DELFT UNIVERSITY OF TECHNOLOGY

MASTERS THESIS

Which error detection tool to choose?

Author:
Martijn VERMEULEN

Supervisor:
Dr. Asterios KATSIFODIMOS

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Web Information Systems Group
Software Technology

Student number: 4390784

Thesis committee:		
	Prof.dr.ir. G.J.P.M. Houben,	TU Delft, chair
	Dr. A. Katsifodimos,	TU Delft, supervisor
	Dr.ir. G. Gousios,	TU Delft

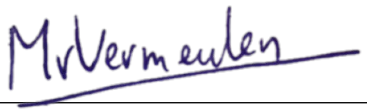
An electronic version of this thesis is available at
https://repository.tudelft.nl/.

August 16, 2020

Declaration of Authorship

I, Martijn VERMEULEN, declare that this thesis titled, "Which error detection tool to choose?" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: 

Date: 16-08-2020

DELFT UNIVERSITY OF TECHNOLOGY

Abstract

Electrical Engineering, Mathematics and Computer Science
Software Technology

Master of Science

Which error detection tool to choose?

by Martijn VERMEULEN

The amount of data being collected is growing exponentially, both in academics as well as in business. Unfortunately, the quality of that data can be poor, leading to poor decisions and increasing costs. Data cleaning, the process of detecting and correcting errors from a dataset, could be the solution to improve bad data.
This research focuses on detecting these errors. There are (semi-)automated error detection tools available, but it is unclear how well these tools perform under varying conditions and on different datasets.
Following from this problem, the main research question was developed: How to choose a fitting error detection algorithm for a specific relational dataset?
To answer this question, a comparative study has been done for error detection tools on relational data. An interactive error detection tool, Raha, performed best from the selected state of the art tools.
Subsequently, an attempt was made to estimate the performance of error detection tools and particular configurations on unseen datasets, based on high-level profiles of these datasets. According to the qualitative and quantitative experiments in this research, the proposed estimators have been shown to be effective. Moreover, the performance estimators were analyzed to provide more interpretability on the functioning of the error detection tools on the datasets in this research.
Ultimately, these performance estimators were used to generate suggested rankings of error detection strategies. The produced system outperformed the set baseline and was able to create valuable rankings. The proposed strategy ranking system could help real-world computer scientists and data experts choose a fitting error detection algorithm for a specific relational dataset.

Acknowledgements

First of all, I would like to thank my supervisors Dr. Asterios Katsifodimos and Christos Koutras, for guiding me with this research. I have great respect for the recovery of Asterios and am truly grateful that he was still able to be part of the completion of this thesis. Furthermore, I would like to thank Christos for always being available for all the discussions we had, even if, due to circumstances in 2020, the meetings could not be in person. This thesis would not have been possible without all your help, conversations and ideas.

Also, I would like to thank my other committee members, Prof.dr.ir. Geert-Jan Houben and Dr.ir. Georgios Gousios, for taking the time and effort to be part of this thesis's final process.

In addition, thanks to the TU Delft and especially the Faculty of Electrical Engineering, Mathematics & Computer Science, which has been my academic home for the past six years, completing a BSc Electrical Engineering and now the MSc Computer Science.

Last but not least, I would like to thank friends and family for helping me throughout the process, by reading my thesis, thinking along or just having a good conversation with me about all kinds of subjects.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	**A**rtificial **I**ntelligence
CFD	**C**onditional **F**unctional **D**ependency
CS	**C**omputer **S**cience
DC	**D**enial **C**onstraint
DCG	**D**iscounted **C**umulative **G**ain
FD	**F**unctional **D**ependency
MAE	**M**edian **A**bsolute **E**rror
MAR	**M**issing-**A**t-**R**andom
MCAR	**M**issing-**C**ompletely-**A**t-**R**andom
ML	**M**achine **L**earning
MSE	**M**ean **S**quared **E**rror
NDCG	**N**ormalized **D**iscounted **C**umulative **G**ain
NP	**N**on-deterministic **P**olynomial time
P	**P**recision
PCA	**P**rincipal **C**omponente **A**nalysis
R	**R**ecall
RDF	**R**esource **D**escription **F**ramework
RDFS	**R**esource **D**escription **F**ramework **S**chema
Regex / RE	**Reg**ular **Ex**pression
SHAP	**SH**apley **A**dditive ex**P**lanations

Chapter 1

Introduction

This first chapter will cover an introduction to the main research topic and motivation of this thesis in Section 1.1 and Section 1.2. Then the research questions will be introduced in Section 1.3. The main contributions will be covered in Section 1.4 and lastly, the outline of this thesis will be shown in Section 1.5.

1.1 Error detection

1.1.1 A data story

In 2020, everybody knows the word *data*. Everybody wants to use data to improve their business, their academic work or even their daily lives. However, what does it actually mean to *use data*? To give a clear idea of what *using data* could look like, we take a look at an example scenario of a fictional character named "Robert".

Robert is working at a local bakery, selling different french baked goods. His bakery sells croissants, baguettes and macarons. Robert heard from a friend that he should use *data* in order to improve his business. He came up with the idea to keep an Excel sheet with all the products that were sold. Then, after a full day of hard work, he wanted to show the owner of the bakery which product contributes the most to the revenue, so that they might focus more on that product. At the end of the day, the sheet looked like shown in Figure 1.1. Each line (row) of data contains which order number the product belonged to, the product name and price, time sold and the seller of the product.

Because it was the first time gathering data, some of the values might have been entered incorrectly in the Excel sheet. Nevertheless, Robert goes and makes a nice pie chart to show his boss the findings of that day. The graph in Figure 1.2 shows the pie chart with the revenue contribution for that day.

Immediately, his boss tells him that he does not believe this graph and walks out of the office. If you look closely at Figure 1.2, you see that something weird has

Order Id	Product Name	Price	Time Sold	Sold By
1	Croissant	2.00	10.56	James
2	Croissant	2.00	11.57	James
3	Baguette	3.00	13.42	Robert
a	Macarons	1.50	13.42	Robert
4	Baguette	30.00	32.10	Charle
5	Krosant	2.00	23.55	Charles

FIGURE 1.1: Robert's data after a day of work

happened. Suddenly, 4 product categories are shown. "Krosant" does not even exist, so this graph cannot be correct either. Also, it could not be true that the shop got 81% of their revenue through "Baguette", that did not feel right for Robert. Robert just experienced an example of **"Garbage in, Garbage out."** Somehow, using the data that was gathered, was not helpful at all.

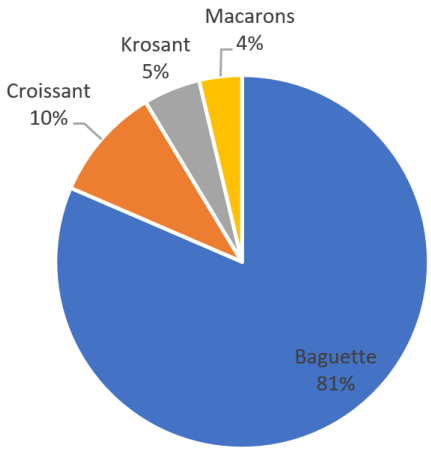


FIGURE 1.2: Robert's first revenue contribution chart

Robert quickly goes back to his sheet to see what went wrong. Taking a closer look, Robert sees that mistakes have been made. He sees that values in Figure 1.1 cannot be true, as they do not belong to the values that can be possible. For example, "Krosant" must be "Croissant" and a "Baguette" only sells for 3.00 euros, not 30.00. The values that he altered, were **errors**. After Robert correct the data, it looked like Figure 1.3.

Order Id	Product Name	Price	Time Sold	Sold By
1	Croissant	2.00	10.56	James
2	Croissant	2.00	11.57	James
3	Baguette	3.00	13.42	Robert
3	Macarons	1.50	13.42	Robert
4	Baguette	**3.00**	**23.10**	**Charles**
5	**Croissant**	2.00	23.55	Charles

FIGURE 1.3: Robert's corrected data

Being more confident that the saved data is better now, Robert creates a new pie chart, as shown in Figure 1.4. Going back to his boss, he shows the new results. The boss now trusts Robert's results and graph. The chart gives a proper idea on which of the 3 products drives revenue, giving more insights and the possibility of acting upon that insight. And this is all the result of a single chart and well-cleaned data!

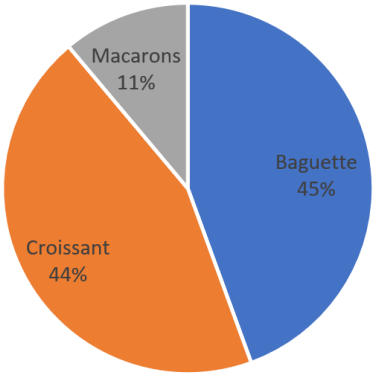


FIGURE 1.4: Robert's corrected revenue contribution chart

1.1.2 Research topic

The story told in the previous subsection was a story about using data. It started with collecting data. Then finding out that the collected data was not fit for generating insights. The data had to be cleaned. Data cleaning is a process that is the foundation of every individual or group that wants to do data science.

> Data cleaning is the process of ***detecting*** and correcting (or removing) corrupt or inaccurate records from a **record set, table, or database** and refers to **identifying incomplete, incorrect, inaccurate or irrelevant parts** of the data and then replacing, modifying, or deleting the dirty or coarse data.[1]

In this thesis, the research topic will be the data cleaning process. As this process is a fairly broad concept, a more concentrated part will be focused on. Only the first step of data cleaning, the *detection* of dirty data or so-called **error detection** will be the scope of this research. As seen in the definition of data cleaning, the goal of error detection is to identify incomplete, incorrect, inaccurate or irrelevant parts of the data.

Moreover, the focus will be on relational data sources. These types of sources follow the relational model Codd, 1970. This is a structure that presents the data in tuples or rows. Each row is grouped into relations that are defined for the complete table. A relational dataset has different attributes or columns. Also, each row in that dataset will have a value for these attributes, also called a data cell. Data sources in the relational model are queryable using SQL (Structured Query Language).

The example data of Subsection 1.1.1 is also relational data, as shown in Figure 1.1 and Figure 1.3. While the data in both Figure 1.1 and Figure 1.3 refer to the same real-world entities, events or other characteristics, there is a difference between the two. The difference between the two versions is the correction of unwanted values to the desired values. The unwanted differences are called errors.
So an error in relational data is a deviation from the desired ground truth values.

The problem a user tries to solve in this domain is to identify all erroneous cells in a relational dataset. In the examples given above, manual cleaning is still possible,

[1] Definition by Wikipedia contributors, 2020

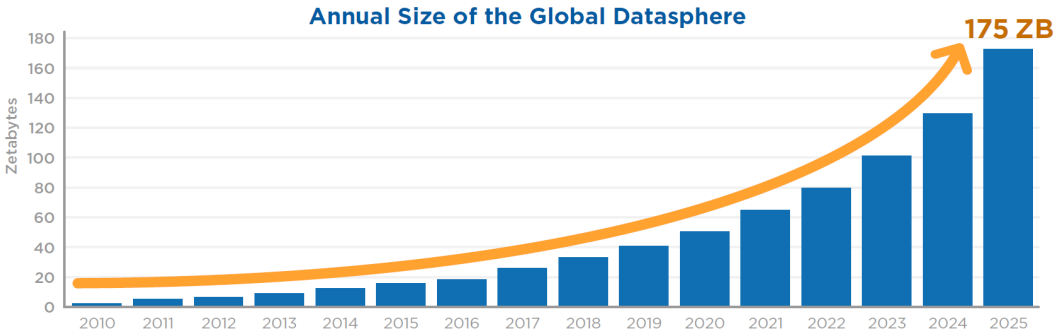


FIGURE 1.5: Annual size of the global datasphere (Rydning, 2018)

but with increasing size of datasets, increasing costs in cleaning arise. Luckily, there exist (semi-)automated error detection tools. These algorithms or tools try to identify erroneous cells, based on common deviations from the ground truth. These common errors could be outlying numerical values (like the "baguette" costing 30.00 euros), missing values or misspellings ("Krosant" vs "Croissant"). Questions that arise when trying to find a tool that helps to detect errors are:

- Which algorithm is the best?
- How will it perform when I run it on my data?
- How to configure that tool?
- Why would it work and can we understand when it does not work?

These questions are the foundation of the work presented in this thesis. Now that the topic has been introduced, the motivation of the subject will be covered in the next section.

1.2 Motivation

Errors in relational data are present in all fields, both in academics as well as in the industry.

In business, it is known throughout various studies and news articles that the cost of bad data is high. Poor data across businesses and the government cost the U.S. economy $3.1 trillion a year in 2012 (Ilyas and Chu, 2015). Every company nowadays wants to be more 'data-driven', but does not foresee the challenge that arises with processing data. Also, the amount of data created and stored is growing exponentially. The total estimated amount of data stored has grown by an approximated ten-fold since 2012 and will only continue to grow according to industry experts (Figure 1.5), to an estimated 175 zettabytes[2] by 2025. This growth in data creation will lead to a growth in the amount of poor data and consequently growth in costs if not properly dealt with.

Companies want to use data for various reasons, such as: maintaining good customer relationships, improving organisation efficiency and drive useful data-driven insights. And at the same time following all the latest privacy regulations. But, in order to pursue these aspirations and be able to extract value from data, numerous

[2] 1 zettabyte = 1 billion terabytes = 10^{21} bytes

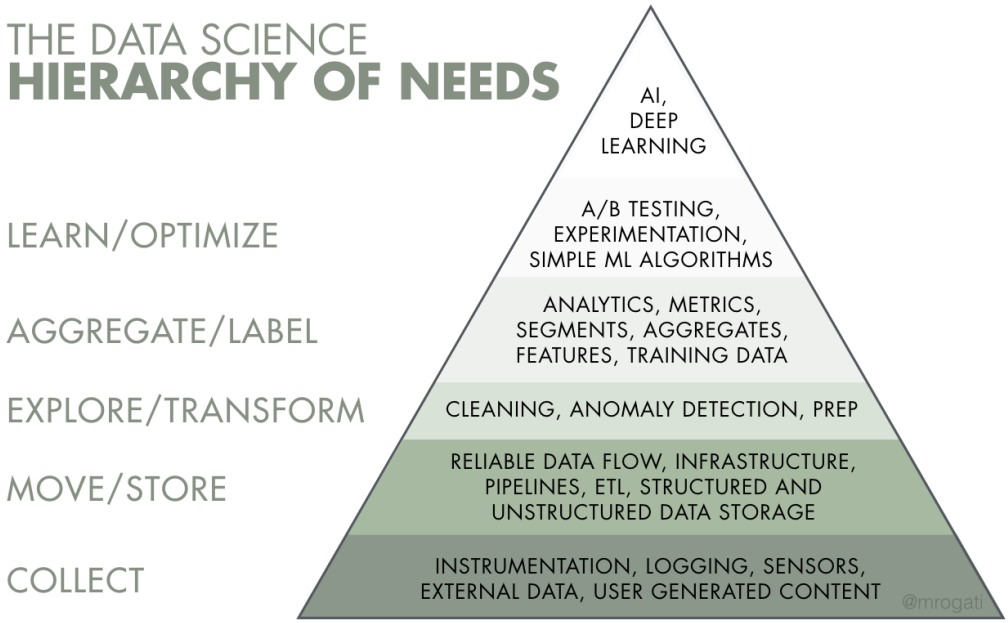


FIGURE 1.6: The data science hierarchy of needs[3]

steps have to be taken beforehand. A pyramid of data science or artificial intelligence (AI) needs is shown in Figure 1.6. This shows that before any aggregating or learning from data can be done, the data needs to be explored and transformed properly. This is where error detection comes into play. If the data cleaning step with error detection as the foundation is not correctly executed, the upper levels of the data science hierarchy cannot be reached properly. Without data cleaning, valuable actions like analytics, experimentation or even artificial intelligence will not be possible.

For the past few years, governments, industry and academics have been investing both money and effort into big data and the data processing pipelines in general (Cai and Zhu, 2015). As data quality issues in relational data sources have been a pressing problem for the industry for a longer period of time, researchers now try to take learning from experience in the field in corporate businesses (Stonebraker and Ilyas, 2018). Working with industry allows scientists to have access to more resources and real-life examples of the complications of errors in relational data, which could accelerate the solution-finding process. This is beneficial for academics, as the importance of data quality in research is also prevalent. Poor data quality might not have the monetary consequences in academics like it has in business, but bad quality in academics could lead to falsely substantiated conclusions.

Unfortunately, data cleaning does not have a single solution that is omnipotent and time-efficient. The complexity and heterogeneous nature of relational data make the error detection task more difficult to tackle. There is no single dominant tool for general-purpose error detection and domain-specific tools achieved better precision and recall scores than general-purpose tools (Abedjan et al., 2016). This leads to the difficulty of selecting the right tool for the job, with the right configuration. A

[3]Monica Rogati - https://hackernoon.com/the-ai-hierarchy-of-needs-18f111fcc007

user that is not familiar with the range of available options will not be able to select the best tool. Running all tools will be too time-consuming and will not have a guaranteed result. Preferably, a user wants automatic error detection, without the requirement of human expertise or any sort of configuration.

If it even would be possible to provide an automatic tool, without the need of a human expert, new questions arise. If a machine learning (ML) model performs well, why do we not just trust the model and ignore why it made a certain decision? "The problem is that a single metric, such as classification accuracy, is an incomplete description of most real-world tasks." (Molnar, 2020). That is where interpretability comes into play. Not all ML systems require interpretability, but those where the problem is not studied and validated enough, or the consequences of unacceptable results are too significant need an explanation of how it works (Doshi-Velez and Kim, 2017). Interpret means to explain or to present in understandable terms. Because of the complexity and all different forms relational data can have, error detection in that landscape is too complex and can have too significant consequences to take its workings for granted.

Whilst there have been many solutions proposed for the task of error detection, there are still many improvements to be made. The following main problems have occurred in recent public work:

Usability problems: There are many error detection tools and other data cleaning solutions that are not available to the public. Tools that are openly available have their challenges to prepare for usage in real life. Expertise is needed to configure and set up the tools. Also, the input and output formats deviate per tool.

Completeness problems: Due to the great amount of error detection tools available and the inherent costs that come with them to run, not all proposed tools in research have been tested on a broad range of datasets. Vice versa, not all types of datasets have been cleaned with the different tools available. This means that there are many unknowns when it comes to the performance of tools in all sorts of datasets or selecting the best tool for a specific dataset.

Interpretability problem: All proposed tools have underlying common principles and techniques that they are designed to work with. In real-life datasets, these techniques might produce different results than expected. At the moment, no research has used interpretability methods to understand how the error detection tools work in retrospect, after evaluation. This would give better insight if and how the tools would work on unseen real-life datasets, and if there are any hidden side-effects or influences on the performance of these tools.

1.3 Research Questions

Resulting from this motivation, the following research questions will tried to be answered in this thesis:

Main Research Question:
How to choose a fitting error detection algorithm for a specific relational dataset?

With the following sub-questions:

RQ1 What is the current state of the art and what is the performance of these tools?

RQ2 Is it possible to create an extensive data profile to estimate performance on unseen datasets?

RQ3 Is it possible to generate a ranking of tools according to their performance on unseen datasets?

RQ4 Do these data profiles provide more interpretability of error detection tools?

1.4 Contributions

By answering the research questions above, the following main contributions have been made in this research:

- An error detection framework with six different error detection tools
- A comparison study of error detection algorithms for relational data
- A suggestion system for error detection tools and configuration for unseen relational datasets

1.5 Outline

First, in Chapter 2, the background on error detection in relational data is given. The foundational information on error types, scores and error detection benchmarking, state of the art tools and methods will be covered. Also, the background on dataset profiling and interpretability with respect to error detection will be given. In Chapter 3, the tools explicitly used in this research are discussed. In Chapter 4, the methodology of this research is highlighted in order to answer the research questions. Specifically used datasets, error detection tools and machine learning techniques will be covered. In Chapter 5, the experimental setup of this study is explained. Then, in Chapter 6, the results of the research will be presented. Lastly in Chapter 7, conclusions and future work will be discussed.

Chapter 2

Background

2.1 Errors in relational data

The perspective in this research is about comparing two versions of the data. One "dirty" dataset and one "clean" dataset. Values from the "dirty" version that are not equal to their "clean" counterparts are counted as errors. These errors come in all different types, in terms of content or methods of detection. Also, different metrics or scores can be used to evaluate the found errors or to analyze a "dirty" dataset compared to the "clean" version.

In the following subsections, different error types and metrics to evaluate error detection methods will be discussed. A summary of different available error detection tools and their workings will be discussed. Also, the basics datasets in this field and synthetic error generation are discussed to sketch a comprehensive view of the current error detection environment in research.

2.1.1 Error types

In research, errors have been described with error types. A compilation of these types can be found in the survey by Abedjan et al., 2016. It gives the following types as broad categories for errors:

- Outliers
- Duplicates
- Rule violations
- Pattern violations

These categories are overlapping and non-exhaustive. The types above mainly describe how the errors can be found, not necessarily how they are introduced.

Outliers include data values that deviate from the distribution of values in a column of a table. “Characterizing, locating, and in some cases eliminating these outliers offers interesting insight about the data under scrutiny and reinforces the confidence that one may have in conclusions drawn from otherwise noisy datasets. Pit-Claudel et al., 2016”.

Duplicates are different tuples, referring to the same real-world entity. If attribute values do not match, this could signify an error in the data, but could also be the result of bad schema design, where redundancy of information is present.

Rule violations refer to values that violate any kind of integrity constraints, such as Not Null constraints and Uniqueness constraints.

Pattern violations refer to values that violate syntactic and semantic constraints, such as alignment, formatting, misspelling, and semantic data types.

Other errors include wrong references (Rahm and Do, 2000), wrong content or other types of errors do not fall into the other categories, but are different from the ground truth.

It is also important noting that this research will focus on single-source data error problems (Rahm and Do, 2000). In real-world environments, the problems and challenges that arise out of data errors will become more and more complex with a growing number of sources. Fixing errors with a single source as the scope is less complex. Also, most error detection methods only support using a single dataset. However, when a single source is cleaned, the detected errors and repair value pairs can be iterated through other sources.

2.1.2 Metrics & Data quality

Data quality A term often used in data cleaning is "data quality". Data quality describes a dataset in a number of dimensions. Depending on the task at hand, these dimensions might differ. The basic set of dimensions is accuracy, completeness, consistency, and timeliness (Batini et al., 2009). One can refer to either the data values or the schema with regards to the quality. Data quality also gives information about the usability of the data in analysis. If the quality is high, the results of visualizations, aggregate values or other grouped actions are close to the expected results. So higher data quality gives more confidence in using a dataset.

Accuracy is the metric consisting of the sum of all true positives and true negatives, divided by the total number of instances. Whenever working with imbalanced datasets, this number could give misleading confidence in the score, whereas the larger type (positives or negatives) dominates the score.

Precision is the metric consisting of the count of all true positives, divided by all the true positives and false positives. This metric is the fraction of relevant instances among all the "detected" positives. High precision means that the quality of errors detected is good, without having a large fraction of false positives. This metric could be helpful for completely automated processes. If precision is 1, no mistakes were made in the detected errors and no human interaction is needed. This metric however, does not tell anything about the portion of total errors retrieved and errors still left in the dataset, only about the quality of the instances predicted to be errors.

Recall is the metric consisting of the count of all true positives, divided by the count of all true positives and false negatives. Whereas precision measures the quality of positively labeled instances, recall is the fraction of correctly detected errors divided by the total number of true errors in the dataset. This metric focuses on retrieving all errors from the dataset, without any regard for the quality of the predictions.

F1-score is the metric that is the harmonic mean of precision and recall. So it is a combination of both recall and precision, measuring each score with equal importance. Due to this mix of importance, it is usable as a single metric, while having a larger scope of measurement.

Automation

Depending on the error detection task at hand, a metric from the list above could be selected to rate the error detection method. For automating the error detection task, high precision could be emphasized. As discussed by Huang and He, 2018, for automatically detecting errors, the aim should be to set a very high bar in terms of precision, as users would quickly lose confidence if the system keeps generating spurious alerts. Methods that achieve higher recall could be useful in scenarios where manual checking after detection is done, to ensure higher coverage, but at a higher human cost.

Benchmarks & Evaluation

Public benchmark methods for error detection and data cleaning in relational data sources have been scarce.

CleanML (Li et al., 2019) investigates the impact of data cleaning on downstream machine learning models. Besides providing numerous datasets for testing, it focuses on the impact of cleaning separate error types on the end machine learning task. It tackles the following error types:

1. Inconsistencies
2. Duplicates
3. Mislabels
4. Outliers
5. Missing Values

Types 2, 3 and 5 are known from other research, whereas inconsistencies and mislabels are more niche category descriptions. Inconsistencies can be covered as rule violations and mislabels would fall under the "Other errors" from Subsection 2.1.1. Unfortunately, this method does not cover holistic data cleaning. Due to the high overlap in both the descriptions of error types, in combination with treating single error types separately, it cannot properly tackle the task at hand in this thesis.

Data quality estimation is a way of evaluating the results of a cleaning method, without knowing the ground truth. The research by Chung, Krishnan, and Kraska, 2017 shows a method to estimate data quality, by letting a human iteratively judge parts of the data, to get a good estimate of the accuracy and quality, without having access to the ground truth beforehand. Its main point of view is estimating the number of errors based on the initial "dirty" dataset. This aligns with the goals of this thesis. However, because the ground truth is known for all the used datasets and due to the stochastic nature of this method, it is not suited for the empirical study in the upcoming sections. This concept might be used as an extension to this work, when adding larger datasets where the ground truth is not known beforehand.

2.1.3 Datasets & Error generation

In contrary to the limited number of benchmark methods in data cleaning, there have been many efforts to publish clean and dirty dataset pairs, to be used for evaluating error detection and repairing methods. Also, it has been used in research to synthetically introduce errors in datasets that were clean beforehand.

