlackOak	0.46	0.43	0.41	(+5%) 0.68

4.7 Domain Specific Tools

For the MIT VPF and BlackOak data sets, we can use an address verification tool, AddressCleaner[5], that matches any given address to the best address in its database and formats it uniformly. The API reads an address string and returns a JSON object with the matched address and additional components, such as longitude/latitude coordinates and zip+4 values. Since the free service has access limitations, we could only apply it on a 1000 row sample of MIT VPF and BlackOak.

Table 8 lists the precision and recall of AddressCleaner on each data set and its contribution to the combined recall of all tools. The recall of this tool alone can be at most 67% on MIT VPF and 61% on BlackOak since the service applies only on fields referring to address, city, state, and zip code. Limiting the dataset to only those columns the recall on both datasets exceeds 97%. Interestingly, AddressCleaner does not have 100% precision, which is due to wrong zip+4 determinations, which occur if the address is too ambiguous or if different representations of neighborhood areas are present. For example, Jamaica Plain and Boston are both valid city representations for the same address. The inclusion of the AddressCleaner to the tool union slightly increases the combined recall on both data sets, with 2 and 13 new errors detected for MIT VPF and BlackOak, respectively. However, we can see that general-purpose tools cover a significant portion of the errors in these data sets. Therefore, the added utility of a domain specific tool is questionable on our study data sets.

4.8 Enrichment

Some strategies for error identification fail because of a lack of evidence. In particular, rule-based systems and duplicate detection strategies would benefit from additional attributes that can be utilized for new rules or for further disambiguation of entities. Naturally, a promising direction is to first enrich a given data set by adding more attributes to each record that are obtained from other data sources. Additional data sources can be relational web tables from the public internet or corporate knowledge or databases. We show the benefit of data enrichment using a straightforward approach.

For the MIT VPF and BlackOak data sets, we obtained additional tables from the data owners that could be joined to the dirty tables we had. Using this technique, we manually enriched the original data sets with additional columns. We only used columns that did not introduce additional duplicate rows. As a result, the results below are a best case scenario. In the case of BlackOak, we appended three more columns that included the person name, address information and the date of birth in a different representation. Using these 3 columns, we defined four more functional dependencies. In the case of MIT VPF, we appended seven more columns by joining the vendor master table with another table called Remit-To-Address. These seven attributes contained additional address data that we were able to compare with the address of the same company in our data set using seven new DCs. Moreover, the added attributes can be used to assist with duplicate detection.

Table 9 shows the impact of the new rules and the improvement in duplicate detection in the two data sets. Duplicate detection leads to 5% better recall on BlackOak because Tamr found more duplicate pairs that contained conflicting values. In the case of MIT VPF, however, no additional improvement was seen, because Tamr already found all the duplicates prior to the enrichment process. Rule-based error detection did not improve the results on BlackOak, since all new rules overlapped with previously defined ones. On the other hand, MIT VPF clearly benefited from the new rules, which exposed 32 more errors and generated higher precision and recall. In summary, targeted enrichment has the potential to improve the performance of duplicate detection and rule-based systems.

5. CONCLUSION AND FUTURE WORK

In this paper, we assembled a collection of real-world data sets that exhibit most of the errors found in practice as well a collection of data cleaning tools that have the potential to detect the various types of errors. We performed an extensive empirical study by running each tool on each data set and reported on its best possible performance. We also examined how combinations of these tools can improve performance. Our experimental results led us to the following conclusions: (1) There is no single dominant tool for the various data sets and diversified types of errors. Single tools achieved on average 47% precison and 36% recall (Table 5), showing that a combination of tools is needed to cover all the errors. (2) Picking the right order in applying the tools can improve the precision and help reduce the cost of validation by humans. As shown in Figure 4, compared to the simple union of all tools, a benefit-based ordering algorithm achieved 28% average precision gain with only 3.5% average recall loss on all the data sets when the threshold was 0.1. (3) Domain specific tools can achieve high precision and recall compared to general-purpose tools, achieving on average 71% precision and 64% recall, but are limited to certain domains (Table 8). (4) Rule-based systems and duplicate detection benefited from data enrichment. In our experiments, we achieved an improvement of up to 10% more precision and 7% more recall (Table 9).

There are several promising future directions:

(1) A holistic combination of tools. We showed that there is no single dominant data cleaning tool for all data sets and blindly combining tools will likely decrease precision. Our ordering algorithm is a start in this direction. However, new holistic approaches to combining tools may be able to perform even better.

(2) A data enrichment system. The more knowledge and context available for a data set, the easier it is to expose

[5]Anonymized by authors.

erroneous values. While we showed some improvements using enrichment, it was applied in an ad-hoc way. More work is needed on how to enrich the data set with data from an added data source that would be most useful for data cleaning. In particular, we need to find automatic approaches to enrichment via corporate knowledge bases. Furthermore, data sets with public information, such as the Rayyan Bib data set, could be enriched through web tables and other open data. For example the attributes journal_title and journal_abbreviation, which suffer from missing values, can be enriched through tools, such as DataXFormer [3], by looking for semantic transformations of journal title to its abbreviation.

(3) A novel interactive dashboard. Throughout this paper, it is clear that user engagement is central to any data cleaning activities. At the same time, such engagement is costly and must be comensurate with the potential benefit from cleaning. It is therefore crucial to devise novel interactive dashboards to help users better understand the data and be efficiently involved in the data cleaning process.