Datasets

A list of datasets extracted from different papers and researches will be used in this thesis. There are multiple ways of communicating a dirty version of a dataset and its ground truth counterpart. The most useful way for this research of denoting a dataset and its ground truth is explained by Mahdavi et al., 2019. A clean and dirty version are handled as deterministic and known values. The datasets will be provided by 2 different files (or in-memory tables), like depicted in Figure 2.1a and Figure 2.1b. Whenever such a dirty-clean dataset pair is used, it is assumed to be correct. The difficulty of data cleaning lies in the ambiguity of what the ground truth might be.

Order Id	Product Name	Price	Time Sold	Sold By
1	Croissant	2.00	10.56	James
2	Croissant	2.00	11.57	James
3	Baguette	3.00	13.42	Robert
a	Macarons	1.50	13.42	Robert
4	Baguette	30.00	32.10	Charle
5	Krosant	2.00	23.55	Charles

(A) A dirty dataset d

Order Id	Product Name	Price	Time Sold	Sold By
1	Croissant	2.00	10.56	James
2	Croissant	2.00	11.57	James
3	Baguette	3.00	13.42	Robert
3	Macarons	1.50	13.42	Robert
4	Baguette	**3.00**	**23.10**	**Charles**
5	**Croissant**	2.00	23.55	Charles

(B) Its ground truth d*

FIGURE 2.1: A dirty-clean dataset pair

Other research provides errors in other structures. For example, with MDedup (Koumarelas, Papenbrock, and Naumann, 2020), dirty datasets are provided. Separately, files with duplicate rows are provided, stating each source row id and duplicate row id.

CleanML (Li et al., 2019) provides separate clean and dirty datasets for each different error type. For this thesis, error detection tools will be evaluated holistically, meaning that the dirty and clean dataset should include all different error types in one version. To merge the split versions, the dirty datasets are merged together, as the ground truth stays the same for all versions. This will leave 2 datasets, one dirty and one clean.

Error generation

To evaluate all error detection tools, a ground truth needs to be known and errors need to exist in the dirty dataset.

Crowdsourcing & Expert labeling To generate dirty-clean dataset pairs, real-life dirty datasets can be used and corrected by experts. This is a timely and costly process, with increasingly higher complexity with growing datasets. The cost of labeling causes a limited number of train and test datasets for this research field. The work presented by Sheng, Provost, and Ipeirotis, 2008 shows that repeated labeling can improve the data quality for supervised learning. Even when the individual accuracy of labeling from crowdsource labelers is not 100%, it is possible to generally retrieve higher quality labels than a single labeler would generate. Repeated labeling could be a way of creating the ground truth from a real-world dataset, by letting users correct cells in a given dataset.

Synthetic errors To circumvent the high costs of labeling, BART (Arocena et al., 2015) was proposed to synthetically introduce detectable errors into clean datasets. The error-generation problem was shown to be NP-complete[1]. The authors made a distinction between detectability and repairability. When introducing errors into a dataset, it is important that generated new values can actually be considered errors. Therefore, a user of BART must provide certain constraints on the data to specify when and if a value is a violation. Also, the user can then provide the portion of the dataset that should be turned into errors. Besides the evaluation of error detection tools, BART can also be used to measure error correction (repair) tools. Whenever an error is introduced, it is known whether or not this value lies in the specified domain of values and if the ground truth value can be found in other parts of the data. If this is not the case, the repairability decreases, as it is a hard process to find the ground truth. However, repairing is outside the scope of this thesis and the main focus will lie on the error generation with detection as the priority task.

2.2 Error detection tools

In order to automatically identify incomplete, incorrect, inaccurate or irrelevant parts of the data, numerous methods have been proposed in research. First, different base error detection methods & frameworks will be discussed that follow a single mechanism. In the second subsection, composite methods will be discussed that have state-of-the-art performance and incorporate some combination of different base techniques.

2.2.1 Error Detection Methods & Frameworks

To start, the methods underlying the most common methods will be discussed.

Regular expression violation In many error detection methods, some form of text pattern recognition is used to detect errors. This mostly comes down to verifying a regular expression (regex). It is an expression that describes a set of strings (Mitkov, 2004). If a cell value in a relational dataset does not fit in this set describing these cells, it will be a violation of this regular expression. An example of this could be a column in a dataset which contains year values. A regex that would take correct 4 digit year values would be: `^\d{4}$` (match only those strings that begin and end with 4 digits).
1996 would be a correct value, but 2oi9 would be a violation.

[1] NP-complete problems are in NP (Non-deterministic Polynomial time), the set of all decision problems whose solutions can be verified in polynomial time

Product	Price
Croissant	*€2*
Baguette	€3
Croissant	*€4*
Baguette	€3

TABLE 2.1: A table with FD *Product* $\rightarrow$ *Price*, with violations for product "Croissant"

Despite the straightforward workings of this method, it can be costly in terms of time and human expertise to derive these regular expressions by hand. Automatic regular expression inference is possible, for instance as proposed by Bartoli et al., 2016. This means that it is possible to learn the regular expression (if there is any), describing a column or part of a text. This then could be used to find errors, as regular expression violations might indicate an error. But one must take into account that if a dataset contains many errors, the regular expression learned might be that of the erroneous values, possibly creating a faulty detection method.

(Conditional) Functional Dependency violation & Denial Constraints In this part, (conditional) functional dependencies and denial constraints will be discussed. They are among the family of rules/constraints that are similar in the detection of violations, characteristics and approaches of creating. The discussed methods are part of a non-exhaustive list of methods that could be used to find errors that are categorized as rule violations. A functional dependency (FD) is a relationship between two attributes of a relational dataset. Meaning that values in a certain column will indicate what values in other columns will be. Given a functional dependency R, a set of attributes (a column) X functionally determines another set of attributes (another column) Y, denoted as $X \rightarrow Y$. Each value of X maps to another single value Y. So if there is a FD set on columns *Product* and *Price* in Table 2.1, a violation occurs for product "Croissant", indicating that one of them must be an error.

Conditional Functional Dependencies (CFDs) are an extension of FDs, where another condition must be met before the relation withholds. This allows greater configurability and the possibility of exceptions. Denial Constraints (DCs), as discussed in Chu, Ilyas, and Papotti, 2013, is another method to enforce correct application semantics to a database. It allows more expressive language for defining rules, but still has enough structure to allow automatic discovery and to be usable. A DC states that all the predicates cannot be true at the same time. Otherwise, it indicates a rule violation.

Work by Asghar and Ghenai, 2015 shows that there is a great number of automatic (conditional) functional dependency discovery tools. Also, newer work (Rammelaere and Geerts, 2019b) still dedicates energy to finding, optimizing and comparing new methods. Similar to finding regular expressions, manually creating rules in the form of dependencies of constraints can be costly. Most automatic discovery tools however, rely on data without too many errors. This introduces difficulties for detecting errors in datasets with bad data quality.

NADEEF (Dallachiesa et al., 2013) is a data cleaning tool that has a programming interface where users can define different types of data quality rules about both their static semantics and dynamic semantics, and a core that implements algorithms to detect and repair dirty data by treating multiple types of quality rules holistically. It

is a commonly used tool to detect data errors in research. Especially when certain (C)FDs are known, they are easily implementable and NADEEF has been proven to be effective. However, for this research and as this tool requires manually programmed rules, this tool is not suited for standalone error detection.

dBoost (Pit-Claudel et al., 2016) is "Outlier Detection in Heterogeneous Datasets using Automatic Tuple Expansion". This work demonstrates the expansion of data tuples using knowledge about the schema and field types. For example, a timestamp 1424866716 could be expanded into year 2015, Wednesday, etc.. Then using statistical analysis of these expansions, the data is modeled using machine-learning algorithms (Histograms, Gaussian, and Mixtures). Whenever a part of the tuple expansion does not lie in the data model for a certain user-defined threshold, the entry is flagged as an outlier.

FAHES (Qahtan et al., 2018) is a disguised missing values detector. Whereas most missing value detectors focus only on NULL or empty values, this tool takes a different approach. They categorize detectable disguised missing values into five different cases:

1. Out of range data values
2. Outliers
3. String with repeated characters or characters that are next to each other on the used keyboard
4. Values with non-conforming data types
5. Valid values that are randomly distributed within the range of the data and used frequently in the data set

Cases 1-4 can be treated as outliers, whereas case 5 is an inlier in the data distribution. The research shows detection using syntactic outliers, where syntactic patterns are discovered. By comparing pattern frequencies and checking for repeated patterns (for example, pushing the same key repeatedly), possible disguised missing values can be found. The research also proposes a more simple density-based numeric outlier detection method. For the last case, detecting errors as inliers in statistical models is proposed. The method is capable of detecting disguised missing values that are used frequently throughout the dataset (and are therefore inliers), by verifying if data columns are following two statistical missing value models, MCAR (missing-completely-at-random) and MAR (missing-at-random).

Forbidden Itemsets (Rammelaere and Geerts, 2019a) applies a constraint-like method to detect and repair invalid entries in a dataset. However, it assumes real-world scenarios, where constraints or rules specifying when data is dirty are not available. The proposed so-called forbidden itemsets capture unlikely value co-occurrences, similar to denial constraints. Experiments show high precision on detected errors. However, one main requirement is that a large body of clean data is needed to detect forbidden value combinations: if most of the data is dirty, inconsistencies are no longer “unlikely”, and errors will not be considered forbidden.

ActiveClean (Krishnan et al., 2016b) is a general-purpose error detection tool. Using interactive human labeling, it iteratively trains a machine learning model to specify between clean and dirty data entries. Because the proposed method is using a generalized featurizer, the method can be used for different data cleaning problems, not only for relational data cleaning. But, in order to use it for relational data, each cell needs to be converted to a set of numerical input features using a user-defined featurizer. A predictive model will then be trained on these numerical features, for every step in the process. After each step, the training and test set will be updated, in the traditional active learning way, sampling from both the most certain predictions and the human labeling input. This will continue until all input cells are classified with high probability or labeled by a human, or some predefined human sampling budget is reached.

KATARA (Chu et al., 2015) is a knowledge base powered error detection tool. It is capable of table pattern definition, discovery and validation using a combination of knowledge bases like DBpedia (Auer et al., 2007) and crowdsourced human labeling. In this research, only knowledge bases are used as a source to find errors. The main problem KATARA can tackle is finding approximate table patterns and relations. Knowledge bases are considered RDF-based data consisting of resources, whose schema is defined using the Resource Description Framework Schema (RDFS). A resource in this schema is any real-world entity. The RDF-based data then describes the relation between different resources for a specific domain. Take, for example, the resources in the domains of countries and cities. The two resources respectively could be `France` and `Paris`. A knowledge base entry could then describe the relation `hasCapital`, with a specific entry `hasCapital(France, Paris)`. KATARA passes through all columns and finds the overlap of that column for a specific resource. If a column is covered for a certain threshold by a resource, then it is supposed to be that type. So if a large part of a column matches country resources (not necessarily all) and another matches cities, KATARA will treat the relation between the columns as a functional dependency defined by the relation `hasCapital`. All pairs that do not match this functional dependency will be marked erroneous.

2.2.2 Composite methods

Combining all these different base techniques, the following methods proposed in research uses a combination of previous knowledge to achieve state of the art error detection performance.

Auto-Detect (Wang and He, 2019) is a data-driven error detection approach (very similar to another recent tool Uni-Detect Huang and He, 2018, but Auto-Detect outperforms Uni-Detect). It uses a large corpus consisting of 100M web tables extracted from a commercial search engine. Because it is extracted from a commercial search engine, the data quality is expected to be high. It tries to learn about the four common classes of errors. It performs single-column error detection and creates a generalization tree to translate actual values to patterns to compare. This tool however, needs a specific corpus related to the subject dataset. While this tool performs really well in its own tests, it is not tested on common benchmark datasets. Also, the implementation of this tool is not available publicly.

Metadata-Driven Error Detection Visengeriyeva and Abedjan, 2018 makes use of combining different algorithms by either bagging or stacking and adds content-based metadata to the features to enhance error detection classification. Augmenting system features with automatically generated metadata information will improve the error predicting outcome. This insight might help future work. The base error detection strategies in the ensemble learning methods need to be chosen and configured by hand, which is again a hard process. Also, this work has not been developed any further for a few months and other methods outperform this tool now.

HoloDetect Heidari et al., 2019 is a few-shot learning framework for error detection that uses data augmentation to create more synthetic training examples resulting in high performance. It also has an expressive model to learn representations and syntactic and semantic differences in errors. This work uses a lot of feature engineering fed into a neural network. Feature engineering is hard and might influence the performance, especially because the exact implementation is not public. Also, depending on the amount of training time and samples, this method varies in performance. One other note is that the released F_1 scores are not calculated correctly for some of their tests, making HoloDetect come out on top every time, whereas it would not have been the best. The key takeaway is that data augmentation could solve the imbalanced data problem and even outperforms active learning methods, reducing human efforts.

UGuide Thirumuruganathan et al., 2017 is an interactive tool that detects the set of functional dependency detectable errors under a fixed human effort budget. This work has a good focus on the human budget in data cleaning. One downside of this research is that all the errors were synthetically generated using BART (Arocena et al., 2015). It would be interesting to see if finding FDs will still be possible with highly erroneous datasets. It asks the user to verify tuple-based, FD-based and cell-based questions about the validity of dependencies. However, only using functional dependencies is not expressive enough to tackle the whole error detection problem, but the methodology of this work gave extra insights, especially on a human budget.

Raha Mahdavi et al., 2019 is a (partially) configuration-free error detection system. Using data profiles (like in REDS by Mahdavi and Abedjan, 2019, covered in Section 2.3), it selects different pre-configured strategies automatically, based on previously cleaned datasets. Then, it incorporates the outputs from various error detection strategies as a feature vector for the error detection task. Using these feature vectors, it creates clusters of which samples will be labeled in order to reduce user involvement. Drawbacks could be that information might get lost in the stacking process. Also, the (time) complexity of this tool increases vastly when adding more base learners.

ED2 Neutatz, Mahdavi, and Abedjan, 2019 or Example-Driven Error Detection, is a machine learning-based error detection technique that also makes use of active human labeling. Like HoloDetect Heidari et al., 2019, it creates features that cover information on the attribute, tuple (across attributes), and dataset level for each data cell of the dataset. It iteratively selects columns that will be sampled from, humanly labeled and propagated using different strategies. The main contribution is the effective selection of columns, reducing the user input effort by selecting columns & cells to be labeled, that will have the most positive impact on the learning. As with

all active learning tools, the drawback is the human interaction. The number of examples needed to be labeled by the user differs greatly. Also, no confidence intervals are given for the performance metrics, making it unsure what the worst-case performance will be.

2.3 Dataset profiling

In this section, dataset profiling will be discussed. Features need to be engineered to differentiate datasets in certain intrinsic properties. First, it will be covered on how a dataset profile could be of use for the estimation of the performance of error detection strategies. Afterwards, different approaches and challenges for profiling relational data will be discussed.

REDS (Mahdavi and Abedjan, 2019) estimates the performance of error detection strategies based on datasets using their so-called "dirtiness profiles". It takes a number of samples from the dataset, extracts content, structure and quality features from this sample. Then it trains a regression model to predict the performance for each strategy. For **N** strategies, there will be **N** different models to predict the performance. This is promising as their research did not require unrealistically large repositories. However, their estimation error was still decreasing in their tests by adding more data profiles to the repository. So a large repository could be possible and would lead to better estimations of the performance of strategies. The idea is also used in Raha Mahdavi et al., 2019, as a plugin, but the idea will be used in this research as a standalone recommendation for using error detection tools. The main idea is that similar datasets will have similar "dirtiness profiles". And similar datasets would require similar error detection methods. Key ideas from schema matching can be used to check for similarity between different datasets.

A survey of approaches to automatic schema matching (Rahm and Bernstein, 2001) This research covers different schema matching approaches using a taxonomy. “A fundamental operation in the manipulation of schema information is Match, which takes two schemas as input and produces a mapping between elements of the two schemas that correspond semantically to each other” Historically, schema matching has been a part of 4 main processes. Schema integration (creating a global view of independent schemas), designing of source-to-warehouse transformations for raw input to a central warehouse with fixed schemas, e-commerce with a focus on message transformation and semantic query processing. The common divider in these processes is the transformation of the input to some fixed, known output. Then the research distinguished different approaches between schema and instance-level, element- and structure-level, and language and constraint-based matchers and discussed the combination of multiple matchers. Combining that with additional information (like dictionaries), will give a toolkit for matching schemas and showing similarities. Using all these tools with the purpose of error detection, could give insight into the performance of a tool on an unseen dataset, which has similar similarities to already known datasets.

Profiling relational data: a survey (Abedjan, Golab, and Naumann, 2015) Goes with a focus on determining metadata about a given dataset. One of the insights is on the challenges of data profiling: (i) managing the input, (ii) performing the computation, and (iii) managing the output. For different input shapes and types,

some certain transformation will be done beforehand. Also, computing the different steps of data profiling might differ based on different content types, schemas or even completely different documents. Then managing the output to be useful for a specific task is a challenge on itself. Depending on the task, limitations on the output can be existing. In the case of this thesis, the output must be homogeneous, in order to be used for a machine learning task. This fixed output limits the possible profiling tasks that can be done or used in further calculations. According to this work, data profiling can be split up in 3 types: single column profiling, multiple columns profiling and dependencies profiling. Single-column profiling has different tasks that can be directly converted to quantitative input features for a machine learning task. Examples of tasks are finding the number of null values, number of distinct values or calculating the percentage of cells that are of a number data type. These tasks can be executed with a single pass over the data, making the time complexity linear. Multi-column (for example, clustering of rows) and dependencies (for example, number of functional dependencies) profiling can also be aggregated to fixed numerical outputs. However, the main drawback of these techniques is the higher computational costs. These costs make the profiling methods less favored than single-column profiling tasks. So the main take-away from this work will be that there are different usable single-column profiling tasks that can be aggregated to fixed quantitative outputs usable for machine learning.

2.4 Interpretability

The challenge of interpreting a machine learning model is based upon the desire to understand how and why computers and algorithms work. When models can be interpreted, they can be improved as well. This is part of the learning process. First, feature selection with dimensionality reduction as main goal will be discussed. Then, research about feature importance as a goal itself will be covered.

2.4.1 Feature selection

Research has shown that using too many input features for machine learning will decrease performance (Trunk, 1979), especially when the number of training samples is not sufficient. This means that from a set of selected input features, a smaller subset of features as input could improve the model.
Feature selection methods to find a representative set of features for the machine learning task at hand have been proposed by numerous researchers. Feature selection methods can be categorized into mainly two types: *wrappers* and *filters*.

A wrapper feature selection algorithm is one that executes the models themselves to evaluate which features to use. Although this can be costly in terms of time consumption, this gives a purpose-specific and comparable result on features. The most extensive wrapper feature selection method would be to run the model with every possible subset of features (powerset) and pick the best subset.

A filter feature selection algorithm is one that evaluates the features using heuristics based on assumptions of the data. The objective machine learning model is not used in this feature selection process. In practice, this method is faster than a wrapper method, but if wrong assumptions are used, filter methods could lead to worse performance results. Examples of filter methods would be to exclude features with

low variance or specific types. Also, correlation-based feature selection has been proposed in research, like by Hall, 2000. The work by Hall, 2000 proposed a feature subset correlation analysis, showing that a filter would not have to be limited by a single feature at a time. Also, it has shown that application in both machine learning classification and regression can be improved by using feature selection.

Feature selection is still a very important topic in machine learning. Especially in this era of big data, only finding the most representative features will improve the whole machine learning pipeline, from data mining to the actual performance of the models (Li et al., 2017). Also, newer feature selection types for different data types were discussed, such as streaming data or heterogeneous data like multi-source, multi-view, linked data. However, these advanced data types are beyond the scope of this thesis. Feature selection for unstructured conventional data suffices. This leaves the following types of feature selection: similarity-based, information-theoretical-based, sparse-learning-based, and statistical-based methods.

In this thesis, filter-based statistical-based methods and a wrapper based feature selection will be used. These methods filter out unwanted features in a simple and straightforward manner. Simple methods are still commonly used, even before applying other more sophisticated feature selection algorithms (Li et al., 2017).

2.4.2 Feature importance

Feature importance refers to methods that assign a score to input features based on how useful they are at predicting a target variable. One exemplary research on feature importance was published in 1990 by Porter, Bareiss, and Holte, 1990. The goal of the research was to not only to learn to classify, but also explain why a classification had happened. A conceptual dialogue between an expert and their introduced system shows the importance of having explainable machine learning. This allowed for the expert to tweak certain weights and importances that were not correctly learned and reflect on the given suggestions by a machine learning model. Besides directly learnable binary relations (if A holds, then B must hold), the authors also discussed featural importances. Meaning that some features would have a differential influence on other features or the classification (or regression) outcome. One of the main goals of statistical learning or analysis of machine learning models is to find the impact of predictive features on the variable of interest, given that the prediction model performs reasonably well (Altmann et al., 2010). Common feature importance methods rely on the learned model coefficients, support vectors or other directly learnable parameters. However, when more complex and less studied methods are used, the direct inference of importances might become impossible.

Permutation importance (Altmann et al., 2010) is a possible technique to analyze feature importance for complex machine learning models. The general idea proposed was that a permuted version of the features was used to train a model. Each feature x was randomly permuted to break the association between the feature and the predictive outcome. Then, a comparison between the non-permuted value and the permuted value was made to see how the performance would increase or decrease by permutation (randomization for x).

SHAP (SHapley Additive exPlanations) (Lundberg and Lee, 2017) is another promising technique to explain machine learning models. The Shapley value (Shapley,

1953) was a metric used in game theory to explain the utility of different users in a multi-person game. This later was tailored to visualize and interpret the contribution of features on the performance of a machine learning model (Lundberg and Lee, 2017). This method was able to explain which fraction of the performance can be attributed to which feature. Permutation feature importance is based on the decrease in model performance, whereas SHAP is based on the magnitude of feature attributions.

Chapter 3

Selected tools

In this subsection, the tools explicitly used in this research will be discussed. Besides the summaries in Section 2.2, a review focused on the inner workings with regards to error types (from Subsection 2.1.1) will be given below. The selected tools below represent the current state of the art in error detection tools. From the background research in Chapter 2, a subset of tools that was open-source or easily implementable was adapted and used in this research. The selected tools cover all the four main error types, of which a reference can be found in Section 3.2.

3.1 Tools

3.1.1 ActiveClean (Krishnan et al., 2016a)

This tool uses a **human in the loop to actively learn** which rows are errors. The main idea of active learning will be reused for this thesis. The researchers have proposed a basic data flow that describes how to implement an active cleaning method. The implementation created for this work uses tf-idf features (frequency-inverse document frequency), created using the scikit-learn `TfidfVectorizer`. Then, a linear model binary classifier is trained on the humanly labeled samples first and in following iterations, also on self-labeled erroneous rows. The implementation treats each row as a document. Treating each row as a document allows the tool to capture relational information. This allows for catching **rule violations**. If values across columns co-occur multiple times, the classifier can learn from human labeling whether these co-occurrences are erroneous or valid. The downside of this technique is that a full row will be labeled as either the truth or erroneous. However, on the positive side, it might be able to catch duplicate rows and mark these rows completely as erroneous, making it an error detection tool that could spot **duplicates**.

3.1.2 dBoost (Pit-Claudel et al., 2016)

dBoost is a tool that uses **expansion of data tuples using knowledge about the schema and field types to detect outliers**. The values retrieved from the expanded data tuples will then be modeled using histograms, univariate Gaussian models or multivariate Mixture models. Whenever the value of a histogram category for a data value is too low, it will be marked as erroneous. Also, whenever a value lies in the part of a Gaussian model with a density lower than a configured threshold, it will be marked as erroneous.

The data tuple expansion firstly allows for finding **pattern violations**. Due to the multivariate Gaussian models, co-occurrences of values across columns can be taken into account. This makes it possible to find **rule violations**. Lastly, of course, (numerical) **outliers** are detectable with this method. dBoost will model numerical

values with the Gaussian model. Outliers in this model will be simply marked as erroneous.

3.1.3 FAHES (Qahtan et al., 2018)

FAHES is a **disguised missing values detector**. Whereas most missing value detectors focus only on NULL or empty values, this tool takes a different approach. They categorize detectable disguised missing values into five different cases: 1. Out of range data values 2. Outliers 3. String with repeated characters or characters that are next to each other on the used keyboard 4. Values with non-conforming data types 5. Valid values that are randomly distributed within the range of the data and used frequently in the data set. Because missing values are categorized as **rule violations**, as stated in Subsection 2.1.1, this method can only detect those types of errors.

3.1.4 Forbidden Itemsets (Rammelaere and Geerts, 2019a)

Applies **constraint-like method** to detect and repair invalid entries in a dataset. The proposed forbidden itemsets capture unlikely value co-occurrences, similar to denial constraints. Denial constraints are a formal method of describing rule violations. Due to the similarity of the Forbidden Itemsets with these rules, this method is only capable of detecting **rule violations**.

3.1.5 KATARA (Chu et al., 2015)

A **knowledge base** and crowd-powered data cleaning system that, given a table, a knowledge base, and a crowd, interprets table semantics to align it with the knowledge base. KATARA tries to find relations between attributes that have many overlapping values with a type in the knowledge base provided. If, with sound confidence, a relation is found, the complete set of rows will be checked. If there are mismatches with the knowledge base and the rows, cells will be marked erroneous. As discussed in Section 2.2, KATARA makes use of knowledge bases with relations. It finds if a column in a target dataset is of a particular type, by finding the overlap of a column and a specific attribute type in the knowledge base. Then if possible, relations between found column types are proposed. These matching type relations will be checked as functional dependencies. Violations of these relations are rule violations. So KATARA is capable of finding **rule violations**.

3.1.6 Raha (Mahdavi et al., 2019)

Raha is a human-guided error detection system. It selects different pre-configured strategies automatically, based on previously cleaned datasets. It then incorporates the **outputs from various error detection strategies as a feature vector for the error detection task**. Using these feature vectors, it creates clusters of which samples. These clusters will be humanly labeled to actively learn which values to mark as erroneous. Because Raha uses different underlying methods to learn which values are errors, it is capable of detecting **pattern violations**, **rule violations** and **outliers**. Because of the active learning of errors, it is also possible that the classifier consequently learns to mark **duplicates** as erroneous, as (near) duplicate rows will be in the same feature vector clusters.

3.2 Error type coverage

The summarized underlying techniques are explained by the common error types that are detectable by the tools, as seen in Table 3.1.

Tool name	Pattern violations	Rule violations	Outliers	Duplicates
ActiveClean		✓		✓
dBoost	✓	✓	✓	
FAHES		✓		
Forbidden Itemsets		✓		
KATARA		✓		
Raha	✓	✓	✓	✓

TABLE 3.1: Table with the tools used and the common error types that are detectable by the tools

Chapter 4

Methodology

The methodology of this research will contain mainly four distinct parts. The first part covers research question 1. In this first section, the empirical study is clarified. The second section of this chapter covers research question 2, where the performance prediction of tools and strategies is enlightened. The third section will cover the ranking of these tools and strategies, covering research question 3. Finally, in the fourth section, research question 4 is covered about the interpretability of the tools.

4.1 Empirical study

This section will be dedicated to the methodology to answer the first research question: *What is the current state of the art and what is the performance of these tools?*;

The first part of the research question is covered in Chapter 3. Those tools are selected to represent the state of the art of error detection tools. To answer the latter part of the question and substantiate further research questions with research data, an empirical study on error detection tools and different configurations is designed. To cover a broad range of configurations (strategies) of these tools, a range of permutations of configurations will be generated for each error detection tool and run on a wide range of datasets with different characteristics and errors. The source code of this framework can be found on GitHub[1].
Because the source languages of error detection tools differ, a high-level general-purpose programming language suits to be used for the framework to connect all the different underlying tools. Together with the fact that it has well-supported libraries for relational data handling, Python[2] was chosen as the primary development language.

4.1.1 Setup

First, the setup for the empirical study will be discussed. The basic setup is shown in Figure 4.1. For each tool in the study, a number of configurations are created, depending on the configurability of the tool. Each configuration, in combination with that tool, will be tested on all the available datasets. So a single experiment exists of:

- An error detection tool
- A tool-specific configuration
- A target dataset

[1] https://github.com/MartijnVermeulen96/master-thesis
[2] https://www.python.org/

Each experiment will be pushed to the queue of experiments, allowing for retrieving an experiment session without having to rerun all the finished experiments. Incrementally, all the experiment results are pushed into the results database, allowing for the empirical study and further usage with performance prediction and tool ranking.

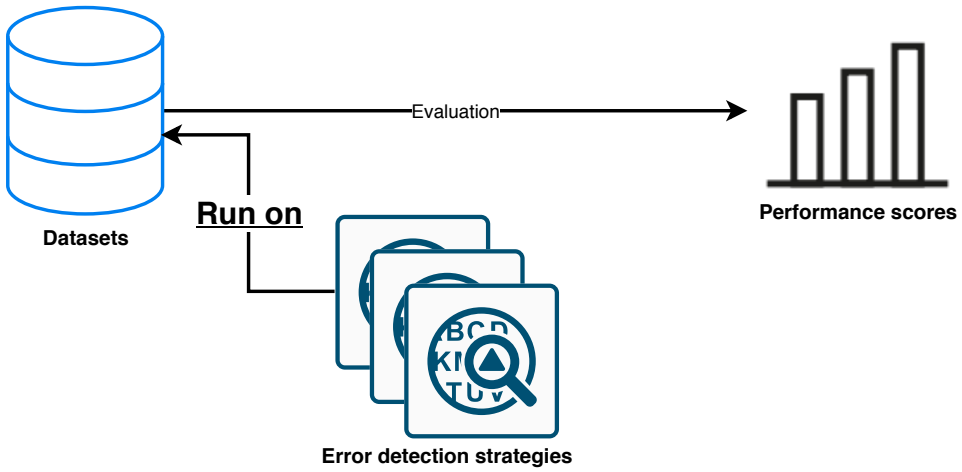


FIGURE 4.1: Setup of the empirical study

4.1.2 Error detection framework

A framework for running error detection tools was designed. The purpose of creating a single framework is to have homogeneous in and output for each selected error detection strategy, creating the possibility of running batches of experiments in a structured manner. The content and structure of the error detection framework are as follows:

- ➔ dataset
- ➔ experiment
- ➔ profiler
- ➔ tool
- ➔ tools (see Chapter 3)

dataset contains the class for handling datasets for the experiments. It is based upon the work done by Mahdavi et al., 2019, where each dirty dataset has a clean counterpart. Errors can be calculated both cell-wise as well as row-wise. Information about a dataset can be uploaded for later use and metrics like precision, recall and the F1-measure can be calculated from within this class.

experiment contains the workflow depicted in Figure 4.1. Objects from this class can create the tool-configuration-dataset triples for the empirical study. Also, it contains the incremental queuing system for the execution of experiments, which can be interrupted and resumed at will. Besides the workings shown in Figure 4.1, this part of the framework supports timeouts of the experiments, to shut down experiments whenever the preconfigured time limit is set. Lastly, it will upload the result metrics,

configuration and runtime to a centralized database to further filter and analyze the results.

profiler contains the elements to profile datasets and predict performance for different tools. This will be further discussed in Section 4.2.

tool contains the classes for creating error detection tool instances and the base class for tool. The tool creator dynamically finds added tools and is possible to return "Tool" instances for each error detection tool. Each error detection tool is adjusted to work in a similar fashion. The tool base class contains an abstract method to run the code, expecting similar input and output for each tool. The tool base class has subprocess functionality to support the timeouts set by the experiment and to run non-Python code. The tool base class also cleans up leftover subprocesses whenever timeouts are reached.

tools contains each error detection specific implementation of the Tool base class. Only the initialization and the `run` method need to be implemented for each tool work, making it easy to extend this framework. Every tool is implemented in a different submodule. The implemented tools are discussed in Chapter 3.

4.1.3 Datasets

The datasets selected contained different types of errors, to create a heterogeneous test set to compare the tools and configurations. Errors in a dataset are defined by the difference in the dirty dataset and clean dataset. So, only the ground truth makes up an error. Errors can be seen as a transformation of the ground truth to some dirty value. These transformations could specify why the dirty version is wrong. Below are different "error types" describing errors that have similar transformation between the dirty instance and the ground truth. Of course, these transformations might overlap, as one type of transformation could have the same effect as another type, but these categorizations give an overview of the dataset quality one has to deal with.

Error types

The error types are derived from Subsection 2.1.1, with the addition of the "other errors", which contains errors that cannot be found or described by the default error type categories. These are summaries and typical examples of the main error types:

- *Outliers:* Values that are erroneous and do not fit in the (column) distribution of the dataset
- *Pattern violations:* Values that are erroneous and do not fit in the common pattern of that column (i.e., 2k20 instead of 2020). Also error values that contain wrong spelling belong to this category.
- *Rule violations:* This category contains the violations of dependencies and/or more complex rules between columns. Also, empty values or missing value placeholders (e.g., N/A) belong to this category. Lastly, inconsistent abbreviations or references to entities (like states, companies, universities, etc..) also count as rule violations.

- *Duplicates:* These errors are different tuples, referring to the same real-world entity, while only one reference should be valid.
- *Other error types:* Examples of this type are values that result from a classification (output) based on the other given columns, but the output is wrong. Alternatively, an error value where the entity's content is erroneous, but other rules and patterns do hold.

4.1.4 Execution & Evaluation

In this section, different parts from the sections above will be combined to answer the last part of research question 1: *What is the current state of the art and what is the* ***performance of these tools?***
The tools from Chapter 3 will be run in selected configurations (tool-specific) on a variety of the datasets (Section 5.1). Different performance score metrics will be kept for each experiment. In a single experiment, a timeout limit is set to filter out tool configurations that do not satisfy runtime needs. The main performance scores that will be kept are:

- Precision
- Recall
- F1-score

The performance scores (results from experiments depicted in Figure 4.1) will be **aggregated into a table** showing the best results for a *single tool* (regardless of its configuration) and the *target dataset*, sorted on the F1-score. In the table, the rows represent a single dataset, and each column represents a tool. Such a table will allow comparison to see which of the tools performs best for which dataset. This result will show the performance of tools over a wide variety of datasets, holistically, for every error type. Secondly, for each error type (excluding "other error types") as shown in the previous section, separate results will be shown. Instead of tackling the error detection task holistically, experiment results will be shown only for **overlapping error types** occurring in the dataset and errors that are presumably detectable by the error detection tool (for error type coverage, see Table 3.1). Combining the findings from the overall analysis and analysis per error type will give a complete overview of the performance of the selected tools representing the state of the art error detection methods.

4.2 Performance prediction

In this section, the results given by the empirical study (Section 4.1) will be used for predicting performance on unseen new datasets, in order to answer the second research question: *Is it possible to create an extensive data profile to estimate performance on unseen datasets?*

Performance prediction aims to estimate the performance scores of an error detection strategy, on a specific dataset, without having to run the strategy beforehand or having the clean dataset at hand. The dataflow/workflow of this process is shown in Figure 4.2. When estimators are capable of estimating the performance scores, one could use this estimation to choose an error detection strategy. This contributes to the main research question of choosing a fitting error detection algorithm for a specific relational dataset.

To make predictions on the performance scores, information about the dataset is extracted beforehand, called a dataset profile. This profile contains high-level information that is of low cost computationally. This high-level information will most presumably give an idea on how to fix the errors inside the dataset, which could be input to predict the performance score a specific error detection strategy will get on that particular dataset. An estimator is then trained for each error detection strategy, with all the data profiles. The main notion is that error detection strategies assumably have comparable performance scores on similar datasets in terms of these high-level information profiles.

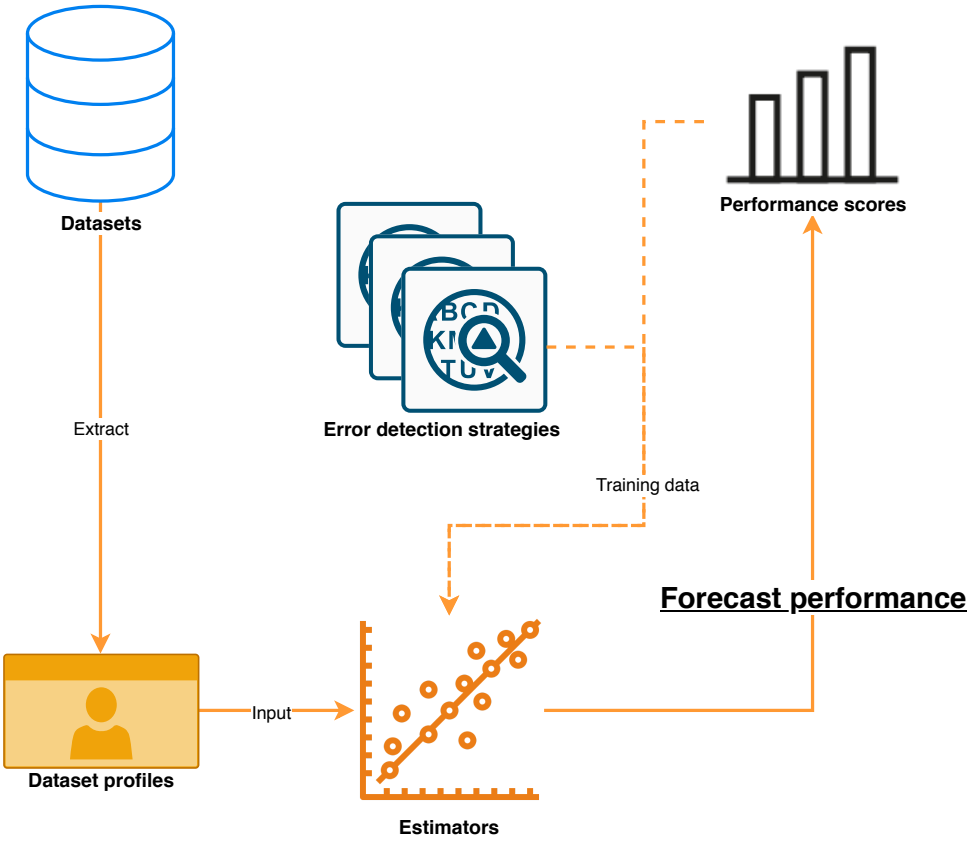


FIGURE 4.2: Performance scores estimation

Analogy One can think of this like going into a car garage with a broken car. The garage employee will take a quick look at the outside of the car, at the brand of the car and will ask you some questions about the problems you have. This small ***assessment*** could give an idea on what fixing strategy will most probably work best to fix the car. In his mind, the employee will have a list of strategies to fix a car (replace tyres, oil change, etc..) and will ***estimate the possible outcomes*** (or performance) of those strategies (speed of fixing, longevity, cost, etc..), based on the performance of these strategies on previous cars with a similar assessment. Although in the execution, the results might differ from what the employee had in mind. The estimation might not give the exact performance outcome.

This is the same with the prediction of the performance scores of error detection strategies, as depicted in Figure 4.2. An estimator, in this case a regression model, will look at a dataset and do the small assessment (***dataset profile***), like looking at the car. From this small assessment, the estimator will try to ***estimate the performance scores*** for each of the error detection strategies in mind based on previous datasets and the performance on those datasets, like the car garage employee guessed the outcome of replacing the tyres based on experience with previous cars. However, there will be a difference between the real outcome of that strategy (experimental results from Section 4.1) and the estimations of the performance scores.

This idea was proposed by Mahdavi and Abedjan, 2019. This paper proposes the idea of a 'dirtiness profile'. The idea is that error detection tools would have similar performance on similar datasets. Regression models would be trained and tested on characteristics of the dataset, also called 'dirtiness profile' (input) and F1-scores (output). The following section will be a modification of the research by Mahdavi and Abedjan, 2019. The goal will be to estimate performance and predict values as close to the real performance scores as possible, which can be measured with the mean squared error. The train and test in and outputs are retrieved from the empirical study (Section 4.1). One main difference is that this research will solely focus on automated prediction, where no rules or patterns from the ground truth are taken, and no user-defined configurations need to be created in order to perform the predictions. Besides, multiple new input features will be introduced to see if the performance can be increased. Lastly, instead of directly estimating the F1-score, as done by Mahdavi and Abedjan, 2019, estimators will be trained on both the recall and precision scores. The F1-score can be calculated using both estimated scores. A regression model is trained for every strategy (error tool and specific configuration, see Figure 4.3).

4.2.1 Data profiles

This section will discuss the methodology of creating a data profile for a dataset, to elaborate the first part of the second research question: *Is it possible to* ***create an extensive data profile*** *to estimate performance on unseen datasets?*

The data profiles are created using only top-level features, without having any previous knowledge about the datasets. In this research, the focus lies only on features that are extractable without knowing the ground truth, only the dirty dataset. The basic set of features used is a subset of the proposed features from Mahdavi and Abedjan, 2019. However, all features where some information about the ground truth was necessary (rules, patterns or other heuristics found in the clean dataset), were left out. The features are extracted from each column. Then, aggregates of these column-wise extracted features are aggregated using the mean, variance, min

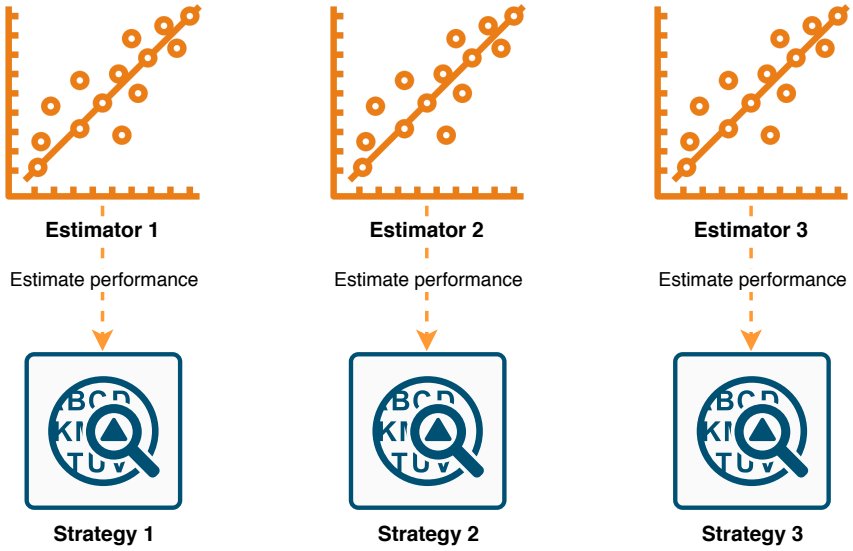


FIGURE 4.3: Each strategy will result in a separate estimator

and max to get a representation of feature distributions for each column in a dataset, with homogeneous output in size for training the regression models later on. These aggregated features representing the dataset profiles are the input values for the estimators.

Feature	Type	Description
characters_unique	characters	The number of unique characters
characters_alphabet	characters	The number of only alphabetical characters
characters_numeric	characters	The number of only numerical characters
characters_punctuation	characters	The number of punctuation characters
characters_miscellaneous	characters	The number of characters not included in the categories above
words_unique	words	The number of unique words
words_alphabet	words	The number of fully alphabetical words
words_numeric	words	The number of fully numerical words (decimals included)
words_punctuation	words	The number of characters used solely as punctuation
words_miscellaneous	words	The number of words not included in the categories above
words_length	words	The average word length
cells_unique	cells	The number of unique cells
cells_alphabet	cells	The number of completely alphabetical cells
cells_numeric	cells	The number of fully numerical cells (decimals included)
cells_punctuation	cells	The number of cells with only punctuation characters
cells_miscellaneous	cells	The number of cells not included in the categories above
cells_length	cells	The total length of cells combined
cells_null	cells	The number of empty cells

TABLE 4.1: All column-wise extracted features for the dataset profiles

4.2.2 Experimental setup

This section will describe the experiments to test the estimation capabilities of possible estimators, to answer the other part of the second research question: *Is it **possible** to create an extensive data profile **to estimate performance** on unseen datasets?*

To discover whether the discussed data profiles are useful in estimating the performance of error detection tools, the setup from Section 4.1 is updated. Figure 4.2 shows the flow of estimating the performance scores of the run error detection tools on the datasets. The goal of this workflow is to estimate the performance scores of error detection strategies on unseen datasets.

The estimators will be created as scikit-learn (Pedregosa et al., 2011) pipelines. Not only does this allow interchangeable regression models, but also configurable data preparation. The different configurable parts all have certain assumptions. A combination of all possible settings will be created and tested in order to find the best configuration possible.

To evaluate the estimators and find the best configuration for the estimators, an experiment has to be conducted to evaluate the prediction potential of these estimators. In reality, the estimators will be trained on all available training datasets and used to estimate performance scores on unseen datasets. However, with unseen datasets, only the dirty dataset is available, not the clean dataset. This means that it is not possible to evaluate the prediction capabilities of these estimators in a straight-forward way. **Leave-one-out cross-validation** is a method to simulate the behaviour of training on as many training datasets as possible, while still being able to evaluate the results afterwards, by leaving only one dataset out of that training set at the time. Table 4.2 shows such an example for leave-one-out training with 4 datasets available. In each iteration, a single dataset is left out of the training set, simulating the performance prediction for the left out dataset. This will result in a performance score prediction $\hat{y}(s_i, d_j)$ for a certain strategy s_i and dataset d_j.

d_1	d_2	d_3	d_4		**Output**
Estimate	Train	Train	Train	$\rightarrow$	$\hat{y}(s_i, d_1)$
Train	*Estimate*	Train	Train	$\rightarrow$	$\hat{y}(s_i, d_2)$
Test	Train	*Estimate*	Train	$\rightarrow$	$\hat{y}(s_i, d_3)$
Test	Train	Train	*Estimate*	$\rightarrow$	$\hat{y}(s_i, d_4)$

TABLE 4.2: Leave one out training example for a strategy s_i and datasets d_{1-4}

For each dataset d from D datasets and error detection strategy s from all strategies S, the leave-one-out evaluation as described above and in Table 4.2 will be executed. This will leave a $m \times n$ matrix of estimated performance scores as shown in Table 4.3a, representing a m strategies and the n available datasets. For comparison, the real scores that are obtained from the empirical study from Section 4.1 are shown in the same format in Table 4.3b.

TABLE 4.3: $m \times n$ (strategies $\times$ datasets) matrix of scores

	d_1	d_2	**...**	d_n
s_1	$\hat{y}(s_1, d_1)$	$\hat{y}(s_1, d_2)$	...	$\hat{y}(s_1, d_n)$
s_2	$\hat{y}(s_2, d_1)$	$\hat{y}(s_2, d_2)$	...	$\hat{y}(s_2, d_n)$
...	...	...	...	...
s_m	$\hat{y}(s_m, d_1)$	$\hat{y}(s_m, d_2)$	...	$\hat{y}(s_m, d_n)$

(A) Estimated performance scores

	d_1	d_2	**...**	d_n
s_1	$y(s_1, d_1)$	$y(s_1, d_2)$	...	$y(s_1, d_n)$
s_2	$y(s_2, d_1)$	$y(s_2, d_2)$	...	$y(s_2, d_n)$
...	...	...	...	...
s_m	$y(s_m, d_1)$	$y(s_m, d_2)$	...	$y(s_m, d_n)$

(B) Real performance scores

To assess the estimated performance scores as shown in Table 4.3a, the scores will be compared element-wise with the real performance scores shown in Table 4.3b. To do so, the difference between the estimation of the score and the score itself will be

calculated. This is the **estimation error**. The real score is denoted as y, which is a performance score result of the empirical study, for example precision, recall or f1-score. The estimated score is denoted as $\hat{y}$. The error for a single score, for a specific error detection strategy and dataset is the following:

$$e(s,d) = \hat{y}(s,d) - y(s,d) \tag{4.1}$$

To quantify the ability of an estimator pipeline to estimate the real performance scores of error detection strategies, the mean squared error for all the (strategy, dataset) tuples is taken.

$$MSE = \frac{1}{|D||S|} \sum_{d \in D} \sum_{s \in S} e(s,d)^2 \tag{4.2}$$

This mean squared error will give a method of comparison between different estimator configurations. For the best estimator configuration, the estimation error distribution will be shown to find how the estimator generally performs, for example by comparing the number of over- and under-estimations.

4.2.3 Estimator selection

Now that a score for comparing estimators has been covered, the range of estimator configurations will be discussed. The estimator pipeline consists of four possible parts:

1. Feature normalization
2. Feature selection
3. Principal component analysis
4. Regression model

Each part of the pipeline is configurable. The first three parts can be left out, leaving only a regression model with the normal dataset profile as input.

Feature normalization or standardization

The three chosen possibilities for feature normalization are:

- None: No normalization or scaling is applied.
- StandardScaler (`sklearn.preprocessing.StandardScaler`): Standardize features by removing the mean and scaling to unit variance. The scaler will be trained on the training sample so that after the training, each feature can be scaled using the distributions of the training set.
- Normalizer (`sklearn.preprocessing.Normalizer`): Normalize samples individually to unit norm. Each sample (i.e., each row of the data matrix) with at least one non zero component is rescaled independently of other samples so that its norm equals one. It is executed with a default l_2-norm

Feature selection

For feature selection, different trainable configurations are possible to optimize the number of input dimensions for the later parts of the pipeline. Research has shown that using too many input features for machine learning will decrease performance (Trunk, 1979). To circumvent the curse of dimensionality, feature selection methods can be put in the machine learning pipeline to learn which features to use as well.

- None: No dimensionality reduction is applied in the pipeline.
- Variance Threshold (`sklearn.feature_selection.VarianceThreshold`): This removes all features that have a variance below a certain threshold in the training dataset. The threshold has to be set to some value.
- Select From Model (`sklearn.feature_selection.SelectFromModel`): It selects the features based upon the importance weights of the trained regressor. On default, all features that have a weight above the mean of all feature weights are selected.
- Select K Best (`sklearn.feature_selection.SelectKBest`): This method selects the K best features according to a specified score. In this case, the `sklearn.feature_selection.f_regression` method is used, where the correlation between the target values and each separate (univariate) feature is calculated.

Besides feature selection methods that are used directly in the machine learning pipeline, feature selection can also be done in retrospect, which will be discussed in Subsection 4.4.1.

Principal component analysis

Principal component analysis is a dimensionality reduction method invented over a century ago (Pearson, 1901), but still has a strong use case in machine learning today. As said in the previous section, too many input features will lead to worse performance with the small number of input samples to train. A linear PCA will reduce the whole feature space to N number of components holding the most variance for all the input points. To transform and capture non-linear relations, also Kernel PCA will be used. This will be useful, as most regression models will not be capable of distinguishing between non-linear relations.

- None: No dimensionality reduction using PCA will be done.
- PCA (`sklearn.decomposition.PCA`): Linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower-dimensional space. Highly sensitive to scaling of the features (recommended to scale beforehand).
- Kernel PCA (`sklearn.decomposition.KernelPCA`): Allows for non-linear dimensionality reduction through using specified kernels. An RBF (radial basis function) kernel is commonly used for projecting non-linear relation onto a linear separable projection.

Regression model

As the final step in the pipeline, a regression model is present. It takes in the input features from the last step in the pipeline, depending on which normalization, feature selection and/or principal component analysis methods are chosen. The regression model will take the transformed data profile inputs from all the training datasets and tries to estimate the selected real scores. This score always will be between 0 and 1, inclusive. So at the end of each estimator, a limiter will be put into place not to have any impossible scores. The possible regression models are stated below. All models have certain assumptions that the workings build upon.