(4) Reasoning on real-world data. Analysis of synthetic errors is important for testing the functionality of a single tool but it does not help to identify the usefulness of a tool in the real-world. We believe that it is vital to shift the focus from optimizing single error-type tools towards creating end-to-end data quality solution for real-world data sets.

6. ACKNOWLEDGMENTS

We thank the Office of the VPF at MIT, the IT department of Merck, and BlackOak Analytics for kindly providing their data sets. We also thank William D. Tietje from UC Berkeley for providing the Animal data set.

7. REFERENCES

[1] Z. Abedjan, C. Akcora, M. Ouzzani, P. Papotti, and M. Stonebraker. Temporal rules discovery for web data cleaning. *PVLDB*, 9(4):336 –347, 2015.

[2] Z. Abedjan, L. Golab, and F. Naumann. Profiling relational data: a survey. *VLDB Journal*, 24(4):557–581, 2015.

[3] Z. Abedjan, J. Morcos, I. F. Ilyas, P. Papotti, M. Ouzzani, and M. Stonebraker. DataXFormer: A robust data transformation system. In *ICDE*, 2016.

[4] P. C. Arocena, B. Glavic, G. Mecca, R. J. Miller, P. Papotti, and D. Santoro. Messing-Up with BART: Error Generation for Evaluating Data Cleaning Algorithms. *PVLDB*, 9(2):36–47, 2015.

[5] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009.

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

[7] 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*, 2015.

[8] M. Dallachiesa, A. Ebaid, A. Eldawy, A. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang. Nadeef: A commodity data cleaning system. In *SIGMOD*, 2013.

[9] T. Dasu and J. M. Loh. Statistical distortion: Consequences of data cleaning. *PVLDB*, 5(11):1674–1683, 2012.

[10] A. Doan, A. Y. Halevy, and Z. G. Ives. *Principles of Data Integration*. Morgan Kaufmann, 2012.

[11] X. L. Dong and D. Srivastava. *Big Data Integration*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2015.

[12] A. Elmagarmid, Z. Fedorowicz, H. Hammady, I. Ilyas, M. Khabsa, and O. Mourad. Rayyan: a systematic reviews web app for exploring and filtering searches for eligible studies for cochrane reviews. In *Abstracts of the 22nd Cochrane Colloquium*, page 9. John Wiley & Sons, Sept. 2014.

[13] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 19(1):1–16, 2007.

[14] W. Fan and F. Geerts. *Foundations of Data Quality Management*. Morgan & Claypool, 2012.

[15] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Towards certain fixes with editing rules and master data. *VLDB Journal*, 21(2):213–238, 2012.

[16] F. Geerts, G. Mecca, P. Papotti, and D. Santoro. Mapping and Cleaning. In *ICDE*, 2014.

[17] J. M. Hellerstein. Quantitative data cleaning for large databases, 2008.

[18] H. Hemila and E. Chalker. Vitamin c for preventing and treating the common cold. *Cochrane Database Syst Rev*, 1, 2013.

[19] I. F. Ilyas and X. Chu. Trends in cleaning relational data: Consistency and deduplication. *Foundations and Trends in Databases*, 5(4):281–393, 2015.

[20] E. T. Jaynes. On the rationale of maximum-entropy methods. *Proceedings of the IEEE*, 70(9):939–952, 1982.

[21] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. New York, NY, USA, 2011.

[22] S. Kandel, A. Paepcke, J. M. Hellerstein, and J. Heer. Enterprise data analysis and visualization: An interview study. *IEEE Trans. Vis. Comput. Graph.*, 18(12):2917–2926, 2012.

[23] Z. Khayyat, I. F. Ilyas, A. Jindal, S. Madden, M. Ouzzani, P. Papotti, J.-A. Quiané-Ruiz, N. Tang, and S. Yin. Bigdansing: A system for big data cleansing. In *SIGMOD*, pages 1215–1230, 2015.

[24] W. Kim, B.-J. Choi, E.-K. Hong, S.-K. Kim, and D. Lee. A taxonomy of dirty data. *Data Min. Knowl. Discov.*, 7(1):81–99, Jan. 2003.

[25] S. Kolahi and L. V. S. Lakshmanan. On Approximating Optimum Repairs for Functional Dependency Violations. In *ICDT*, 2009.

[26] F. Naumann and M. Herschel. *An Introduction to Duplicate Detection*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

[27] 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, 32 Vassar Street, Cambridge MA 02139, February 2016.

[28] N. Prokoshyna, J. Szlichta, F. Chiang, R. J. Miller, and D. Srivastava. Combining quantitative and logical data cleaning. *PVLDB*, 9(4):300–311, 2015.

[29] E. Rahm and H.-H. Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23(4):3–13, 2000.

[30] M. Stonebraker, D. Bruckner, I. F. Ilyas, G. Beskales, M. Cherniack, S. Zdonik, A. Pagan, and S. Xu. Data curation at scale: The Data Tamer system. In *CIDR*, 2013.

[31] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *WWW*, pages 697–706, 2007.

[32] M. Vartak, S. Rahman, S. Madden, A. Parameswaran, and N. Polyzotis. Seedb: Efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 8(13):2182–2193, Sept. 2015.

[33] J. Wang and N. Tang. Towards dependable data repairing with fixing rules. In *SIGMOD*, pages 457–468, 2014.

[34] E. Wu and S. Madden. Scorpion: Explaining away outliers in aggregate queries. *PVLDB*, 6(8):553–564, June 2013.