- Linear Regression (`sklearn.linear_model.LinearRegression`): This regression model fits a linear model that minimizes the sum of errors between the observed estimation and real observations. This method can be sensitive to outliers, due to its reliance on the square of errors.
- K-nearest Neighbors Regression (`sklearn.neighbors.KNeighborsRegressor`): The output is estimated by comparing it to the nearest neighbors in the training samples. The K nearest points are weighted equally and the training scores will be combined as an estimation.
- Ridge Regression (`sklearn.linear_model.Ridge`): This model solves a regression model where the loss function is the linear least squares function and regularization is given by the l_2-norm
- Bayesian Ridge Regression (`sklearn.linear_model.BayesianRidge`): This model can be used for estimating the regularization parameters as well, the regularization parameter is not set in a hard sense but tuned to the data at hand. It then estimates the ridge regression model.
- Decision Tree Regression (`sklearn.tree.DecisionTreeRegressor`): This is a regression model that outputs different discrete output levels based on an outcome of a decision tree, comparing individual features to certain learned values.
- Support Vector Regression (`sklearn.svm.SVR`): A support vector machine-based regression model. Like normal Support Vector Classification, the regression model depends only on a subset of the training data, making it more robust against outliers.
- Gradient Boosting Regression (`sklearn.ensemble.GradientBoostingRegressor`): Produces a predictive model from an ensemble of weak predictive models. In each stage, a regression tree is fit on the negative gradient of the least squares regression loss.
- AdaBoost Regression (`sklearn.ensemble.AdaBoostRegressor`): A meta-estimator that trains multiple regression models (default `DecisionTreeRegressor`) and consequently adjusts weights of input samples depending on the error of the previous estimator.
- Multi-layer Perceptron Regression (`sklearn.neural_network.MLPRegressor`): A neural network-based regression model. Capable of capturing non-linear relations, however, that would require multiple layers, which will, on its turn, require more training samples.

	d_1 (Train)	d_2 (Train)	d_3 (Test)
F1	1	0.2	?
Estimation	-	-	**0.6**

TABLE 4.4: Example baseline performance estimation for some example strategy s

4.2.4 Combination of estimators

Now that all the components for the estimator have been described, one can try to create ensemble estimators. In the study of Mahdavi and Abedjan, 2019, the F1-score was directly estimated. Knowing that this metric is directly built from both the precision and recall, it intuitively makes sense to first create estimators for the precision and recall (two separate estimator configurations), and use the output of these two configurations in combination to estimate the F1-score. Therefore, besides only calculating the MSE for an F1-score estimator, it will be compared to the MSE of the ensemble F1-score estimator.

4.2.5 Evaluation

To evaluate the experiment results as discussed above, the results will be both qualitatively and quantitatively analyzed to find an answer the research question: *Is it possible to create an extensive data profile to estimate performance on unseen datasets?*;

Qualitative First, the best estimator configuration will be chosen according to the lowest mean squared error of the estimations of real performance scores. This chosen estimator configuration will be qualitatively judged on the distribution of estimation errors. To deem the estimation of performance successful, the error distribution should be heavily centered around 0, indicating that most estimations are close to perfect. Also, the distribution should not be uniform (which will indicate random errors) and should not have clusters of outliers, which indicate underestimations or overestimations.

Quantitative Second, to add more substance to the evaluation, a baseline estimator method will be executed to see if the selected estimator has an improvement over that baseline method. The baseline estimator will be simply taking the average of the performance scores achieved on the training datasets. Only taking the average of the input performance scores is similar to regression without any weights to the input variables. An example is shown in Table 4.4 where the baseline estimators "trains" on d_1 and d_2. The estimation of a new dataset d_3 will be the average of the already known dataset performance scores, namely the average of 1 and 0.2, which is 0.6.
If the estimators improve over this baseline method in terms of mean squared error, median absolute error and mean absolute error, it shows that the estimator is capable of learning information from the dataset profiles and giving a useful estimation, better than the baseline.

If the assessments are both qualitatively and quantitatively positive, it can be deemed possible to create an extensive data profile to estimate performance on unseen datasets.

4.3 Ranking

Following the performance prediction as described in the previous section, this section will contain the methodology to answer research question 3: *Is it possible to generate a ranking of tools according to their performance on unseen datasets?*

An estimator for future error detection tool performance was introduced in the previous section. Now, the goal of generating an estimated ranking of tools with regards to their performance will be discussed. An absolute score is helpful to know with a single tool at your disposal, but a ranking or recommendation of tools and their strategies would be of greater use for the end-user. The question proposed here is to see if such a ranking could be made, ranking the better performing tools higher in the ranking. The flow of creating such a ranking is shown in Figure 4.4. For a new dataset as input, all the estimators will give an estimation of what the performance score on that new dataset will be. These estimations will then be ordered into a ranking, with the estimated highest performing strategies on top and worst estimated performing strategies on the bottom. A user could then make a substantiated choice for which error detection strategy to choose, based on expected performance.

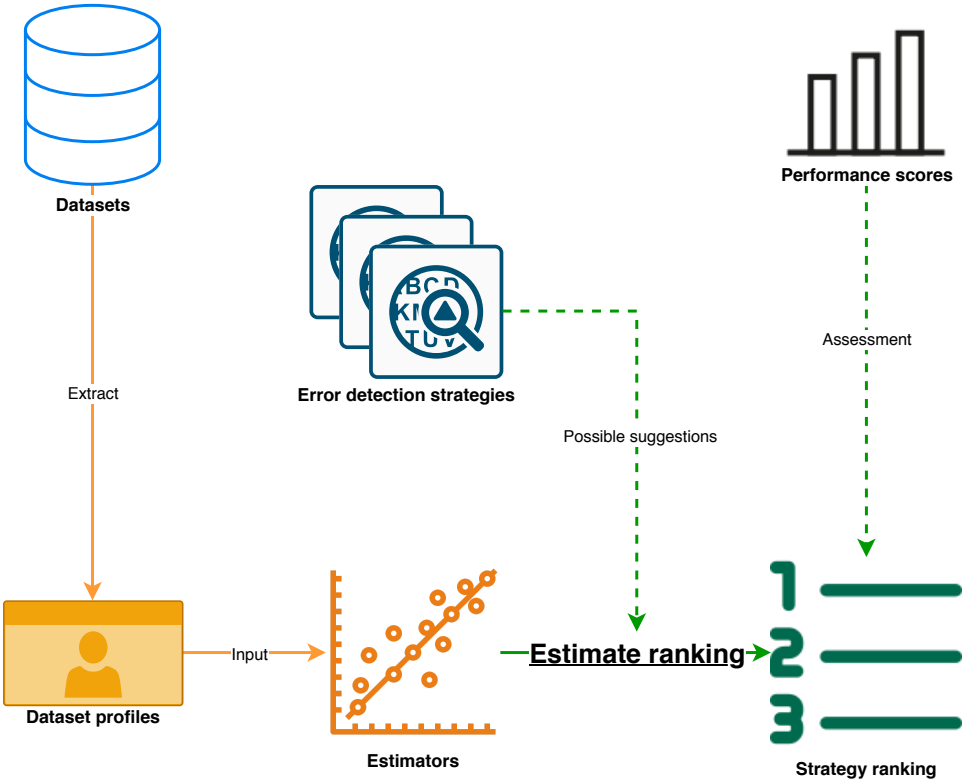


FIGURE 4.4: Flow of the estimation of strategy ranking

4.3.1 Method of ranking

A user should be able to retrieve an estimated ranking of high performing strategies without running these tools on an unseen dataset. A ranking should be K strategies long, with a predefined K. It should be so that a user can a variety of tools and configurations to pick from. To evaluate whether it is possible to create such an estimated ranking, two types of rankings will be made and scored. First, a ranking

based on finding the **best tool** for the dataset will be created. Then, a ranking based on finding the **best configuration for a tool** will be created. To test if estimated strategy ranking is viable, this section will focus on F1 estimation and ranking.
To generate an estimated ranking for a new unseen dataset, the data profile as described in Subsection 4.2.1 is created. Then, all estimators as found by Subsection 4.2.3 are given this data profile as input, generating estimated scores for all strategies $s \in S$. These estimated scores (like a single column in Table 4.3a) are then sorted. Depending on the task (best tool or best configuration), the results are filtered and only the top suggestions are kept.

Best tool First, the ranking of the best tool for a dataset will be discussed. This is the use case where a user does not know which tool he or she wants to use. The tools will be ranked according to the estimated performance score, with one output row per tool. An example of such a ranking is shown in Table 4.5. In this example, there are six error detection tools available, just like the six tools used in this research. In the "Tool" column, it is shown which tool is suggested at that ranking. Also, in this ranking, the estimated best configuration for that tool will be suggested in the "Configuration". Lastly, the predicted F1 score will be shown for comparison of predicted performances of the strategies. A user will be able to choose an error detection strategy directly from this list.

Rank	Tool	Configuration	Estimated F1
1	Tool 1	Config 1-a	$\hat{y}(s_{1-a}, d)$
2	Tool 2	Config 2-a	$\hat{y}(s_{2-a}, d)$
3	Tool 3	Config 3-a	$\hat{y}(s_{3-a}, d)$
4	Tool 4	Config 4-a	$\hat{y}(s_{4-a}, d)$
5	Tool 5	Config 5-a	$\hat{y}(s_{5-a}, d)$
6	Tool 6	Config 6-a	$\hat{y}(s_{6-a}, d)$

TABLE 4.5: Strategy ranking for best tool

Best configuration per tool Another ranking that could be made is a configuration ranking per tool, for a specific dataset. An example of this ranking is shown in Table 4.6. It might happen that a user has decided to work with a specific tool. However, the user is not aware of the best configuration for that tool. So, for one single tool, K different configurations will be outputted in a ranking. The format of the ranking will be the same as the "best tool" ranking, but only 1 tool type will be shown. As shown in the Table 4.6, in column "Tool", only 1 tool is selected. Then, in the column "Configuration", the different configurations will be ordered from best to worst. Again, the predicted performance score will be displayed in the last column in descending order.

4.3.2 Performance measure of ranking

Quantitative metrics will be calculated to compare and analyze the results of these rankings. Average Precision is a metric that takes into account the ranking of the returned documents. Unfortunately, this metric uses a binary relevance scoring, where the last relevant document is equally important to rank high as the first relevant document. In this research, a cutoff point for a score should then be chosen to evaluate, for example, any strategy with a score > 0.5 would be considered relevant. Taking

Rank	Tool	Configuration	Estimated F1
1	Tool 1	Config 1-a	$\hat{y}(s_{1-a}, d)$
2	Tool 1	Config 1-b	$\hat{y}(s_{1-b}, d)$
3	Tool 1	Config 1-c	$\hat{y}(s_{1-c}, d)$
4	Tool 1	Config 1-d	$\hat{y}(s_{1-d}, d)$
5	Tool 1	Config 1-e	$\hat{y}(s_{1-e}, d)$
6	Tool 1	Config 1-f	$\hat{y}(s_{1-f}, d)$
7	Tool 1	Config 1-g	$\hat{y}(s_{1-g}, d)$
8	Tool 1	Config 1-h	$\hat{y}(s_{1-h}, d)$
9	Tool 1	Config 1-i	$\hat{y}(s_{1-i}, d)$
10	Tool 1	Config 1-j	$\hat{y}(s_{1-j}, d)$

TABLE 4.6: Strategy ranking for best configuration per tool

the mean average precision for all different unseen datasets and the produced rankings would then have some meaning, but does not fit this purpose best.

A metric that does take the relative importance into account is the discounted cumulative gain (DCG). It allows degrees of relevance, which is suited for our purpose. The top ranks count the most, meaning whenever a highly relevant (high scoring strategy) is actually listed at the top, the DCG reflects that in a positive way. The utility also decreases with the rank, meaning that lower ranks count less towards the DCG. The DCG is a summation of the relevance score discounted by a log value relative to the given place in the ranking (shown in Equation 4.3).
Also, a normalized version of the discounted cumulative gain exists. It takes the ideal ranking and calculates the DCG ($IDCG_k$). The output metric will be the DCG_k for the produced ranking, divided by the DCG for the best ranking (see Equation 4.4).

$$DCG_k = \sum_{r=1}^{k} \frac{rel_r}{\log(r+1)} \tag{4.3}$$

$$NDCG_k = \frac{DCG_k}{IDCG_k} \tag{4.4}$$

The relevance of a returned item in a ranking must be determined beforehand. It should have a relation to how good a strategy performs. That is why, for this research, the actual real score (precision, recall or F1) of a strategy run on a specific dataset will be the relevance. This allows this for automatic relevance scoring. As shown in Figure 4.4 and as described in the previous sections, it is possible to assess the produced strategy ranking by using real performance scores from the empirical study as relevance.

Best tool ranking scoring The best tool ranking will be scored in two ways. When a ranking like displayed in Table 4.5 is given, scoring can be done tool-wise, but also strategy-wise. The first way is to get the highest relevance for all the configurations of that tool and using that value as rel_r. This will judge a ranking solely on the order of tools given. The second way of ranking will be taking the relevance of the suggested strategy for scoring. This is a more difficult task for the recommendation system, as not only does the tool suggestion need to be correct, but the only configuration for that tool should also be the best configuration.

4.3.3 Evaluation

Again, similarly to Section 4.2, the experiments as proposed above will be assessed qualitatively and quantitatively to answer the third research question: *Is it possible to generate a ranking of tools according to their performance on unseen datasets?*;

In total, three types of experiments will be conducted.

- Best tool ranking - Scored on tool + configuration relevance
- Best tool ranking - Scored on tool relevance
- Best configuration ranking - Scored on configuration relevance

The estimated rankings will be produced using leave-one-out cross-validation, like in Section 4.2.

For the first two types of experiments, the same rankings are used. These rankings will consist of 6 entries ($K = 6$), namely for the six possible tools to be suggested. For the last type of experiments, the rankings will consist of 10 suggestions[3] ($K = 10$) of configurations for a tool. Therefore, only tools with 10+ configurations will be tested in this experiment (dBoost & Raha), as this will test the capability of the ranking system to produce the right order of ranking, but also cutting off the majority of configurations.

For each dataset, the ranking will be scored with NDCG, using the best possible ranking from the ground truth as a reference ideal DCG score. This will result in a NDCG for all datasets available. These scores will the main subject of analysis for evaluation.

Qualitative The NDCG values per dataset for each experiment will be sorted and plotted for comparison. The focus of the qualitative analysis will be that of the ordering and general relative results between datasets in one experiment. Preferably, NDCG values per dataset are similarly high, without many lower outliers. If there is a cluster of lower NDCG values, the ranking will be deemed meaningless.

Quantitative Similar to Section 4.2, the results of the experiments will be compared to a baseline method. This baseline method estimates performance as described in Subsection 4.2.5. From these baseline estimations, rankings of the same structure as in the experiments will be created and scored with NDCG. While the focus lies on the F1-score for this section, a comparison of mean NDCG for all datasets will be compared for precision-based ranking, recall-based ranking and F1-based ranking, both directly estimated and combinedly estimated (recall + precision estimation to calculate the estimated F1).

The main comparison to answer if it is possible to generate a ranking of tools according to their performance on unseen datasets, will be to see if there is an increase in the mean NDCG for all experiments from the proposed estimator compared to the baseline, described in the paragraph above. If there is no increase in NDCG, the estimator will be deemed not suitable for estimated strategy ranking.

[3]For this research, a K of 10 will be used. 10 is the number of search suggestions in the basic Google search bar (`https://www.google.com/`).

4.4 Interpretability of tools

Finally, this section will cover research question 4: *Do these data profiles provide more interpretability of error detection tools?*

Based on Section 4.2 and Section 4.3, with a set of input features and output values in the two tasks, the regression models can be analyzed to "explain" the inner workings of tools. Using feature importance identification methods depending on the specific regressor or general importance methods, the "weight" of a certain feature of the data profile can be translated to the performance of a certain tool with a specific configuration. The importance of features for each strategy can than be analyzed, to see if they correspond with the underlying techniques in these methods. If there are matches and mismatches, try to find why the misalignment occurs and see if improvements can be suggested for the error detection tools. A visual representation of making the error detection models interpretable is shown in Figure 4.5.

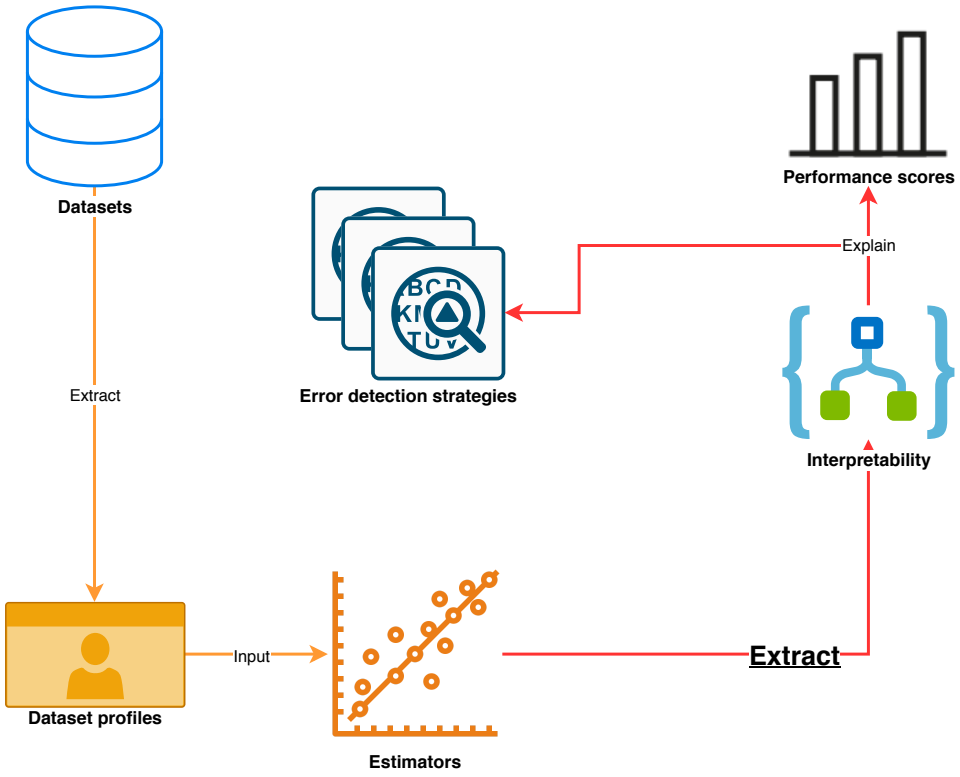


FIGURE 4.5: Adding interpretability into the flow of error detection

4.4.1 Feature selection

Besides the learnable feature selection that will be used in the performance estimate pipeline of Section 4.2, feature selection can also be done at the end of the estimation loop. For this part, permutation importance will be used to determine which features contribute to the estimation results. All features that actually seem to have a positive impact on the model will be kept for the dataset profile with fewer features. With this reduced number of features, the performance prediction of Section 4.2 will be repeated to see if there is an improvement in estimation.

4.4.2 Feature importance

Finding which data profile features are important for the estimator model to estimate its performance gives insights into how a certain error detection strategy behaves.

To answer whether the dataset profile features could provide more insight on when and how the selected error detection tools perform correctly, the direct F1-score, recall and precision estimators from Section 4.2 will be analyzed using SHAP values. For each tool, the best performing configuration (highest mean F1-score) will be analyzed. First, the features with the highest impact on the estimator output will be inspected. Then, the correlation between the most impacting features and the SHAP contribution of that feature will be calculated to find if higher or lower feature values have a positive or negative impact on the model outputs.

4.4.3 Evaluation

In this section, an analysis will be proposed for both the feature selection and feature importance part of the previous sections to evaluate the fourth research question: *Do these data profiles provide more interpretability of error detection tools?*;

The feature selection results produced from repeating the experiments from Section 4.2 will be compared on the mean squared error and mean absolute error to check for improvements. These comparisons will be made for performance metrics (precision, recall and direct F1 estimation), as well as for the combined F1 estimation. If there is a clear improvement in mean squared error and mean absolute error, it will show that automated interpretability can be beneficial for the estimator models.

The main focus of the evaluation will lie on the produced feature importance, explained by the second experiment in the section above. Finding the most impactful features and their correlation with the performance scores could be useful for interpretability if for every tool:

- The top features have a substantial impact on the estimator model output
- The correlations are translatable to theories or logic about the impact of features on the model output

If both requirements can be met, the dataset profiles will add more interpretability of when, how and if the error detection tools will perform adequately.

Chapter 5

Experimental setup

5.1 Selected datasets

A summary of the list of datasets used for the empirical study, performance estimation and ranking of error detection strategies can be found below (Table 5.1). The datasets were taken from published data from the following papers:

- ED2 by Neutatz, Mahdavi, and Abedjan, 2019
- Raha by Mahdavi et al., 2019
- REDS by Mahdavi and Abedjan, 2019
- CleanML by Li et al., 2019

Table 5.1 shows each of the datasets and the common error types that occur in these datasets. Statistics about the used datasets can be found in Table 5.2. Below, a general description of all datasets will be presented and the source of the dataset will be mentioned.

Airbnb This dataset contains 42,492 records on hotels in the top 10 tourist destinations and major US metropolitan areas scraped from Airbnb.com. Each record has 40 attributes, including the number of bedrooms, price, location, etc. Demographic and economic attributes were scraped from city-data.com. The classification task is to determine whether the rating of each hotel is 5 or not.
(Source: CleanML)

Beers This dataset contains a list of different beers. The attributes are numeric ID's, foreign keys, names, City, state abbreviation, amount of ounces & percentage of alcohol.
(Source: ED2, REDS, Raha)

EEG This is a dataset of 14,980 EEG recordings. Each record has 14 EEG attributes. One of the attributes is a classification task is to predict whether the eye-state is closed or open. In the dirty version of this dataset, some of these predictions are wrong.
(Source: CleanML)

Flights This dataset contains a flight schedule. Attributes are id, source of data, flight number alphanumeric with dashes, scheduled and actual departure and arrival times in XX:XX a.m. format.
(Source: ED2, REDS, Raha)

Dataset name	Pattern violations	Rule violations	Outliers	Duplicates	Other error types
Airbnb		✓	✓	✓	
Beers	✓	✓			
EEG			✓		✓
Flights	✓	✓			
Hospital	✓				
Marketing		✓			✓
Movie		✓		✓	
Movies	✓	✓			✓
Rayyan	✓	✓			
Restaurant	✓	✓			
Restaurants	✓				
Toy		✓			✓
University		✓			
Uscensus		✓			✓

TABLE 5.1: Datasets used and the common error types present in these datasets

Hospital This source contains a list of hospital experiments. Attributes include ID, hospital names, addresses, measurement types, measurement named, experiment scores and other statistics about the hospital and experiments.
(Source: REDS, Raha)

Marketing This dataset consists of 8,993 records about household income from a survey. Each record has 14 demographic attributes, including sex, education, etc. Another attribute is a classification task is to predict if the annual household income is less than $25,000.
(Source: CleanML)

Movies This test set is a collection of popular movies. Attributes of this source are: Id, Name, Year, Release Date, Director, Creator, Actors, Cast, Language, Country, Duration, RatingValue, RatingCount, ReviewCount, Genre, Filming Locations, Description
(Source: ED2, REDS, Raha)

Movie Same as movies, but less and different columns
(Source: CleanML)

Rayyan This dataset contains a set of published articles. The attributes include the title, which journal it was published in, the date, pagination and other information important for citing the articles.
(Source: REDS, Raha)

Restaurants This source is a list of restaurants. Attributes contain the name, address, phone number, website and other attributes that are commonly found on restaurant review websites (like rating value and price range).
(Source: ED2)

Dataset	Data quality	Rows	Columns	Number of errors	Total cells
airbnb	85.4 %	18406	40	107625	736240
beers	83.5 %	2410	11	4362	26510
eeg	97.6 %	14980	15	5496	224700
flights	70.4 %	2376	7	4920	16632
hospital	97.5 %	1000	20	509	20000
marketing	78.7 %	8993	14	26815	125902
movie	63.3 %	6531	8	19174	52248
movies	98.8 %	7390	17	1501	125630
rayyan	91.4 %	1000	11	948	11000
restaurant	99.5 %	12007	11	635	132077
restaurants	99.9 %	28787	16	565	460592
toy	77.8 %	6	3	4	18
university	97.4 %	210	18	100	3780
uscensus	99.1 %	32561	15	4262	488415

TABLE 5.2: Dataset statistics

Restaurant This dataset contains similar information as "Restaurants", but has fewer columns.
(Source: CleanML)

Toy Dummy set for testing purposes & testing for generalizability
(Source: REDS, Raha)

University This dataset contains 286 records about universities. Each record has 17 attributes, including state, university name, SAT scores, etc. It contains a column with a classification task is to predict whether the expenses are greater than 7,000 for each university. This dataset contains inconsistent representations for states and locations.
(Source: CleanML)

Uscensus This dataset contains 32,561 US Census records for adults. Each record has 14 attributes, including age, education, sex, etc. It also contains an attribute with a classification goal of predicting whether the adult earns more than $50,000.
(Source: CleanML)

5.2 Metrics

To measure the performance of tools on test datasets, metrics of two types can be used. Cell-based and row-based metrics. The difference lies in what is taken as an entity while calculating scores.

Cell-based metrics

Cell-based metrics identify each cell in a dataset as a separate entity in the test set. Each cell will be counted as one positive or negative error value.

Row-based metrics

Row-based metrics identify each row in a dataset as a separate entity in the test set. A row could contain multiple errors, but only the whole row is counted as one positive or negative error value.

Scores

In the case of error detection, instances are either erroneous (positive) or clean (negative). Because the ground truth of the datasets in Subsection 4.1.3 is known, the performance metrics can be calculated directly after the execution of a tool. The following performance scores will be kept after experiments and estimated in the later stages of the research:

- Precision $(= \frac{\text{true positives}}{\text{true positives}+\text{false positives}})$
- Recall $(= \frac{\text{true positives}}{\text{true positives}+\text{false negatives}})$
- F1-score $(= 2 \cdot \frac{\text{precision}\cdot\text{recall}}{\text{precision}+\text{recall}})$

Examples

To provide visual aid for the different metrics proposed, two examples of error detection outputs are shown in Figure 5.1. In the figure, the real errors (difference between dirty dataset and the ground truth) are labeled cell-wise by underling and displaying in bold. The "detected" errors by the fictional error detection strategies are shown with a red background. In contrast to the cell-wise detection, the erroneous rows are marked by the black arrows to the left. The "detected" faulty rows are marked by the red arrows on the right.

	Order Id	Product Name	Price	Time Sold	Sold By	
	1	Croissant	2.00	10.56	James	
	2	Croissant	2.00	11.57	James	
	3	Baguette	3.00	13.42	Robert	
→	**a**	Macarons	1.50	13.42	Robert	←
→	4	Baguette	**30.00**	**32.10**	**Charle**	←
→	5	**Krosant**	2.00	23.55	Charles	←

(A) A perfect error detection output example

	Order Id	Product Name	Price	Time Sold	Sold By	
	1	Croissant	2.00	10.56	James	←
	2	Croissant	2.00	11.57	James	←
	3	Baguette	3.00	13.42	Robert	
→	**a**	Macarons	1.50	13.42	Robert	
→	4	Baguette	**30.00**	**32.10**	**Charle**	←
→	5	**Krosant**	2.00	23.55	Charles	

(B) A worse error detection output example

FIGURE 5.1: Two examples of error detection outputs. Real errors are underlined and bold. Example outputs by error detection methods are marked in red.

Cell-wise scores in the examples are as follows.
For Figure 5.1a, there is a total of 5 real error cells. The detection algorithm marks all 5 errors correctly. So $precision = \frac{5}{5} = 1$, $recall = \frac{5}{5} = 1$ and $F1 = 2 \times \frac{1 \times 1}{1+1} = 1$.
For Figure 5.1b, there are also the same 5 real error cells. The detection algorithm marks only 1 error correctly and 3 more incorrectly. So $precision = \frac{1}{4} = 0.25$, $recall = \frac{1}{5} = 0.2$ and $F1 = 2 \times \frac{0.25 \times 0.2}{0.25+0.2} = 0.22$.

Row-wise scores for the two examples are different from the cell-wise performance scores.
For Figure 5.1a, there is a total of 3 real error rows. The detection algorithm marks all 3 rows correctly. So $precision = \frac{3}{3} = 1$, $recall = \frac{3}{3} = 1$ and $F1 = 2 \times \frac{1 \times 1}{1+1} = 1$.
For Figure 5.1b, there are also the same 3 real error rows. The detection algorithm marks only 1 row correctly and 2 more incorrectly. So $precision = \frac{1}{3} = 0.33$, $recall = \frac{1}{3} = 0.2$ and $F1 = 2 \times \frac{0.33 \times 0.33}{0.33+0.33} = 0.33$.

	Precision	**Recall**	**F1**
(A)	1 / 1	1 / 1	1 / 1
(B)	0.25 / 0.33	0.2 / 0.33	0.22 / 0.33

TABLE 5.3: Cell-wise / Row-wise performance score differences for the two examples in Figure 5.1

For this research, **cell-wise scores** will be used to measure the performance of error detection strategies. The goal of error detection in this thesis is to find all different types of errors holistically. Unlike, for example, duplicate rows, most other errors are defined only for single cells. To also include duplicate rows in the scoring, the complete row is marked as erroneous. The error detection algorithms will be scored accordingly.

5.3 Error detection tool configurations

After generating configurations with the settings shown in Section 4.1, the number of strategies per tool is shown in Table 5.4. However, not all strategies will be able to produce results within the timeouts.

Tool	Number of strategies
dBoost	72
Raha	37
ForbiddenItemSets	7
ActiveClean	7
KATARA	4
FAHES	4

TABLE 5.4: Number of configurations per tool

5.4 Computer specification

Experiments were run on a computer with an Intel Core i7 8750H processor using 16 GB of RAM, running Ubuntu 18.04 as Windows Subsystem for Linux 2. The maximum runtime for a single experiment was limited at 1800 seconds (half-hour). A timeout error would occur after that, and the experiment would be canceled.

Chapter 6

Results

This chapter will go through the main results of the experiments and methods as described in Chapter 4.

6.1 Empirical study

This section will be dedicated to the results of the first research question: *What is the current state of the art and what is the performance of these tools?*.
The tools that have been selected to represent the state of the art have been discussed in both Chapter 2 and Chapter 4. The results will now dive into the second part of the research question, as the performance results from the empirical study executed will be presented below.

6.1.1 Overall results

An aggregate of all the performance scores is shown in Table 6.1. All human-aided tools with a maximum of 20 (perfectly accurate) human interactions and all the other non-human interactive tools are used in this aggregate. Then, the performance scores with the highest F1-score for all configurations of a tool are kept per dataset. Precision, recall and the F1-score are shown from left to right per column. The best score for each dataset is marked bold.

TABLE 6.1: |Precision Recall F1| for tool as column & dataset as row (best scores in bold)

	ActiveClean			FAHES			ForbiddenItemSets			KATARA			Raha			dBoost		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
airbnb	0.15	**1.00**	0.26	**0.54**	0.01	0.02	0.13	0.29	0.18	Other error			0.42	0.13	0.20	0.23	0.38	**0.28**
beers	0.16	**1.00**	0.28	0.83	0.02	0.04	0.34	0.30	0.32	0.14	0.26	0.18	**0.97**	0.69	**0.80**	0.68	0.55	0.61
eeg	0.04	0.03	0.03	0.00	0.00	0.00	0.02	0.26	0.04	0.00	0.00	0.00	**0.56**	0.71	**0.63**	0.13	**1.00**	0.23
flights	0.30	**0.98**	0.46	0.23	0.01	0.02	0.56	0.16	0.24	0.09	0.09	0.09	0.90	0.83	**0.86**	**0.94**	0.59	0.72
hospital	0.03	0.47	0.05	0.02	0.09	0.04	0.01	0.06	0.02	0.08	0.37	0.13	**0.98**	**0.57**	**0.72**	0.03	0.43	0.06
marketing	0.25	0.36	0.30	0.24	0.01	0.01	0.25	0.46	0.33	0.21	0.32	0.25	**0.50**	0.32	0.39	0.34	**0.67**	**0.45**
movie	0.37	**1.00**	0.54	0.00	0.00	0.00	0.31	0.08	0.13	0.43	0.43	0.43	**0.47**	0.64	**0.54**	0.37	**1.00**	0.54
movies	0.02	0.00	0.01	0.01	0.10	0.02	0.01	0.06	0.01	0.02	0.16	0.03	**0.72**	**0.76**	**0.74**	0.01	0.09	0.03
rayyan	0.09	**1.00**	0.16	0.07	0.04	0.05	Other error			0.01	0.02	0.01	**0.86**	0.84	**0.85**	0.22	0.77	0.34
restaurant	0.01	**0.83**	0.02	0.00	0.00	0.00	0.01	0.07	0.01	0.00	0.13	0.01	**0.14**	0.11	**0.12**	0.03	0.03	0.03
restaurants	0.00	0.00	0.00	**0.00**	0.07	**0.01**	Other error			0.00	0.22	0.00	0.00	**1.00**	0.00	0.00	0.08	0.00
toy	Other error			0.00	0.00	0.00	0.00	0.00	0.00	0.21	0.75	0.33	0.22	**1.00**	0.36	**0.33**	0.75	**0.50**
university	0.03	0.09	0.04	0.00	0.00	0.00	Other error			0.06	0.29	0.10	**0.99**	0.91	**0.95**	0.32	**1.00**	0.49
uscensus	0.02	0.00	0.00	0.01	0.18	0.02	0.02	0.26	0.04	0.00	0.00	0.00	**1.00**	**1.00**	**1.00**	0.41	**1.00**	0.58

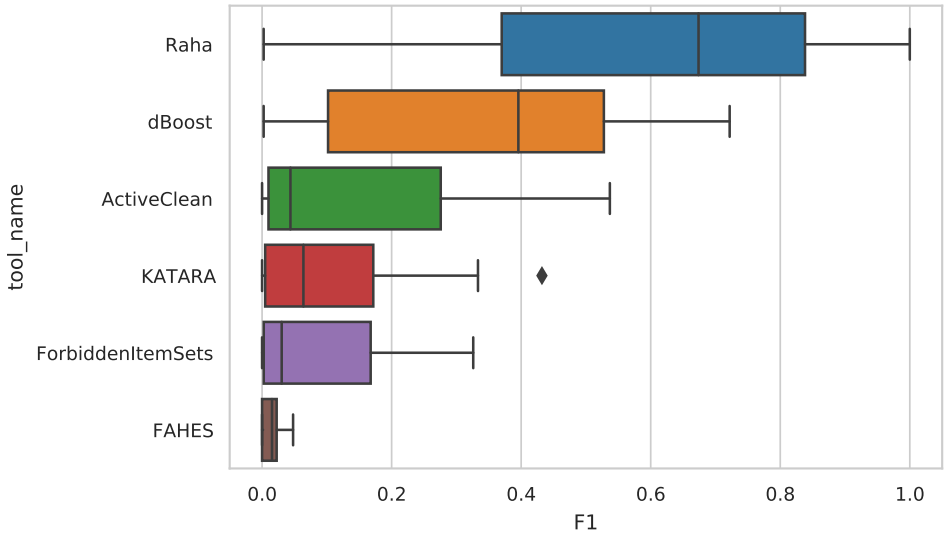


FIGURE 6.1: F1 boxplots for the best tool results

The results above are also shown as boxplots in Figure 6.1 to show a direct overview of the best performing configurations per tool, measured by the F1 score. From the table and figures shown, the conclusion can be drawn that Raha performs best overall for all the tools. Because Raha uses different underlying error detection techniques, it can detect a wide range of errors. In this setup were the error detection task was to detect not only specific errors, but all sorts, the more general methods perform better.
From the non-interactive tools, dBoost performs the best. This might mean that in the majority of datasets used, errors were detectable using rule violations, pattern violations or outliers. Besides Raha, the other human interactive tool ActiveClean is the next best tool on average. However, this tool mostly scores best when marking all the values as errors, when the datasets have relatively more errors. Example datasets with low data quality (high percentage of errors) include the datasets movie or flights (see statistics in Table 5.2). The last tools, FAHES, ForbiddenItemSets and KATARA perform worst overall. FAHES is a relatively specific tool, as it mainly tries to find disguised missing values. It could maybe fit in the data cleaning pipeline somewhere where only the missing values need to be detected. ForbiddenItemSets scores a decent average F1-score, as shown in Figure 6.1, but does not cut it against the top 3. It does have higher precision than FAHES and KATARA overall, making it maybe possible to use in an automated fashion, or as a submodule in a composite method like Raha. Lastly, KATARA also performs below average. This is due to the fact that KATARA is highly dependent on the domain the dataset has and the domain given to KATARA as a knowledge base. Apparently, the combination of knowledge base parts was not suited well enough for the test datasets.

Execution times Beside the performance scores in Figure 6.1 and Table 6.1, the execution times in seconds of the best configuration are visualized in Figure 6.2. Raha does have the best performance scores overall, but also takes the most time to execute. Raha is a composite method and uses the result of multiple simpler error detection methods. Consequently, the runtime also depends significantly on the underlying methods. So the sum of all the underlying error detection strategies makes it so that it is also the most time-consuming tool. The second best tool, dBoost, has a

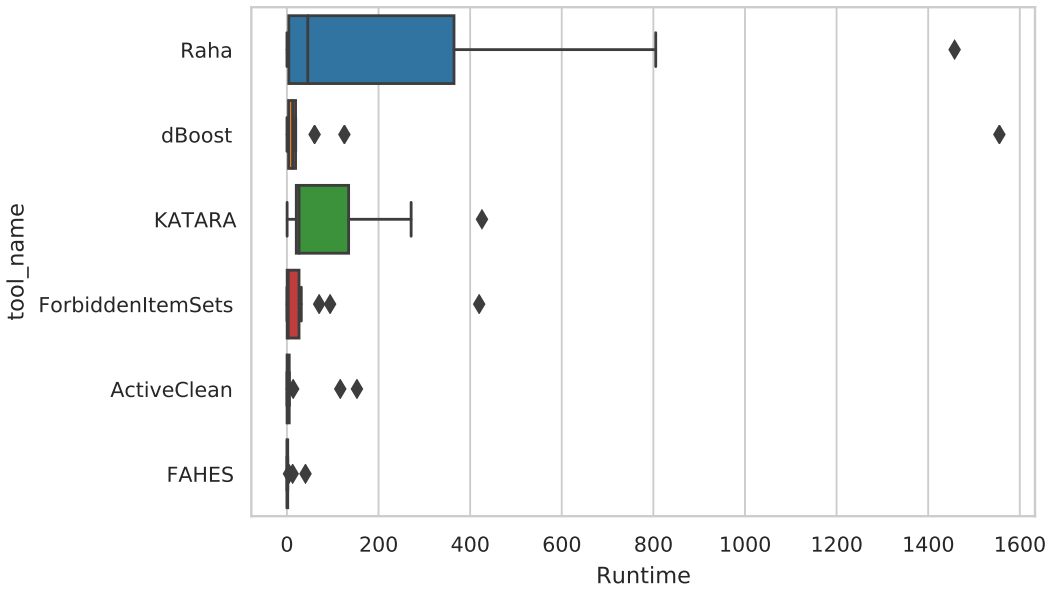


FIGURE 6.2: Boxplot of the runtime in seconds for the best tool results

lesser median runtime than most tools, but has some outliers for larger datasets. In general, the runtime of these error detection tools will grow more than linearly with increasing input sizes.

6.1.2 Human accuracy influence

If a user is willing and able to actively label cells during error detection, there still remains the question of how accurate those labels are. To show the implications of lesser accurate labeling, all the configuration of Raha were executed again, but the human interactions were simulated to be only 75% accurate, instead of 100%. The results are shown in Table 6.2, with the fully accurate labeling on the left side, and the 75% accuracy on the rights side. Immediately can be seen that tool is highly dependent on human accuracy. Only for 1 out of the 14 datasets, the tool randomly performed better with lower labeling accuracy according to the F1-score, then it did with full accuracy, due to the randomness in Raha. Nevertheless, precision-wise, for none of the datasets, the less accurate experiments produced better results than the fully accurate strategies. A user of interactive error detection tools should take this into account when deciding for an error detection tool. Solutions like repeated labeling by users could maybe mitigate the risk (as examples have shown by Sheng, Provost, and Ipeirotis, 2008).

6.1.3 Results by error type

In the following subsections, a more detailed review of results for the empirical study will be given. Where the previous sections would discuss overall performance, this section separates each error type for analysis, covering only overlapping tools that supposedly can detect such error types and datasets that contain these error types.

TABLE 6.2: |Precision Recall F1| for Raha with different human labeling accuracy (best scores in bold)

	Raha 100%			Raha 75%		
	P	**R**	**F1**	**P**	**R**	**F1**
airbnb	**0.42**	0.13	0.20	0.19	**0.36**	**0.25**
beers	**0.97**	0.69	**0.80**	0.70	**0.81**	0.75
eeg	**0.56**	**0.71**	**0.63**	0.10	0.44	0.16
flights	**0.90**	**0.83**	**0.86**	0.60	0.73	0.66
hospital	**0.98**	**0.57**	**0.72**	0.09	0.54	0.15
marketing	**0.50**	0.32	**0.39**	0.30	**0.47**	0.37
movie	**0.47**	0.64	**0.54**	0.41	**0.71**	0.52
movies	**0.72**	**0.76**	**0.74**	0.11	0.35	0.17
rayyan	**0.86**	0.84	**0.85**	0.71	**0.84**	0.77
restaurant	**0.14**	0.11	**0.12**	0.01	**0.58**	0.03
restaurants	**0.00**	**1.00**	**0.00**	0.00	0.00	0.00
toy	**0.22**	**1.00**	**0.36**	Other error		
university	**0.99**	**0.91**	**0.95**	0.25	0.78	0.38
uscensus	**1.00**	**1.00**	**1.00**	0.05	0.64	0.09

Rule violations

To start off, rule violations will be discussed. All the selected error detection tools have capabilities of detecting rule violations of some sort. Rule violations are also the most common error type occurring in datasets. All datasets except `EEG`, `Hospital` and `Restaurants` contain errors detectable by rule violations. Because all tools and almost all datasets contain these error types, the general findings from Subsection 6.1.1 apply to this specific error type.

Pattern violations

The two tools capable of detecting pattern violations are Raha and dBoost. Both error detection tools are capable of detecting multiple error types, but it is clear that Raha outperforms dBoost. In most datasets that have pattern violation errors, will also contain rule violations. dBoost has limited multicolumn error detection (with the Gaussian mixed models), compared to Raha, which could lead to the lower results. However, for highly structured datasets like `Flights`, that contained information like time and flight number, dBoost is capable of finding outliers in the patterns from these columns. Because Raha has underlying pattern detection tools it learns from (dBoost implementation) and the aid of a human expert, it is more capable of capturing all errors holistically, including pattern violations.

TABLE 6.3: Pattern violations - |Precision Recall F1| for tool as column & dataset as row (best scores in bold)

	Raha			dBoost		
	P	**R**	**F1**	**P**	**R**	**F1**
beers	**0.97**	**0.69**	**0.80**	0.68	0.55	0.61
flights	0.90	**0.83**	**0.86**	**0.94**	0.59	0.72
hospital	**0.98**	**0.57**	**0.72**	0.03	0.43	0.06
movies	**0.72**	**0.76**	**0.74**	0.01	0.09	0.03
rayyan	**0.86**	**0.84**	**0.85**	0.22	0.77	0.34
restaurant	**0.14**	**0.11**	**0.12**	0.03	0.03	0.03
restaurants	0.00	**1.00**	**0.00**	**0.00**	0.08	0.00

One of the surprising results is those of the `Restaurants` dataset. This dataset has a data quality of 99.9%, meaning that less than 0.1% of the data cells are errors. Rounded, this will result in a precision of 0.00 for Raha, but with a recall of 1.00. This means that the best error detection job it could do, was simply marking all the values as erroneous, leading to this rather uncommon precision and recall tuple of 0.00 and 1.00.

Outliers

The errors that can be detected as outliers are detectable by again Raha and dBoost. The only two datasets containing classical outliers are `Airbnb` and `EEG`. In terms of F1-score, dBoost outperforms for `Airbnb`. For dBoost, there are configurations that achieve higher precision, at the cost of lower recall. This means that for detecting outliers, if dBoost is configured with more conservative thresholds for outlier detection, it could be of great value in an automated error detection pipeline.

TABLE 6.4: Outliers - |Precision Recall F1| for tool as column & dataset as row (best scores in bold)

	Raha			dBoost		
	P	**R**	**F1**	**P**	**R**	**F1**
airbnb	**0.42**	0.13	0.20	0.23	**0.38**	**0.28**
eeg	**0.56**	0.71	**0.63**	0.13	**1.00**	0.23

Duplicates

The two tools capturing complete information about a row are ActiveClean and Raha, making it possible for these two tools to detect duplicates that are errors. `Airbnb` and `Movie` are the datasets containing duplicates. ActiveClean has a higher F1-score for `Airbnb`, but always lower precision. The precision of ActiveClean is equal to the portion of errors in those datasets, showing that ActiveClean simply labels all values as erroneous, which is not a valuable error detection result.

TABLE 6.5: Duplicates - |Precision Recall F1| for tool as column & dataset as row (best scores in bold)

	ActiveClean			Raha		
	P	**R**	**F1**	**P**	**R**	**F1**
airbnb	0.15	**1.00**	**0.26**	**0.42**	0.13	0.20
movie	0.37	**1.00**	0.54	**0.47**	0.64	**0.54**

6.1.4 Evaluation

The results from the empirical study have one clear outcome. Raha has the highest performance scores, at the cost of human expertise while detecting errors and the highest execution times. Raha outperforms or scores equally for grouped experiments based on error types. This could be because of the following:

- Raha is capable to "capture" more complex characteristics due to the human in the loop.
- Only two datasets contained a single error type. All others contained a mixture of error types.

In other studies, like done by Abedjan et al., 2016, the researchers included data cleaning tools with predefined cleaning rules or functional dependencies. This research aims to find completely automated or active learning error detection tools with no extra human configuration. Without user-defined function or rules as input to the error detection tools, automated cleaning tools will perform less on these heterogeneous datasets. With an active cleaning tool like Raha, these rules can be implicitly be transferred by actively labeling error clusters.

In the empirical study, there was no tool included for designated duplicate detection. Although datasets in the empirical study contained duplicates, there was no specific tool to detect duplicates row-wise explicitly. The study might benefit from having an error type-specific error detection tool for each error type, to give a complete overview of their workings on real-life datasets.

Also, ActiveClean and Raha are partially based on random sampling for labeling errors actively during cleaning. Different samples could lead to different results for the same configuration and dataset. Running strategies is a time-consuming process, so repeated experiments were not feasible for this stage of the research. This means that, depending on the (partially) random sampling, the same configuration could perform differently. This should be taken into account in further research.

To summarize, the current state of the art was represented by 6 different error detection tools, namely: ActiveClean, dBoost, FAHES, Forbidden Itemsets, KATARA and Raha. The performance of Raha, a human-interactive tool was the best according to the F1-measure, both overall as well as in the detailed analysis grouped on error types. In using Raha, there is a caveat, namely that the tool is highly dependent on the accuracy of the human labeling the errors in the active learning process.

6.2 Performance prediction

The results from the experiments as described in Section 4.2 will be shown in this section, to answer the second research question: *Is it possible to create an extensive data profile to estimate performance on unseen datasets?*

Strategies to estimate The goal is to predict the performance scores (recall, precision & F1-score) for all datasets and strategies available. But, not all strategies were able to produce results that are meaningful enough to estimate. From all the executed strategies, only strategies with an average F1-score for all datasets over 0 will be kept. As the estimation of performance results can only be done by learning from training examples, a certain amount of training samples is necessary to generate and test the estimations. Only strategies that generated results for more than 10 datasets (at least 75% of the datasets), will be discarded for the next steps of the research. Also, for Raha, only the strategies with fully accurate human labeling simulation are taken into account. The remaining number of strategies per tool are shown in Table 6.6. In total, there are 1157 dataset-strategy pairs that will be used as input and output for training and testing.

Tool	# of configurations
dBoost	60
Raha	20
ForbiddenItemSets	7
FAHES	3
KATARA	2
ActiveClean	1

TABLE 6.6: Number of configurations per tool filtered with F1-score > 0 and at least 75% completion

Estimation pipeline selection As discussed in Subsection 4.2.3, an estimator pipeline will be constructed based on different regression models. The pipeline exists of four stages:

1. Feature normalization/standardization (optional)
2. Feature selection (optional)
3. Principle component analysis (optional)
4. Regression model (required)

For each performance score, one estimation pipeline setting is chosen using the minimum mean square error of estimations. In each of the following subsections, the best specific estimator pipeline is discussed and the estimation results are shown. Also, the estimation errors are shown and discusses, to understand how the estimator performs.

6.2.1 Precision estimation

For the precision estimation, the best scoring estimation pipeline was:
Dataset profiles → Feature normalization → Gradient Boosting Regressor
No principle component analysis or feature selection was done.

In Figure 6.3, the distribution of estimation errors for the precision is shown. Each count represents a single estimation of the precision of an error detection strategy (tool + configuration) and a specific dataset, using the leave-one-out error. In the figure, it shows that for the precision, most of the estimations are slightly over the real values. The peak is just over 0. The median for the estimation errors is 0.0046, and the mean is -0.0092. This shows that overall, more precision estimates are over-estimations, but some of the worse estimations are underestimations (negative values).
The mean square error is 0.0537 and the median absolute error is 0.0946. This means that for most of the strategy-dataset pairs, it was capable of estimating the precision within an error of 0.1.

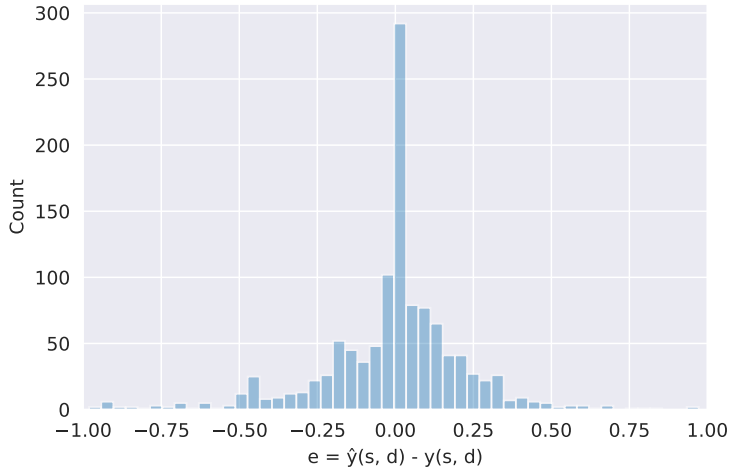


FIGURE 6.3: Precision estimation errors distribution

To put these scores in perspective, if all an example random estimator would only output 0.5 as estimated precision, the mean square error would be 0.1646 and the absolute median error would be 0.45. Whereas for the selected estimator above, the estimations would make sense, having an error of around 0.45 would not provide any beneficial information before running an error detection strategy.

6.2.2 Recall estimation

For the recall estimation, the best scoring estimation pipeline was:
Dataset profiles → Select best 2 features → 3 nearest neighbours regression
No principle component analysis was done.

In Figure 6.4, the distribution of estimation errors for the recall is shown. The best estimator pipeline for estimating the recall performs worse than it does for the precision. A mean square error 0.1045 and a mean absolute error of 0.166 is achieved. The lower estimation performance can be caused by the distribution of real recall performance scores. Around 70 dataset-strategy pairs are scoring between 0 and 0.1, and approximately 300 between 0.9 and 1. Because a high recall can be achieved by marking all the cells as erroneous, this spike in the high recall score is bigger than it is with the precision estimation (around 600 between 0 and 0.1 and 50 between 0.9 and 1). Because it is easier to achieve high recall (at the cost of precision) with error

detection on a dataset, the profile and the characteristics of that datasets might not determine the recall of an error detection strategy. Without determining factors in the dataset profiles, it becomes harder (to impossible) to estimate the recall correctly.

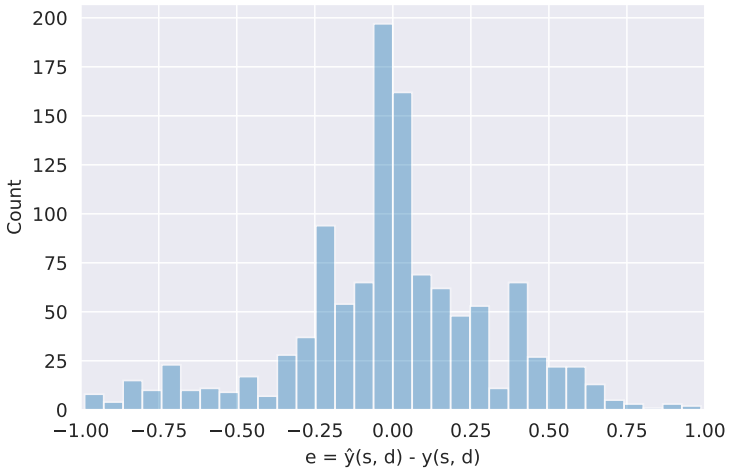


FIGURE 6.4: Recall estimation errors distribution

6.2.3 F1-score estimation

Direct estimation

For the direct F1 estimation, the best scoring estimation pipeline was:
Dataset profiles → Feature standardization → PCA with RBF-kernel to 2 dimensions → Support Vector Regression

The errors of estimation are displayed in Figure 6.5. With a mean square error of 0.0442 and median absolute error of 0.1006, this estimator is performing generally the best of the presented estimators, with respect to the mean square error. This best estimator for directly estimating the F1 score is mostly over-estimating values with small error margin.

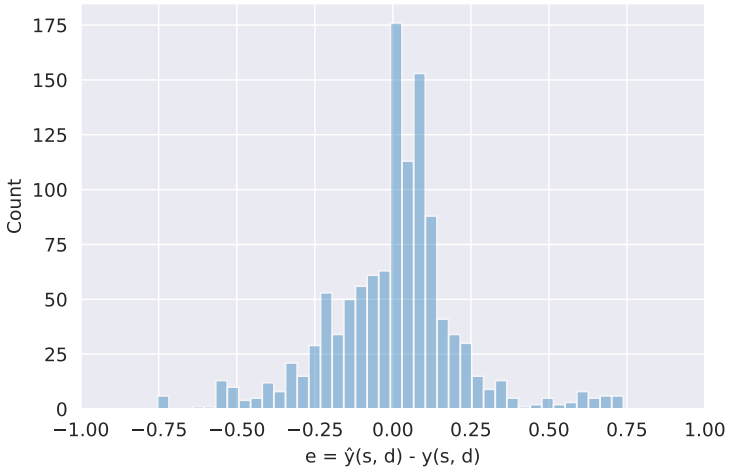


FIGURE 6.5: F1 estimation errors distribution

Combined estimation

The combined estimator is the using the best estimator pipelines from the precision and recall together to output an F1-score. The error distribution is shown in Figure 6.6. The main noticeable finding is that the distribution shifted more towards the center, performing better in general than the direct F1 score estimator. With a median absolute error of only 0.0762, it outperforms the direct estimator for most of the samples, but due to the under-estimated outliers on the left of Figure 6.6, the mean square error is negligibly higher (0.0449).

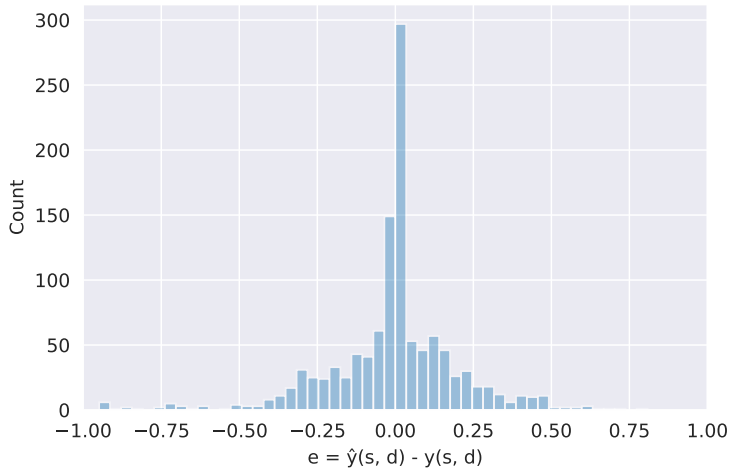


FIGURE 6.6: Combined F1 estimation errors distribution

6.2.4 Results per tool

To verify how the specific estimators perform for each tool, the mean squared error grouped for each tool is shown in Figure 6.7. It shows that for Raha, the best performing tool, has the highest mean square error. ActiveClean follows in the highest error. This could imply that it is harder to estimate error detection strategies with a human in the loop. Also, both Raha and ActiveClean are based on partially random sampling for the human labeling, which could give different results for different runs. Lastly, the fact that Raha performs best overall, also contributes to possible over-estimations for a dataset like "restaurants", where no tool performed better than 0.01 F1-score.

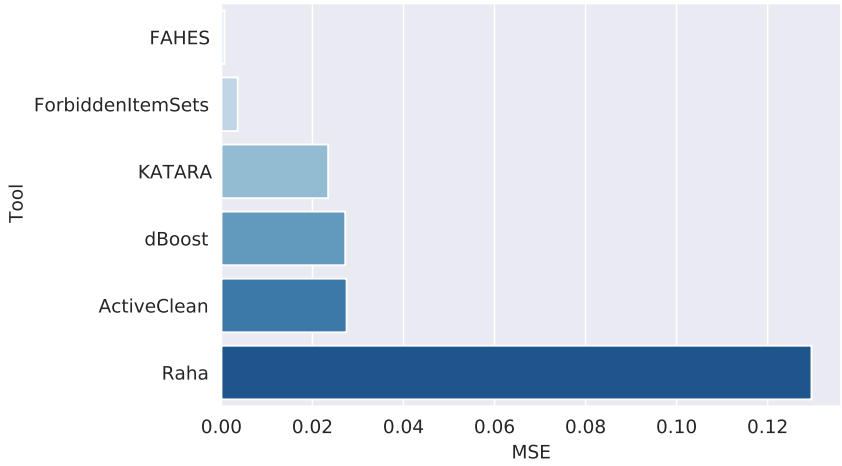


FIGURE 6.7: Mean squared error for combined F1 estimation, grouped on tool

6.2.5 Evaluation

Qualitatively, the error distributions in the section above meet the criteria set in Subsection 4.2.5. The distributions are heavily centered around 0, indicating that most estimations are close to perfect. Also, the distributions are clearly non-uniform. Only the recall estimation shows larger clusters of outliers, but combined with the precision estimation, it gives a promising output without clusters of errors for the combined F1 estimation, as seen in Figure 6.6. For comparison, the error estimation distribution of the combined F1 baseline estimation is shown in Figure 6.8. This clearly shows that the proposed estimator has an improved error distribution (Figure 6.6) over the baseline, which is mostly overestimating.

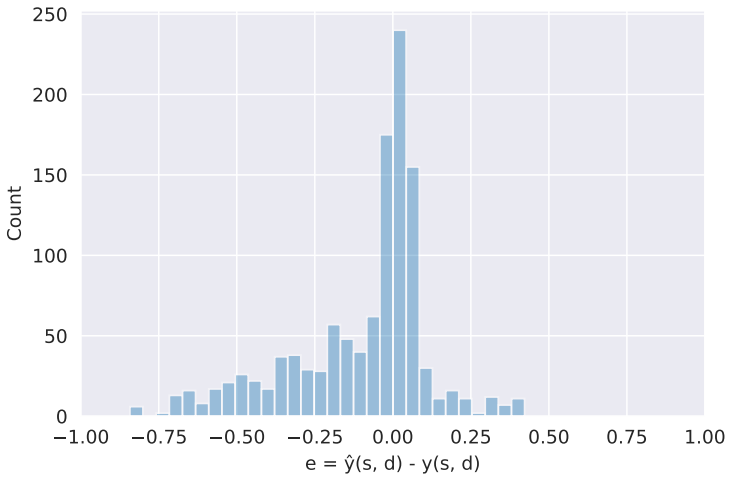


FIGURE 6.8: Baseline combined F1 estimation errors distribution

In Table 6.7, the comparison with the baseline for the mean squared error (MSE) and median absolute error (MAE) is given. For all estimation experiments, the proposed estimators score better than the baseline method. This means that it the estimator models are capable of learning information from the dataset profiles and are not just taking average scores of previously seen entries.

		Baseline	Estimator
Precision	**MSE**	0.0598	**0.0537**
	MAE	0.1357	**0.0946**
Recall	**MSE**	0.1126	**0.1045**
	MAE	0.2084	**0.1660**
Direct F1	**MSE**	0.0484	**0.0442**
	MAE	0.1264	**0.1006**
Combined F1	**MSE**	0.0586	**0.0449**
	MAE	0.0787	**0.0762**

TABLE 6.7: Estimator model and baseline method comparison for performance prediction (best scores in bold)

There are some limitations to these experiments. Like stated in Subsection 6.1.4, due the infeasibility of re-running experiments, all configurations were run once on each dataset. ActiveClean and Raha rely on random sampling for human labeling. This could imply that a performance estimation made by the models might be correct on historical data, but not on new tests, due to randomness. Also, there would have been repetitions in experiments, it would be unclear on which performance results (latest, best, first, etc..) or which possible aggregate of results (mean, max, min, etc..) to use for estimation. Therefore, in this stage, there has been chosen to only include single experiment results.

Also, the settings of each estimator pipeline are chosen to be same for each performance metric (precision, recall and F1 separately). Another possibility would be picking the optimal setting of the estimator pipeline for each specific strategy. This could improve estimation, but also leads to more complexity and possible overfitting. It should also be taken into account that the minimization for finding the best estimator was done using the mean squared error, but other metrics could be fitting as well, like the mean absolute error. This would change the resulting "best" model and its estimation error distribution.

Concluding, the experiments show positive results, both qualitatively and quantitatively. So, from the experimental results, it seems possible to create an extensive data profile to estimate performance on unseen datasets, with acceptable distributions of estimation errors, using the proposed estimator models.

6.3 Ranking

In this section, the estimators from the previous section will be used to give an estimated ranking for best strategies that could be used on an unseen dataset, to answer research question 3: *Is it possible to generate a ranking of tools according to their performance on unseen datasets?*

In the following section, the results will be based on the ranking generated by the F1-score estimators from the previous section. Only the NDCG based on the F1-score will be used to give the qualitative evaluation. For each experiment, the best strategy ranking estimator (either the combined F1-estimator or direct F1-estimator) will be shown and evaluated. In the last section, overall performance across all the rankings will be compared to the baseline method to evaluate the results quantitatively.

6.3.1 Best tool ranking

For finding the best tool, the strategy ranking with only a single configuration per tool was created. Like stated in Section 4.3, there were two ways of scoring this ranking:

Strategy-based scoring

The first way of scoring was strategy-based scoring. For this experiment the direct F1 estimator was the best performing underlying estimator for generating the ranking. In figure Figure 6.9, the NDCG scores are shown with relevance according to the suggested specific strategy.

As shown in the figure, for two out of the fourteen datasets, the score is less than half the top score (1). This implies that, for those datasets, the suggested strategy ranking placed less performing strategies higher at the top and/or gave less performing configurations of a tool priority over a better configuration.

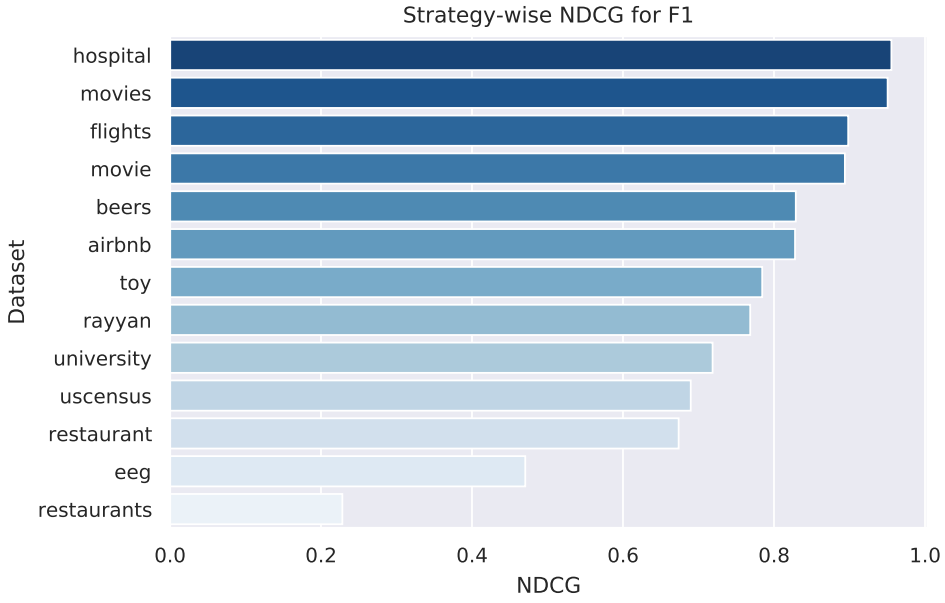


FIGURE 6.9: NDCG per dataset for strategy ranking of the direct F1 estimator

Tool-based scoring

The other way of scoring, tool-wise scoring, was were the NDCG scores calculated with relevance according to the highest scoring strategy of that tool. In this scoring way, the configurations of the ranking are disregarded and there is only a focus on the ordering of tools. The direct F1-estimator performed slightly (+0.01 increase in NDCG) better than the combined F1-estimator. The results can be seen in Figure 6.10. The resulting tool-wise NDCG scores are 0.211 (0.956 vs 0.745) higher than the NDCG scores for the strategy-wise scoring. This means that the recommended strategy ranking is better at finding the right tool, but not necessarily selects the best single configuration for that tool. The ordering for each dataset is nearly perfect, except for `Toy` (really small dataset) and `Restaurants` (where the highest F1 score was 0.01, so an outlier among the datasets).

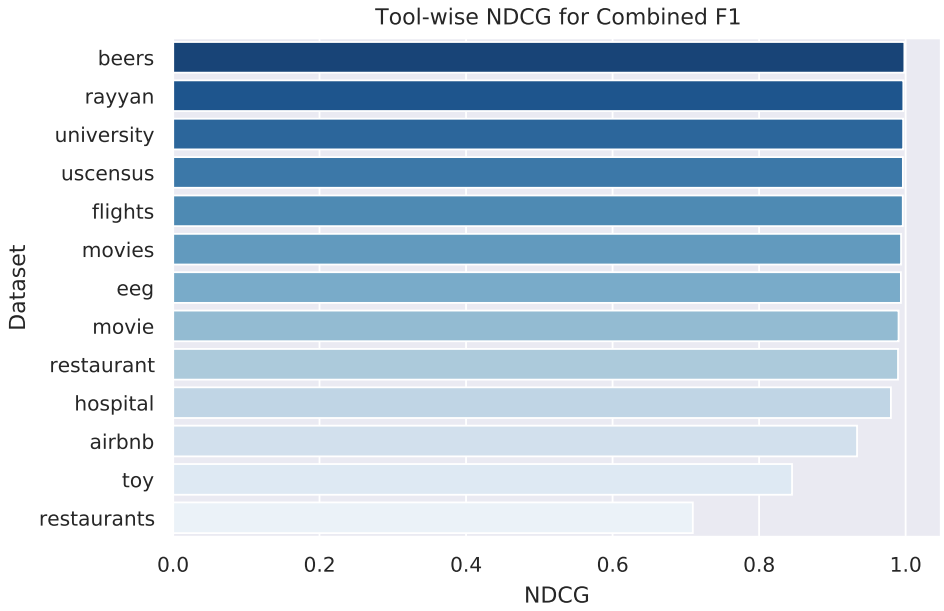


FIGURE 6.10: NDCG per dataset for tool ranking of the direct F1 estimator

6.3.2 Best configuration ranking

Following the results from Figure 6.10, where it has been shown that a good ranking of tools can be made, the next section will discuss configuration ranking for a single tool. The two error detection tools that will be covered are Raha and dBoost. The two tools are the best performing tools overall and have both have numerous configuration options, making it worthwhile to generate a configuration ranking.

Raha

For estimating ranking of the best configurations for Raha, the direct F1 estimator performed best. With a mean NDCG of 0.824, the configuration ranking for Raha is really promising. For most of the datasets, the ranking is near-optimal. As shown in Figure 6.11, only for two datasets, the score is below half optimal. This means that for those datasets, wrong suggestions will be made for the configurations to use. For the majority of datasets, the ranking system is able to output the configuration in a

ordering where the top suggestions also correspond with high performance results from the empirical study. For the worst-performing dataset, `Restaurants`, the F1-score for Raha was 0.00 for the empirical study. This results in no possible best ranking.

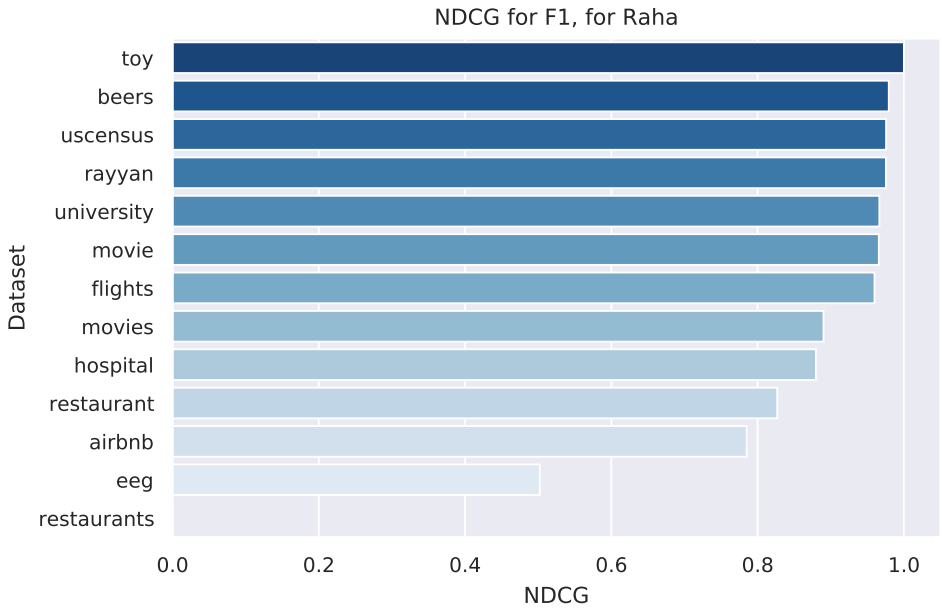


FIGURE 6.11: NDCG per dataset for configuration ranking of the direct F1 estimator for Raha

dBoost

For estimating ranking of the best configurations for dBoost, the combined F1 estimator performed best. With a mean NDCG of 0.572, the configuration ranking for dBoost is worse than the generated configuration rankings of Raha. A score of 0.572 corresponds to an just above decent ranking on average. This means that, for some datasets, it is able to output correct rankings, but for others, it can completely not. With dBoost, the difference between the best and worst-performing strategies is more significant than the difference between Raha strategies.

Also, certain dBoost configurations are locally non-sensitive, meaning that small changes in the configurations do not have a great impact on the results. These locally insensitive configurations can be grouped together based on more influential parameters. This also implies that, if an estimate for a configuration is given, the configuration in the same group will also have similar estimates. If one of these estimates is an over-estimation, multiple estimations will be too high, resulting in a worse ranking. Future improvements of the ranking system could include protection against these grouped estimations causing worse results.

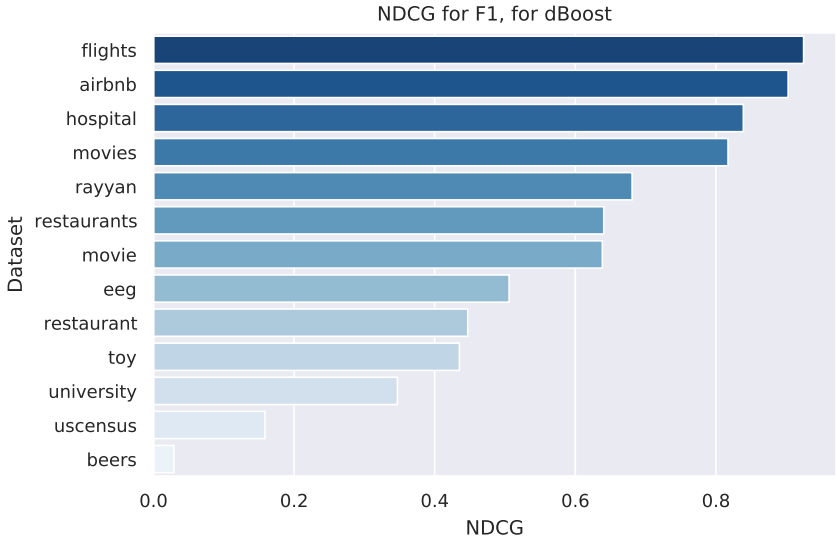


FIGURE 6.12: NDCG per dataset for configuration ranking of the combined F1 estimator for dBoost

6.3.3 Evaluation

In the sections above, the different NDCG values for the experiment per dataset have been shown. Below, a further evaluation will follow. Qualitatively, the ranking scores are promising. For the tool ranking with tool-wise NDCG scoring, the ranking system was able to generate high-quality output, without any lower outliers. When looking at the strategy-wise scored rankings (where the suggested configuration is also taken into account), the NDCG is of course "discounted", due to the stricter scoring criteria. Still, only two datasets have an NDCG below 0.5. For all other datasets, more than decent rankings have been returned, so there are no large clusters of low outliers in terms of ranking results. The same holds for the configuration ranking for Raha. With the `Restaurants` dataset, it was not possible to generate any good rankings, due to the 0 F1-score, and for the `EEG` dataset, it was close to half optimal. For the other datasets, the ranking system was capable of generating a well ordered ranking with respect to optimal configurations for Raha. Lastly, for the configuration suggestion ranking of dBoost, the NDCG values were the lowest. It contained more lower-ranking results, showing that is was harder to estimate the right order of configurations.

Quantitatively, a comparison between the proposed strategy ranking system with performance estimators based on the dataset profiles and the baseline method will be discussed below. Similarly to the evaluation of the previous section and research question (Subsection 6.2.5), there will be looked at a quantitative improvement over the scores between both systems. For all four experiments described in the results and figures above, this comparison will be made. First, the mean strategy-wise NDCG tool ranking results will be compared, as shown in Table 6.8. Overall, for all the ranking/estimator types (Precision, Recall, F1 and Combined F1), the proposed method improves over the baseline. The scores for the precision and recall are also included. It is a possibility to investigate these more in future research. Especially the results for estimated precision ranking could be used in automated error detection workflows.

Metric	Baseline Mean NDCG	Estimator Mean NDCG	Improvement
Combined F1	0.540	**0.654**	0.114
F1	0.690	**0.745**	0.055
Precision	0.495	**0.571**	0.077
Recall	0.885	**0.905**	0.020

TABLE 6.8: Mean strategy-wise NDCG comparison for tool ranking (best scores in bold)

For the other 3 experiments (tool-wise NDCG tool ranking, Raha configuration ranking & dBoost ranking), the same results are shown in Appendix A. For the tool-wise scored tool ranking results, the baseline and proposed system score close to equal on all estimation tasks. Also, the mean NDCG of these rankings are close to 1. This implies that for tool ranking and looking only at the tools for scoring (not at the suggested configurations), simply taking the best previous tool is enough to generate good rankings and cannot be improved much further. For the configuration ranking for Raha, the proposed system scores better than the baseline on all metrics. For dBoost, this not the case. The proposed system performs best on estimating the F1 score (using the combined estimator) and estimating the precision. However, the baseline performs better in estimating recall. However, the emphasis for ranking lies more on F1 and precision, as recall by itself is not meaningful (selecting all values as erroneous will give 1.00 as recall), so combining the improvement over the baseline of the F1 rankings and precision rankings, still shows the potential of the proposed ranking system.

The proposed ranking system based on the performance estimators has shown to be promising both qualitatively and quantitatively among the majority of datasets. The proposed system outperformed the set baseline and was able to create valuable rankings for different tasks, namely tool ranking and configuration ranking for a specific tool. This has shown that it is possible to generate a ranking of tools according to their performance on unseen datasets.

6.4 Interpretability

The results from the experiments, as described in Section 4.4, will be shown in this section, to answer the fourth research question: *Do these data profiles provide more interpretability of error detection tools?*

6.4.1 Feature selection

To improve machine learning models, dimensionality reduction can be a solution. Permutation importance was calculated for all the features of the dataset profiles for the best performing strategies, after all the estimators were trained. All the features that had a positive impact on the model prediction outcome were kept. 46 out of the original 72 features were kept. Now, using this hindsight information, the models were retrained with only those features. To compare the two sets of features used, the mean squared error (MSE) and median absolute error (MAE) are shown in Table 6.9. It shows that the improvements are negligible.

Metric	MSE 1	MAE 1	MSE 2	MAE 2
Precision	0.0543	0.0949	**0.0532**	**0.0948**
Recall	**0.1045**	0.166	0.1048	**0.1578**
Direct F1	0.0442	**0.1006**	**0.0440**	0.1007
Combined F1	**0.0452**	0.0790	0.0455	**0.0775**

TABLE 6.9: Estimation score comparison - 1 all features / 2 reduced number of features (best scores in bold)

6.4.2 Feature importance

The feature importance of the dataset profile features will be analyzed using SHAP values of the estimators produced in Section 4.2. The first part below will be contributed to a general overview of feature importance for all direct F1 estimator models. Then, from the second subsection, a more detailed precision and recall feature importance analysis will be conducted for every error detection tool.

General F1-score analysis

For every best scoring strategy per tool, the feature importance analysis of the direct F1-estimator was done. The extensive results can be found in Section B.1. For FAHES & Forbidden Itemsets, no feature importance could be extracted using the direct F1 estimator. For every other tool, The key takeaways were that an increase in *cells_length_variance* and/or *characters_alphabet_variance* would have a negative impact on the performance (F1). This can be attributed to the fact that, whenever lengths of cells and the number of alphabetical characters vary across columns, the data is less structured. Less structured data seems harder to detect errors in. While these two features are most important for the direct F1 estimations of all error detection tools, the model output still varies for each tool. A less structured dataset will have a different relative impact for each tool, but it is still a common issue for all datasets.

ActiveClean

To start the detailed review, ActiveClean will be discussed first. The first step in the feature performance analysis was to find the top influential features of the dataset profile to estimate the performance of the tool. In Figure 6.13, the ten most influential features for precision are shown and in Figure 6.14 for recall.

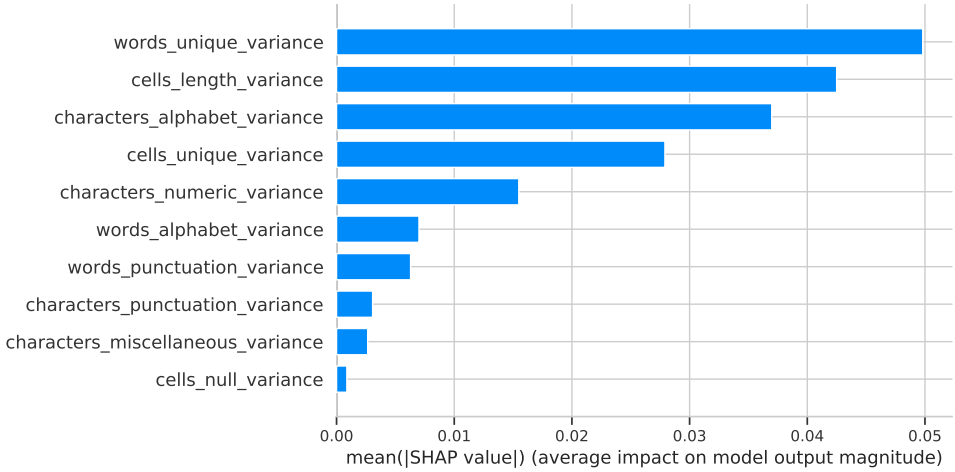


FIGURE 6.13: ActiveClean - Top 10 Precision influential features according to SHAP values

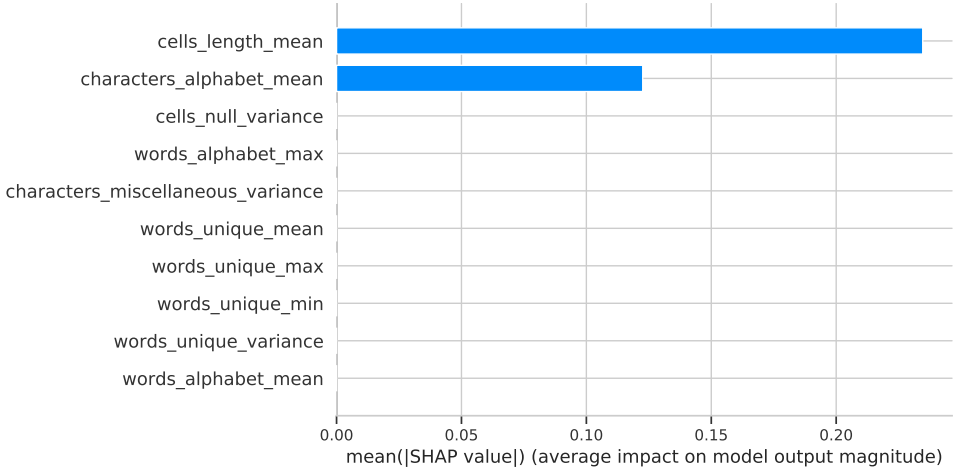


FIGURE 6.14: ActiveClean - Top 10 Recall influential features according to SHAP values

The figures above give a clear indication on the average impact on the model outputs. It shows that for the precision estimation, all top 10 features have impact on the model output. However, for the recall estimation, only 2 features are found to be beneficial for estimation.
The next step in the analysis for extract interpretability was trying to correlate the input features to the relative contribution they make on the model output. Then, these correlations need to be translated into logic or relations between the dataset profiles and the functioning of an error detection tool.

TABLE 6.10: Top feature influences - ActiveClean

#	Feature	Influence
1	words_unique_variance	0.39 ↗
2	cells_length_variance	-0.42 ↘
3	characters_alphabet_variance	-0.39 ↘

(A) Precision feature influences - ActiveClean

#	Feature	Influence
1	cells_length_mean	-0.09 ↘
2	characters_alphabet_mean	-0.12 ↘
3	-	-

(B) Recall feature influences - ActiveClean

In Table 6.10a, the correlations/influences of the three most impactful features are shown for precision. It shows that a higher *words_unique_variance* will have a positive impact on the prediction and *cells_length_variance* as well as *characters_alphabet_variance* have a negative influence with higher values. The positive impact of the variance of the number of unique words across columns can be attributed to the fact that ActiveClean makes use of the frequency of (co-)occurrences of words, so having more unique words in some columns and more similar words in other columns, makes rows distinguishable to find duplicates or rule violations. Having higher *cells_length_variance* and *characters_alphabet_variance* will manifest as unstructured data with varying character types across columns, which has a negative impact on ActiveClean. In Table 6.10b, the influences of the most important features for recall are shown. Because only 2 features had an impact, only two will be discussed. The average length of cells and the average number of alphabetical characters per column negatively impact the model output. This could be attributed to the fact that datasets with large values and only alphabetical values will be indistinguishable for ActiveClean. This results in marking each value as erroneous, which will lead to a higher recall.

dBoost

For dBoost and the following tools in this chapter, only the correlation tables will be shown. The full impact bar plots for the top 10 influential features can be found in Section B.2 of the appendix. Also, more condensed explanations for the feature importance will be given, as the main thinking pattern was shown in the section above.

TABLE 6.11: Top feature influences - dBoost

#	Feature	Influence
1	cells_length_variance	-0.79 ↘
2	characters_alphabet_variance	-0.59 ↘
3	cells_unique_variance	0.72 ↗

(A) Precision feature influences - dBoost

#	Feature	Influence
1	cells_punctuation_variance	-0.39 ↘
2	cells_null_max	0.29 ↗
3	-	-

(B) Recall feature influences - dBoost

As seen in Table 6.11a, for the precision feature importance, the two most impactful features are the same as the general F1 feature importance analysis. dBoost is negatively impacted in terms of precision if the dataset is less structured (varying cell lengths and varying character types). Being based on multivariate outliers for detecting errors, the third feature *cells_unique_variance* can positively influence the results if higher. A variance across columns for unique cells makes it such that columns can have identifying values in one column (low number of unique cells), like a category, and then outliers of a certain rule (high number of unique cells) in

another column, making errors identifiable. And as seen in Table 6.11b, the recall is negatively impacted if, across columns, the cells containing punctuation vary, which might be attributed to the inability to find appropriate expansion schemes for these cells. If there are many null cells in the dataset, the recall improves, as they are common errors and detectable by dBoost.

FAHES

As can be seen in Table 6.12a, precision for FAHES is positively impacted by varying unique cells across columns, which could make errors more identifiable. Furthermore, it is negatively impacted by unstructured data (*cells_length_variance* and *characters_alphabet_variance*). However, if the cells in the data are varying of type (alphabetic or numeric), across columns, the recall improves (Table 6.12b).

TABLE 6.12: Top feature influences - FAHES

#	Feature	Influence
1	cells_unique_variance	0.67 ↗
2	cells_length_variance	-0.59 ↘
3	characters_alphabet_variance	-0.59 ↘

(A) Precision feature influences - FAHES

#	Feature	Influence
1	cells_alphabet_variance	0.66 ↗
2	cells_numeric_variance	0.61 ↗
3	-	-

(B) Recall feature influences - FAHES

Forbidden Itemsets

For Forbidden Itemsets, the most influential features are shown in Table 6.13a and Table 6.13b, for precision and recall, respectively. If the length of the cells varies across columns, precision improves. Forbidden Itemsets works with identifying "unlikely" value pairs, so if errors have other lengths than non-errors, this could make them detectable by the tool. It is not straightforward to interpret the impact of the unique cells and words on precision, as there is no clear correlation between the features and the impact. This shows that there is not a simple correlation to explain the impact. Also, recall improves if the datasets contain larger words and more varying words.

TABLE 6.13: Top feature influences - Forbidden Itemsets

#	Feature	Influence
1	cells_length_variance	0.36 ↗
2	cells_unique_variance	0.0 →
3	words_unique_variance	-0.18 ↘

(A) Precision feature influences - Forbidden Itemsets

#	Feature	Influence
1	words_length_variance	0.3 ↗
2	words_length_min	0.23 ↗
3	-	-

(B) Recall feature influences - Forbidden Itemsets

KATARA

For KATARA the feature importance for the three most impactful features are shown in Table 6.14a and Table 6.14b. The three features that impact precision show that it needs structured cells, mostly alphabetical characters and words, in order to achieve higher precision. This seems logical, as KATARA is solely based on relations in

knowledge bases based on known words. The recall is mostly slightly negatively influenced by the number of average punctuation characters and miscellaneous words, as these are not present in the knowledge base powering KATARA.

TABLE 6.14: Top feature influences - KATARA

#	Feature	Influence
1	words_alphabet_variance	0.29 ↗
2	cells_length_variance	-0.2 ↘
3	characters_alphabet_variance	0.11 ↗

(A) Precision feature influences - KATARA

#	Feature	Influence
1	characters_punctuation_mean	-0.05 →
2	words_miscellaneous_mean	-0.07 ↘
3	-	-

(B) Recall feature influences - KATARA

Raha

Lastly, the feature importance for Raha is shown in Table 6.15a and Table 6.15b. Raha's precision is negatively impacted by the number of miscellaneous characters and unstructured columns (varying cell length). The recall estimation is not impacted by any of the input features from the data profile. The estimator will most likely have a single recall value as output, meaning that it was not capable of learning from the dataset profiles with regards to the recall of Raha.

TABLE 6.15: Top feature influences - Raha

#	Feature	Influence
1	characters_miscellaneous_mean	-0.42 ↘
2	characters_miscellaneous_variance	-0.35 ↘
3	cells_length_variance	-0.2 ↘

(A) Precision feature influences - Raha

#	Feature	Influence
1	-	-
2	-	-
3	-	-

(B) Recall feature influences - Raha

6.4.3 Evaluation

To see if interpretability in retrospect would help improve the estimator models using feature selection, the estimator models were tested with the regular dataset profiles input and the reduced feature set dataset profiles in Subsection 6.4.1. No clear improvement has been made in automating the feature selection in retrospect. Different selection criteria could yield different results, but that could be looked into in future work.

As stated in the methodology, the main focus laid on whether the dataset profiles and estimators for the best performing configurations per tool would provide interpretability. In general (found by the F1 estimator feature importances), it was found that error detection tools benefit from structured data, with less variance between the length and content (characters) of cells. For the precision feature importance, it was possible to identify at least three features, for every tool, that had a strong impact on the model output and could give an estimated explanation of how and when the tool would work properly. However, for recall, it was not possible to identify multiple impactful features and for some tools even no features.

Besides finding impactful features, it was tested if the impactful features could be correlated with impact to translate the findings to theories or logic about the characteristics of a dataset. An attempt at giving logical explanations was provided in

the previous sections. But, it was not tested in this research whether these findings hold for every situation. It was not possible for every impactful feature to find a strong correlation with respect to the output. However, for the majority of the tools, a link between a dataset characteristic (dataset profile feature) and the influence on a performance score could be made.

Adding interpretability using SHAP values is only possible for a single estimator at a time. It would not be feasible to integrate SHAP analysis in the strategy ranking system, but it could be used to give confidence to a suggestion when a final error detection strategy has already been selected. Further analysis could be done on explaining the predictions of performance scores for a single dataset and error detection strategy at one time in more detail. This would possibly give more confidence in the suggestions to the experts using the predictions and suggested rankings made by the system proposed in this research.

Considering the evaluation from above, it has been shown that it is possible to provide some insights on the performance of error detection tools using interpretability methods with regards to features from the dataset profiles. However, work needs to be done in order to integrate the interpretability methods in an error detection workflow and verify the produced results and logic from the SHAP values. But concluding, it can be deemed true that the data profiles provide more interpretability of error detection tools.

Chapter 7

Conclusion & Future Work

Conclusion

In this thesis, a comparative study has been done for error detection tools on relational data. Subsequently, an attempt was made to estimate performance of error detection tools and particular configurations on unseen datasets, based on high-level profiles of these datasets. These estimators were used to generate suggested rankings of error detection strategies. Ultimately, these estimators were analyzed to provide more interpretability on the functioning of the error detection tools on the datasets in this research. In this research, multiple research questions were proposed in Section 1.3. The methodology for experiments in this work and to answer these research questions was given in Chapter 4. Below, each research question will be restated and answered according to the findings in Chapter 6.

Research Question 1: *What is the current state of the art and what is the performance of these tools?*
The current state of the art was represented by six error detection tools with different underlying techniques: ActiveClean, dBoost, FAHES, Forbidden Itemsets, KATARA and Raha. An empirical study was done on 14 datasets with a range of characteristics. Out of the six tools, Raha, an interactive tool performed best on average, with regards to the F1-score. However, the performance results of human interactive tools were highly dependent on the accuracy of the human in the loop.

Research Question 2: *Is it possible to create an extensive data profile to estimate performance on unseen datasets?*
A data profile about the dataset was created without any foreknowledge about that dataset. Using this data profile, the performance results of the error detection tools and their configuration were estimated. The proposed estimators were evaluated on the test datasets and the empirical results from the previous research question. The errors of these estimations on real performance scores were evaluated qualitatively by looking at the distribution of estimation errors. The distributions were heavily centered around 0, indicating low errors, without any outliers of high errors. The estimator also clearly outperformed the baseline estimation method. So, both qualitatively and quantitatively, it was found to be possible to create an extensive data profile to estimate performance on unseen datasets using the proposed estimator models.

Research Question 3: *Is it possible to generate a ranking of tools according to their performance on unseen datasets?*
Using the proposed performance estimators, an estimated suggested ranking of error detection strategies for an unseen dataset was created. The proposed system outperformed the set baseline and was able to create valuable rankings for different tasks, namely tool ranking and configuration ranking for a specific tool. This has shown that it is possible to generate a ranking of tools according to their performance on unseen datasets, with the proposed ranking system.

Research Question 4: *Do these data profiles provide more interpretability of error detection tools?*
Lastly, the error detection tool performance estimators were analyzed to provide interpretability on how and when an error detection strategy would perform well. The proposed automated feature selection, based on feature importance, did not improve estimation results. However, providing logic and relations between performance scores and the dataset input features was deemed possible. Found was that the F1-measure was mostly impacted by how structured the data was, with most tools performing better when columns of the data have similar length and contained similar numbers of alphabetical characters. For precision and recall, a detailed overview of the impact of dataset profile features was given. For the majority of the tools, a link between a dataset characteristic (dataset profile feature) and the influence on a performance score could be made. The data profiles did provide more interpretability of error detection tools, but in order to integrate the method into the error detection workflow and verify the results, more work needs to be done.

Main Research Question: *How to choose a fitting error detection algorithm for a specific relational dataset?*
By using the knowledge from the empirical study, a user is able to see which tools are performing well in general. Then, for a detailed choice, the user can use a suggested strategy ranking for that specific relational dataset, to find the best tool and configuration in a substantiated manner. The strategy ranking system and underlying performance estimators proposed in this work have shown to be effective in suggesting a particular error detection tool and matching configuration for an unseen dataset. The strategy ranking system's outcomes could help choose a fitting error detection algorithm for a specific relational dataset.

Future work

Besides the possible improvements and considerations discussed in the results from Chapter 6, there are multiple next steps that could additionally improve the research presented in this thesis. Below, a non-exhaustive list of possible next steps is presented:

- Estimating the runtime of error detection tools
To give a complete suggestion for experts in the data cleaning and error detection field, runtime estimations of the tools could also be included. Not only would an expert then be able to choose a preferred error detection strategy based on performance scores, but would also be able to make a cost-effective decision.

- Integrate more error detection tools and add new datasets

The research could be expanded with more state of the art tools and more error type-specific tools. Also, adding more datasets would add more data, substantiating the findings and improving the performance estimators.

- Creating an online repository for the suggestion results
The newly added tools, datasets and performance results for the empirical study could be published and shared online, in order to provide access directly to experts. Also, new tools and datasets can then be easily adapted by outsiders. More knowledge would lead to more expertise in the suggestion system.

- A balanced number of strategies per tool
The estimators were evaluated on all configurations for all tools. However, for some tools, many more configurations exist than for others. This gives an imbalanced training and minimization problem. The number of configurations per tool could be normalized, or scoring could be discounted for the tools with excessive amounts of configurations. Creating a balanced optimization and learning problem could improve the overall predictive capabilities and generated rankings.

- Error correction tools
The focus of this research has been on error detection tools. The next step after error detection would be the correction of these errors. Error detection is a highly complex and heterogeneous problem, but automated error correction is even more so. However, the same steps (empirical study $\rightarrow$ performance prediction $\rightarrow$ strategy ranking) could be applied for evaluating error correction tools. This could be beneficial for the complete data cleaning workflow of computer scientists working on relational data.

Appendix A

Ranking

The following tables are a supplement to the baseline comparison for Subsection 6.3.3.

TABLE A.1: Mean tool-wise NDCG comparison for tool ranking

Metric	Baseline Mean NDCG	Estimator Mean NDCG	Improvement
Combined F1	**0.953**	0.942	-0.012
F1	**0.958**	0.956	-0.002
Precision	**0.939**	0.934	-0.005
Recall	0.964	**0.972**	0.009

TABLE A.2: Mean NDCG comparison for configuration ranking Raha

Metric	Baseline Mean NDCG	Estimator Mean NDCG	Improvement
Combined F1	0.702	**0.807**	0.105
F1	0.731	**0.824**	0.092
Precision	0.652	**0.802**	0.150
Recall	0.913	**0.970**	0.057

TABLE A.3: Mean NDCG comparison for configuration ranking dBoost

Metric	Baseline Mean NDCG	Estimator Mean NDCG	Improvement
Combined F1	0.424	**0.572**	0.149
F1	**0.521**	0.492	-0.029
Precision	0.565	**0.647**	0.082
Recall	**0.911**	0.829	-0.083

Appendix B

Feature importance

B.1 F1 feature importance

The following subsections dive into the F1 feature importances of each tool. These analyses can be used as a reference for the main results given in Section 6.4.

B.1.1 FAHES & ForbiddenItemSets

Due to the low average F1-scores for FAHES & ForbiddenItemSets, it is not possible to extract any SHAP values from the estimators, as the estimate for the F1 is equal to 0. Whilst it was possible to extract interpretations from the precision and recall estimators, these will not be discussed as this is out of scope for this research for now.

B.1.2 ActiveClean

For ActiveClean, the 10 most influential features are shown in figure B.1. The estimators base the F1 prediction mainly on the variance of characteristics across the different columns a dataset has. Meaning that, if the characteristics of columns differ from each other, the variance for that feature will be higher.

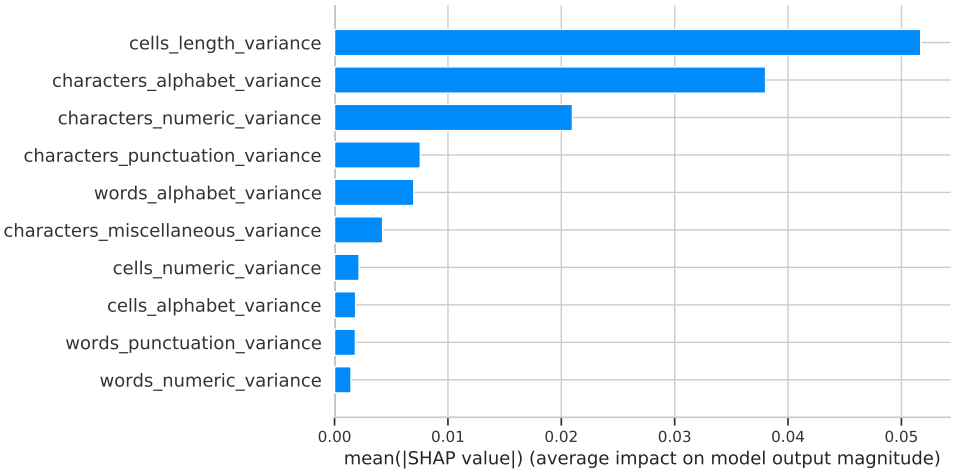


FIGURE B.1: ActiveClean - Top 10 influential features according to SHAP values

The top 3 features and their correlation with impact are displayed in table B.1. When the cells_length_variance is higher, the performance result will be estimated lower. This means that if different columns, have different content sizes, the error

detection job most likely becomes harder. This happens with heterogeneous relational data with many different data types and lengths. Also, when there is a lot of difference between the use of alphabetical characters between columns (characters_alphabet_variance), so some have many alphabetical characters, others do not, the performance decreases. Lastly, whenever the characters_numeric_variance is higher, the performance will increase. This means that if there are columns that have many numerical characters and other columns are non-numerical, it is easier for ActiveClean to differentiate errors. These results can be the effect of the featurizer, tf–idf (term frequency–inverse document frequency), which relies on the frequency of words and word pairs occuring in that data. So if the lengths of attributes and the number of alphabetical characters per column are generally the same, but there are some columns numerical, it is able to correctly group these rows and do error detection.

#	Feature	Influence
1	cells_length_variance	-0.48 ↘
2	characters_alphabet_variance	-0.56 ↘
3	characters_numeric_variance	0.42 ↗

TABLE B.1: ActiveClean - Top 3 feature influence

B.1.3 dBoost

For dBoost, the 10 most influential features are shown in figure B.2.

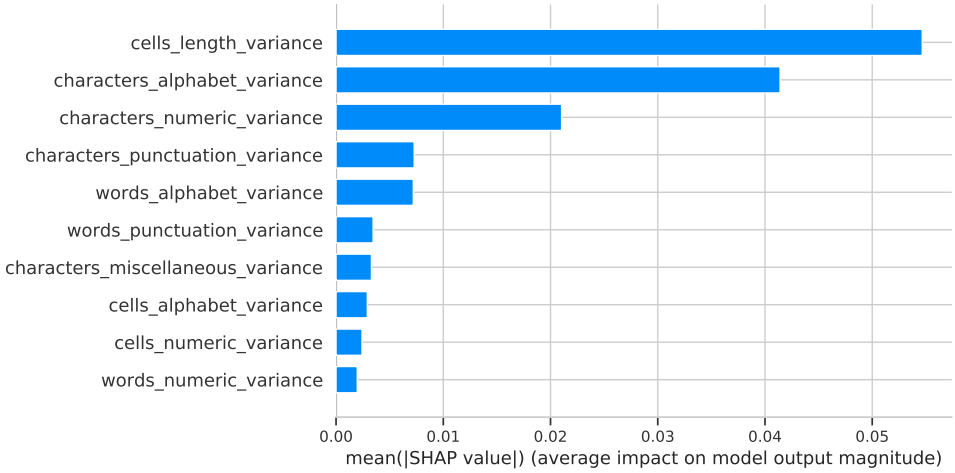


FIGURE B.2: dBoost - Top 10 influential features according to SHAP values

The top 3 impactful features for dBoost are the same as for ActiveClean. However, the influence of the features is different B.2. if the length of cells across varies more, the performance of dBoost decreases more drastically. The variance in numerical characters does have a high impact on the performance, but no direct correlation, meaning that depending on different features, it might have a positive or negative impact. Concluding, dBoost will perform best in structured data with the same length of cells across the columns.

#	Feature	Influence
1	cells_length_variance	-0.67 ↘
2	characters_alphabet_variance	-0.27 ↘
3	characters_numeric_variance	-0.04 →

TABLE B.2: dBoost - Top 3 feature influence

B.1.4 KATARA

For KATARA, the 10 most influential features are shown in figure B.3.

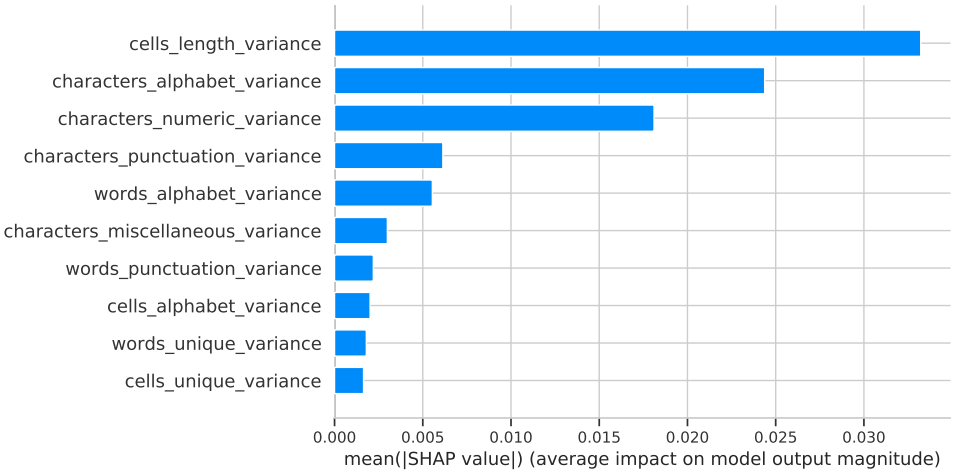


FIGURE B.3: KATARA - Top 10 influential features according to SHAP values

As shown in table B.3, again the most influential features for the F1 estimation are the same as for the other tools. The influences are comparable to that of ActiveClean. This can be attributed to the fact that both capture row-wise characteristics. ActiveClean tries to find common or more important words using tf-idf and KATARA tries to find common words and then map those columns of the common words to relations.

#	Feature	Influence
1	cells_length_variance	-0.2 ↘
2	characters_alphabet_variance	-0.09 ↘
3	characters_numeric_variance	0.36 ↗

TABLE B.3: KATARA - Top 3 feature influence

B.1.5 Raha

For Raha, the 10 most influential features are shown in figure B.4. Whereas the previous tools were most impacted by the variance based features across columns, Raha is also depending slightly on other features like the maximum of punctuation words used in a column.

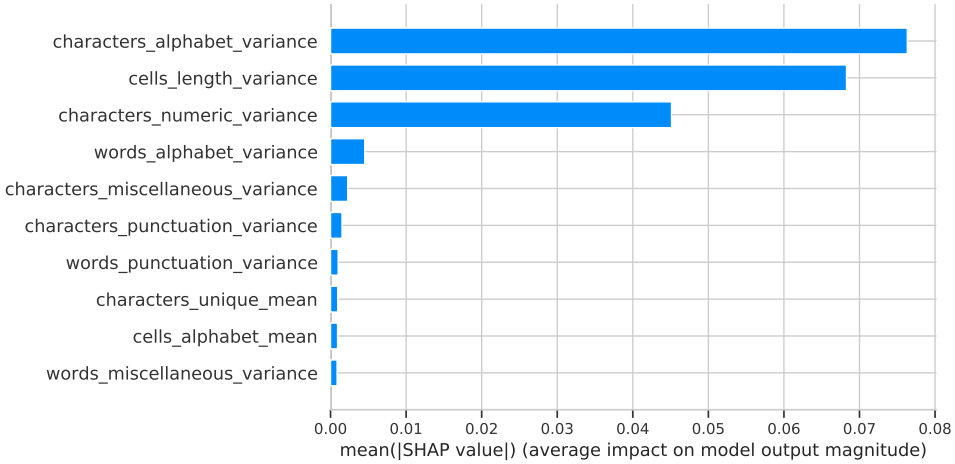


FIGURE B.4: Raha - Top 10 influential features according to SHAP values

But, again the most influential features are characters_alphabet_variance, cells_length_variance and characters_numeric_variance. All have a strongly negative correlation with the performance results estimation, meaning that Raha does better when these characteristics have low variance. This means that it performs best with structured data, with same length content and without too many difference in terms of numerical characters between attributes. Because Raha is also based on the combination of underlying tools like dBoost and KATARA, it is as expected that the F1 score will be influenced by the same characteristics as these tools.

#	Feature	Influence
1	characters_alphabet_variance	-0.49 ↘
2	cells_length_variance	-0.42 ↘
3	characters_numeric_variance	-0.42 ↘

TABLE B.4: Raha - Top 3 feature influence

B.2 Precision & Recall importance

The following figures are given additionally with the results described in the precision and recall feature importance analysis in Section 6.4. For every selected tool and best performing (highest mean F1) configuration, the top 10 most impactful features according to SHAP value analysis are given below.

B.2.1 dBoost

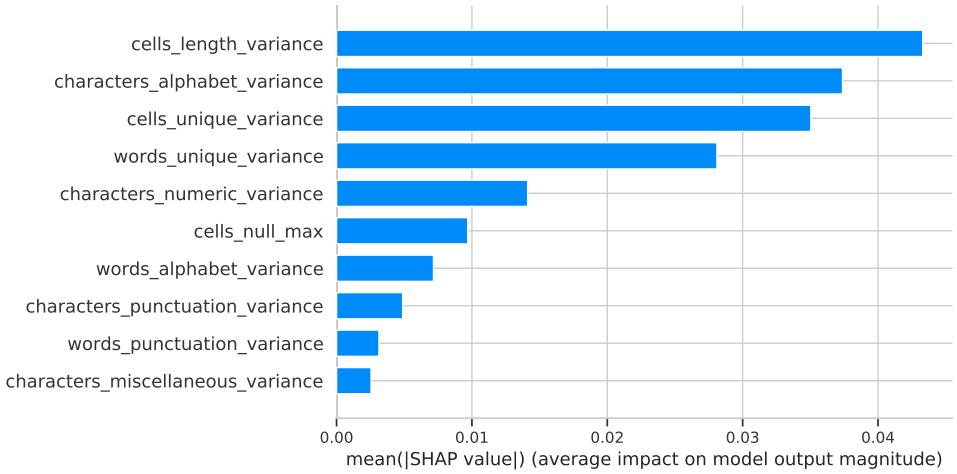


FIGURE B.5: dBoost - Top 10 Precision influential features according to SHAP values

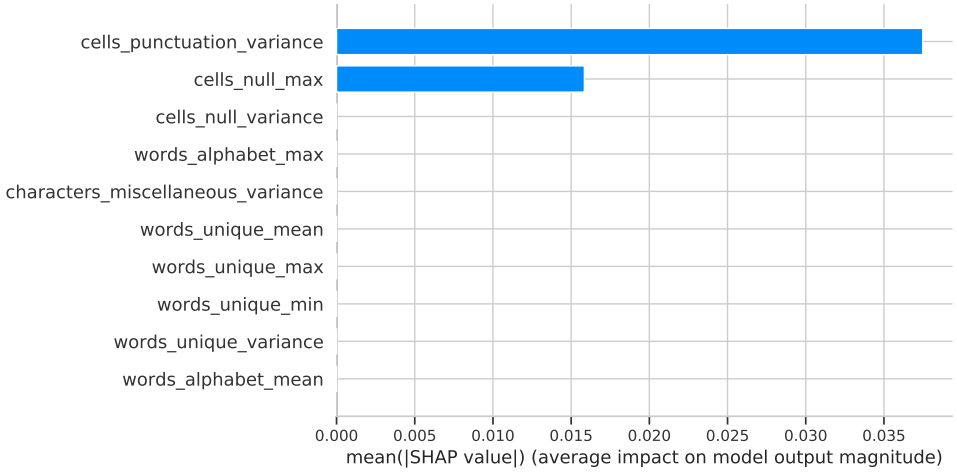


FIGURE B.6: dBoost - Top 10 Recall influential features according to SHAP values

B.2.2 FAHES

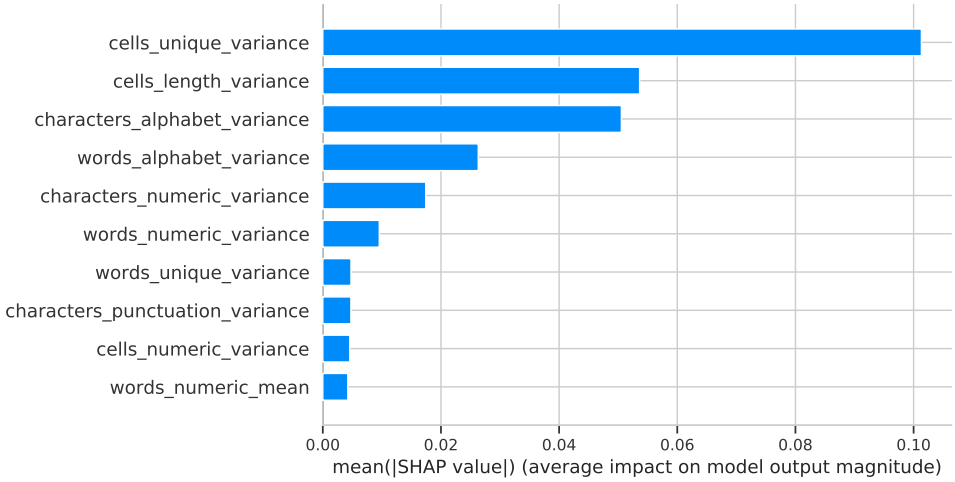


FIGURE B.7: FAHES - Top 10 Precision influential features according to SHAP values

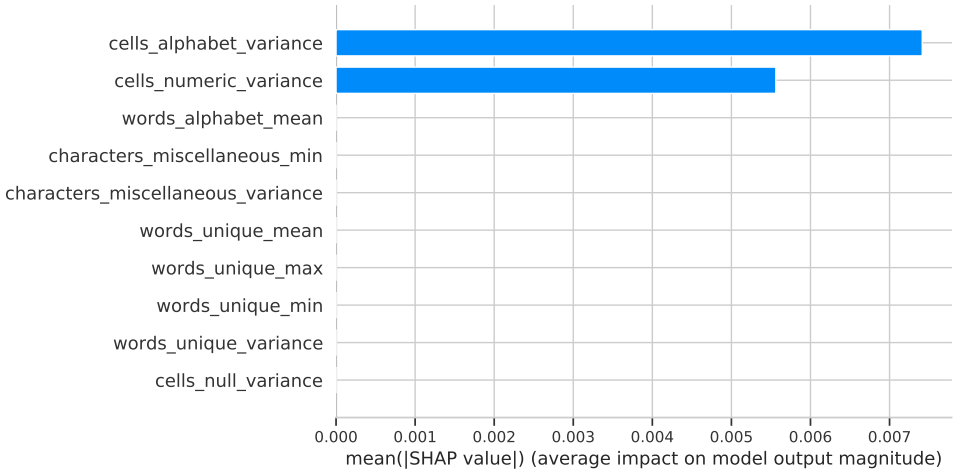


FIGURE B.8: FAHES - Top 10 Recall influential features according to SHAP values

B.2.3 Forbidden Itemsets

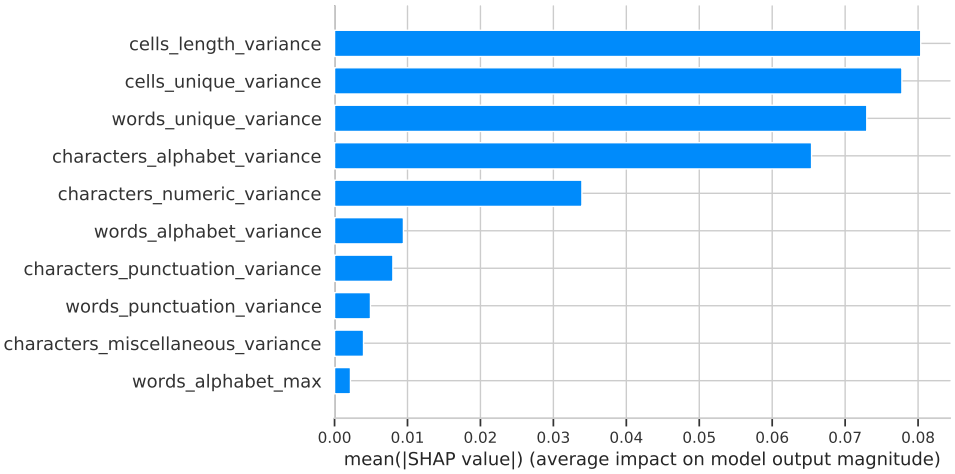


FIGURE B.9: Forbidden Itemsets - Top 10 Precision influential features according to SHAP values

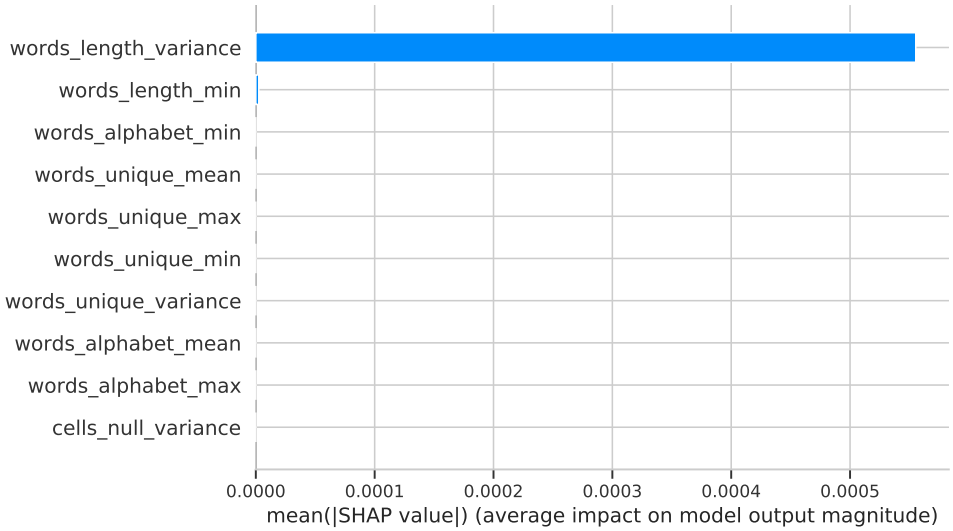


FIGURE B.10: Forbidden Itemsets - Top 10 Recall influential features according to SHAP values

B.2.4 KATARA

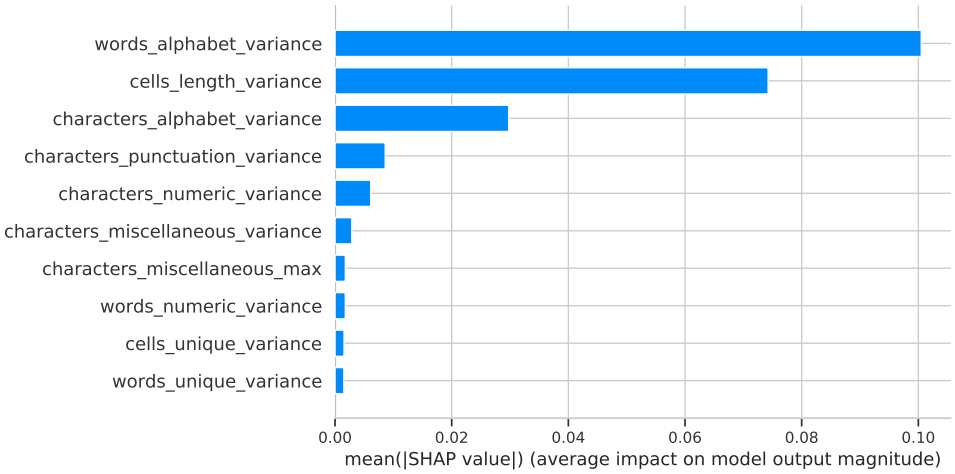


FIGURE B.11: KATARA - Top 10 Precision influential features according to SHAP values

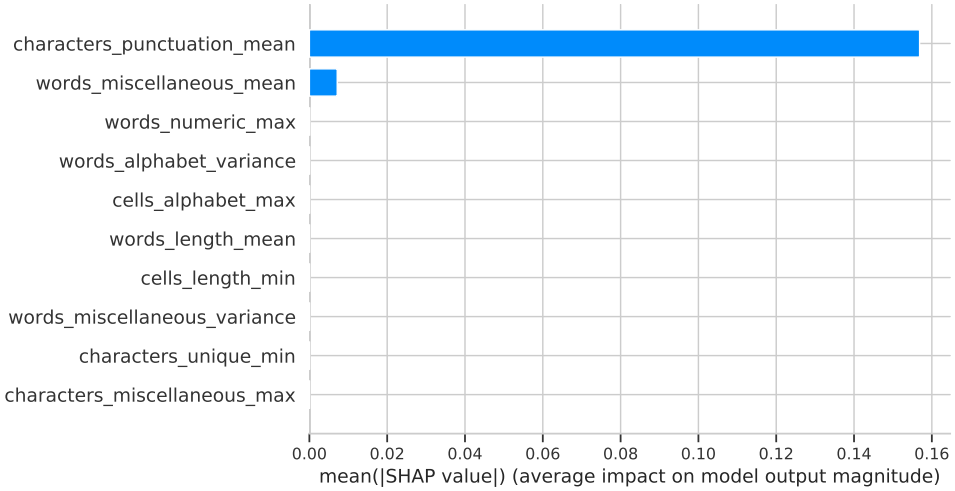


FIGURE B.12: KATARA - Top 10 Recall influential features according to SHAP values

B.2.5 Raha

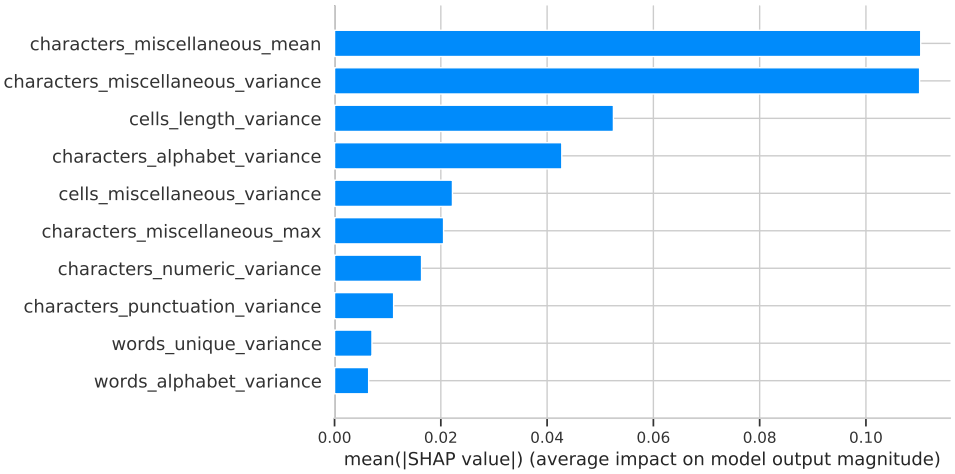


FIGURE B.13: Raha - Top 10 Precision influential features according to SHAP values

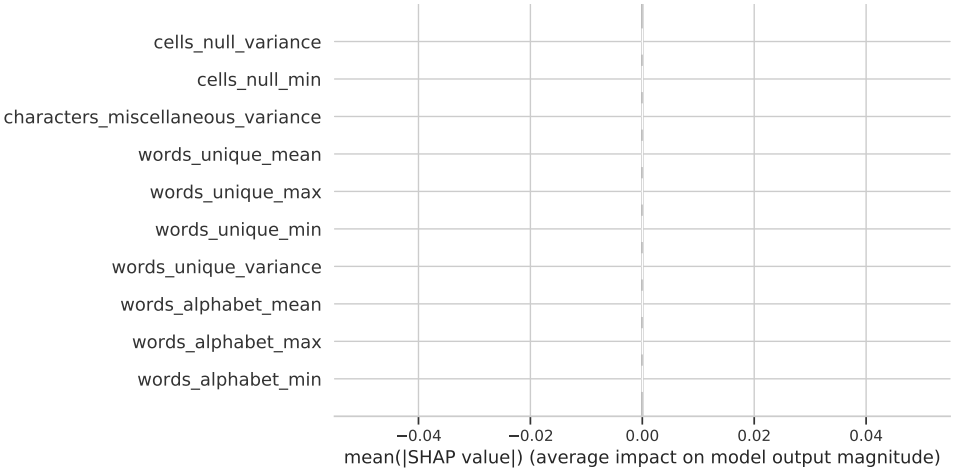


FIGURE B.14: Raha - Top 10 Recall influential features according to SHAP values

Bibliography

Abedjan, Ziawasch, Lukasz Golab, and Felix Naumann (Aug. 2015). "Profiling relational data: a survey". In: *VLDB J.* 24.4, pp. 557–581.

Abedjan, Ziawasch et al. (Aug. 2016). "Detecting Data Errors: Where Are We and What Needs to Be Done?" In: *Proceedings VLDB Endowment* 9.12, pp. 993–1004.

Altmann, André et al. (May 2010). "Permutation importance: a corrected feature importance measure". en. In: *Bioinformatics* 26.10, pp. 1340–1347.

Arocena, Patricia C et al. (Oct. 2015). "Messing Up with BART: Error Generation for Evaluating Data-cleaning Algorithms". In: *Proceedings VLDB Endowment* 9.2, pp. 36–47.

Asghar, Nabiha and Amira Ghenai (2015). "Automatic discovery of functional dependencies and conditional functional dependencies: A comparative study". In: *university of Waterloo*.

Auer, Sören et al. (2007). "DBpedia: A Nucleus for a Web of Open Data". In: *The Semantic Web*. Springer Berlin Heidelberg, pp. 722–735.

Bartoli, A et al. (May 2016). "Inference of Regular Expressions for Text Extraction from Examples". In: *IEEE Trans. Knowl. Data Eng.* 28.5, pp. 1217–1230.

Batini, Carlo et al. (July 2009). "Methodologies for data quality assessment and improvement". In: *ACM Comput. Surv.* 41.3, pp. 1–52.

Cai, Li and Yangyong Zhu (May 2015). "The Challenges of Data Quality and Data Quality Assessment in the Big Data Era". en. In: *Data Science Journal* 14.0, p. 2.

Chu, Xu, Ihab F Ilyas, and Paolo Papotti (Aug. 2013). "Discovering denial constraints". In: *Proceedings VLDB Endowment* 6.13, pp. 1498–1509.

Chu, Xu et al. (May 2015). "KATARA: A Data Cleaning System Powered by Knowledge Bases and Crowdsourcing". In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. SIGMOD '15. Melbourne, Victoria, Australia: Association for Computing Machinery, pp. 1247–1261.

Chung, Y, S Krishnan, and T Kraska (2017). "A data quality metric (DQM): how to estimate the number of undetected errors in data sets". In: *Proceedings VLDB Endowment*.

Codd, E F (June 1970). "A relational model of data for large shared data banks". In: *Commun. ACM* 13.6, pp. 377–387.

Dallachiesa, Michele et al. (2013). "NADEEF: a commodity data cleaning system". In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. dl.acm.org, pp. 541–552.

Doshi-Velez, Finale and Been Kim (Feb. 2017). "Towards A Rigorous Science of Interpretable Machine Learning". In: arXiv: 1702.08608 [stat.ML].

Hall, Mark A (2000). "Correlation-based feature selection of discrete and numeric class machine learning". In:

Heidari, Alireza et al. (June 2019). "HoloDetect: Few-Shot Learning for Error Detection". In: *Proceedings of the 2019 International Conference on Management of Data*. ACM, pp. 829–846.

Huang, Zhipeng and Yeye He (2018). "Auto-Detect: Data-Driven Error Detection in Tables". In: *Proceedings of the 2018 International Conference on Management of Data*. SIGMOD '18. Houston, TX, USA: ACM, pp. 1377–1392.

Ilyas, Ihab F and Xu Chu (2015). "Trends in Cleaning Relational Data: Consistency and Deduplication". In: *Foundations and Trends® in Databases* 5.4, pp. 281–393.

Koumarelas, Loannis, Thorsten Papenbrock, and Felix Naumann (Jan. 2020). "MDedup: duplicate detection with matching dependencies". In: *Proceedings VLDB Endowment* 13.5, pp. 712–725.

Krishnan, Sanjay et al. (2016a). "ActiveClean: An Interactive Data Cleaning Framework For Modern Machine Learning". In: *Proceedings of the 2016 International Conference on Management of Data*. SIGMOD '16. San Francisco, California, USA: ACM, pp. 2117–2120.

Krishnan, Sanjay et al. (Aug. 2016b). "ActiveClean: Interactive Data Cleaning for Statistical Modeling". In: *Proceedings VLDB Endowment* 9.12, pp. 948–959.

Li, Jundong et al. (Dec. 2017). "Feature Selection: A Data Perspective".

Li, Peng et al. (Apr. 2019). "CleanML: A Benchmark for Joint Data Cleaning and Machine Learning [Experiments and Analysis]". In: arXiv: 1904.09483 [cs.DB].

Lundberg, Scott M and Su-In Lee (2017). "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems 30*. Ed. by I Guyon et al. Curran Associates, Inc., pp. 4765–4774.

Mahdavi, Mohammad and Ziawasch Abedjan (2019). "REDS: Estimating the Performance of Error Detection Strategies Based on Dirtiness Profiles". In: *Proceedings of the 31st International Conference on Scientific and Statistical Database Management*. SSDBM '19. Santa Cruz, CA, USA: ACM, pp. 193–196.

Mahdavi, Mohammad et al. (2019). "Raha: A Configuration-Free Error Detection System". In: *Proceedings of the 2019 International Conference on Management of Data*. SIGMOD '19. Amsterdam, Netherlands: ACM, pp. 865–882.

Mitkov, Ruslan (2004). *The Oxford Handbook of Computational Linguistics*. en. OUP Oxford.

Molnar, Christoph (Feb. 2020). *Interpretable Machine Learning*. en. Lulu.com.

Neutatz, Felix, Mohammad Mahdavi, and Ziawasch Abedjan (Nov. 2019). "ED2: A Case for Active Learning in Error Detection". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. ACM, pp. 2249–2252.

Pearson, Karl (Nov. 1901). "LIII. On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11, pp. 559–572.

Pedregosa, F et al. (2011). "Scikit-learn: Machine Learning in Python". In: *J. Mach. Learn. Res.* 12, pp. 2825–2830.

Pit-Claudel, Clément et al. (Feb. 2016). "Outlier Detection in Heterogeneous Datasets using Automatic Tuple Expansion". In:

Porter, Bruce W, Ray Bareiss, and Robert C Holte (1990). *Concept learning and heuristic classification in weak-theory domains*. Tech. rep. TEXAS UNIV AT AUSTIN ARTIFICIAL INTELLIGENCE LAB.

Qahtan, Abdulhakim A et al. (2018). "FAHES: A Robust Disguised Missing Values Detector". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '18. London, United Kingdom: ACM, pp. 2100–2109.

Rahm, Erhard and Philip A Bernstein (Dec. 2001). "A Survey of Approaches to Automatic Schema Matching". In: *VLDB J.* 10.4, pp. 334–350.

Rahm, Erhard and Hong Hai Do (2000). "Data cleaning: Problems and current approaches". In:

Rammelaere, J and F Geerts (2019a). "Cleaning Data with Forbidden Itemsets". In: *IEEE Trans. Knowl. Data Eng.* Pp. 1–1.

Rammelaere, Joeri and Floris Geerts (2019b). "Revisiting Conditional Functional Dependency Discovery: Splitting the "C" from the "FD"". In: *Machine Learning and Knowledge Discovery in Databases*. Springer International Publishing, pp. 552–568.

Rydning, David Reinsel-John Gantz-John (2018). "The digitization of the world from edge to core". In: *Framingham: International Data Corporation*.

Shapley, Lloyd S (1953). "A value for n-person games". In: *Contributions to the Theory of Games* 2.28, pp. 307–317.

Sheng, Victor S, Foster Provost, and Panagiotis G Ipeirotis (Aug. 2008). "Get another label? improving data quality and data mining using multiple, noisy labelers". In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD '08. Las Vegas, Nevada, USA: Association for Computing Machinery, pp. 614–622.

Stonebraker, Michael and Ihab F Ilyas (2018). "Data Integration: The Current Status and the Way Forward". In: *IEEE Data Eng. Bull.* 41.2, pp. 3–9.

Thirumuruganathan, Saravanan et al. (2017). "UGuide: User-Guided Discovery of FD-Detectable Errors". In: *Proceedings of the 2017 ACM International Conference on Management of Data*. SIGMOD '17. Chicago, Illinois, USA: ACM, pp. 1385–1397.

Trunk, G V (Mar. 1979). "A problem of dimensionality: a simple example". en. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 1.3, pp. 306–307.

Visengeriyeva, Larysa and Ziawasch Abedjan (2018). "Metadata-driven Error Detection". In: *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*. SSDBM '18. Bozen-Bolzano, Italy: ACM, 1:1–1:12.

Wang, Pei and Yeye He (2019). "Uni-Detect: A Unified Approach to Automated Error Detection in Tables". In: *Proceedings of the 2019 International Conference on Management of Data*. SIGMOD '19. Amsterdam, Netherlands: ACM, pp. 811–828.

Wikipedia contributors (2020). *Data cleansing — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Data_cleansing&oldid=973104849.